

CONVEX SCILIB User's Guide
Document No. 710-013630-001

First Edition
August 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX SCILIB User's Guide
Order No.DSW-360
First Edition

© 1991 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, stored electronically, or reduced to machine-readable form without prior written consent from CONVEX Computer Corporation.

Although the material contained herein has been carefully reviewed, CONVEX Computer Corporation (CONVEX) does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

UNLESS PROVIDED OTHERWISE IN WRITING WITH CONVEX COMPUTER CORPORATION (CONVEX), THE PROGRAM DESCRIBED HEREIN IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES. THE ABOVE EXCLUSION MAY NOT BE APPLICABLE TO ALL PURCHASERS BECAUSE WARRANTY RIGHTS CAN VARY FROM STATE TO STATE. IN NO EVENT WILL CONVEX BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS PROGRAM. CONVEX WILL NOT BE LIABLE EVEN IF IT HAS BEEN NOTIFIED OF THE POSSIBILITY OF SUCH DAMAGE BY THE PURCHASER OR ANY THIRD PARTY.

CONVEX and the CONVEX logo ("C"), and C200 Series architecture are registered trademarks of CONVEX Computer Corporation.
C200 Series and C1 are trademarks of CONVEX Computer Corporation.
VECLIB and ASAP are trademarks of CONVEX Computer Corporation.
ConvexOS is a trademark of CONVEX Computer Corporation.
IEEE is a trademark of the Institute of Electrical and Electronics Engineers, Inc.
VAX and VMS are trademarks of Digital Equipment Corporation.
VectorPak is a trademark of The Boeing Company.
Cray and UNICOS are registered trademarks of Cray Research, Inc.
UNIX is a registered trademark of AT&T Bell Laboratories.

Printed in the United States of America

1. The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that this is crucial for ensuring the integrity of the financial statements and for providing a clear audit trail.

2. The second part of the document outlines the various methods used to collect and analyze data. It includes a detailed description of the sampling process and the statistical techniques employed to interpret the results.

3. The third part of the document provides a comprehensive overview of the findings. It highlights the key areas where discrepancies were identified and discusses the potential causes of these issues.

4. The final part of the document offers recommendations for improving the internal control system. It suggests several practical measures that can be implemented to reduce the risk of errors and to enhance the overall reliability of the financial reporting process.

1. The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that this is crucial for ensuring the integrity of the financial statements and for providing a clear audit trail.

2. The second part of the document outlines the various methods used to collect and analyze data. It includes a detailed description of the sampling process and the statistical techniques employed to interpret the results.

3. The third part of the document provides a comprehensive overview of the findings. It highlights the key areas where discrepancies were identified and discusses the potential causes of these issues.

4. The final part of the document offers recommendations for improving the internal control system. It suggests several practical measures that can be implemented to reduce the risk of errors and to enhance the overall reliability of the financial reporting process.

Table of Contents

1 Introduction to SCILIB	
Overview	1-1
Chapter Objectives	1-1
What You Need to Know to Use SCILIB	1-2
Optimizations	1-2
Standardization	1-2
Relationship to VECLIB and VECLIB8 Libraries	1-2
IEEE Compatibility	1-3
Accessing SCILIB	1-3
Profiling with SCILIB	1-3
Optimizing with the Application Compiler	1-4
CONVEX Architecture	1-4
Roundoff Effects	1-6
SCILIB Data Types and Naming Conventions	1-7
Error Handling	1-8
SCILIB Programmer's Reference	1-8
Support Services	1-8
Public Domain Software	1-9
Supplemental Reading	1-10
2 Basic Vector Operations	
Overview	2-1
Chapter Objectives	2-1
What You Need to Know to Use These Subprograms	2-1
BLAS Storage Conventions	2-1
Supplemental Reading	2-4
Subprogram Descriptions	2-4
3 Basic Matrix Operations	
Overview	3-1
Chapter Objectives	3-1
What You Need to Know to Use These Subprograms	3-1
Subroutine Naming Convention	3-1
Supplemental Reading	3-3
Subprogram Descriptions	3-3
4 Linear Equations	
Overview	4-1
Chapter Objectives	4-1
What You Need to Know to Use These Subprograms	4-2
Subroutine Naming Convention	4-2
Condition Number	4-4
Determinant and Inverse	4-4
Supplemental Reading	4-5
Subprogram Descriptions	4-5
LINPACK Subprograms not in the <i>SCILIB User's Guide</i>	4-52
5 Eigenvalues and Eigenvectors	
Overview	5-1
Chapter Objectives	5-1
What You Need to Know to Use These Subprograms	5-1
Supplemental Reading	5-2
Subprogram Descriptions	5-2
EISPACK Subprograms not in the <i>SCILIB User's Guide</i>	5-13

6 Fast Fourier Transforms	
Overview	6-1
Chapter Objectives	6-1
What You Need to Know to Use These Subprograms	6-1
Supplemental Reading	6-2
Subprogram Descriptions	6-2
7 Correlation and Convolution Subprograms	
Overview	7-1
Chapter Objectives	7-1
What You Need to Know to Use These Subprograms	7-1
Supplemental Reading	7-1
Subprogram Descriptions	7-1
8 Linear Recurrences	
Overview	8-1
Chapter Objectives	8-1
What You Need to Know to Use These Subprograms	8-1
Subprogram Descriptions	8-1
9 Miscellaneous Routines	
Overview	9-1
Chapter Objectives	9-1
What You Need to Know to Use These Subprograms	9-1
Subprogram Descriptions	9-1

Appendices

A Reporting Problems	A-1
Technical Assistance Center	A-1
The <i>contact</i> Utility	A-1
Prerequisites	A-1
Tips on Using the <i>contact</i> Utility	A-2
Using the <i>contact</i> Utility	A-4

List of Tables

1-1 SCILIB Data Types and Naming Convention	1-7
3-1 Extended BLAS Naming Convention — Data Type	3-2
3-2 Extended BLAS Naming Convention — Matrix Form	3-2
3-3 Extended BLAS Naming Convention — Computation	3-2
3-4 Extended BLAS Naming Convention — Subprogram Names	3-3
4-1 LINPACK Naming Convention — Data Type	4-2
4-2 LINPACK Naming Convention — Form or Decomposition	4-2
4-3 LINPACK Naming Convention — Computation	4-3
4-4 LINPACK Naming Convention — Subprogram Names	4-3
4-5 LINPACK Subprograms not in the <i>SCILIB User's Guide</i>	4-52
5-1 EISPACK Subprograms not in the <i>SCILIB User's Guide</i>	5-13

Permuted Index

Sum of Absolute Values of the Elements of a Vector	2-SASUM	
ELMHES Accumulate the Transformations in the Reduction by	5-ELTRAN	
ORTHES Accumulate the Transformations in the Reduction by	5-ORTRAN	
Matrix-Vector Product Added to a Vector	3-SMXPY	
Vector-Matrix Product Added to a Vector	3-SXMPY	
	Apply a Givens Rotation	2-SROT
	Apply a Modified Givens Rotation	2-SROTM
Back Transform Eigenvectors following	BALANC	5-BALBAK
	Balance a Complex General Matrix	5-CBAL
	Balance a Real General Matrix	5-BALANC
Solve Linear Equations with a Triangular	Band Matrix	3-STBSV
Determinant of a General	Band Matrix	4-SGBDI
Factor a General	Band Matrix	4-SGBFA
Solve Linear Equations with a General	Band Matrix	4-SGBSL
Determinant of a Positive Definite	Band Matrix	4-SPBDI
Factor a Positive Definite	Band Matrix	4-SPBFA
Solve Linear Equations with a Positive Definite	Band Matrix	4-SPBSL
Determine Some Eigenvectors of a Real Symmetric	Band Matrix	5-BANDV
Determine Some Eigenvalues of a Real Symmetric	Band Matrix	5-BQR
Determine the Eigenvalues/vectors of a Real Symmetric	Band Matrix	5-RSB
Factor a General	Band Matrix and Estimate its Condition Number	4-SGBCO
Factor a Positive Definite	Band Matrix and Estimate its Condition Number	4-SPBCO
Reduce a Real Symmetric	Band Matrix to Real Symmetric Tridiagonal Form	5-BANDR
General	Band Matrix-Vector Multiply	3-SGBMV
Symmetric	Band Matrix-Vector Multiply	3-SSBMV
Triangular	Band Matrix-Vector Multiply	3-STBMV
Back Transform Eigenvectors following	CBAL	5-CBABK2
Initialization subprogram for	CFFTMLT	6-CFFTAX
Matrix Recompute the	Cholesky Decomposition of a DOWDATED Symmetric	4-SCHDD
Recompute the	Cholesky Decomposition of a Permuted Symmetric Matrix	4-SCHEX
Compute the	Cholesky Decomposition of a Symmetric Matrix	4-SCHDC
Recompute the	Cholesky Decomposition of an Updated Symmetric Matrix	4-SCHUD
Find Indices of	Clusters of Elements = the Target Within a Vector	2-CLUSEQ
Find Indices of	Clusters of Elements ≈ the Target Within a Vector	2-CLUSFGE
Find Indices of	Clusters of Elements > the Target Within a Vector	2-CLUSFGT
Find Indices of	Clusters of Elements ≤ the Target Within a Vector	2-CLUSFLE
Find Indices of	Clusters of Elements < the Target Within a Vector	2-CLUSFLT
Find Indices of	Clusters of Elements ≥ the Target Within a Vector	2-CLUSIGE
Find Indices of	Clusters of Elements > the Target Within a Vector	2-CLUSIGT
Find Indices of	Clusters of Elements ≤ the Target Within a Vector	2-CLUSILE
Find Indices of	Clusters of Elements < the Target Within a Vector	2-CLUSILT
Find Indices of	Clusters of Elements ≠ the Target Within a Vector	2-CLUSNE
Solve a Least Squares Problem with a Real Rectangular	Coefficient Matrix	5-MINFIT
First Order Linear Recurrence with Constant	Coefficients	8-FOLRC
Back Transform Eigenvectors following	COMHES	5-COMBAK
Eigenproblem	Complete the Reduction of a Real General Generalized	5-QZIT
Complex to	Complex Discrete Fourier Transform	6-CFFT2
Real to	Complex Discrete Fourier Transform	6-RCFFT2
Sets Complex to	Complex Discrete Fourier Transform of Multiple Data	6-CFFTMLT
Sets Real to	Complex Discrete Fourier Transform of Multiple Data	6-RFFTMLT
Balance a	Complex General Matrix	5-CBAL
Determine Eigenvalues/vectors of a	Complex General Matrix	5-CG
Form Reduce a	Complex General Matrix to Complex Upper Hessenberg	5-COMHES
Form Reduce a	Complex General Matrix to Complex Upper Hessenberg	5-CORTH
Determine Eigenvalues/vectors of a	Complex Hermitian Matrix	5-CH
Tridiagonal Form Reduce a	Complex Hermitian Matrix to Real Symmetric	5-HTRID3
Tridiagonal Form Reduce a	Complex Hermitian Matrix to Real Symmetric	5-HTRIDI
Determine the Eigenvalues/vectors of a	Complex Hessenberg Matrix	5-COMLR2
Multiple Data Sets	Complex to Complex Discrete Fourier Transform	6-CFFT2
	Complex to Complex Discrete Fourier Transform of	6-CFFTMLT
	Complex to Real Discrete Fourier Transform	6-CRFFT2
Reduce a Complex General Matrix to	Complex Upper Hessenberg Form	5-COMHES
Reduce a Complex General Matrix to	Complex Upper Hessenberg Form	5-CORTH
Determine Some Eigenvectors of a	Complex Upper Hessenberg Matrix	5-CINVT
Determine the Eigenvalues of a	Complex Upper Hessenberg Matrix	5-COMLR
Determine the Eigenvalues of a	Complex Upper Hessenberg Matrix	5-COMQR

Permuted Index

Determine the Eigenvalues/vectors of a	Complex Upper Hessenberg Matrix	5-COMQR2
Gather a Sparse Vector into	Compressed Form	2-GATHER
Matrix	Compute the Cholesky Decomposition of a Symmetric	4-SCHDC
Recurrence	Compute the Last Term of a First Order Linear	8-FOLRN
Recurrence	Compute the Last Term of a First Order Linear	8-FOLRNP
Recurrence	Compute the Last Term of a Second Order Linear	8-SOLRN
Rectangular Matrix	Compute the Singular Value Decomposition of a General	4-SSVDC
Rectangular Matrix	Compute the Singular Value Decomposition of a Real	5-SVD
Factor a General Band Matrix and Estimate its	Condition Number	4-SGBCO
Factor a General Matrix and Estimate its	Condition Number	4-SGECO
a Positive Definite Band Matrix and Estimate its	Condition Number Factor	4-SPBCO
Factor a Positive Definite Matrix and Estimate its	Condition Number	4-SPOCO
a Positive Definite Packed Matrix and Estimate its	Condition Number Factor	4-SPPCO
Factor a Symmetric Indefinite Matrix and Estimate its	Condition Number	4-SSICO
a Symmetric Indefinite Packed Matrix and Estimate its	Condition Number Factor	4-SSPCO
Estimate the	Condition Number of a Triangular Matrix	4-STRCO
First Order Linear Recurrence with	Constant Coefficients	8-FOLRC
	Construct a Givens Rotation	2-SROTG
	Construct a Modified Givens Rotation	2-SROTMG
	Convolution of Two Vectors	7-FILTERG
	Convolution of Two Vectors	7-FILTERS
	Copy a Vector	2-SCOPY
	CORTH	5-CORTB
Back Transform Eigenvectors following	Count Number of True or Negative Elements in a Vector	2-ILSUM
to Complex Discrete Fourier Transform of Multiple	Data Sets Complex	6-CFFTMLT
to Complex Discrete Fourier Transform of Multiple	Data Sets Real	6-RFFTMLT
Solve Linear Equations using the <i>QR</i>	Decomposition	4-SQRSL
Recompute the Cholesky	Decomposition of a DOWDATED Symmetric Matrix	4-SCHDD
<i>QR</i>	Decomposition of a General Rectangular Matrix	4-SQRDC
Compute the Singular Value	Decomposition of a General Rectangular Matrix	4-SSVDC
Recompute the Cholesky	Decomposition of a Permuted Symmetric Matrix	4-SCHEX
Compute the Singular Value	Decomposition of a Real Rectangular Matrix	5-SVD
Compute the Cholesky	Decomposition of a Symmetric Matrix	4-SCHDC
Recompute the Cholesky	Decomposition of an Updated Symmetric Matrix	4-SCHUD
	Determinant and Inverse of a General Matrix	4-MINV
	Determinant and Inverse of a General Matrix	4-SGEDI
	Determinant and Inverse of a Positive Definite Matrix	4-SPODI
	Determinant and Inverse of a Triangular Matrix	4-STRDI
Indefinite Matrix	Determinant, Inverse, and Inertia of a Symmetric	4-SSIDI
Indefinite Packed Matrix	Determinant, Inverse, and Inertia of a Symmetric	4-SSPDI
	Determinant of a General Band Matrix	4-SGBDI
	Determinant of a Positive Definite Band Matrix	4-SPBDI
	Determinant of a Positive Definite Packed Matrix	4-SPPDI
Real Symmetric Matrix	Determine All Eigenvalues and Some Eigenvectors of a	5-RSM
Matrix	Determine Eigenvalues/vectors of a Complex General	5-CG
Matrix	Determine Eigenvalues/vectors of a Complex Hermitian	5-CH
Matrix	Determine Some Eigenvalues of a Real Symmetric Band	5-BQR
Tridiagonal Matrix	Determine Some Eigenvalues of a Real Symmetric	5-TRIDIB
Symmetric Tridiagonal Matrix	Determine Some Eigenvalues/vectors of a Real	5-TSTURM
Hessenberg Matrix	Determine Some Eigenvectors of a Complex Upper	5-CINVT
Matrix	Determine Some Eigenvectors of a Real Symmetric Band	5-BANDV
Tridiagonal Matrix	Determine Some Eigenvectors of a Real Symmetric	5-BISECT
Tridiagonal Matrix	Determine Some Eigenvectors of a Real Symmetric	5-TINVT
Hessenberg Matrix	Determine Some Eigenvectors of a Real Upper	5-INVT
Symmetric Tridiagonal Matrix	Determine Some Extreme Eigenvalues of a Real	5-RATQR
Hessenberg Matrix	Determine the Eigenvalues of a Complex Upper	5-COMLR
Hessenberg Matrix	Determine the Eigenvalues of a Complex Upper	5-COMQR
Tridiagonal Matrix	Determine the Eigenvalues of a Real Symmetric	5-IMTQL1
Tridiagonal Matrix	Determine the Eigenvalues of a Real Symmetric	5-IMTQLV
Tridiagonal Matrix	Determine the Eigenvalues of a Real Symmetric	5-TQL1
Tridiagonal Matrix	Determine the Eigenvalues of a Real Symmetric	5-TQLRAT
Matrix	Determine the Eigenvalues of a Real Upper Hessenberg	5-HQR
Generalized Eigenproblem	Determine the Eigenvalues of a Reduced Real General	5-QZVAL
Hessenberg Matrix	Determine the Eigenvalues/vectors of a Complex	5-COMLR2
Hessenberg Matrix	Determine the Eigenvalues/vectors of a Complex Upper	5-COMQR2
Generalized Eigenproblem	Determine the Eigenvalues/vectors of a Real General	5-RGG
Matrix	Determine the Eigenvalues/vectors of a Real General	5-RG
Band Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-RSB
Generalized Eigenproblem	Determine the Eigenvalues/vectors of a Real Symmetric	5-RSG
Generalized Eigenproblem	Determine the Eigenvalues/vectors of a Real Symmetric	5-RSGAB
Generalized Eigenproblem	Determine the Eigenvalues/vectors of a Real Symmetric	5-RSGBA
Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-RS

Packed Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-RSP
Tridiagonal Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-IMTQL2
Tridiagonal Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-RST
Tridiagonal Matrix	Determine the Eigenvalues/vectors of a Real Symmetric	5-TQL2
Tridiagonal Matrix	Determine the Eigenvalues/vectors of a Real	5-RT
Hessenberg Matrix	Determine the Eigenvalues/vectors of a Real Upper	5-HQR2
Generalized Eigenproblem	Determine the Eigenvalues/vectors of a Reduced Real General	5-QZVEC
Complex to Complex	Discrete Fourier Transform	6-CFFT2
Complex to Real	Discrete Fourier Transform	6-CRFFT2
Real to Complex	Discrete Fourier Transform	6-RCFFT2
Complex to Complex	Discrete Fourier Transform of Multiple Data Sets	6-CFFTMLT
Real to Complex	Discrete Fourier Transform of Multiple Data Sets	6-RFFTMLT
	Dot Product of Two Vectors	2-SDOT
	Dot Product of Two Vectors	2-SPDOT
	Downdated Symmetric Matrix	5-QZHH
Recompute the Cholesky Decomposition of a Partially Reduce a Real General Generalized Eigenproblem	Eigenproblem	5-QZHES
Complete the Reduction of a Real General Generalized the Eigenvalues of a Reduced Real General Generalized Eigenproblem	Eigenproblem Determine	5-QZIT
Eigenvalues/vectors of a Real General Generalized the Eigenvalues/vectors of a Real General Generalized Eigenproblem	Eigenproblem Determine the	5-QZVAL
Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem	Eigenproblem Determine	5-RGVE
Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem	Eigenproblem Determine	5-RGG
Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem	Eigenproblem Determine the	5-RSG
Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem	Eigenproblem Determine the	5-RSGB
Reduce a Real Symmetric Generalized Eigenproblem	Eigenproblem Determine the	5-RSGBA
Reduce a Real Symmetric Generalized Matrix	Eigenproblem to Standard Form	5-REDUC
Determine All Eigenvalues and Some Eigenvectors of a Real Symmetric	Eigenvalues and Some Eigenvectors of a Real Symmetric	5-RSM
Determine the Eigenvalues of a Complex Upper Hessenberg Matrix	Eigenvalues of a Complex Upper Hessenberg Matrix	5-COMLR
Determine the Eigenvalues of a Complex Upper Hessenberg Matrix	Eigenvalues of a Complex Upper Hessenberg Matrix	5-COMQR
Determine Some Eigenvalues of a Real Symmetric Band Matrix	Eigenvalues of a Real Symmetric Band Matrix	5-BQR
Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-IMTQL1
Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-IMTQLV
Determine Some Extreme Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-RATQR
Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-TQL
Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-TQLRAT
Determine Some Eigenvalues of a Real Symmetric Tridiagonal Matrix	Eigenvalues of a Real Symmetric Tridiagonal Matrix	5-TRIDIB
Determine the Eigenvalues of a Real Upper Hessenberg Matrix	Eigenvalues of a Real Upper Hessenberg Matrix	5-HQR
Eigenproblem Determine the Eigenvalues of a Reduced Real General Generalized	Eigenvalues of a Reduced Real General Generalized	5-QZVAL
Determine the Eigenvalues/vectors of a Complex General Matrix	Eigenvalues/vectors of a Complex General Matrix	5-CG
Determine the Eigenvalues/vectors of a Complex Hermitian Matrix	Eigenvalues/vectors of a Complex Hermitian Matrix	5-CH
Determine the Eigenvalues/vectors of a Complex Hessenberg Matrix	Eigenvalues/vectors of a Complex Hessenberg Matrix	5-COMLR2
Matrix Determine the Eigenvalues/vectors of a Complex Upper Hessenberg	Eigenvalues/vectors of a Complex Upper Hessenberg	5-COMQR2
Eigenproblem Determine the Eigenvalues/vectors of a Real General Generalized	Eigenvalues/vectors of a Real General Generalized	5-RGG
Determine the Eigenvalues/vectors of a Real General Matrix	Eigenvalues/vectors of a Real General Matrix	5-RG
Determine the Eigenvalues/vectors of a Real Symmetric Band Matrix	Eigenvalues/vectors of a Real Symmetric Band Matrix	5-RSB
Eigenproblem Determine the Eigenvalues/vectors of a Real Symmetric Generalized	Eigenvalues/vectors of a Real Symmetric Generalized	5-RSG
Eigenproblem Determine the Eigenvalues/vectors of a Real Symmetric Generalized	Eigenvalues/vectors of a Real Symmetric Generalized	5-RSGB
Eigenproblem Determine the Eigenvalues/vectors of a Real Symmetric Generalized	Eigenvalues/vectors of a Real Symmetric Generalized	5-RSGBA
Determine the Eigenvalues/vectors of a Real Symmetric Matrix	Eigenvalues/vectors of a Real Symmetric Matrix	5-RS
Determine the Eigenvalues/vectors of a Real Symmetric Packed Matrix	Eigenvalues/vectors of a Real Symmetric Packed Matrix	5-RSP
Matrix Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal	Eigenvalues/vectors of a Real Symmetric Tridiagonal	5-IMTQL2
Matrix Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal	Eigenvalues/vectors of a Real Symmetric Tridiagonal	5-RST
Matrix Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal	Eigenvalues/vectors of a Real Symmetric Tridiagonal	5-TQL2
Matrix Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal	Eigenvalues/vectors of a Real Symmetric Tridiagonal	5-TSTURM
Determine the Eigenvalues/vectors of a Real Tridiagonal Matrix	Eigenvalues/vectors of a Real Tridiagonal Matrix	5-RT
Determine the Eigenvalues/vectors of a Real Upper Hessenberg Matrix	Eigenvalues/vectors of a Real Upper Hessenberg Matrix	5-HQR2
Back Transform Eigenvectors following BALANC	Eigenvectors following BALANC	5-BALBAK
Back Transform Eigenvectors following CBAL	Eigenvectors following CBAL	5-CBBAK2
Back Transform Eigenvectors following COMHES	Eigenvectors following COMHES	5-COMBAK
Back Transform Eigenvectors following CORTH	Eigenvectors following CORTH	5-CORTB
Back Transform Eigenvectors following ELMHES	Eigenvectors following ELMHES	5-ELMBAK
Back Transform Eigenvectors following FIGI	Eigenvectors following FIGI	5-BAKVEC
Back Transform Eigenvectors following HTRID3	Eigenvectors following HTRID3	5-HTRIB3
Back Transform Eigenvectors following HTRIDI	Eigenvectors following HTRIDI	5-HTRIBK
Back Transform Eigenvectors following ORTHES	Eigenvectors following ORTHES	5-ORTBAK
Back Transform Eigenvectors following REDUC or REDUC2	Eigenvectors following REDUC or REDUC2	5-REBAK
Back Transform Eigenvectors following REDUC2	Eigenvectors following REDUC2	5-REBAKB
Back Transform Eigenvectors following TRED1	Eigenvectors following TRED1	5-TREBAK1
Back Transform Eigenvectors following TRED3	Eigenvectors following TRED3	5-TREBAK3
Determine Some Eigenvectors of a Complex Upper Hessenberg Matrix	Eigenvectors of a Complex Upper Hessenberg Matrix	5-CINVIT
Determine Some Eigenvectors of a Real Symmetric Band Matrix	Eigenvectors of a Real Symmetric Band Matrix	5-BANDV
Determine All Eigenvalues and Some Eigenvectors of a Real Symmetric Matrix	Eigenvectors of a Real Symmetric Matrix	5-RSM
Determine Some Eigenvectors of a Real Symmetric Tridiagonal Matrix	Eigenvectors of a Real Symmetric Tridiagonal Matrix	5-BISECT

Determine Some	Eigenvectors of a Real Symmetric Tridiagonal Matrix . . .	5-TINVIT
Determine Some	Eigenvectors of a Real Upper Hessenberg Matrix	5-INVIT
Eigenproblem Determine the	Eigenvectors of a Reduced Real General Generalized . . .	5-QZVEC
	Elementary Vector Operation	2-SAXPY
	Elementary Vector Operation	2-SPAXPY
	ELMHES	5-ELMBAK
	ELMHES	5-ELTRAN
Back Transform Eigenvectors following	Equalities Search	2-OSRCHF
Accumulate the Transformations in the Reduction by	Equalities Search	2-OSRCHI
Ordered Vector for First Equality and Number of	Equalities Search Ordered	2-OSRCHM
Ordered Vector for First Equality and Number of	Equality and Number of Equalities	2-OSRCHF
Masked Vector for First Equality and Number of	Equality and Number of Equalities	2-OSRCHI
Search Ordered Vector for First	Equality and Number of Equalities	2-OSRCHM
Search Ordered Vector for First	Equation	4-OPFLT
Search Ordered Masked Vector for First	Equations	3-STPSV
Solve Weiner-Levinson Optimal Filter	Equations using the <i>QR</i> Decomposition	4-SQRSL
Solve Triangular Packed Linear	Equations with a General Band Matrix	4-SGBSL
Solve Linear	Equations with a General Matrix	4-SGESL
Solve Linear	Equations with a General Tridiagonal Matrix	4-SGTSL
Solve Linear	Equations with a Positive Definite Band Matrix	4-SPBSL
Solve Linear	Equations with a Positive Definite Matrix	4-SPOSL
Solve Linear	Equations with a Positive Definite Packed Matrix	4-SPPSL
Solve Linear	Equations with a Positive Definite Tridiagonal Matrix	4-SPTSL
Solve Linear	Equations with a Symmetric Indefinite Matrix	4-SSISL
Solve Linear	Equations with a Symmetric Indefinite Packed Matrix	4-SSPSL
Solve Linear	Equations with a Triangular Band Matrix	3-STBSV
Solve Multiple Sets of Linear	Equations with a Triangular Matrix	3-STRSM
Solve Linear	Equations with a Triangular Matrix	3-STRSV
Solve Linear	Equations with a Triangular Matrix	4-STRSL
	Error routine	3-XERBLA
	Error routine	9-XERSCI
Factor a General Band Matrix and	Estimate its Condition Number	4-SGBCO
Factor a General Matrix and	Estimate its Condition Number	4-SGECO
Factor a Positive Definite Band Matrix and	Estimate its Condition Number	4-SPBCO
Factor a Positive Definite Matrix and	Estimate its Condition Number	4-SPOCO
Factor a Positive Definite Packed Matrix and	Estimate its Condition Number	4-SPPCO
Factor a Symmetric Indefinite Matrix and	Estimate its Condition Number	4-SSICO
Factor a Symmetric Indefinite Packed Matrix and	Estimate its Condition Number	4-SSPCO
	Estimate the Condition Number of a Triangular Matrix	4-STRCO
	Euclidean Norm of a Vector	2-SNRM2
Matrix Determine Some	Extreme Eigenvalues of a Real Symmetric Tridiagonal	5-RATQR
	Factor a General Band Matrix	4-SGBFA
Condition Number	Factor a General Band Matrix and Estimate its	4-SGBCO
	Factor a General Matrix	4-SGEFA
Number	Factor a General Matrix and Estimate its Condition	4-SGECO
	Factor a Positive Definite Band Matrix	4-SPBFA
its Condition Number	Factor a Positive Definite Band Matrix and Estimate	4-SPBCO
	Factor a Positive Definite Matrix	4-SPOFA
Condition Number	Factor a Positive Definite Matrix and Estimate its	4-SPOCO
	Factor a Positive Definite Packed Matrix	4-SPPFA
its Condition Number	Factor a Positive Definite Packed Matrix and Estimate	4-SPPCO
	Factor a Symmetric Indefinite Matrix	4-SSIFA
Condition Number	Factor a Symmetric Indefinite Matrix and Estimate its	4-SSICO
	Factor a Symmetric Indefinite Packed Matrix	4-SSPFA
Estimate its Condition Number	Factor a Symmetric Indefinite Packed Matrix and	4-SSPCO
Number of Leading	False or Positive Elements in a Vector	2-ILLZ
Back Transform Eigenvectors following	FIGI	5-BAKVEC
Solve Weiner-Levinson Optimal	Filter Equation	4-OPFLT
Vector	Find Indices of All Elements = the Target Within a	2-WHENEQ
Vector	Find Indices of All Elements \geq the Target Within a	2-WHENFGE
Vector	Find Indices of All Elements $>$ the Target Within a	2-WHENFGT
Vector	Find Indices of All Elements \leq the Target Within a	2-WHENFLE
Vector	Find Indices of All Elements $<$ the Target Within a	2-WHENFLT
Vector	Find Indices of All Elements \geq the Target Within a	2-WHENIGE
Vector	Find Indices of All Elements $>$ the Target Within a	2-WHENIGT
Vector	Find Indices of All Elements \leq the Target Within a	2-WHENILE
Vector	Find Indices of All Elements $<$ the Target Within a	2-WHENILT
Vector	Find Indices of All Elements \neq the Target Within a	2-WHENNE
Within a Vector	Find Indices of All Masked Elements = the Target	2-WHENMEQ
Within a Vector	Find Indices of All Masked Elements \geq the Target	2-WHENMGE
Within a Vector	Find Indices of All Masked Elements $>$ the Target	2-WHENMGT
Within a Vector	Find Indices of All Masked Elements \leq the Target	2-WHENMLE

Within a Vector	Find Indices of All Masked Elements < the Target	2-WHENMLT
Within a Vector	Find Indices of All Masked Elements ≠ the Target	2-WHENMNE
Within a Vector	Find Indices of Clusters of Elements = the Target	2-CLUSEQ
Within a Vector	Find Indices of Clusters of Elements ≥ the Target	2-CLUSFGE
Within a Vector	Find Indices of Clusters of Elements > the Target	2-CLUSFGT
Within a Vector	Find Indices of Clusters of Elements ≤ the Target	2-CLUSFLE
Within a Vector	Find Indices of Clusters of Elements < the Target	2-CLUSFLT
Within a Vector	Find Indices of Clusters of Elements ≥ the Target	2-CLUSIGE
Within a Vector	Find Indices of Clusters of Elements > the Target	2-CLUSIGT
Within a Vector	Find Indices of Clusters of Elements ≤ the Target	2-CLUSILE
Within a Vector	Find Indices of Clusters of Elements < the Target	2-CLUSILT
Within a Vector	Find Indices of Clusters of Elements ≠ the Target	2-CLUSNE
a Vector	Find the Index of the Element of Maximum Magnitude of	2-ISAMAX
Vector	Find the Index of the Element of Maximum Value of a	2-ISMAX
a Vector	Find the Index of the Element of Minimum Magnitude of	2-ISAMIN
Vector	Find the Index of the Element of Minimum Value of a	2-ISMIN
	Find the Index of the First Element = the Objective	2-ISRCH EQ
Objective	Find the Index of the First Element ≥ the	2-ISRCHFGE
	Find the Index of the First Element > the Objective	2-ISRCHF GT
Objective	Find the Index of the First Element ≤ the	2-ISRCHFLE
	Find the Index of the First Element < the Objective	2-ISRCHF LT
Objective	Find the Index of the First Element ≥ the	2-ISRCHIGE
	Find the Index of the First Element > the Objective	2-ISRCHIGT
Objective	Find the Index of the First Element ≤ the	2-ISRCHILE
	Find the Index of the First Element < the Objective	2-ISRCHILT
Objective	Find the Index of the First Element ≠ the	2-ISRCHNE
Objective	Find the Index of the First Masked Element = the	2-ISRCHMEQ
Objective	Find the Index of the First Masked Element ≥ the	2-ISRCHMGE
Objective	Find the Index of the First Masked Element > the	2-ISRCHMGT
Objective	Find the Index of the First Masked Element ≤ the	2-ISRCHMLE
Objective	Find the Index of the First Masked Element < the	2-ISRCHMLT
Objective	Find the Index of the First Masked Element ≠ the	2-ISRCHMNE
of an Integer Vector	Find the Index of the Masked Element of Maximum Value	2-INFLMAX
of an Integer Vector	Find the Index of the Masked Element of Minimum Value	2-INFLMIN
Solve Linear Equations using the	QR Decomposition	4-SQRSL
Matrix	QR Decomposition of a General Rectangular	4-SQRDC
Back Transform Eigenvectors	following BALANC	5-BALBAK
Back Transform Eigenvectors	following CBAL	5-CBABK2
Back Transform Eigenvectors	following COMHES	5-COMBAK
Back Transform Eigenvectors	following CORTH	5-CORTB
Back Transform Eigenvectors	following ELMHES	5-ELMBAK
Back Transform Eigenvectors	following FIGI	5-BAKVEC
Back Transform Eigenvectors	following HTRID3	5-HTRIB3
Back Transform Eigenvectors	following HTRIDI	5-HTRIBK
Back Transform Eigenvectors	following ORTHES	5-ORTBAK
Back Transform Eigenvectors	following REDUC or REDUC2	5-REBAK
Back Transform Eigenvectors	following REDUC2	5-REBAKB
Back Transform Eigenvectors	following TRED1	5-TRBAK1
Back Transform Eigenvectors	following TRED3	5-TRBAK3
Complex to Complex Discrete	Fourier Transform	6-CFFT2
Complex to Real Discrete	Fourier Transform	6-CRFFT2
Real to Complex Discrete	Fourier Transform	6-RCFFT2
Complex to Complex Discrete	Fourier Transform of Multiple Data Sets	6-CFFTMLT
Real to Complex Discrete	Fourier Transform of Multiple Data Sets	6-RFFTMLT
Scatter a Sparse Vector into	Full Form	2-SCATTER
	Gather a Sparse Vector into Compressed Form	2-GATHER
Determinant of a	General Band Matrix	4-SGBDI
Factor a	General Band Matrix	4-SGBFA
Solve Linear Equations with a	General Band Matrix	4-SGBSL
Factor a	General Band Matrix and Estimate its Condition Number	4-SGBCO
	General Band Matrix-Vector Multiply	3-SGBMV
Partially Reduce a Real	General Generalized Eigenproblem	5-QZHES
Complete the Reduction of a Real	General Generalized Eigenproblem	5-QZIT
Determine the Eigenvalues of a Reduced Real	General Generalized Eigenproblem	5-QZVAL
Determine the Eigenvectors of a Reduced Real	General Generalized Eigenproblem	5-QZVEC
Determine the Eigenvalues/vectors of a Real	General Generalized Eigenproblem	5-RGG
Determinant and Inverse of a	General Matrix	4-MINV
Determinant and Inverse of a	General Matrix	4-SGEDI
Factor a	General Matrix	4-SGEFA
Solve Linear Equations with a	General Matrix	4-SGESL
Balance a Real	General Matrix	5-BALANC
Balance a Complex	General Matrix	5-CBAL

Permuted Index

Determine Eigenvalues/vectors of a Complex	General Matrix	5-CG
Determine the Eigenvalues/vectors of a Real	General Matrix	5-RG
Factor a	General Matrix and Estimate its Condition Number	4-SGECO
General Matrix-Matrix Multiplication,	General Matrix Storage	3-MXMA
General Matrix-Vector Multiplication,	General Matrix Storage	3-MXVA
Reduce a Complex	General Matrix to Complex Upper Hessenberg Form	5-COMHES
Reduce a Complex	General Matrix to Complex Upper Hessenberg Form	5-CORTH
Reduce a Real	General Matrix to Real Upper Hessenberg Form	5-ELMHES
Reduce a Real	General Matrix to Real Upper Hessenberg Form	5-ORTHES
	General Matrix-Matrix Multiplication	3-MXM
Storage	General Matrix-Matrix Multiplication, General Matrix	3-MXMA
	General Matrix-Vector Multiply	3-SGEMM
Method	General Matrix-Matrix Multiply using Strassen's	3-SGEMMS
	General Matrix-Vector Multiplication	3-MXV
Storage	General Matrix-Vector Multiplication, General Matrix	3-MXVA
	General Matrix-Vector Multiply	3-SGEMV
	General Rank-1 Update	3-SGER
<i>QR</i> Decomposition of a	General Rectangular Matrix	4-SQRDC
Compute the Singular Value Decomposition of a	General Rectangular Matrix	4-SSVDC
Solve Linear Equations with a	General Tridiagonal Matrix	4-SGTSL
Partially Reduce a Real General	Generalized Eigenproblem	5-QZHES
Complete the Reduction of a Real General	Generalized Eigenproblem	5-QZIT
Determine the Eigenvalues of a Reduced Real General	Generalized Eigenproblem	5-QZVAL
Determine the Eigenvectors of a Reduced Real General	Generalized Eigenproblem	5-QZVEC
Determine the Eigenvalues/vectors of a Real General	Generalized Eigenproblem	5-RGG
Determine the Eigenvalues/vectors of a Real Symmetric	Generalized Eigenproblem	5-RSG
Determine the Eigenvalues/vectors of a Real Symmetric	Generalized Eigenproblem	5-RSGAB
Determine the Eigenvalues/vectors of a Real Symmetric	Generalized Eigenproblem	5-RSGBA
Reduce a Real Symmetric	Generalized Eigenproblem to Standard Form	5-REDUC
Reduce a Real Symmetric	Generalized Eigenproblem to Standard Form	5-REDUC2
Apply a	Givens Rotation	2-SROT
Construct a	Givens Rotation	2-SROTG
Apply a Modified	Givens Rotation	2-SROTM
Construct a Modified	Givens Rotation	2-SROTMG
Determine Eigenvalues/vectors of a Complex	Hermitian Matrix	5-CH
Reduce a Complex	Hermitian Matrix to Real Symmetric Tridiagonal Form	5-HTRID3
Reduce a Complex	Hermitian Matrix to Real Symmetric Tridiagonal Form	5-HTRIDI
Reduce a Complex General Matrix to Complex Upper	Hessenberg Form	5-COMHES
Reduce a Complex General Matrix to Complex Upper	Hessenberg Form	5-CORTH
Reduce a Real General Matrix to Real Upper	Hessenberg Form	5-ELMHES
Reduce a Real General Matrix to Real Upper	Hessenberg Form	5-ORTHES
Determine Some Eigenvectors of a Complex Upper	Hessenberg Matrix	5-CINVT
Determine the Eigenvalues of a Complex Upper	Hessenberg Matrix	5-COMLR
Determine the Eigenvalues/vectors of a Complex	Hessenberg Matrix	5-COMLR2
Determine the Eigenvalues of a Complex Upper	Hessenberg Matrix	5-COMQR
Determine the Eigenvalues/vectors of a Complex Upper	Hessenberg Matrix	5-COMQR2
Determine the Eigenvalues of a Real Upper	Hessenberg Matrix	5-HQR
Determine the Eigenvalues/vectors of a Real Upper	Hessenberg Matrix	5-HQR2
Determine Some Eigenvectors of a Real Upper	Hessenberg Matrix	5-INVIT
Back Transform Eigenvectors following	HTRID3	5-HTRIB3
Back Transform Eigenvectors following	HTRIDI	5-HTRIBK
Determinant, Inverse, and Inertia of a Symmetric	Indefinite Matrix	4-SSIDI
Factor a Symmetric	Indefinite Matrix	4-SSIFA
Solve Linear Equations with a Symmetric	Indefinite Matrix	4-SSISL
Factor a Symmetric	Indefinite Matrix and Estimate its Condition Number	4-SSICO
Determinant, Inverse, and Inertia of a Symmetric	Indefinite Packed Matrix	4-SSPDI
Factor a Symmetric	Indefinite Packed Matrix	4-SSPFA
Solve Linear Equations with a Symmetric	Indefinite Packed Matrix	4-SSPSL
Number Factor a Symmetric	Indefinite Packed Matrix and Estimate its Condition	4-SSPCO
Find the	Index of the Element of Maximum Magnitude of a Vector	2-ISAMAX
Find the	Index of the Element of Maximum Value of a Vector	2-ISMAX
Find the	Index of the Element of Minimum Magnitude of a Vector	2-ISAMIN
Find the	Index of the Element of Minimum Value of a Vector	2-ISMIN
Find the	Index of the First Element = the Objective	2-ISRCHCQ
Find the	Index of the First Element \geq the Objective	2-ISRCHFG
Find the	Index of the First Element $>$ the Objective	2-ISRCHFGT
Find the	Index of the First Element \leq the Objective	2-ISRCHFL
Find the	Index of the First Element $<$ the Objective	2-ISRCHFLT
Find the	Index of the First Element \geq the Objective	2-ISRCHIGE
Find the	Index of the First Element $>$ the Objective	2-ISRCHIGT
Find the	Index of the First Element \leq the Objective	2-ISRCHILE
Find the	Index of the First Element $<$ the Objective	2-ISRCHILT

Find the	Index of the First Element \neq the Objective	2-ISRCHNE
Find the	Index of the First Masked Element = the Objective	2-ISRCHMEQ
Find the	Index of the First Masked Element \geq the Objective	2-ISRCHMGE
Find the	Index of the First Masked Element $>$ the Objective	2-ISRCHMGT
Find the	Index of the First Masked Element \leq the Objective	2-ISRCHMLE
Find the	Index of the First Masked Element $<$ the Objective	2-ISRCHMLT
Find the	Index of the First Masked Element \neq the Objective	2-ISRCHMNE
Integer Vector Find the	Index of the Masked Element of Maximum Value of an	2-INFLMAX
Integer Vector Find the	Index of the Masked Element of Minimum Value of an	2-INFLMIN
Find	Indices of All Elements = the Target Within a Vector	2-WHENEQ
Vector Find	Indices of All Elements \geq the Target Within a	2-WHENFGE
Find	Indices of All Elements $>$ the Target Within a Vector	2-WHENFGT
Vector Find	Indices of All Elements \leq the Target Within a	2-WHENFLE
Find	Indices of All Elements $<$ the Target Within a Vector	2-WHENFLT
Vector Find	Indices of All Elements \geq the Target Within a	2-WHENIGE
Find	Indices of All Elements $>$ the Target Within a Vector	2-WHENIGT
Vector Find	Indices of All Elements \leq the Target Within a	2-WHENILE
Find	Indices of All Elements $<$ the Target Within a Vector	2-WHENILT
Vector Find	Indices of All Elements \neq the Target Within a	2-WHENNE
Vector Find	Indices of All Masked Elements = the Target Within a	2-WHENMEQ
a Vector Find	Indices of All Masked Elements \geq the Target Within	2-WHENMGE
Vector Find	Indices of All Masked Elements $>$ the Target Within a	2-WHENMGT
a Vector Find	Indices of All Masked Elements \leq the Target Within	2-WHENMLE
Vector Find	Indices of All Masked Elements $<$ the Target Within a	2-WHENMLT
a Vector Find	Indices of All Masked Elements \neq the Target Within	2-WHENMNE
Vector Find	Indices of Clusters of Elements = the Target Within a	2-CLUSEQ
Within a Vector Find	Indices of Clusters of Elements \geq the Target	2-CLUSFGE
Vector Find	Indices of Clusters of Elements $>$ the Target Within a	2-CLUSFGT
Within a Vector Find	Indices of Clusters of Elements \leq the Target	2-CLUSFLE
Vector Find	Indices of Clusters of Elements $<$ the Target Within a	2-CLUSFLT
Within a Vector Find	Indices of Clusters of Elements \geq the Target	2-CLUSIGE
Vector Find	Indices of Clusters of Elements $>$ the Target Within a	2-CLUSIGT
Within a Vector Find	Indices of Clusters of Elements \leq the Target	2-CLUSILE
Vector Find	Indices of Clusters of Elements $<$ the Target Within a	2-CLUSILT
Within a Vector Find	Indices of Clusters of Elements \neq the Target	2-CLUSINE
Determinant, Inverse, and	Inertia of a Symmetric Indefinite Matrix	4-SSIDI
Determinant, Inverse, and	Inertia of a Symmetric Indefinite Packed Matrix	4-SSPDI
	Initialization subprogram for CFFTMLT	6-CFTFAX
	Initialization subprogram for RFFTMLT	6-FFTFAQ
Index of the Masked Element of Maximum Value of an	Integer Vector Find the	2-INFLMAX
Index of the Masked Element of Minimum Value of an	Integer Vector Find the	2-INFLMIN
Determinant,	Inverse, and Inertia of a Symmetric Indefinite Matrix	4-SSIDI
Matrix Determinant,	Inverse, and Inertia of a Symmetric Indefinite Packed	4-SSPDI
Determinant and	Inverse of a General Matrix	4-MINV
Determinant and	Inverse of a General Matrix	4-SGEDI
Determinant and	Inverse of a Positive Definite Matrix	4-SPODI
Determinant and	Inverse of a Triangular Matrix	4-STRDI
Number of	Leading False or Positive Elements in a Vector	2-ILLZ
Number of	Leading True or Negative Elements in a Vector	2-IILZ
Coefficient Matrix Solve a	Least Squares Problem with a Real Rectangular	5-MINFIT
Solve Triangular Packed	Linear Equations	3-STPSV
Solve	Linear Equations using the QR Decomposition	4-SQRSL
Solve	Linear Equations with a General Band Matrix	4-SGBSL
Solve	Linear Equations with a General Matrix	4-SGESL
Solve	Linear Equations with a General Tridiagonal Matrix	4-SGTSL
Solve	Linear Equations with a Positive Definite Band Matrix	4-SPBSL
Solve	Linear Equations with a Positive Definite Matrix	4-SPOS
Matrix Solve	Linear Equations with a Positive Definite Packed	4-SPPSL
Matrix Solve	Linear Equations with a Positive Definite Tridiagonal	4-SPTSL
Solve	Linear Equations with a Symmetric Indefinite Matrix	4-SSISL
Matrix Solve	Linear Equations with a Symmetric Indefinite Packed	4-SSPSL
Solve	Linear Equations with a Triangular Band Matrix	3-STBSV
Solve Multiple Sets of	Linear Equations with a Triangular Matrix	3-STRSM
Solve	Linear Equations with a Triangular Matrix	3-STRSV
Solve	Linear Equations with a Triangular Matrix	4-STRSL
First Order	Linear Recurrence	8-FOLR
First Order	Linear Recurrence	8-FOLR2
First Order	Linear Recurrence	8-FOLR2P
Compute the Last Term of a First Order	Linear Recurrence	8-FOLRN
Compute the Last Term of a First Order	Linear Recurrence	8-FOLRNP
First Order	Linear Recurrence	8-FOLRP
Second Order	Linear Recurrence	8-SOLR

Permuted Index

	Second Order	Linear Recurrence	8-SOLR3
Compute the Last Term of a Second Order	Linear Recurrence	Linear Recurrence	8-SOLRN
	First Order	Linear Recurrence with Constant Coefficients	8-FOLRC
Find the Index of the Element of Maximum	Magnitude of a Vector	Magnitude of a Vector	2-ISAMAX
Find the Index of the Element of Minimum	Magnitude of a Vector	Magnitude of a Vector	2-ISAMIN
Find the Index of the First	Masked Element = the Objective	Masked Element = the Objective	2-ISRCHMEQ
Find the Index of the First	Masked Element ≥ the Objective	Masked Element ≥ the Objective	2-ISRCHMGE
Find the Index of the First	Masked Element > the Objective	Masked Element > the Objective	2-ISRCHMGT
Find the Index of the First	Masked Element ≤ the Objective	Masked Element ≤ the Objective	2-ISRCHMLE
Find the Index of the First	Masked Element < the Objective	Masked Element < the Objective	2-ISRCHMLT
Find the Index of the First	Masked Element ≠ the Objective	Masked Element ≠ the Objective	2-ISRCHMNE
Find the Index of the	Masked Element of Maximum Value of an Integer Vector	Masked Element of Maximum Value of an Integer Vector	2-INFLMAX
Find the Index of the	Masked Element of Minimum Value of an Integer Vector	Masked Element of Minimum Value of an Integer Vector	2-INFLMIN
Find Indices of All	Masked Elements = the Target Within a Vector	Masked Elements = the Target Within a Vector	2-WHENMEQ
Find Indices of All	Masked Elements ≥ the Target Within a Vector	Masked Elements ≥ the Target Within a Vector	2-WHENMGE
Find Indices of All	Masked Elements > the Target Within a Vector	Masked Elements > the Target Within a Vector	2-WHENMGT
Find Indices of All	Masked Elements ≤ the Target Within a Vector	Masked Elements ≤ the Target Within a Vector	2-WHENMLE
Find Indices of All	Masked Elements < the Target Within a Vector	Masked Elements < the Target Within a Vector	2-WHENMLT
Find Indices of All	Masked Elements ≠ the Target Within a Vector	Masked Elements ≠ the Target Within a Vector	2-WHENMNE
Equalities Search Ordered	Masked Vector for First Equality and Number of	Masked Vector for First Equality and Number of	2-OSRCHM
Solve Linear Equations with a Triangular Band	Matrix	Matrix	3-STBSV
Multiple Sets of Linear Equations with a Triangular	Matrix Solve	Matrix Solve	3-STRSM
Solve Linear Equations with a Triangular	Matrix	Matrix	3-STRSV
Determinant and Inverse of a General	Matrix	Matrix	4-MINV
Compute the Cholesky Decomposition of a Symmetric	Matrix	Matrix	4-SCHDC
the Cholesky Decomposition of a DOWndated Symmetric	Matrix Recompute	Matrix Recompute	4-SCHDD
the Cholesky Decomposition of a Permuted Symmetric	Matrix Recompute	Matrix Recompute	4-SCHEX
the Cholesky Decomposition of an Updated Symmetric	Matrix Recompute	Matrix Recompute	4-SCHUD
Determinant of a General Band	Matrix	Matrix	4-SGBDI
Factor a General Band	Matrix	Matrix	4-SGBFA
Solve Linear Equations with a General Band	Matrix	Matrix	4-SGBSL
Determinant and Inverse of a General	Matrix	Matrix	4-SGEDI
Factor a General	Matrix	Matrix	4-SGEFA
Solve Linear Equations with a General	Matrix	Matrix	4-SGESL
Solve Linear Equations with a General Tridiagonal	Matrix	Matrix	4-SGTSL
Determinant of a Positive Definite Band	Matrix	Matrix	4-SPBDI
Factor a Positive Definite Band	Matrix	Matrix	4-SPBFA
Solve Linear Equations with a Positive Definite Band	Matrix	Matrix	4-SPBSL
Determinant and Inverse of a Positive Definite	Matrix	Matrix	4-SPODI
Factor a Positive Definite	Matrix	Matrix	4-SPOFA
Solve Linear Equations with a Positive Definite	Matrix	Matrix	4-SPOSL
Determinant of a Positive Definite Packed	Matrix	Matrix	4-SPPDI
Factor a Positive Definite Packed	Matrix	Matrix	4-SPPFA
Linear Equations with a Positive Definite Packed	Matrix Solve	Matrix Solve	4-SPPSL
Linear Equations with a Positive Definite Tridiagonal	Matrix Solve	Matrix Solve	4-SPTSL
QR Decomposition of a General Rectangular	Matrix	Matrix	4-SQRDC
Inverse, and Inertia of a Symmetric Indefinite	Matrix Determinant,	Matrix Determinant,	4-SSIDI
Factor a Symmetric Indefinite	Matrix	Matrix	4-SSIFA
Solve Linear Equations with a Symmetric Indefinite	Matrix	Matrix	4-SSISL
Inverse, and Inertia of a Symmetric Indefinite Packed	Matrix Determinant,	Matrix Determinant,	4-SSPDI
Factor a Symmetric Indefinite Packed	Matrix	Matrix	4-SSPFA
Linear Equations with a Symmetric Indefinite Packed	Matrix Solve	Matrix Solve	4-SSPSL
Singular Value Decomposition of a General Rectangular	Matrix Compute the	Matrix Compute the	4-SSVDC
Estimate the Condition Number of a Triangular	Matrix	Matrix	4-STRCO
Determinant and Inverse of a Triangular	Matrix	Matrix	4-STRDI
Solve Linear Equations with a Triangular	Matrix	Matrix	4-STRSL
Balance a Real General	Matrix	Matrix	5-BALANC
Determine Some Eigenvectors of a Real Symmetric Band	Matrix	Matrix	5-BANDV
Some Eigenvectors of a Real Symmetric Tridiagonal	Matrix Determine	Matrix Determine	5-BISECT
Determine Some Eigenvalues of a Real Symmetric Band	Matrix	Matrix	5-BQR
Balance a Complex General	Matrix	Matrix	5-CBAL
Determine Eigenvalues/vectors of a Complex General	Matrix	Matrix	5-CG
Determine Eigenvalues/vectors of a Complex Hermitian	Matrix	Matrix	5-CH
Some Eigenvectors of a Complex Upper Hessenberg	Matrix Determine	Matrix Determine	5-CINVIT
the Eigenvalues of a Complex Upper Hessenberg	Matrix Determine	Matrix Determine	5-COMLR
the Eigenvalues/vectors of a Complex Hessenberg	Matrix Determine	Matrix Determine	5-COMLR2
the Eigenvalues of a Complex Upper Hessenberg	Matrix Determine	Matrix Determine	5-COMQR
the Eigenvalues/vectors of a Complex Upper Hessenberg	Matrix Determine	Matrix Determine	5-COMQR2
Determine the Eigenvalues of a Real Upper Hessenberg	Matrix	Matrix	5-HQR
the Eigenvalues/vectors of a Real Upper Hessenberg	Matrix Determine	Matrix Determine	5-HQR2
the Eigenvalues of a Real Symmetric Tridiagonal	Matrix Determine	Matrix Determine	5-IMTQL1
Eigenvalues/vectors of a Real Symmetric Tridiagonal	Matrix Determine the	Matrix Determine the	5-IMTQL2

the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Determine	5-IMTQLV
Some Eigenvectors of a Real Upper Hessenberg Matrix	Determine	5-INVIT
Squares Problem with a Real Rectangular Coefficient Matrix	Solve a Least	5-MINFIT
Extreme Eigenvalues of a Real Symmetric Tridiagonal Matrix	Determine Some	5-RATQR
Determine the Eigenvalues/vectors of a Real General Matrix		5-RG
Determine the Eigenvalues/vectors of a Real Symmetric Matrix		5-RS
the Eigenvalues/vectors of a Real Symmetric Band Matrix	Determine	5-RSB
Eigenvalues and Some Eigenvectors of a Real Symmetric Matrix	Determine All	5-RSM
the Eigenvalues/vectors of a Real Symmetric Packed Matrix	Determine	5-RSP
Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix	Determine the	5-RST
the Eigenvalues/vectors of a Real Tridiagonal Matrix	Determine	5-RT
Singular Value Decomposition of a Real Rectangular Matrix	Compute the	5-SVD
Some Eigenvectors of a Real Symmetric Tridiagonal Matrix	Determine	5-TINVIT
the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Determine	5-TQL1
Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix	Determine the	5-TQL2
the Eigenvalues of a Real Symmetric Tridiagonal Matrix	Determine	5-TQLRAT
Some Eigenvalues of a Real Symmetric Tridiagonal Matrix	Determine	5-TRIDIB
Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix	Determine Some	5-TSTURM
Factor a General Band Matrix	and Estimate its Condition Number	4-SGBCO
Factor a General Matrix	and Estimate its Condition Number	4-SGECO
Factor a Positive Definite Band Matrix	and Estimate its Condition Number	4-SPBCO
Factor a Positive Definite Matrix	and Estimate its Condition Number	4-SPOCO
Factor a Positive Definite Packed Matrix	and Estimate its Condition Number	4-SPPCO
Factor a Symmetric Indefinite Matrix	and Estimate its Condition Number	4-SSICO
Factor a Symmetric Indefinite Packed Matrix	and Estimate its Condition Number	4-SSPCO
General Matrix-Matrix Multiplication, General Matrix	Storage	3-MXMA
General Matrix-Vector Multiplication, General Matrix	Storage	3-MXVA
Reduce a Complex General Matrix	to Complex Upper Hessenberg Form	5-COMHES
Reduce a Complex General Matrix	to Complex Upper Hessenberg Form	5-CORTH
Transform a Real Non-symmetric Tridiagonal Matrix	to Real Symmetric Form	5-FIG1
Transform a Real Non-symmetric Tridiagonal Matrix	to Real Symmetric Form	5-FIG2
Reduce a Real Symmetric Band Matrix	to Real Symmetric Tridiagonal Form	5-BANDR
Reduce a Complex Hermitian Matrix	to Real Symmetric Tridiagonal Form	5-HTRID3
Reduce a Complex Hermitian Matrix	to Real Symmetric Tridiagonal Form	5-HTRID1
Reduce a Real Symmetric Matrix	to Real Symmetric Tridiagonal Form	5-TRED1
Reduce a Real Symmetric Matrix	to Real Symmetric Tridiagonal Form	5-TRED2
Reduce a Real Symmetric Matrix	to Real Symmetric Tridiagonal Form	5-TRED3
Reduce a Real General Matrix	to Real Upper Hessenberg Form	5-ELMHES
Reduce a Real General Matrix	to Real Upper Hessenberg Form	5-ORTHES
General Matrix-Matrix Multiplication		3-MXM
General Matrix-Matrix Multiplication, General Matrix	Storage	3-MXMA
General Matrix-Matrix Multiply		3-SGEMM
Symmetric Matrix-Matrix Multiply		3-SSYMM
Triangular Matrix-Matrix Multiply		3-STRMM
General Matrix-Matrix Multiply using Strassen's Method		3-SGEMMS
General Matrix-Vector Multiplication		3-MXV
General Matrix-Vector Multiplication, General Matrix	Storage	3-MXVA
General Band Matrix-Vector Multiply		3-SGBMV
General Matrix-Vector Multiply		3-SGEMV
Symmetric Band Matrix-Vector Multiply		3-SSBMV
Symmetric Packed Matrix-Vector Multiply		3-SSPMV
Symmetric Matrix-Vector Multiply		3-SSYMV
Triangular Band Matrix-Vector Multiply		3-STBMV
Triangular Packed Matrix-Vector Multiply		3-STPMV
Triangular Matrix-Vector Multiply		3-STRMV
Matrix-Vector Product Added to a Vector		3-SMXPY
Find the Index of the Element of Maximum Magnitude of a Vector		2-ISAMAX
Find the Index of the Element of Maximum Value of a Vector		2-ISMAX
Find the Index of the Masked Element of Maximum Value of an Integer Vector		2-INFLMAX
General Matrix-Matrix Multiply using Strassen's Method		3-SGEMMS
Find the Index of the Element of Minimum Magnitude of a Vector		2-ISAMIN
Find the Index of the Element of Minimum Value of a Vector		2-ISMIN
Find the Index of the Masked Element of Minimum Value of an Integer Vector		2-INFLMIN
Apply a Modified Givens Rotation		2-SROTM
Construct a Modified Givens Rotation		2-SROTMG
Complex to Complex Discrete Fourier Transform of Multiple Data Sets		6-CFFTMLT
Real to Complex Discrete Fourier Transform of Multiple Data Sets		6-RFFTMLT
Matrix Solve Multiple Sets of Linear Equations with a Triangular Matrix		3-STRSM
General Matrix-Matrix Multiplication		3-MXM
General Matrix-Vector Multiplication		3-MXV
General Matrix-Matrix Multiplication, General Matrix	Storage	3-MXMA
General Matrix-Vector Multiplication, General Matrix	Storage	3-MXVA

Permuted Index

General Band Matrix-Vector	Multiply	3-SGBMV
General Matrix-Matrix	Multiply	3-SGEMM
General Matrix-Vector	Multiply	3-SGEMV
Symmetric Band Matrix-Vector	Multiply	3-SBBMV
Symmetric Packed Matrix-Vector	Multiply	3-SSPMV
Symmetric Matrix-Matrix	Multiply	3-SSYMM
Symmetric Matrix-Vector	Multiply	3-SSYMV
Triangular Band Matrix-Vector	Multiply	3-STBMV
Triangular Packed Matrix-Vector	Multiply	3-STPMV
Triangular Matrix-Matrix	Multiply	3-STRMM
Triangular Matrix-Vector	Multiply	3-STRMV
General Matrix-Matrix	Multiply using Strassen's Method	3-SGEMMS
Number of Leading True or	Negative Elements in a Vector	2-ILZ
Count Number of True or	Negative Elements in a Vector	2-ILSUM
Form Transform a Real	Non-symmetric Tridiagonal Matrix to Real Symmetric	5-FIGI
Form Transform a Real	Non-symmetric Tridiagonal Matrix to Real Symmetric	5-FIGI2
Euclidean	Norm of a Vector	2-SNRM2
Find the Index of the First Element = the	Objective	2-ISRCH EQ
Find the Index of the First Element \geq the	Objective	2-ISRCHFGT
Find the Index of the First Element $>$ the	Objective	2-ISRCHFLT
Find the Index of the First Element \leq the	Objective	2-ISRCHFLE
Find the Index of the First Element $<$ the	Objective	2-ISRCHIGT
Find the Index of the First Element \geq the	Objective	2-ISRCHIGE
Find the Index of the First Element $>$ the	Objective	2-ISRCHIGT
Find the Index of the First Element \leq the	Objective	2-ISRCHILE
Find the Index of the First Element $<$ the	Objective	2-ISRCHILT
Find the Index of the First Masked Element = the	Objective	2-ISRCHMEQ
Find the Index of the First Masked Element \geq the	Objective	2-ISRCHMGE
Find the Index of the First Masked Element $>$ the	Objective	2-ISRCHMGT
Find the Index of the First Masked Element \leq the	Objective	2-ISRCHMLE
Find the Index of the First Masked Element $<$ the	Objective	2-ISRCHMLT
Find the Index of the First Masked Element \neq the	Objective	2-ISRCHMNE
Find the Index of the First Element \neq the	Objective	2-ISRCHNE
Elementary Vector	Operation	2-SAXPY
Sparse Elementary Vector	Operation	2-SPAXPY
Solve Weiner-Levinson	Optimal Filter Equation	4-OPFILT
First	Order Linear Recurrence	8-FOLR
First	Order Linear Recurrence	8-FOLR2
First	Order Linear Recurrence	8-FOLR2P
Compute the Last Term of a First	Order Linear Recurrence	8-FOLRN
Compute the Last Term of a First	Order Linear Recurrence	8-FOLRNP
First	Order Linear Recurrence	8-FOLRP
Second	Order Linear Recurrence	8-SOLR
Second	Order Linear Recurrence	8-SOLR3
Compute the Last Term of a Second	Order Linear Recurrence	8-SOLRN
First	Order Linear Recurrence with Constant Coefficients	8-FOLRC
of Equalities Search	Ordered Masked Vector for First Equality and Number	2-OSRCHM
Equalities Search	Ordered Vector for First Equality and Number of	2-OSRCHF
Equalities Search	Ordered Vector for First Equality and Number of	2-OSRCHI
Back Transform Eigenvectors following	ORTHES	5-ORTBAK
Accumulate the Transformations in the Reduction by	ORTHES	5-ORTRAN
Solve Triangular	Packed Linear Equations	3-STPSV
Determinant of a Positive Definite	Packed Matrix	4-SPPDI
Factor a Positive Definite	Packed Matrix	4-SPPFA
Solve Linear Equations with a Positive Definite	Packed Matrix	4-SPPSL
Inverse, and Inertia of a Symmetric Indefinite	Packed Matrix Determinant,	4-SPPDI
Factor a Symmetric Indefinite	Packed Matrix	4-SPPFA
Solve Linear Equations with a Symmetric Indefinite	Packed Matrix	4-SPPSL
Determine the Eigenvalues/vectors of a Real Symmetric	Packed Matrix	5-RSP
Factor a Positive Definite	Packed Matrix and Estimate its Condition Number	4-SPPCO
Factor a Symmetric Indefinite	Packed Matrix and Estimate its Condition Number	4-SPPCO
Symmetric	Packed Matrix-Vector Multiply	3-SSPMV
Triangular	Packed Matrix-Vector Multiply	3-STPMV
Symmetric	Packed Rank-1 Update	3-SSPR
Symmetric	Packed Rank-2 Update	3-SSPR2
Solve a	Partial Products Problem	8-RECP
Solve a	Partial Summation Problem	8-RECPS
Eigenproblem	Partially Reduce a Real General Generalized	5-QZHE
Recompute the Cholesky Decomposition of a	Permuted Symmetric Matrix	4-SCHEX
Determinant of a	Positive Definite Band Matrix	4-SPBDI
Factor a	Positive Definite Band Matrix	4-SPBFA
Solve Linear Equations with a	Positive Definite Band Matrix	4-SPBSL

Condition Number Factor a	Positive Definite Band Matrix and Estimate its	4-SPBCO
Determinant and Inverse of a	Positive Definite Matrix	4-SPODI
Factor a	Positive Definite Matrix	4-SPOFA
Solve Linear Equations with a	Positive Definite Matrix	4-SPOSL
Number Factor a	Positive Definite Matrix and Estimate its Condition	4-SPOCO
Determinant of a	Positive Definite Packed Matrix	4-SPPDI
Factor a	Positive Definite Packed Matrix	4-SPPFA
Solve Linear Equations with a	Positive Definite Packed Matrix	4-SPPSL
Condition Number Factor a	Positive Definite Packed Matrix and Estimate its	4-SPPCO
Solve Linear Equations with a	Positive Definite Tridiagonal Matrix	4-SPTSL
Number of Leading False or	Positive Elements in a Vector	2-ILLZ
Solve a Partial Products	Problem	8-RECPP
Solve a Partial Summation	Problem	8-RECPS
Solve a Least Squares	Problem with a Real Rectangular Coefficient Matrix	5-MINFIT
Matrix-Vector	Product Added to a Vector	3-SMXPY
Vector-Matrix	Product Added to a Vector	3-SXMPY
Dot	Product of Two Vectors	2-SDOT
Sparse Dot	Product of Two Vectors	2-SPDOT
Solve a Partial	Products Problem	8-RECPP
General	Rank-1 Update	3-SGER
Symmetric Packed	Rank-1 Update	3-SSPR
Symmetric	Rank-1 Update	3-SSYR
Symmetric Packed	Rank-2 Update	3-SSPR2
Symmetric	Rank-2 Update	3-SSYR2
Symmetric	Rank-2k Update	3-SSYR2K
Symmetric	Rank-k Update	3-SSYRK
Complex to	Real Discrete Fourier Transform	6-CRFFT2
Partially Reduce a	Real General Generalized Eigenproblem	5-QZHES
Complete the Reduction of a	Real General Generalized Eigenproblem	5-QZIT
Determine the Eigenvalues of a Reduced	Real General Generalized Eigenproblem	5-QZVAL
Determine the Eigenvectors of a Reduced	Real General Generalized Eigenproblem	5-QZVEC
Determine the Eigenvalues/vectors of a	Real General Generalized Eigenproblem	5-RGG
Balance a	Real General Matrix	5-BALANC
Determine the Eigenvalues/vectors of a	Real General Matrix	5-RG
Reduce a	Real General Matrix to Real Upper Hessenberg Form	5-ELMHES
Reduce a	Real General Matrix to Real Upper Hessenberg Form	5-ORTHES
Symmetric Form Transform a	Real Non-symmetric Tridiagonal Matrix to Real	5-FIG1
Symmetric Form Transform a	Real Non-symmetric Tridiagonal Matrix to Real	5-FIG2
Solve a Least Squares Problem with a	Real Rectangular Coefficient Matrix	5-MINFIT
Compute the Singular Value Decomposition of a	Real Rectangular Matrix	5-SVD
Determine Some Eigenvectors of a	Real Symmetric Band Matrix	5-BANDV
Determine Some Eigenvalues of a	Real Symmetric Band Matrix	5-BQR
Determine the Eigenvalues/vectors of a	Real Symmetric Band Matrix	5-RSB
Tridiagonal Form Reduce a	Real Symmetric Band Matrix to Real Symmetric	5-BANDR
Transform a Real Non-symmetric Tridiagonal Matrix to	Real Symmetric Form	5-FIG1
Transform a Real Non-symmetric Tridiagonal Matrix to	Real Symmetric Form	5-FIG2
Determine the Eigenvalues/vectors of a	Real Symmetric Generalized Eigenproblem	5-RSG
Determine the Eigenvalues/vectors of a	Real Symmetric Generalized Eigenproblem	5-RSGAB
Determine the Eigenvalues/vectors of a	Real Symmetric Generalized Eigenproblem	5-RSGBA
Form Reduce a	Real Symmetric Generalized Eigenproblem to Standard	5-REDUC
Form Reduce a	Real Symmetric Generalized Eigenproblem to Standard	5-REDUC2
Determine the Eigenvalues/vectors of a	Real Symmetric Matrix	5-RS
Determine All Eigenvalues and Some Eigenvectors of a	Real Symmetric Matrix	5-RSM
Form Reduce a	Real Symmetric Matrix to Real Symmetric Tridiagonal	5-TRED1
Form Reduce a	Real Symmetric Matrix to Real Symmetric Tridiagonal	5-TRED2
Form Reduce a	Real Symmetric Matrix to Real Symmetric Tridiagonal	5-TRED3
Determine the Eigenvalues/vectors of a	Real Symmetric Packed Matrix	5-RSP
Reduce a Real Symmetric Band Matrix to	Real Symmetric Tridiagonal Form	5-BANDR
Reduce a Complex Hermitian Matrix to	Real Symmetric Tridiagonal Form	5-HTRID3
Reduce a Complex Hermitian Matrix to	Real Symmetric Tridiagonal Form	5-HTRIDI
Reduce a Real Symmetric Matrix to	Real Symmetric Tridiagonal Form	5-TRED1
Reduce a Real Symmetric Matrix to	Real Symmetric Tridiagonal Form	5-TRED2
Reduce a Real Symmetric Matrix to	Real Symmetric Tridiagonal Form	5-TRED3
Determine Some Eigenvectors of a	Real Symmetric Tridiagonal Matrix	5-BISECT
Determine the Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-IMTQL1
Determine the Eigenvalues/vectors of a	Real Symmetric Tridiagonal Matrix	5-IMTQL2
Determine the Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-IMTQLV
Determine Some Extreme Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-RATQR
Determine the Eigenvalues/vectors of a	Real Symmetric Tridiagonal Matrix	5-RST
Determine Some Eigenvectors of a	Real Symmetric Tridiagonal Matrix	5-TINVIT
Determine the Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-TQL1
Determine the Eigenvalues/vectors of a	Real Symmetric Tridiagonal Matrix	5-TQL2

Determine the Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-TQLRAT
Determine Some Eigenvalues of a	Real Symmetric Tridiagonal Matrix	5-TRIDIB
Determine Some Eigenvalues/vectors of a	Real Symmetric Tridiagonal Matrix	5-TSTURM
	Real to Complex Discrete Fourier Transform	6-RCFFT2
Multiple Data Sets	Real to Complex Discrete Fourier Transform of	6-RFFTMLT
Determine the Eigenvalues/vectors of a	Real Tridiagonal Matrix	5-RT
Reduce a Real General Matrix to	Real Upper Hessenberg Form	5-ELMHES
Reduce a Real General Matrix to	Real Upper Hessenberg Form	5-ORTHES
Determine the Eigenvalues of a	Real Upper Hessenberg Matrix	5-HQR
Determine the Eigenvalues/vectors of a	Real Upper Hessenberg Matrix	5-HQR2
Determine Some Eigenvectors of a	Real Upper Hessenberg Matrix	5-INVIT
Symmetric Matrix	Recompute the Cholesky Decomposition of a DOWDATED	4-SCHDD
Symmetric Matrix	Recompute the Cholesky Decomposition of a Permuted	4-SCHEX
Symmetric Matrix	Recompute the Cholesky Decomposition of an Updated	4-SCHUD
Solve a Least Squares Problem with a Real	Rectangular Coefficient Matrix	5-MINFIT
QR Decomposition of a General	Rectangular Matrix	4-SQRDC
Compute the Singular Value Decomposition of a General	Rectangular Matrix	4-SSVDC
Compute the Singular Value Decomposition of a Real	Rectangular Matrix	5-SVD
First Order Linear	Recurrence	8-FOLR
First Order Linear	Recurrence	8-FOLR2
First Order Linear	Recurrence	8-FOLR2P
Compute the Last Term of a First Order Linear	Recurrence	8-FOLRN
Compute the Last Term of a First Order Linear	Recurrence	8-FOLRNP
First Order Linear	Recurrence	8-FOLRP
Second Order Linear	Recurrence	8-SOLR
Second Order Linear	Recurrence	8-SOLR3
Compute the Last Term of a Second Order Linear	Recurrence	8-SOLRN
First Order Linear	Recurrence with Constant Coefficients	8-FOLRC
Back Transform Eigenvectors following	REDUC or REDUC2	5-REBAK
Back Transform Eigenvectors following REDUC or	REDUC2	5-REBAK
Back Transform Eigenvectors following	REDUC2	5-REBAKB
Hessenberg Form	Reduce a Complex General Matrix to Complex Upper	5-COMHES
Hessenberg Form	Reduce a Complex General Matrix to Complex Upper	5-CORTH
Tridiagonal Form	Reduce a Complex Hermitian Matrix to Real Symmetric	5-HTRID3
Tridiagonal Form	Reduce a Complex Hermitian Matrix to Real Symmetric	5-HTRIDI
Partially	Reduce a Real General Generalized Eigenproblem	5-QZHES
Form	Reduce a Real General Matrix to Real Upper Hessenberg	5-ELMHES
Form	Reduce a Real General Matrix to Real Upper Hessenberg	5-ORTHES
Tridiagonal Form	Reduce a Real Symmetric Band Matrix to Real Symmetric	5-BANDR
Standard Form	Reduce a Real Symmetric Generalized Eigenproblem to	5-REDUC
Standard Form	Reduce a Real Symmetric Generalized Eigenproblem to	5-REDUC2
Tridiagonal Form	Reduce a Real Symmetric Matrix to Real Symmetric	5-TRED1
Tridiagonal Form	Reduce a Real Symmetric Matrix to Real Symmetric	5-TRED2
Tridiagonal Form	Reduce a Real Symmetric Matrix to Real Symmetric	5-TRED3
Determine the Eigenvalues of a	Reduced Real General Generalized Eigenproblem	5-QZVAL
Determine the Eigenvectors of a	Reduced Real General Generalized Eigenproblem	5-QZVEC
Accumulate the Transformations in the	Reduction by ELMHES	5-ELTRAN
Accumulate the Transformations in the	Reduction by ORTHES	5-ORTRAN
Complete the	Reduction of a Real General Generalized Eigenproblem	5-QZIT
Initialization subprogram for	RFFTMLT	6-FFTFAX
Apply a Givens	Rotation	2-SROT
Construct a Givens	Rotation	2-SROTG
Apply a Modified Givens	Rotation	2-SROTM
Construct a Modified Givens	Rotation	2-SROTMG
Scale a Vector by a	Scalar	2-SSCAL
	Scale a Vector by a Scalar	2-SSCAL
	Scatter a Sparse Vector into Full Form	2-SCATTER
Number of Equalities	Search Ordered Masked Vector for First Equality and	2-OSRCHM
of Equalities	Search Ordered Vector for First Equality and Number	2-OSRCHF
of Equalities	Search Ordered Vector for First Equality and Number	2-OSRCHI
	Second Order Linear Recurrence	8-SOLR
	Second Order Linear Recurrence	8-SOLR3
Compute the Last Term of a	Second Order Linear Recurrence	8-SOLRN
Complex Discrete Fourier Transform of Multiple Data	Sets Complex to	6-CFFTMLT
Complex Discrete Fourier Transform of Multiple Data	Sets Real to	6-RFFTMLT
Solve Multiple	Sets of Linear Equations with a Triangular Matrix	3-STRSM
Matrix Compute the	Singular Value Decomposition of a General Rectangular	4-SSVDC
Matrix Compute the	Singular Value Decomposition of a Real Rectangular	5-SVD
Coefficient Matrix	Solve a Least Squares Problem with a Real Rectangular	5-MINFIT
	Solve a Partial Products Problem	8-RECPP
	Solve a Partial Summation Problem	8-RECPS
Decomposition	Solve Linear Equations using the QR	4-SQRSL

	Solve Linear Equations with a General Band Matrix . . .	4-SGBSL
	Solve Linear Equations with a General Matrix	4-SGESL
Matrix	Solve Linear Equations with a General Tridiagonal	4-SGTSL
Matrix	Solve Linear Equations with a Positive Definite Band	4-SPBSL
Matrix	Solve Linear Equations with a Positive Definite	4-SPOSL
Packed Matrix	Solve Linear Equations with a Positive Definite	4-SPPSL
Tridiagonal Matrix	Solve Linear Equations with a Positive Definite	4-SPTSL
Matrix	Solve Linear Equations with a Symmetric Indefinite	4-SSISL
Packed Matrix	Solve Linear Equations with a Symmetric Indefinite	4-SSPSL
	Solve Linear Equations with a Triangular Band Matrix	3-STBSV
	Solve Linear Equations with a Triangular Matrix	3-STRSV
	Solve Linear Equations with a Triangular Matrix	4-STRSL
Triangular Matrix	Solve Multiple Sets of Linear Equations with a	3-STRSM
	Solve Triangular Packed Linear Equations	3-STPSV
	Solve Weiner-Levinson Optimal Filter Equation	4-OPFILT
	Sort Array	9-ORDERS
	Sparse Dot Product of Two Vectors	2-SPDOT
	Sparse Elementary Vector Operation	2-SPAXPY
Gather a	Sparse Vector into Compressed Form	2-GATHER
Scatter a	Sparse Vector into Full Form	2-SCATTER
Matrix	Squares Problem with a Real Rectangular Coefficient	5-MINFIT
Reduce a Real Symmetric Generalized Eigenproblem to	Standard Form	5-REDUC
Reduce a Real Symmetric Generalized Eigenproblem to	Standard Form	5-REDUC2
General Matrix-Matrix Multiplication, General Matrix	Storage	3-MXMA
General Matrix-Vector Multiplication, General Matrix	Storage	3-MXVA
General Matrix-Matrix Multiply using	Strassen's Method	3-SGEMMS
Initialization	subprogram for CFFTMLT	6-CFTFAX
Initialization	subprogram for RFFTMLT	6-FFTFAK
	Sum of Absolute Values of the Elements of a Vector	2-SASUM
	Sum of the Elements of a Vector	2-SSUM
Solve a Partial	Summation Problem	8-RECPS
	Swap Two Vectors	2-SSWAP
Determine Some Eigenvectors of a Real	Symmetric Band Matrix	5-BANDV
Determine Some Eigenvalues of a Real	Symmetric Band Matrix	5-BQR
Determine the Eigenvalues/vectors of a Real	Symmetric Band Matrix	5-RSB
Form Reduce a Real	Symmetric Band Matrix to Real Symmetric Tridiagonal	5-BANDR
a Real Non-symmetric Tridiagonal Matrix to Real	Symmetric Band Matrix-Vector Multiply	3-SSBMV
a Real Non-symmetric Tridiagonal Matrix to Real	Symmetric Form Transform	5-FIGI
Determine the Eigenvalues/vectors of a Real	Symmetric Form Transform	5-FIGI2
Determine the Eigenvalues/vectors of a Real	Symmetric Generalized Eigenproblem	5-RSG
Determine the Eigenvalues/vectors of a Real	Symmetric Generalized Eigenproblem	5-RSGAB
Determine the Eigenvalues/vectors of a Real	Symmetric Generalized Eigenproblem	5-RSGBA
Reduce a Real	Symmetric Generalized Eigenproblem to Standard Form	5-REDUC
Reduce a Real	Symmetric Generalized Eigenproblem to Standard Form	5-REDUC2
Determinant, Inverse, and Inertia of a	Symmetric Indefinite Matrix	4-SSIDI
Factor a	Symmetric Indefinite Matrix	4-SSIFA
Solve Linear Equations with a	Symmetric Indefinite Matrix	4-SSISL
Condition Number Factor a	Symmetric Indefinite Matrix and Estimate its	4-SSICO
Determinant, Inverse, and Inertia of a	Symmetric Indefinite Packed Matrix	4-SSPDI
Factor a	Symmetric Indefinite Packed Matrix	4-SSPFA
Solve Linear Equations with a	Symmetric Indefinite Packed Matrix	4-SSPSL
Condition Number Factor a	Symmetric Indefinite Packed Matrix and Estimate its	4-SSPCO
Compute the Cholesky Decomposition of a	Symmetric Matrix	4-SCHDC
Recompute the Cholesky Decomposition of a DOWNDATED	Symmetric Matrix	4-SCHDD
Recompute the Cholesky Decomposition of a Permuted	Symmetric Matrix	4-SCHEX
Recompute the Cholesky Decomposition of an Updated	Symmetric Matrix	4-SCHUD
Determine the Eigenvalues/vectors of a Real	Symmetric Matrix	5-RS
All Eigenvalues and Some Eigenvectors of a Real	Symmetric Matrix Determine	5-RSM
Reduce a Real	Symmetric Matrix to Real Symmetric Tridiagonal Form	5-TRED1
Reduce a Real	Symmetric Matrix to Real Symmetric Tridiagonal Form	5-TRED2
Reduce a Real	Symmetric Matrix to Real Symmetric Tridiagonal Form	5-TRED3
	Symmetric Matrix-Matrix Multiply	3-SSYMM
	Symmetric Matrix-Vector Multiply	3-SSYMV
Determine the Eigenvalues/vectors of a Real	Symmetric Packed Matrix	5-RSP
	Symmetric Packed Matrix-Vector Multiply	3-SSPMV
	Symmetric Packed Rank-1 Update	3-SSPR
	Symmetric Packed Rank-2 Update	3-SSPR2
	Symmetric Rank-1 Update	3-SSYR
	Symmetric Rank-2 Update	3-SSYR2
	Symmetric Rank-2k Update	3-SSYR2K
	Symmetric Rank-k Update	3-SSYRK
Reduce a Real Symmetric Band Matrix to Real	Symmetric Tridiagonal Form	5-BANDR

Reduce a Complex Hermitian Matrix to Real	Symmetric Tridiagonal Form	5-HTRID3
Reduce a Complex Hermitian Matrix to Real	Symmetric Tridiagonal Form	5-HTRIDI
Reduce a Real Symmetric Matrix to Real	Symmetric Tridiagonal Form	5-TRED1
Reduce a Real Symmetric Matrix to Real	Symmetric Tridiagonal Form	5-TRED2
Reduce a Real Symmetric Matrix to Real	Symmetric Tridiagonal Form	5-TRED3
Determine Some Eigenvectors of a Real	Symmetric Tridiagonal Matrix	5-BISECT
Determine the Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-IMTQL1
Determine the Eigenvalues/vectors of a Real	Symmetric Tridiagonal Matrix	5-IMTQL2
Determine the Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-IMTQLV
Determine Some Extreme Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-RATQR
Determine the Eigenvalues/vectors of a Real	Symmetric Tridiagonal Matrix	5-RST
Determine Some Eigenvectors of a Real	Symmetric Tridiagonal Matrix	5-TINVIT
Determine the Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-TQL1
Determine the Eigenvalues/vectors of a Real	Symmetric Tridiagonal Matrix	5-TQL2
Determine the Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-TQLRAT
Determine Some Eigenvalues of a Real	Symmetric Tridiagonal Matrix	5-TRIDIB
Determine Some Eigenvalues/vectors of a Real	Symmetric Tridiagonal Matrix	5-TSTURM
Find Indices of Clusters of Elements = the	Target Within a Vector	2-CLUSEQ
Find Indices of Clusters of Elements ≥ the	Target Within a Vector	2-CLUSFGE
Find Indices of Clusters of Elements > the	Target Within a Vector	2-CLUSFGT
Find Indices of Clusters of Elements ≤ the	Target Within a Vector	2-CLUSFLE
Find Indices of Clusters of Elements < the	Target Within a Vector	2-CLUSFLT
Find Indices of Clusters of Elements ≥ the	Target Within a Vector	2-CLUSIGE
Find Indices of Clusters of Elements > the	Target Within a Vector	2-CLUSIGT
Find Indices of Clusters of Elements ≤ the	Target Within a Vector	2-CLUSILE
Find Indices of Clusters of Elements < the	Target Within a Vector	2-CLUSILT
Find Indices of Clusters of Elements * the	Target Within a Vector	2-CLUSNE
Find Indices of All Elements = the	Target Within a Vector	2-WHENEQ
Find Indices of All Elements ≥ the	Target Within a Vector	2-WHENFGE
Find Indices of All Elements > the	Target Within a Vector	2-WHENFGT
Find Indices of All Elements ≤ the	Target Within a Vector	2-WHENFLE
Find Indices of All Elements < the	Target Within a Vector	2-WHENFLT
Find Indices of All Elements ≥ the	Target Within a Vector	2-WHENIGE
Find Indices of All Elements > the	Target Within a Vector	2-WHENIGT
Find Indices of All Elements ≤ the	Target Within a Vector	2-WHENILE
Find Indices of All Elements < the	Target Within a Vector	2-WHENILT
Find Indices of All Masked Elements = the	Target Within a Vector	2-WHENMEQ
Find Indices of All Masked Elements ≥ the	Target Within a Vector	2-WHENMGE
Find Indices of All Masked Elements > the	Target Within a Vector	2-WHENMGT
Find Indices of All Masked Elements ≤ the	Target Within a Vector	2-WHENMLE
Find Indices of All Masked Elements < the	Target Within a Vector	2-WHENMLT
Find Indices of All Masked Elements * the	Target Within a Vector	2-WHENMNE
Find Indices of All Elements * the	Target Within a Vector	2-WHENNE
Compute the Last	Term of a First Order Linear Recurrence	8-FOLRN
Compute the Last	Term of a First Order Linear Recurrence	8-FOLRNP
Compute the Last	Term of a Second Order Linear Recurrence	8-SOLRN
Complex to Complex Discrete Fourier	Transform	6-CFFT2
Complex to Real Discrete Fourier	Transform	6-CRFFT2
Real to Complex Discrete Fourier	Transform	6-RCFFT2
Real Symmetric Form	Transform a Real Non-symmetric Tridiagonal Matrix to	5-FIGI
Real Symmetric Form	Transform a Real Non-symmetric Tridiagonal Matrix to	5-FIGI2
Back	Transform Eigenvectors following BALANC	5-BALBAK
Back	Transform Eigenvectors following CBAL	5-CBABK2
Back	Transform Eigenvectors following COMHES	5-COMBAK
Back	Transform Eigenvectors following CORTH	5-CORTB
Back	Transform Eigenvectors following ELMHES	5-ELMBAK
Back	Transform Eigenvectors following FIGI	5-BAKVEC
Back	Transform Eigenvectors following HTRID3	5-HTRIB3
Back	Transform Eigenvectors following HTRIDI	5-HTRIBK
Back	Transform Eigenvectors following ORTHES	5-ORTBAK
Back	Transform Eigenvectors following REDUC or REDUC2	5-REBAK
Back	Transform Eigenvectors following REDUC2	5-REBAKB
Back	Transform Eigenvectors following TRED1	5-TRBAK1
Back	Transform Eigenvectors following TRED3	5-TRBAK3
Complex to Complex Discrete Fourier	Transform of Multiple Data Sets	6-CFFTMLT
Real to Complex Discrete Fourier	Transform of Multiple Data Sets	6-RFFTMLT
Accumulate the	Transformations in the Reduction by ELMHES	5-ELTRAN
Accumulate the	Transformations in the Reduction by ORTHES	5-ORTRAN
Back Transform Eigenvectors following	TRED1	5-TRBAK1
Back Transform Eigenvectors following	TRED3	5-TRBAK3
Solve Linear Equations with a	Triangular Band Matrix	3-STBSV
Solve Linear Equations with a	Triangular Band Matrix-Vector Multiply	3-STBMV

Solve Multiple Sets of Linear Equations with a	Triangular Matrix	3-STRSM
Solve Linear Equations with a	Triangular Matrix	3-STRSV
Estimate the Condition Number of a	Triangular Matrix	4-STRCO
Determinant and Inverse of a	Triangular Matrix	4-STRDI
Solve Linear Equations with a	Triangular Matrix	4-STRSL
	Triangular Matrix-Matrix Multiply	3-STRMM
	Triangular Matrix-Vector Multiply	3-STRMV
	Triangular Packed Linear Equations	3-STPSV
	Triangular Packed Matrix-Vector Multiply	3-STMV
	Tridiagonal Form	5-BANDR
Reduce a Real Symmetric Band Matrix to Real Symmetric	Tridiagonal Form	5-HTRID3
Reduce a Complex Hermitian Matrix to Real Symmetric	Tridiagonal Matrix	5-HTRID1
Reduce a Complex Hermitian Matrix to Real Symmetric	Tridiagonal Matrix	5-TRED1
Reduce a Real Symmetric Matrix to Real Symmetric	Tridiagonal Matrix	5-TRED2
Reduce a Real Symmetric Matrix to Real Symmetric	Tridiagonal Matrix	5-TRED3
Reduce a Real Symmetric Matrix to Real Symmetric	Tridiagonal Matrix	4-SGTSL
Solve Linear Equations with a General	Tridiagonal Matrix	4-SPTSL
Solve Linear Equations with a Positive Definite	Tridiagonal Matrix	5-BISECT
Determine Some Eigenvectors of a Real Symmetric	Tridiagonal Matrix	5-IMTQL1
Determine the Eigenvalues of a Real Symmetric	Tridiagonal Matrix	5-IMTQL2
Determine the Eigenvalues/vectors of a Real Symmetric	Tridiagonal Matrix	5-IMTQLV
Determine the Eigenvalues of a Real Symmetric	Tridiagonal Matrix	5-RATQR
Some Extreme Eigenvalues of a Real Symmetric	Tridiagonal Matrix	5-RST
Determine the Eigenvalues/vectors of a Real Symmetric	Tridiagonal Matrix	5-RT
Determine the Eigenvalues/vectors of a Real	Tridiagonal Matrix	5-TINVIT
Determine Some Eigenvectors of a Real Symmetric	Tridiagonal Matrix	5-TQL1
Determine the Eigenvalues of a Real Symmetric	Tridiagonal Matrix	5-TQL2
Determine the Eigenvalues/vectors of a Real Symmetric	Tridiagonal Matrix	5-TQLRAT
Determine the Eigenvalues of a Real Symmetric	Tridiagonal Matrix	5-TRIDIB
Some Eigenvalues/vectors of a Real Symmetric	Tridiagonal Matrix	5-TSTURM
Transform a Real Non-symmetric	Tridiagonal Matrix to Real Symmetric Form	5-FIGI
Transform a Real Non-symmetric	Tridiagonal Matrix to Real Symmetric Form	5-FIGI2
Number of Leading	True or Negative Elements in a Vector	2-ILLZ
Count Number of	True or Negative Elements in a Vector	2-ILSUM
General Rank-1	Update	3-SGER
Symmetric Packed Rank-1	Update	3-SSPR
Symmetric Packed Rank-2	Update	3-SSPR2
Symmetric Rank-1	Update	3-SSYR
Symmetric Rank-2	Update	3-SSYR2
Symmetric Rank-2k	Update	3-SSYR2K
Symmetric Rank-k	Update	3-SSYRK
Recompute the Cholesky Decomposition of an	Updated Symmetric Matrix	4-SCHUD
Reduce a Complex General Matrix to Complex	Upper Hessenberg Form	5-COMHES
Reduce a Complex General Matrix to Complex	Upper Hessenberg Form	5-CORTH
Reduce a Real General Matrix to Real	Upper Hessenberg Form	5-ELMHES
Reduce a Real General Matrix to Real	Upper Hessenberg Form	5-ORTHES
Determine Some Eigenvectors of a Complex	Upper Hessenberg Matrix	5-CINVIT
Determine the Eigenvalues of a Complex	Upper Hessenberg Matrix	5-COMLR
Determine the Eigenvalues of a Complex	Upper Hessenberg Matrix	5-COMQR
Determine the Eigenvalues/vectors of a Complex	Upper Hessenberg Matrix	5-COMQR2
Determine the Eigenvalues of a Real	Upper Hessenberg Matrix	5-HQR
Determine the Eigenvalues/vectors of a Real	Upper Hessenberg Matrix	5-HQR2
Determine the Eigenvalues/vectors of a Real	Upper Hessenberg Matrix	5-INVIT
Determine Some Eigenvectors of a Real	Upper Hessenberg Matrix	4-SSVDC
Compute the Singular	Value Decomposition of a General Rectangular Matrix	5-SVD
Compute the Singular	Value Decomposition of a Real Rectangular Matrix	2-ISMAX
Find the Index of the Element of Maximum	Value of a Vector	2-ISMIN
Find the Index of the Element of Minimum	Value of a Vector	2-INFLMAX
Find the Index of the Masked Element of Maximum	Value of an Integer Vector	2-INFLMIN
Find the Index of the Masked Element of Minimum	Value of an Integer Vector	2-SASUM
Sum of Absolute	Values of the Elements of a Vector	3-SXMPY
	Vector-Matrix Product Added to a Vector	4-OFFILT
	Weiner-Levinson Optimal Filter Equation	2-CLUSEQ
	Solve	2-CLUSFGE
Find Indices of Clusters of Elements = the Target	Within a Vector	2-CLUSFGT
Find Indices of Clusters of Elements ≥ the Target	Within a Vector	2-CLUSFLE
Find Indices of Clusters of Elements > the Target	Within a Vector	2-CLUSFLT
Find Indices of Clusters of Elements ≤ the Target	Within a Vector	2-CLUSIGE
Find Indices of Clusters of Elements < the Target	Within a Vector	2-CLUSIGT
Find Indices of Clusters of Elements ≥ the Target	Within a Vector	2-CLUSILE
Find Indices of Clusters of Elements > the Target	Within a Vector	2-CLUSILT
Find Indices of Clusters of Elements ≤ the Target	Within a Vector	2-CLUSNE
Find Indices of Clusters of Elements < the Target	Within a Vector	
Find Indices of Clusters of Elements ≠ the Target	Within a Vector	

Permuted Index

Find Indices of All Elements = the Target	Within a Vector	2-WHENEQ
Find Indices of All Elements \neq the Target	Within a Vector	2-WHENFGE
Find Indices of All Elements > the Target	Within a Vector	2-WHENFGT
Find Indices of All Elements \leq the Target	Within a Vector	2-WHENFLE
Find Indices of All Elements < the Target	Within a Vector	2-WHENFLT
Find Indices of All Elements \neq the Target	Within a Vector	2-WHENIGE
Find Indices of All Elements > the Target	Within a Vector	2-WHENIGT
Find Indices of All Elements \leq the Target	Within a Vector	2-WHENILE
Find Indices of All Elements < the Target	Within a Vector	2-WHENILT
Find Indices of All Masked Elements = the Target	Within a Vector	2-WHENMEQ
Find Indices of All Masked Elements \neq the Target	Within a Vector	2-WHENMGE
Find Indices of All Masked Elements > the Target	Within a Vector	2-WHENMGT
Find Indices of All Masked Elements \leq the Target	Within a Vector	2-WHENMLE
Find Indices of All Masked Elements < the Target	Within a Vector	2-WHENMLT
Find Indices of All Masked Elements \neq the Target	Within a Vector	2-WHENMNE
Find Indices of All Elements \neq the Target	Within a Vector	2-WHENNE

Preface

Purpose and Audience

This guide describes the SCILIB software library and shows you how to use it. The SCILIB library is a collection of FORTRAN-callable subprograms identical in name and operation to those found in the Cray Research Incorporated's UNICOS Math and Scientific Library, V5.0. SCILIB subprograms have been optimized for use on the CONVEX family of supercomputers.

The *SCILIB User's Guide* addresses experienced FORTRAN programmers who:

- convert programs written in Cray FORTRAN to CONVEX FORTRAN.
- optimize existing software to improve performance and increase productivity on CONVEX supercomputers.
- use CONVEX FORTRAN to develop new programs that rely heavily on matrix operations.

Familiarity with the ConvexOS operating system is helpful, but not required, to use this guide. If you are not familiar with ConvexOS, refer to the bibliography at the end of this preface.

Organization

To learn fundamental information necessary for using the SCILIB library, read the introductory sections of each chapter. These sections of background information will help you efficiently use SCILIB subprograms.

To learn more about the subject of any given chapter, refer to documentation cited in the Supplemental Reading section of each chapter.

To identify subprograms by function, refer to the Permuted Index which lists subprogram functions and their chapter numbers and names. To find the page number on which the subprogram is described, use the Index or refer to "Subprogram Descriptions" in the chapter introductions.

Specifically, this guide is organized into the following chapters and appendices:

- Chapter 1 introduces general concepts about SCILIB.
- Chapter 2 describes basic vector operations included in SCILIB.
- Chapter 3 explains basic matrix operations included in SCILIB.
- Chapter 4 describes linear equation subprograms in SCILIB.
- Chapter 5 explains the eigenanalysis capabilities available to SCILIB users.
- Chapter 6 describes the discrete Fourier transforms in SCILIB.
- Chapter 7 describes SCILIB subprograms that compute convolutions and correlations of data sets.
- Chapter 8 describes SCILIB subprograms that deal with linear recurrences.
- Chapter 9 describes miscellaneous subprograms to sort elements of a vector and

to report errors detected in the usage of SCILIB routines.

- Appendix A describes how to use the *contact* utility to submit a software or documentation report or to suggest an enhancement.
- An index is included at the back of the manual.

Notational Conventions

The following conventions are used in this manual:

- *Italics* within text indicate mathematical entities used or manipulated by the program; for example, solve the n by n system of linear equations $Ax=b$.

Italics within command lines indicate generic commands, filenames, or subprogram names. Substitute actual commands, filenames, or subprograms for the *italicized* words. For example, the command line

fc prog_name.o

instructs you to type the command *fc*, followed by the name of a program or subprogram that ends in “.o”.

- **UPPERCASE BOLDFACE** within text and in prototype FORTRAN statements indicates FORTRAN keywords and subprogram names that must be typed just as they appear; for example:

CALL SGESL.

- Type in **lowercase boldface** indicates FORTRAN generic variable or array names in which case you should substitute actual variable or array names. The *italicized* mathematical entities frequently correspond to the **lowercase boldface** variable and array names. For example, A will be the matrix that the FORTRAN array **a** contains in the call:

CALL SGESL (a, lda, n, ipvt, b, job)

- **UPPERCASE CONSTANT WIDTH** represents FORTRAN code.
- Brackets ([]) designate optional entries.
- Many SCILIB subprogram names are prefixed to indicate the type of data they operate on.

For example, the subprograms to copy a vector are SCOPY and CCOPY, for REAL and COMPLEX vector types, respectively.

Note that in Cray FORTRAN, the single precision REAL type is 64 bits (one Cray word) in length. This is essentially equivalent to the type DOUBLE PRECISION in CONVEX FORTRAN. Similarly, the Cray FORTRAN type COMPLEX is 128 bits long; this is equivalent to the CONVEX FORTRAN type DOUBLE COMPLEX. The Cray double precision versions of REAL and COMPLEX types (128 and 256 bit, respectively) are not supported in SCILIB because of processing time considerations.

Associated Documents

Using this guide successfully may require information not specific to the tasks described herein or not within the scope of this guide. The following documents are provided by CONVEX Computer Corporation to help you:

- *CONVEX VECLIB User's Guide*. This guide provides definitions for many subprograms available to SCILIB users through inclusion of VECLIB but not documented in the *SCILIB User's Guide*. Appropriate references to these descriptions are made where necessary.
- *CONVEX Performance Analyzer (CXpa) User's Guide*. This guide explains the operation of the CONVEX Performance Analyzer (CXpa) and the steps needed to create and interpret a CXpa profile.
- *CONVEX FORTRAN Language Reference Manual*. This reference describes in detail the CONVEX extensions and the ANSI standard functions of CONVEX FORTRAN.
- *CONVEX FORTRAN User's Guide*. This guide describes the functions and operations of CONVEX FORTRAN using the ConvexOS operating system.
- *CONVEX FORTRAN Optimization Guide*. This guide describes methods for optimizing FORTRAN programs.
- *CONVEX Application Compiler User's Guide*. This guide describes the CONVEX Application Compiler and how to use it to optimize programs.
- *CONVEX Ada User's Guide*. This guide describes the Ada compiler and support tools and reviews basic Ada concepts.
- *ConvexOS Programmer's Manual, Parts I and II*. This book is the standard reference source for the ConvexOS operating system.
- *CONVEX Consultant User's Guide*. This guide describes the functions and operations of the CONVEX *csd* debugger, the post-mortem dump (*pmd*) facility, and the *prof*, *bprof*, and *gprof* profilers.
- Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart. 1979. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia. This guide provides information on non-optimized LINPACK routines available to SCILIB users through CONVEX VECLIB but not documented in the *CONVEX VECLIB User's Guide*.
- Garbow, B.S., et al. 1977. "Matrix Eigensystem Routines—EISPACK Guide Extension." *Lecture Notes in Computer Science*, Vol. 51. Springer-Verlag, New York. This guide provides information on non-optimized EISPACK routines available to SCILIB users through the inclusion of VECLIB, but not documented in the *VECLIB User's Guide*.

- Smith, B.T., et al. 1976. "Matrix Eigensystem Routines—EISPACK Guide." *Lecture Notes in Computer Science*, Vol. 6, 2nd edition. Springer-Verlag, New York. This guide provides information on non-optimized EISPACK routines available to SCILIB through the inclusion of VECLIB, but not documented in the *VECLIB User's Guide*.

Ordering Documentation

To order CONVEX documentation, contact your local CONVEX office.

To order the current edition of this or any other CONVEX document, you need to know the exact title or the six-character order number. See the copyright page of this document for its six-character order number. See the *CONVEX COMPUTER Price Book* or call your local CONVEX office for the order numbers of other CONVEX documents.

To order an edition other than the current edition, you need to know the 12-digit document number. See the title page of this document for its 12-digit document number. Call your local CONVEX office for the document numbers of other CONVEX documents.

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC). Use the following phone numbers:

Within the continental U.S.	1(800)952-0379
Outside continental U.S.	Contact local CONVEX office

Reader Response

If you have comments or questions about the contents of this book, please notify the CONVEX documentation department by using the Reader's Forum located at the end of this guide.

Introduction to SCILIB

Overview

This chapter provides information necessary for efficient use of SCILIB. This information includes a description of the following: SCILIB optimizations and software standardization, IEEE compatibility, SCILIB subprogram capabilities, CONVEX supercomputer architecture, roundoff effects, and error handling.

The SCILIB software library is a collection of FORTRAN-callable mathematical subprograms which provides a look-alike implementation of the Scientific Library portion of Cray Research Incorporated's UNICOS Math and Scientific Library, V5.0.

All SCILIB routines are optimized for use on the CONVEX family of supercomputers. This general library addresses a variety of linear algebra operations on multiple data types. It contains subprograms for:

- dense vector operations
- sparse vector operations
- matrix operations
- linear equation solution
- discrete Fourier transforms
- convolution and correlation
- linear recurrences
- sorting and error reporting

C and Ada programs can call SCILIB subprograms. These topics are covered in Appendixes A and B, respectively of the *CONVEX VECLIB User's Guide*, which is included in this documentation set.

The UNICOS Math and Scientific Library contains many public domain subprograms that are contained in CONVEX VECLIB and documented in the *CONVEX VECLIB User's Guide* and other books in the *CONVEX VECLIB Documentation Set*, of which this guide is a part. To avoid redundancy in both documentation and software, these subprograms are not included in SCILIB, but can be accessed through VECLIB if necessary. For more information refer to the "Relationship to VECLIB and VECLIB8 Libraries" section that follows.

Chapter Objectives

After reading this chapter you will:

- know how to access SCILIB
- understand pertinent facts about CONVEX architecture
- understand how SCILIB works in a parallel computing environment
- understand roundoff effects
- know SCILIB naming conventions
- understand how SCILIB handles errors
- know how to access the on-line *SCILIB Programmer's Reference*

What You Need to Know to Use SCILIB

You should be familiar with the following information to make efficient use of SCILIB.

Optimizations

Keep in mind that while SCILIB routines are identical in name and purpose to routines found in the UNICOS Math and Scientific Library, they have been optimized for use on CONVEX machines, and this required somewhat different implementations. Since the two machines use different architectures and different methods of carrying out various mathematical operations in hardware and software, SCILIB routines cannot be expected to give identical answers to their Cray counterparts in all cases. However, SCILIB routines have been tested to insure that they give essentially equivalent answers in all circumstances.

Standardization

SCILIB is designed to provide CONVEX users with a look-alike implementation of UNICOS Math and Scientific Library routines. This allows programs written for Cray machines to be easily ported to CONVEX, and it provides a common programmer interface between the two machines, which can (especially when used in conjunction with the `-cfc` command line flag) ease a Cray programmer's transition to writing code for CONVEX.

Relationship to VECLIB and VECLIB8 Libraries

The UNICOS Math and Scientific Library contains many Basic Linear Algebra Subprograms (BLAS), Extended (2-loop and 3-loop) BLAS, LINPACK and EISPACK routines which have already been implemented in CONVEX VECLIB. CONVEX SCILIB is a complete implementation of UNICOS Math and Scientific Library V5.0, and contains all of the functions contained in that library.

The CONVEX VECLIB product consists of three libraries: VECLIB, VECLIB8 and SCILIB. VECLIB is a library of mathematical subprograms with data types that assume default CONVEX FORTRAN sizes; for example, a declaration of type INTEGER is equivalent to INTEGER*4, type REAL is equivalent to REAL*4, DOUBLE PRECISION is equivalent to REAL*8, and COMPLEX is equivalent to COMPLEX*8. VECLIB8 is a similar library which assumes default REAL and INTEGER lengths of 8 bytes and default COMPLEX lengths of 16 bytes; VECLIB8 excludes DOUBLE PRECISION and DOUBLE COMPLEX data types as defined by Cray (i.e. REAL*16 and COMPLEX*32 respectively). VECLIB8's default data typing is therefore identical to that of SCILIB. If you require the use of a VECLIB subprogram that is not included in SCILIB, it is best to use the version supplied in VECLIB8, as it follows SCILIB's typing conventions. For more information, see Table 1-1.

For more information on VECLIB and VECLIB8, refer to the *CONVEX VECLIB User's Guide*, which is included with this documentation set.

IEEE Compatibility

CONVEX supercomputers operate on data represented in either of two floating-point formats: *native* mode or the *IEEE* mode. ANSI/IEEE Standard 754 defines IEEE mode. CONVEX SCILIB operates in either native or IEEE mode. SCILIB automatically determines whether to use native or IEEE mode based on the mode the calling program uses, so you do not need to do anything special to incorporate SCILIB routines into programs compiled to use native or IEEE mode. For further information on IEEE mode, refer to the *CONVEX FORTRAN User's Guide*.

Accessing SCILIB

The SCILIB library of compiled subprograms are ready to be incorporated into your programs by the linker. Simply include the appropriate declarations and **CALL** statements in your FORTRAN source program and specify that SCILIB be used as an object library at link time. Assuming that SCILIB has been installed according to standard procedure, you can access SCILIB subprograms by using the **-l** option on the *fc* command line:

```
fc [options] file [linker-options]
```

where [linker-options] includes the string:

```
-lscilib
```

If you use any VECLIB routines you'll need to include VECLIB8 also. In this case **-lveclib8** must follow SCILIB in the list of linker options. A typical *fc* command line might look like:

```
fc -cfc progr1.f -lscilib -lveclib8
```

The *-cfc* specifies that, among other things, Cray default data types are used in the program. See the "SCILIB Data Types and Naming Conventions" section that follows.

If the **-l** options do not work, SCILIB and/or VECLIB has not been installed according to CONVEX specifications. Contact your system administrator for site-specific access instructions.

Profiling with SCILIB

The CONVEX Performance Analyzer, CXpa, is an interactive tool that gathers and analyzes program execution timing (profiling) data. CXpa provides the programmer with the means to study the timing behavior of a program for the purposes of optimizing, benchmarking, and debugging. To use the performance analyzer, you must first compile your FORTRAN program with either the **-pa**, the **-pab**, or the **-par** compiler option. These options instrument the compiled program so that its performance can be measured at the routine level, the loop level, the block level, or the region level. The SCILIB library has been instrumented at the routine level so that the performance of SCILIB subprograms can be included in the analysis. This instrumentation is nonintrusive, so it is not necessary to use a different version of SCILIB when you desire to profile your program. Also, the CXpa instrumentation in SCILIB does not interfere with the *prof*, *bprof*, or *gprof* instrumentation in your program. However, you may not profile your program with both CXpa and *prof*, *bprof*, or *gprof* at the same time.

Routine-level profiling produces summary information about the subprograms that are called during profiled execution of the program. This information includes

- the number of times each subprogram is called
- the CPU time in each subprogram and the percentage of the program total, either including or excluding the cumulative time in called subprograms
- a dynamic call graph indicating the routine calls that take place within a computer program

CXpa is an optional product. For more information about CXpa, refer to the *CONVEX Performance Analyzer (CXpa) User's Guide*, or contact your CONVEX sales representative.

Optimizing with the Application Compiler

The CONVEX Application Compiler is an interprocedural analyzer that tracks the flow of data and control between procedures. The information generated by this analysis removes scope restrictions on optimization, which allows the Application Compiler to generate more efficient code by taking the entire program, with all its dependencies, into account. The database of program information that the interprocedural analyzer builds up also allows the Application Compiler to perform better error checking, leading to more robust and reliable programs. The SCILIB libraries have been annotated to permit programs built with the Application Compiler to effectively use SCILIB's subroutines.

The CONVEX Application Compiler is an optional product. For more information, refer to the *CONVEX Application Compiler User's Guide*, or contact your CONVEX sales representative.

CONVEX Architecture

The CONVEX architecture:

- treats vectors as single entities
- pipelines arithmetic units
- supports parallel processing

SCILIB is useful because it:

- is a logical interface between FORTRAN and CONVEX supercomputers
- increases performance value

Vectors as Single Entities

CONVEX supercomputers are fast because the architecture allows for efficient operations on vectors as entities. Until recently, most digital computers used scalar processing, that is, each operation involved a single operand or pair of operands and produced a single result.

In contrast, CONVEX supercomputers are vector processors that obtain high computational rates because they operate on vectors as single entities. (A vector is simply an array of numbers that can be processed as a unit; no linear algebra connotations are implied.) For example, if the actual operand is a matrix, entities operated on are not individual elements, but rows or columns treated as vectors.

Pipelined Arithmetic Units

CONVEX supercomputers are also fast because they allow overlapping operations. SCILIB code takes advantage of the pipelined functional units provided in CONVEX architecture. A pipelined arithmetic unit consists of several specialized subsections that work together to execute an operation. As soon as the first section has completed processing the first set of operands, intermediate results are passed to the second section. The first section is then free to begin processing a second set of operands. Meanwhile, the second section does its part of the job before handing the operation to the third section, and so on. If operands are supplied fast enough, the pipeline is filled, and all sections are simultaneously productive. Arithmetic operations proceed in assembly-line fashion.

The vector architecture of CONVEX supercomputers provides a mechanism to maximize the arithmetic production. Furthermore, CONVEX supercomputers have several independent arithmetic units that can operate simultaneously. For example, a vector of operands can be transferred from memory to a vector register while a multiplication and an addition are in progress. Thus, CONVEX supercomputers are fast because they treat vectors as entities, and they overlap arithmetic operations.

Parallel Processing

Parallel processing is available on C200 Series supercomputers with multiple processors. These systems can divide a single computational process into small streams of execution, called *threads*. The result is that you can have more than one processor executing on behalf of the same process.

To support parallel processing, the C200 Series hardware employs automatic self-allocating processors (ASAP) that effectively manage CPU assignment. When another processor can be used to assist in executing a process, the active thread posts a request for any idle CPU(s) to help. New threads are created and executed by available CPUs. If no CPUs are available, the original thread executes all the work for the process.

The SCILIB libraries work in both single processor (C100 Series) and parallel processor (C200 Series) environments. Most SCILIB routines contain both parallel and non-parallel algorithms; at run time, the SCILIB routine determines whether it is being used on a C100 or C200 Series processor. If it is running on a C200 Series processor, it detects if the program is already using multiple threads. It uses this information to automatically choose between a single or parallel processor algorithm. Consequently, you can move programs that use SCILIB between C100 and C200 Series systems freely, without losing compatibility or the advantages of either architecture.

If you are computing on a C200 Series system with multiple processors, you can realize the performance benefits of parallel processing in two ways. First, any SCILIB subprogram that contains parallel code will automatically employ parallelism internally when called. Alternatively, you can call SCILIB subprograms in a parallelized loop or region, in which case the specified loop or region will run multiple copies of the routine's nonparallel algorithm in parallel. To employ this mechanism, you must understand the concept of re-entrancy and use of the `FORCE_PARALLEL`, `BEGIN_TASKS`, `NEXT_TASK`, and `END_TASKS` compiler directives.

SCILIB subprograms are reentrant, which means that they may be called several times in parallel to perform independent computations without one call interfering with another. You can use this feature to call SCILIB subprograms in a parallelized loop or region. The compiler does not automatically parallelize loops containing an SCILIB function reference or subroutine call. You can force it to parallelize such a loop by inserting a `FORCE_PARALLEL` compiler directive before the loop. For example, the following FORTRAN code makes parallel calls to SCILIB subprogram `CLUSEQ`:

```

C$DIR FORCE_PARALLEL
  DO 10 J=1,N
    CALL CLUSEQ (N, X, I, A(J), INDX(I,J), NINDX(J))
  10 CONTINUE

```

While optimizing a parallel program, you might want to make parallel calls to an SCILIB subprogram to perform independent operations, but where the call statements are not in a loop. The compiler does not automatically parallelize code outside a loop. You can use the `BEGIN_TASKS`, `NEXT_TASK`, and `END_TASKS` compiler directives to tell the compiler to parallelize such code. For example, the following FORTRAN code makes parallel calls to SCILIB subprogram `INTMAX`:

```
C$DIR BEGIN_TASKS
      IX = INTMAX(NX,X,1)
C$DIR NEXT_TASK
      IY = INTMAX(NY,Y,1)
C$DIR END_TASKS
```

For more information on compiler directives, including usage cautions and warnings, refer to the documentation set for the specific language you are using.

FORTRAN Language and CONVEX Architecture

FORTRAN is a sequential language. It excludes constructs designed for array operations. One statement is executed at a time—after the preceding statement and before the next statement. A program, therefore, must express an array operation as a sequence of statements that operate on one array element at a time.

SCILIB provides a means for you to express frequently occurring array operations outside the sequential constraints of FORTRAN, which requires you to break an array operation into a sequence of scalar operations. Because you have to break an array operation into a sequence of scalar operations when programming in FORTRAN, the FORTRAN compiler must try to discover what the actual array operation is, then try to determine the best method of evaluating it. There are numerous ways to express an array operation as a sequence of scalar operations, so it is possible, and highly likely, that your program will disguise any but the simplest of operations. This problem is usually the result of an attempt to optimize the computation for a particular computer. If that computer is not a vector processor, these optimizations tend to degrade performance on CONVEX supercomputers.

Performance Value

As computer architectures have become more complicated, it has become more important to know the architecture of the target computer to maximize program performance. When a program is moved from one computer to another, architectural considerations on which the program was based may no longer be valid. If, however, the computationally intensive part of the program is based on highly tuned subprograms from a vendor-supplied library, the vendor's knowledge of the architecture is transferred to the program. SCILIB provides this feature, enabling you to achieve good performance at low cost.

Roundoff Effects

SCILIB subprograms may use a different arithmetic order of evaluation than that employed by the UNICOS Math and Scientific Library or other mathematical software. Different roundoff characteristics may result. Accuracy of results is usually about the same, so using SCILIB should not materially affect the accumulation of roundoff errors in a complete application program. If it does, you should examine the mathematical analysis of the problem, which will likely indicate that the problem is ill-conditioned. Ill-conditioned problems magnify the small roundoff errors that are inadvertently introduced into any computation until they are unproportionally large compared to the desired result. Similarly, if results without and with SCILIB differ materially, both sets of answers are probably inaccurate and you should investigate further. If the program correctly applies stable computational algorithms, the problem itself is probably ill-posed.

SCILIB Data Types and Naming Conventions

Many SCILIB subprograms described in this guide use the naming convention given in the following table of FORTRAN data types and FORTRAN compiler options. For each entry, the letter found under SCILIB precedes the subprogram name according to the subprogram's data type requirements. Dashes ("--") indicate that SCILIB does not support this data type. The remainder of the subprogram name is a mnemonic for the subprogram's function or operation. VECLIB and VECLIB8 libraries are included for reference.

Table 1-1: SCILIB Data Types and Naming Convention

Type	FORTRAN	Option	Length (Bytes)	SCILIB	VECLIB8	VECLIB
Integer	INTEGER		4	--	--	I
Integer	INTEGER*4	-p8, -pd8	4	--	--	I
Integer	INTEGER*4	-cfc	8	I	I	--
Integer	INTEGER	-cfc, -p8, -pd8	8	I	I	--
Integer	INTEGER*8		8	I	I	--
Integer	INTEGER*8	-cfc, -p8, -pd8	8	I	I	--
Real	REAL		4	--	--	S
Real	REAL*4	-p8, -pd8	4	--	--	S
Real	REAL*4	-cfc	8	S	S	D
Real	REAL	-cfc, -p8, -pd8	8	S	S	D
Real	REAL*8		8	S	S	D
Real	REAL*8	-cfc, p8, -pd8	8	S	S	D
Complex	COMPLEX		8	--	--	C
Complex	COMPLEX*8	-p8,-pd8	8	--	--	C
Complex	COMPLEX*8	-cfc	16	C	C	Z
Complex	COMPLEX	-cfc, -p8, -pd8	16	C	C	Z
Complex	COMPLEX*16		16	C	C	Z
Complex	COMPLEX*16	-cfc, -p8, -pd8	16	C	C	Z

For example, subprograms that apply a general matrix-matrix multiply using Strassen's method are named according to data type: SGEMMS and CGEMMS for REAL and COMPLEX matrices, respectively. Some function subprograms use two of these letters, the first describing the data type of the function and the second indicating the type of data on which it operates. For example, the BLAS functions ISAMAX and ICAMAX find the index (INTEGER) of the element of maximum magnitude of a vector for REAL and COMPLEX vectors respectively.

Occasionally, the first two letters of a Cray subprogram name are an abbreviation for the subprogram's purpose or one of its operands. For example, the CLUSEQ subprogram finds clusters in a vector; the SPDOT subprogram computes the sparse dot product of a sparse vector and a dense vector. In cases such as these, the prefix meaning will be made clear by the context in which the subprogram is used.

The table above illustrates SCILIB's use of Cray data type length conventions. Note that SCILIB, like VECLIB8, requires INTEGER, REAL and LOGICAL variables of length 8 bytes and COMPLEX variables of length 16 bytes. These sizes are supplied as defaults when the *-pd8* or *-cfc* command line options are used, which leads us to recommend that you use one of these flags and specify the VECLIB8 library if your program uses both VECLIB and SCILIB routines. SCILIB (and VECLIB8) routines prefixed with I or S will not accept INTEGERS or REALS of length 4 bytes, which is the default for these types in CONVEX FORTRAN and VECLIB. If you call SCILIB or VECLIB8 routines and *do not* compile with *-pd8* or *-cfc*, you must be very careful that all variables and constants passed to routines from either library are of the proper length.

The above table omits DOUBLE PRECISION data types because SCILIB does not support them. For more information on VECLIB and VECLIB8 support of the type DOUBLE PRECISION, refer to Table 1-1 of the *CONVEX VECLIB User's Guide*.

For more information on FORTRAN data types in general, refer to the *CONVEX FORTRAN Language Reference Manual*. For more information on FORTRAN compiler options refer to the *CONVEX FORTRAN User's Guide*. For more information on the *-cfc* flag and the Cray FORTRAN compatibility it provides, see Appendix G of the *CONVEX FORTRAN Language Reference Manual*.

This guide is designed so that related subprograms with similar names are located near one another.

Error Handling

Many SCILIB subprograms use the routine XERSCI to handle errors. XERSCI issues an error message reporting the errant subprogram and, if possible, the offending variable or the error condition that was detected. It then executes a FORTRAN STOP statement. For more information on XERSCI and how to supply your own error handling routine, refer to Chapter 9 of the *SCILIB User's Guide*.

SCILIB Programmer's Reference

The *SCILIB Programmer's Reference* is on-line documentation that includes information from the *SCILIB User's Guide*. This reference contains an introduction to SCILIB and to each set of subprograms in SCILIB, and reference entries for each subprogram. Subprogram entries include descriptions and examples of usage.

This reference is provided for users to easily and efficiently obtain on-line information on SCILIB. The *SCILIB Programmer's Reference* is not a substitute for the *SCILIB User's Guide*; for detailed information on SCILIB, refer to the *User's Guide*.

To access reference entries, type

```
man 3sci entry_name
```

For further explanation and a table of contents of reference entries refer to the *scilib(3sci)* entry by typing

```
man 3sci scilib
```

Support Services

CONVEX maintains a staff to provide technical help if you have difficulty. Located in the CONVEX Technical Assistance Center (TAC), these people are the primary link between you and the company, and they stand ready to assist you with any difficulties. Note, though, that SCILIB has been tested extensively and is very reliable. Therefore, before contacting the TAC about an SCILIB problem, follow this procedure to isolate the cause of the trouble and to simplify the job of resolving it.

- Check any error response provided by the subprogram in question. The subprogram descriptions in this manual describe how to check an error response. If the answer is wrong because an error has been detected, correct the cause of the error and run the job again.
- Verify that the subprogram usage in the program matches the subprogram specifications in this manual. Pay special attention to the number of arguments in the **CALL** statement and to the declarations of arrays and integer constants or variables that describe them. If everything is in order, write out all the arguments immediately before and after the **CALL** statement. Since the values of numeric constants can't be

written out, replace them with variables to which the desired values have been assigned or use the FORTRAN **PARAMETER** statement.

- Make sure there really is a problem. For example, if an apparently incorrect answer is being computed, check to see if the answer does satisfy the problem as defined in the program. Also, for problems with more than one answer, SCILIB may produce a different answer or give the answers in a different order than expected. If the problem is ill-conditioned, SCILIB may not be able to compute a reliable answer at all. Again, error messages often suggest the cause of the problem.
- Isolate the problem. If possible, write a small test program that encounters the same difficulty. Perhaps data causing the problem may be written out from the original program and read into the small one. Try to remove the problem area from a large program and concentrate it in a small program. In this way, you eliminate extraneous code from suspicion. If the problem area is large, try to pare it to a manageable size. For example, if a 50 by 50 linear system fails, try to produce a 2 by 2 system that fails in the same way. Clearly, this is not always possible, but the process often leads to insight.

You will frequently discover a usage error and resolve the problem by following the steps above. If the trouble persists, contact the TAC for help. Providing a small test program and expected answers will help the TAC further analyze the problem. To report a software or documentation problem to the TAC, use the *contact* utility. The *contact* utility allows you to submit a problem or suggest an enhancement directly to the TAC from your own system. Refer to Appendix A for detailed instructions on how to use *contact*.

Public Domain Software

The UNICOS Math and Scientific Library contains LINPACK and EISPACK subprograms. The SCILIB distribution tape includes the LINPACK and EISPACK subprograms as well. LINPACK is a collection of subprograms for analyzing and solving linear algebraic equations. EISPACK is a package of subprograms for computing eigenvalues and/or eigenvectors of several classes of matrices. LINPACK and EISPACK are distributed only in standard Cray precision (64 bit INTEGER, REAL and LOGICAL, and 128 bit COMPLEX). 128 bit REAL and 256 bit COMPLEX data lengths are not supported.

Some of the LINPACK subprograms have already been optimized by CONVEX; these are described in Chapter 4 of the *SCILIB User's Guide*. Similarly, some of the EISPACK routines have been optimized and are covered in Chapter 5 of the *SCILIB User's Guide*. The LINPACK and EISPACK subprograms that have not been optimized are included as compiled from their original source code. You can find documentation for these routines in the *LINPACK Users' Guide*, the *EISPACK Guide*, and the *EISPACK Guide Extension*, all of which are included in the SCILIB documentation set. Future releases of SCILIB will contain more performance-enhanced subprograms from these packages. It is, therefore, to your advantage to build your programs around these subprograms even if their capabilities are not otherwise available in VECLIB or SCILIB.

Supplemental Reading

The SCILIB documentation set includes the *CONVEX VECLIB User's Guide*, the *LINPACK Users' Guide*, the *EISPACK Guide*, and the *EISPACK Guide Extension*.

CONVEX VECLIB User's Guide. CONVEX Computer Corporation, P.O. Box 833851, Richardson, TX 75083.

LINPACK Users' Guide. SIAM Publications, 117 South 17th Street, Philadelphia, PA 19103.

EISPACK Guide. Springer Verlag Lecture Notes in Computer Science. 175 Fifth Avenue, New York, NY 10010.

Matrix Eigensystem Routines—EISPACK Guide Extension. Springer-Verlag Lecture Notes in Computer Science. 175 Fifth Avenue, New York, NY 10010.

Basic Vector Operations

Overview

This chapter explains how to use the SCILIB vector subprograms that serve as building blocks for many user programs. It describes subprograms for performing dense and sparse vector operations, and it includes the FORTRAN equivalent of each subprogram. This set of SCILIB subprograms includes:

- subprograms identical in name and usage to the public-domain Basic Linear Algebra Subprograms (BLAS)
- Cray extensions to the BLAS

Chapter Objectives

After reading this chapter you will:

- understand BLAS storage conventions
- know how to specify array sections
- know how to handle backward storage
- know how to use increment (also called stride) arguments

What You Need to Know to Use These Subprograms

This section discusses commonly used or computationally expensive operations of linear algebra. Even though you can code most of these operations in fewer than 10 lines of FORTRAN, using SCILIB subprograms can improve program performance, as well as program modularity and readability. Note, however, that in some situations you can achieve better computational performance by entering FORTRAN code than by calling one of these subprograms.

BLAS Storage Conventions

The Basic Linear Algebra Subprograms (BLAS) were developed to enhance the portability of published linear algebra codes. In particular, LINPACK, the high-level public-domain package, uses BLAS. Thus, if you use LINPACK from SCILIB you will normally be replacing the standard FORTRAN BLAS with the SCILIB BLAS and increasing the efficiency of LINPACK on your CONVEX supercomputer.

You need not limit your use of SCILIB BLAS to LINPACK. Because these subprograms are portable, modular, self-documenting, and efficient, you can incorporate them into your programs.

To realize the full power of BLAS you must understand the following three subjects:

- FORTRAN storage of arrays
- FORTRAN array argument association
- BLAS indexing conventions

FORTRAN Storage of Arrays

Two-dimensional arrays in FORTRAN are stored by columns. Consider the following specifications:

```
DIMENSION A(N1,N2),B(N3)
EQUIVALENCE (A,B)
```

where $N3 = N1 \times N2$. Then $A(I, J)$ is associated with the same memory location as $B(K)$ where

$$K = I + (J-1) \times N1.$$

Successive elements of a column of A are adjacent in memory, while successive elements of a row of A are stored with a difference of $N1$ storage units between them. Remember that the size of a storage unit depends on the data type.

FORTRAN Array Argument Association

When a FORTRAN subprogram is called with an array element as an argument, the value is not passed. Instead, the subprogram receives the address in memory of the element. Consider the following code segment:

```
REAL A(10,10)
J = 3
L = 10
CALL SUBR (A(1, J), L)
.
.
SUBROUTINE SUBR (X, N)
REAL X(N)
.
.
```

SUBR is given the address of the first element of the third column of A . Since it treats that argument as a one-dimensional array, successive elements $X(1), X(2), \dots$, occupy the same memory locations as the successive elements of the third column of A , that is, $A(1,3), A(2,3), \dots$. Hence, the entire third column of A is available to the subprogram.

BLAS Indexing Conventions

The rest of this section describes dealing with stride arguments and handling forward and backward storage.

A vector in BLAS is defined by three quantities:

1. The vector length.
2. The array or starting element within an array.
3. The increment, sometimes called the *stride*, which defines the number of storage units between successive vector elements.

Forward Storage

Suppose that X is a real array. Let N be the vector length and let $INCX$ be the increment. Suppose that a vector x with components x_i , $i=1, 2, \dots, N$ is stored in X . If $INCX \geq 0$, then x_i is stored in $X(1 + (i-1) \times INCX)$. This is forward storage starting from $X(1)$ with stride equal to $INCX$, ending with $X(1 + (N-1) \times INCX)$. Thus, if $N = 4$ and $INCX = 2$, the vector components x_1 , x_2 , x_3 , and x_4 are stored in the array elements $X(1)$, $X(3)$, $X(5)$, and $X(7)$, respectively.

Backward Storage

Some BLAS routines permit the backward storage of vectors, which is specified by using a negative $INCX$. If $INCX < 0$, then x_i is stored in $X(1 + (N-i) \times |INCX|)$ or equivalently in $X(1 - (i-1) \times INCX)$. This is backward storage starting from $X(1 - (N-1) \times INCX)$ with stride equal to $INCX$, ending with $X(1)$. Thus, if $N = 4$ and $INCX = -2$, the vector components x_1 , x_2 , x_3 , and x_4 are stored in the array elements $X(7)$, $X(5)$, $X(3)$, and $X(1)$, respectively.

$INCX = 0$ is permitted by some BLAS routines and is not permitted by others. When it is allowed, it means that x is a vector of length N , whose components all equal the value of $X(1)$.

The notation $(N, X, INCX)$ describes a BLAS vector. For example, if X is an array of dimension N , then $(N, X, 1)$ represents forward storage and $(N, X, -1)$ represents backward storage. If A is an M by N array, then $(M, A(1, J), 1)$ represents column J and $(N, A(I, 1), M)$ represents row I . Finally, if an M by N matrix is embedded in the upper left-hand corner of an array B of size LDB by $NMAX$, then column J is $(M, B(1, J), 1)$ and row I is $(N, B(I, 1), LDB)$.

Examples

The following examples illustrate how to use increment arguments to perform different operations with the same subprogram. These examples use the function `SDOT` with the following usage:

```
REAL*4 SDOT, S, X(1+(N-1)*|INCX|), Y(1+(N-1)*|INCX|)
S = SDOT (N, X, INCX, Y, INCX)
```

This sets S to the dot product of the vectors $(N, X, INCX)$ and $(N, Y, INCX)$.

Example 1

Compute the dot product $T = X(1)*Y(1) + X(2)*Y(2) + X(3)*Y(3) + X(4)*Y(4)$:

```
REAL*4 SDOT, T, X(4), Y(4)
T = SDOT (4, X, 1, Y, 1)
```

Example 2

Compute the convolution $T = X(1)*Y(4) + X(2)*Y(3) + X(3)*Y(2) + X(4)*Y(1)$:

```
REAL*4 SDOT, T, X(4), Y(4)
T = SDOT (4, X, 1, Y, -1)
```

Example 3

Compute the dot product $Y(2) = A(2,1)*X(1) + A(2,2)*X(2) + A(2,3)*X(3)$, which is the dot product of the second row of an M by 3 matrix A , stored in a 10 by 3 array, with a 3-vector X :

```
PARAMETER (LDA = 10)
REAL*4 SDOT, A(LDA, 3), X(3), Y(LDA)
N = 3
Y(2) = SDOT (N, A(2, 1), LDA, X, 1)
```

Supplemental Reading

Lawson, C., R. Hanson, D. Kincaid, and F. Krogh. September, 1979. "Basic Linear Algebra Subprograms for Fortran Usage." *ACM Transactions on Mathematical Software*, Vol. 5, No. 3.

Subprogram Descriptions

Find Clusters of Selected Vector Elements CLUSEQ, CLUSNE, CLUSFxx, CLUSlxx (xx = GE, GT, LE, or LT)	2-6
Gather a Sparse Vector GATHER	2-9
Count Initial Zero Elements in a Vector IILZ	2-10
Count Initial Positive Elements in a Vector ILLZ	2-11
Count the Number of Occurrences of TRUE Elements of a Vector ILSUM	2-12
Index of the Maximum Element of a Vector INFLMAX	2-13
Index of the Minimum Element of a Vector INFLMIN	2-15
Index of the Element of a Vector of Maximum Magnitude ISAMAX, ICAMAX	2-17
Index of the Element of a Vector of Minimum Magnitude ISAMIN	2-19
Index of the Maximum Element of a Vector ISMAX, INTMAX	2-21
Index of the Minimum Element of a Vector ISMIN, INTMIN	2-23
Search a Vector for a Specified Element ISRCHEQ, ISRCHNE, ISEARCH, ISRCHFxx, ISRCHlxx (xx = GE, GT, LE, or LT)	2-25
Search a Vector for a Specified Element ISRCHMxx (xx = EQ, GE, GT, LE, LT, or NE)	2-27

Search an Ordered Vector for a Specified Element OSRCHF, OSRCHI	2-29
Search an Ordered Vector for a Specified Element OSRCHM	2-31
Sum of Magnitudes of the Elements of a Vector SASUM, SCASUM	2-34
Elementary Vector Operation SAXPY, CAXPY	2-36
Scatter a Sparse Vector SCATTER	2-38
Copy Vector SCOPY, CCOPY	2-40
Dot Product of Two Vectors SDOT, CDOTC, CDOTU	2-42
Euclidean Norm of a Vector SNRM2, SCNRM2	2-45
Sparse Elementary Vector Operation SPAXPY	2-47
Sparse Dot Product of Two Vectors SPDOT	2-49
Apply a Givens Rotation to Two Vectors SROT, CROT	2-51
Construct a Givens Rotation SROTG, CROTG	2-53
Apply a Modified Givens Rotation SROTM	2-55
Construct a Modified Givens Rotation SROTMG	2-57
Sum of the Elements of a Vector SSUM, CSUM	2-61
Swap Two Vectors SSWAP, CSWAP	2-62
Find Indices of Selected Vector Elements WHENEQ, WHENNE, WHENF _{xx} , WHENL _{xx} (xx = GE, GT, LE, or LT)	2-64
Find Indices of Selected Vector Elements WHENM _{xx} (xx = EQ, GE, GT, LE, LT, or NE)	2-67

Purpose Given a real or integer vector x of length n , these subprograms search sequentially through the vector and fill an array with the beginning and ending indices i of the maximal contiguous groups (clusters) of elements which satisfy a specified relationship with a given scalar a .

A cluster is a set of one or more elements $\{x_i, x_{i+1}, \dots, x_j\}$, with the following properties:

- 1) for every k with $i \leq k \leq j$, x_k satisfies the specified relationship with a ,
- 2) either $i = 1$ or x_{i-1} does not satisfy the relationship, and
- 3) either $j = n$ or x_{j+1} does not satisfy the relationship.

At most, there are $\lceil n/2 \rceil$ clusters, where $\lceil x \rceil$ represents the smallest integer greater than or equal to x .

The last two characters of the subprogram name specify the relationship of interest between the elements of the vector and the scalar. These characters and the corresponding cluster relationship may be

<u>xx</u>	<u>Cluster relationship</u>
EQ	$\{i : x_i = a\}$
GE	$\{i : x_i \geq a\}$
GT	$\{i : x_i > a\}$
LE	$\{i : x_i \leq a\}$
LT	$\{i : x_i < a\}$
NE	$\{i : x_i \neq a\}$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 n,incx,indx(2,(n+1)/2),nindx
REAL*8    x(lenx),a
CALL CLUSEQ (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(2,(n+1)/2),nindx
CALL CLUSEQ (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,incx,indx(2,(n+1)/2),nindx
REAL*8    x(lenx),a
CALL CLUSNE (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(2,(n+1)/2),nindx
CALL CLUSNE (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,incx,indx(2,(n+1)/2),nindx
REAL*8    x(lenx),a
CALL CLUSFxx (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(2,(n+1)/2),nindx
CALL CLUSIxx (n, x, incx, a, indx, nindx)
```

Continued

Input	n	Number of elements of vector x to be compared to a . If $n \leq 0$, the subprograms do not reference x or indx .
	x	Array of length $\text{lenx} = (n-1) \times \text{incx} + 1$ containing the n -vector x .
	incx	Increment for the array x : $\text{incx} \geq 0$ x is stored forward in array x , i.e., x_i is stored in $x((i-1) \times \text{incx} + 1)$. $\text{incx} < 0$ x is stored backward in array x , i.e., x_i is stored in $x((i-n) \times \text{incx} + 1)$. Use $\text{incx} = 1$ if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.
	a	The scalar a .
Output	indx	$\text{indx}(1,k)$ and $\text{indx}(2,k)$ contain the indices of the beginning and ending elements of the k th cluster of x respectively. Only the first nindx elements of indx are changed.
	nindx	If $n \leq 0$, then $\text{nindx} = 0$. Otherwise, nindx is the number of clusters of elements of x that satisfy the relationship with a specified by the subprogram name.
Notes		These subprograms are sometimes useful for optimizing a loop containing an IF statement.

**FORTRAN
Equivalent**

```

SUBROUTINE CLUSEQ (N,X, INCX,A, INDX,NINDX)
INTEGER*8 N,X(*), INCX,A, INDX(2,*), NINDX
LOGICAL*8 INCLUS
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
NINDX = 0
INCLUS = .FALSE.
DO 10 I = 1, N
  IF ( .NOT. INCLUS ) THEN
    IF ( X(IX) .EQ. A ) THEN
      NINDX = NINDX + 1
      INDX(1,NINDX) = I
      INCLUS = .TRUE.
    END IF
  ELSE
    IF ( X(IX) .NE. A ) THEN
      INDX(2,NINDX) = I-1
      INCLUS = .FALSE.
    END IF
  END IF
  IX = IX + INCX
10 CONTINUE
IF ( INCLUS ) INDX(2,NINDX) = N
RETURN
END

```

Example Find the clusters of positive elements of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 N, INCX, INDX(2, 11), NINDX
REAL*8    A, X(20)
N = 10
INCX = 1
A = 0.0
CALL CLUSGT (N, X, INCX, A, INDX, NINDX)
```

Gather Sparse Vector**GATHER**

Purpose Given a dense real vector y stored in full storage form, and a set of indices of *interesting* elements of y , this subprogram gathers those elements into a sparse vector x stored in compact form via the set of indices.

More precisely, let $\{k_1, k_2, \dots, k_m\}$ be the indices of the interesting elements. If x is represented by arrays \mathbf{x} and \mathbf{indx} such that $\mathbf{indx}(i) = k_i$ and $\mathbf{x}(i) = x_{k_i}$, then

$$x_i = y_{k_i}, \quad i = 1, 2, \dots, m.$$

Usage SCILIB:

```
INTEGER*8 m,indx(m)
REAL*8    y(n),x(m)
CALL GATHER (m, x, y, indx)
```

Input m Number of interesting elements, $m \leq n$, where n is the length of y . If $m \leq 0$, the subprogram does not reference \mathbf{x} , \mathbf{indx} , or \mathbf{y} .

y Array containing the elements of y , $y(i) = y_i$. Only the elements of y whose indices are included in \mathbf{indx} are accessed.

\mathbf{indx} Array containing the indices $\{k_i\}$ of the interesting elements of y . The indices must satisfy

$$1 \leq \mathbf{indx}(i) \leq n, \quad i = 1, 2, \dots, m,$$

where n is the length of y .

Output \mathbf{x} If $m \leq 0$, then \mathbf{x} is unchanged. Otherwise, the m interesting elements of y : $\mathbf{x}(j) = y_i$ if $\mathbf{indx}(j) = i$.

Notes Cray Research, Inc. had declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

The result is unspecified if any element of \mathbf{indx} is out of range or if \mathbf{x} , \mathbf{indx} , and \mathbf{y} overlap such that any element of y or any index shares a memory location with any element of x .

**FORTTRAN
Equivalent**

```
SUBROUTINE GATHER (M, X, Y, INDX)
INTEGER*8 M,INDX(*)
REAL*8 X(*),Y(*)
DO 10 I = 1, M
    X(I) = Y(INDX(I))
10 CONTINUE
RETURN
END
```

Example

Gather y into x , where y is a vector with interesting elements y_1, y_4, y_5 , and y_9 stored in one-dimensional array Y of dimension 20, and x is a vector stored in compact form in a one-dimensional array X .

```
INTEGER*8 M,INDX(4)
REAL*8    Y(20),X(4)
DATA      INDX / 1, 4, 5, 9 /
M = 4
CALL GATHER (M,X,Y,INDX)
```

Purpose Given an integer vector x of length n , this subprogram counts the number of zero elements of x before the first nonzero element. Given a logical vector x , IILZ counts the number of .FALSE. elements of x before the first .TRUE. element.

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage SCILIB:
 INTEGER*8 i,IILZ,n,x(lenx),incx
 i = IILZ (n, x, incx)

INTEGER*8 i,IILZ,n,incx
 LOGICAL*8 x(lenx)
 i = IILZ (n, x, incx)

Input n Number of elements of vector x . If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

$\text{incx} \geq 0$ x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.
 $\text{incx} < 0$ x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use $\text{incx} = 1$ if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output i If $n \leq 0$, then $i = 0$. If $n > 0$ and all elements of x are zero or .TRUE., then $i = n$. Otherwise, i is the number of the zero or .FALSE. elements x_i of x before the first nonzero or .TRUE. element.

FORTRAN Equivalent

```

INTEGER*8 FUNCTION IILZ (N,X, INCX)
INTEGER*8 N,X(*), INCX
IILZ = 0
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
DO 10 I = 1, N
  IF ( X(IX) .NE. 0 ) RETURN
  IX = IX + INCX
  IILZ = I
10 CONTINUE
RETURN
END

```

Example Determine the number of initial zero elements of an INTEGER*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I, IILZ, N, X(20), INCX
N = 10
INCX = 1
I = IILZ (N, X, INCX)

```

Count Initial Positive Elements**ILLZ**

Purpose Given a vector x of length n , this subprogram counts the number of positive elements of x before the first negative element. In this context, positive means that the leftmost or high-order bit is zero, and negative means that the leftmost bit is one.

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage **SCILIB:**
INTEGER*8 i, ILLZ, n, x(lenx), incx
i = ILLZ (n, x, incx)

Input **n** Number of elements of vector x . If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output **i** If $n \leq 0$, then $i = 0$. If $n > 0$ and all elements of x are zero or **TRUE.**, then $i = n$. Otherwise, i is the number of the positive elements x_i of x before the first negative element.

FORTTRAN Equivalent

```

INTEGER*8 FUNCTION ILLZ (N,X, INCX)
INTEGER*8 N,X(*), INCX
ILLZ = 0
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
DO 10 I = 1, N
IF ( X(IX) .LT. 0 ) RETURN
IX = IX + INCX
ILLZ = I
10 CONTINUE
RETURN
END

```

Example Count the number of initial positive elements of an **INTEGER*8** vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I, ILLZ, N, X(20), INCX
N = 10
INCX = 1
I = ILLZ (N, X, INCX)

```

ILSUM**Count TRUE Vector Elements**

Purpose Given a logical vector x of length n , this subprogram counts the number of elements of the vector that have the logical value `.TRUE.`

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ILSUM,n,incx
LOGICAL*8 x(lenx)
i = ILSUM (n, x, incx)
```

Input

n Number of elements of vector x . If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

```
incx ≥ 0  x is stored forward in array x, i.e.,
           xi is stored in x((i-1) × incx + 1).
incx < 0  x is stored backward in array x, i.e.,
           xi is stored in x((i-n) × incx + 1).
```

Use $\text{incx} = 1$ if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output

i If $n \leq 0$, then $i = 0$. Otherwise, i is the number of elements of x that have the logical value `.TRUE.`

**FORTTRAN
Equivalent**

```
INTEGER*8 FUNCTION ILSUM (N,X, INCX)
LOGICAL*8 X(*)
ILSUM = 0
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
DO 10 I = 1, N
  IF ( X(IX) ) ILSUM = ILSUM + 1
  IX = IX + INCX
10 CONTINUE
RETURN
END
```

Example

Count the number of `.TRUE.` elements of a `LOGICAL*8` vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 I, ILSUM, N, INCX
LOGICAL*8 X(20)
N = 10
INCX = 1
I = ILSUM (N, X, INCX)
```

Index of Maximum Element of Vector**INFLMAX**

Purpose Given a vector x of length n , these subprograms determine the index of the first element x_i in which a specified group of bits attains its maximum value in the vector. Specifically, the subprograms determine the smallest index i such that

$$\text{AND}(\text{SHIFTR}(x_i, rshift), mask) = \max \left\{ \text{AND}(\text{SHIFTR}(x_j, rshift), mask) : j = 1, 2, \dots, n \right\}.$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

INTEGER*8 $i, \text{INFLMAX}, n, x(\text{lenx}), \text{incx}, \text{mask}, rshift$
 $i = \text{INFLMAX}(n, x, \text{incx}, \text{mask}, rshift)$

Input

n Number of elements of vector x to be used. If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,

x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,

x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

mask Mask of 1-bits to extract desired group of bits from the shifted elements of x with a bitwise logical product operation. Refer to "Purpose."

rshift Number of bits by which to right shift each element of x so as to align the specified group of bits with **a**, $0 \leq rshift \leq 63$. Refer to "Purpose."

Output

i If $n \leq 0$, then **i** = 0. Otherwise, **i** is the index of the maximum element of x .

FORTTRAN
Equivalent

```

INTEGER*8 FUNCTION INFLMAX (N,X, INCX, MASK, RSHIFT)
INTEGER*8 N, X(*), INCX, MASK, RSHIFT, TEMP, XMAX
INFLMAX = 1
IF ( N .GT. 1 ) THEN
  IX = 1 + INCX
  IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
  XMAX = AND(SHIFTR(X(IX-INCX), RSHIFT), MASK)
  DO 10 I = 2, N
    TEMP = AND(SHIFTR(X(IX), RSHIFT), MASK)
    IF ( TEMP .GT. XMAX ) THEN
      INFLMAX = I
      XMAX = TEMP
    END IF
    IX = IX + INCX
10  CONTINUE
ELSE IF ( N .LT. 1 ) THEN
  INFLMAX = 0
END IF
RETURN
END

```

Example

Locate the element of an INTEGER*8 vector x in which the field consisting of the rightmost 8 bits achieves its maximum value, where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I, INFLMAX, N, X(20), INCX, MASK, RSHIFT
N = 10
INCX = 1
MASK = 'FF'X
RSHIFT = 0
I = INFLMAX (N, X, INCX, MASK, RSHIFT)

```

Index of Minimum Element of Vector**INFLMIN**

Purpose Given a vector x of length n , these subprograms determine the index of the first element x_i in which a specified group of bits attains its minimum value in the vector. Specifically, the subprograms determine the smallest index i such that

$$\text{AND}(\text{SHIFTR}(x_i, rshift), mask) = \min \left(\text{AND}(\text{SHIFTR}(x_j, rshift), mask) : j = 1, 2, \dots, n \right).$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

INTEGER*8 $i, \text{INFLMIN}, n, x(\text{lenx}), \text{incx}, \text{mask}, rshift$
 $i = \text{INFLMIN}(n, x, \text{incx}, \text{mask}, rshift)$

Input

n Number of elements of vector x to be used. If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

mask Mask of 1-bits to extract desired group of bits from the shifted elements of x with a bitwise logical product operation. Refer to "Purpose."

rshift Number of bits by which to right shift each element of x so as to align the specified group of bits with **a**, $0 \leq rshift \leq 63$. Refer to "Purpose."

Output

i If $n \leq 0$, then $i = 0$. Otherwise, i is the index of the minimum element of x .

FORTRAN
Equivalent

```

INTEGER*8 FUNCTION INFLMIN (N,X, INCX, MASK, RSHIFT)
INTEGER*8 N, X(*), INCX, MASK, RSHIFT, TEMP, XMIN
INFLMIN = 1
IF ( N .GT. 1 ) THEN
  IX = 1 + INCX
  IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
  XMIN = AND(SHIFTR(X(IX-INCX), RSHIFT), MASK)
  DO 10 I = 2, N
    TEMP = AND(SHIFTR(X(IX), RSHIFT), MASK)
    IF ( TEMP .LT. XMIN ) THEN
      INFLMIN = I
      XMIN = TEMP
    END IF
    IX = IX + INCX
10  CONTINUE
ELSE IF ( N .LT. 1 ) THEN
  INFLMIN = 0
END IF
RETURN
END

```

Example

Locate the element of an INTEGER*8 vector x in which the field consisting of the rightmost 8 bits achieves its minimum value, where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I, INFLMIN, N, X(20), INCX, MASK, RSHIFT
N = 10
INCX = 1
MASK = 'FF'X
RSHIFT = 0
I = INFLMIN (N, X, INCX, MASK, RSHIFT)

```

Index of Maximum of Magnitudes

ISAMAX/ICAMAX

Purpose Given a real or integer vector x of length n , ISAMAX determines the index of the element of the vector of maximum magnitude. Specifically, the subprograms determine the smallest index i such that

$$|x_i| = \max \left\{ |x_j| : j = 1, 2, \dots, n \right\}.$$

Given a complex vector x of length n , ICAMAX determines the smallest index i such that

$$|Re(x_i)| + |Im(x_i)| = \max \left\{ |Re(x_j)| + |Im(x_j)| : j = 1, 2, \dots, n \right\}$$

where $Re(x_i)$ and $Im(x_i)$ are the real and imaginary parts of x_i , respectively. The usual definition of complex magnitude is

$$\left\{ Re(x_i)^2 + Im(x_i)^2 \right\}^{1/2}.$$

This definition is not used because of computational speed. If the index i is used for pivot selection in matrix factorization, no significant difference in numerical stability should result.

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage SCILIB:

```
INTEGER*8 i,ISAMAX,n,incx
REAL*8    x(lenx)
i = ISAMAX (n, x, incx)
```

```
INTEGER*8 i,ICAMAX,n,incx
COMPLEX*16 x(lenx)
i = ICAMAX (n, x, incx)
```

Input **n** Number of elements of vector x to be used. If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

$\text{incx} \geq 0$ x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

$\text{incx} < 0$ x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use $\text{incx} = 1$ if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output **i** If $n \leq 0$, then $i = 0$. Otherwise, i is the index of the element of x of maximum magnitude.

Notes The handling of `incx < 0` differs between these subprograms and ISAMAX/ICAMAX in VECLIB.

```

FORTRAN      INTEGER*8 FUNCTION ISAMAX (N,X,INCX)
Equivalent    INTEGER*8 N, INCX
                REAL*8 X(*), TEMP, XMAX
                ISAMAX = 1
                IF ( N .GT. 1 ) THEN
                    IX = 1 + INCX
                    IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
                    XMAX = ABS ( X(IX - INCX) )
                    DO 10 I = 2, N
                        TEMP = ABS ( X(IX) )
                        IF ( TEMP .GT. XMAX ) THEN
                            ISAMAX = I
                            XMAX = TEMP
                        END IF
                        IX = IX + INCX
                    10 CONTINUE
                ELSE IF ( N .LT. 1 ) THEN
                    ISAMAX = 0
                END IF
                RETURN
                END

```

Example Locate the largest element of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

                INTEGER*8 I, ISAMAX, N, INCX
                REAL*8    X(20)
                N = 10
                INCX = 1
                I = ISAMAX (N,X,INCX)

```

Index of Minimum of Magnitudes

ISAMIN

Purpose Given a real or integer vector x of length n , ISAMIN determines the index of element of the vector of minimum magnitude. Specifically, the subprogram determines the smallest index i such that

$$|x_i| = \min \left(|x_j| : j = 1, 2, \dots, n \right).$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ISAMIN,n,incx
REAL*8    x(lenx)
i = ISAMIN (n, x, incx)
```

Input

n Number of elements of vector x to be used. If $n \leq 0$, the subprogram does not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,

x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,

x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output

i If $n \leq 0$, then $i = 0$. Otherwise, i is the index of the element of x of minimum magnitude.

Notes

The handling of **incx** < 0 differs between ISAMIN in SCILIB and VECLIB.

**FORTRAN
Equivalent**

```
INTEGER*8 FUNCTION ISAMIN (N,X, INCX)
INTEGER*8 N, INCX
REAL*8 X(*), TEMP, XMIN
ISAMIN = 1
IF ( N .GT. 1 ) THEN
  IX = 1 + INCX
  IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
  XMIN = ABS ( X(IX-INCX) )
  DO 10 I = 2, N
    TEMP = ABS ( X(IX) )
    IF ( TEMP .LT. XMIN ) THEN
      ISAMIN = I
      XMIN = TEMP
    END IF
    IX = IX + INCX
10  CONTINUE
  ELSE IF ( N .LT. 1 ) THEN
    ISAMIN = 0
  END IF
RETURN
END
```

Example Locate the smallest element of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 I, ISAMIN, N, INCX
REAL*8    X(20)
N = 10
INCX = 1
I = ISAMIN (N, X, INCX)
```

Index of Maximum Element of Vector**ISMAX/INTMAX**

Purpose Given a real or integer vector x of length n , these subprograms determine the index of maximum element of the vector. Specifically, the subprograms determine the smallest index i such that

$$x_i = \max \left(x_j : j = 1, 2, \dots, n \right).$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ISMAX,n,incx
REAL*8    x(lenx)
i = ISMAX (n, x, incx)
```

```
INTEGER*8 i,INTMAX,n,x(lenx),incx
i = INTMAX (n, x, incx)
```

Input

n Number of elements of vector x to be used. If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output

i If $n \leq 0$, then **i** = 0. Otherwise, **i** is the index of the maximum element of x .

Notes

The handling of **incx** < 0 differs between ISMAX in SCILIB and VECLIB.

FORTRAN
Equivalent

```
INTEGER*8 FUNCTION ISMAX (N,X, INCX)
INTEGER*8 N, INCX
REAL*8 X(*), XMAX
ISMAX = 1
IF ( N .GT. 1 ) THEN
  IX = 1 + INCX
  IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
  XMAX = X(IX - INCX)
  DO 10 I = 2, N
    IF ( X(IX) .GT. XMAX ) THEN
      ISMAX = I
      XMAX = X(IX)
    END IF
    IX = IX + INCX
10  CONTINUE
ELSE IF ( N .LT. 1 ) THEN
  ISMAX = 0
END IF
RETURN
END
```

Example

Locate the largest element of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 I, ISMAX, N, INCX
REAL*8    X(20)
N = 10
INCX = 1
I = ISMAX (N,X, INCX)
```

Index of Minimum Element of Vector**ISMIN/INTMIN**

Purpose Given a real or integer vector x of length n , these subprograms determine the index of minimum element of the vector. Specifically, the subprograms determine the smallest index i such that

$$x_i = \min \{ x_j : j = 1, 2, \dots, n \}.$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ISMIN,n,incx
REAL*8    x(lenx)
i = ISMIN (n, x, incx)
```

```
INTEGER*8 i,INTMIN,n,x(lenx),incx
i = INTMIN (n, x, incx)
```

Input

n Number of elements of vector x to be used. If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Output

i If $n \leq 0$, then **i** = 0. Otherwise, **i** is the index of the minimum element of x .

Notes

The handling of **incx** < 0 differs between ISMIN in SCILIB and VECLIB.

FORTTRAN
Equivalent

```

INTEGER*8 FUNCTION ISMIN (N,X, INCX)
INTEGER*8 N, INCX
REAL*8 X(*), XMIN
ISMIN = 1
IF ( N .GT. 1 ) THEN
  IX = 1 + INCX
  IF ( INCX .LT. 0 ) IX = 1 - (N-2) * INCX
  XMIN = X(IX - INCX)
  DO 10 I = 2, N
    IF ( X(IX) .LT. XMIN ) THEN
      ISMIN = I
      XMIN = X(IX)
    END IF
    IX = IX + INCX
10  CONTINUE
ELSE IF ( N .LT. 1 ) THEN
  ISMIN = 0
END IF
RETURN
END

```

Example

Locate the smallest element of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I, ISMIN, N, INCX
REAL*8 X(20)
N = 10
INCX = 1
I = ISMIN (N,X, INCX)

```

Search Vector for Element**ISRCHEQ/ISRCHNE/.../ISRCHLxx**

Purpose Given a real or integer vector x of length n , these subprograms search sequentially through the vector for the first element x_i that satisfies a specified relationship with a given scalar a and return the index i of that element.

The last two characters of the subprogram name specify the relationship of interest between the element of the vector and the scalar. These characters and the corresponding function values may be

xx	Function value
EQ	$\min\{i : x_i = a\}$
GE	$\min\{i : x_i \geq a\}$
GT	$\min\{i : x_i > a\}$
LE	$\min\{i : x_i \leq a\}$
LT	$\min\{i : x_i < a\}$
NE	$\min\{i : x_i \neq a\}$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ISRCHEQ,n,incx
REAL*8    x(lenx),a
i = ISRCHEQ (n, x, incx, a)
```

```
INTEGER*8 i,ISRCHEQ,n,x(lenx),incx,a
i = ISRCHEQ (n, x, incx, a)
```

```
INTEGER*8 i,ISRCHNE,n,incx
REAL*8    x(lenx),a
i = ISRCHNE (n, x, incx, a)
```

```
INTEGER*8 i,ISRCHNE,n,x(lenx),incx,a
i = ISRCHNE (n, x, incx, a)
```

```
INTEGER*8 i,ISEARCH,n,incx
REAL*8    x(lenx),a
i = ISEARCH (n, x, incx, a)
```

```
INTEGER*8 i,ISEARCH,n,x(lenx),incx,a
i = ISEARCH (n, x, incx, a)
```

```
INTEGER*8 i,ISRCHFxx,n,incx
REAL*8    x(lenx),a
i = ISRCHFxx (n, x, incx, a)
```

```
INTEGER*8 i,ISRCHLxx,n,x(lenx),incx,a
i = ISRCHLxx (n, x, incx, a)
```

Input **n** Number of elements of vector x to be compared to a . If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

$\text{incx} \geq 0$ x is stored forward in array x , i.e.,

x_i is stored in $x((i-1) \times \text{incx} + 1)$.

$\text{incx} < 0$ x is stored backward in array x , i.e.,

x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use $\text{incx} = 1$ if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

a The scalar a .

Output **i** If $n \leq 0$, then $i = 0$. If $n > 0$ and no element of x satisfies the relationship with a specified by the subprogram name, then $i = n + 1$. Otherwise, i is the index i of the first element x_i of x that satisfies the relationship with a specified by the subprogram name.

**FORTTRAN
Equivalent**

```

INTEGER*8 FUNCTION ISRCHQ (N,X, INCX,A)
INTEGER*8 N,X(*), INCX,A
ISRCHQ = 0
IF ( N .LE. 0 ) RETURN
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
DO 10 I = 1, N
  IF ( X(IX) .EQ. A ) THEN
    ISRCHQ = I
    RETURN
  END IF
  IX = IX + INCX
10 CONTINUE
ISRCHQ = N+1
RETURN
END

```

Example Search for the first positive element of a REAL*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I,ISRCHFGT,N,INCX
REAL*8 X(20),A
N = 10
INCX = 1
A = 0.0
I = ISRCHFGT (N,X, INCX,A)

```

Search Vector for Element

ISRCHMEQ/ISRCHMGE/.../ISRCHMNE

Purpose Given a vector x of length n , these subprograms search sequentially through the vector for the first element x_i which contains a specified group of bits that satisfies a specified relationship with a given scalar a , and return the index i of that element.

The last two characters of the subprogram name specify the relationship of interest between the element of the vector and the scalar. These characters and the corresponding function values, may be

xx	Function value
EQ	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) = a\}$
GE	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) \geq a\}$
GT	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) > a\}$
LE	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) \leq a\}$
LT	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) < a\}$
NE	$\min\{i : \text{AND}(\text{SHIFTR}(x_i, rshift), mask) \neq a\}$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

Usage**SCILIB:**

```
INTEGER*8 i,ISRCHMxx,n,x(lenx),incx,a,mask,rshift
i = ISRCHMxx (n, x, incx, a, mask, rshift)
```

Input

n Number of elements of vector x to be compared to a . If $n \leq 0$, the subprograms do not reference x .

x Array of length $\text{lenx} = (n-1) \times |\text{incx}| + 1$ containing the n -vector x .

incx Increment for the array x :

incx ≥ 0 x is stored forward in array x , i.e.,
 x_i is stored in $x((i-1) \times \text{incx} + 1)$.

incx < 0 x is stored backward in array x , i.e.,
 x_i is stored in $x((i-n) \times \text{incx} + 1)$.

Use **incx** = 1 if the vector x is stored contiguously in array x , i.e., if x_i is stored in $x(i)$. Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

a The scalar a .

mask Mask of 1-bits to extract desired group of bits from the shifted elements of x with a bitwise logical product operation. Refer to "Purpose."

rshift Number of bits by which to right shift each element of x so as to align the specified group of bits with a , $0 \leq \text{rshift} \leq 63$. Refer to "Purpose."

Output

i If $n \leq 0$, then $i = 0$. If $n > 0$ and no element of x satisfies the relationship with a specified by the subprogram name, then $i = n + 1$. Otherwise, i is the index i of the first element x_i of x that satisfies the relationship with a specified by the subprogram name.

**FORTTRAN
Equivalent**

```

INTEGER*8 FUNCTION ISRCHMEQ (N,X, INCX,A,MASK,RSHIFT)
INTEGER*8 N,X(*), INCX,A,MASK,RSHIFT
ISRCHMEQ = 0
IF ( N .LE. 0 ) RETURN
IX = 1
IF ( INCX .LT. 0 ) IX = 1 - (N-1) * INCX
DO 10 I = 1, N
  IF ( AND(SHIFTR(X(IX),RSHIFT),MASK) .EQ. A ) THEN
    ISRCHMEQ = I
    RETURN
  END IF
  IX = IX + INCX
10 CONTINUE
ISRCHMEQ = N+1
RETURN
END

```

Example

Search for the first odd element of an INTEGER*8 vector x , where x is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 I,ISRCHMGT,N,X(20),INCX,A,MASK,RSHIFT
N = 10
INCX = 1
A = 1
MASK = 1
RSHIFT = 0
I = ISRCHMEQ (N,X, INCX,A,MASK,RSHIFT)

```

Search Ordered Vector for Element

OSRCHF/OSRCHI

- Purpose** Given an ordered real or integer vector x of length n , these subprograms search sequentially through the vector for the first element x_i that equals a given scalar a and return the index i of that element. They also return the number of elements of the vector that are equal to the scalar, and the index of the location within the vector where the scalar should fit in the array, whether they find it or not.
- The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.
- Usage** **SCILIB:**
- ```

INTEGER*8 n,incx,indx,ifound,iwould,icount
REAL*8 x(lenx),a
CALL OSRCHF (n, x, incx, a, ifound, iwould, icount)

INTEGER*8 n,x(lenx),incx,a,ifound,iwould,icount
CALL OSRCHI (n, x, incx, a, ifound, iwould, icount)

```
- Input**
- n** Number of elements of vector  $x$  to be compared to  $a$ . If  $n \leq 0$ , the subprograms do not reference  $x$ .
- x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ . The elements of  $x$  are assumed to be in ascending order.
- incx** Increment for the array  $x$ :
- $\text{incx} \geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .
- $\text{incx} < 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .
- Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.
- a** The scalar  $a$ .
- icount** Flag indicating whether to count the number of occurrences of  $a$  in  $x$ :
- $\text{icount} \neq 0$  do count the number of occurrences of  $a$  in  $x$ .  
 $\text{icount} = 0$  do not count the number of occurrences of  $a$  in  $x$ .
- Output**
- ifound** If  $n \leq 0$ , then  $\text{ifound} = 0$ . If  $n > 0$  and no element of  $x$  equals  $a$ , then  $\text{ifound} = n + 1$ . Otherwise,  $\text{ifound}$  is the index  $i$  of the first element  $x_i$  of  $x$  that equals  $a$ .
- iwould** If  $n \leq 0$ , then  $\text{iwould} = 0$ . If  $n > 0$ , then  $\text{iwould}$  is the index  $i$  of the first element  $x_i$  of  $x$  such that  $x_i \leq a \leq x_{i-1}$ . If  $a$  is found in  $x$ , then  $\text{iwould} = \text{ifound}$ .
- icount** If  $\text{icount} \neq 0$  on entry, the number of occurrences of  $a$  in  $x$ . Not used as output if  $\text{icount} = 0$ .
- Notes** No check is made to ensure that the elements of  $x$  are in ascending order. The output is undefined if the elements are unordered.

**FORTTRAN**  
Equivalent

```

SUBROUTINE ORSCHF (N,X, INCX,A, IFOUND, IWOULD, ICOUNT)
INTEGER*8 N, INCX, IFOUND, IWOULD, ICOUNT
REAL*8 X(*), A
IF (N .LE. 0) RETURN
 IFOUND = 0
 IWOULD = 0
 IF (ICOUNT .NE. 0) ICOUNT = 0
 RETURN
END IF
IX = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
DO 20 I = 1, N
 IF (X(IX) .GE. A) THEN
 IF (X(IX) .EQ. A) THEN
 IFOUND = I
 IWOULD = I
 IF (ICOUNT .NE. 0) THEN
 ICOUNT = 1
 IX = IX + INCX
 DO 10 J = I+1, N
 IF (X(IX) .EQ. A) THEN
 ICOUNT = ICOUNT + 1
 IX = IX + INCX
 ELSE
 RETURN
 END IF
 CONTINUE
 END IF
 ELSE
 IFOUND = N+1
 IWOULD = I
 IF (ICOUNT .NE. 0) ICOUNT = 0
 END IF
 RETURN
END IF
IX = IX + INCX
20 CONTINUE
IFOUND = N+1
IWOULD = N+1
IF (ICOUNT .NE. 0) ICOUNT = 0
RETURN
END

```

**Example**

Search for the first element of a REAL\*8 vector  $x$  equal to 10, where  $x$  is a ordered vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 N, INCX, IFOUND, IWOULD, ICOUNT
REAL*8 X(20), A
N = 10
INCX = 1
A = 10.0
CALL OSRCHF (N,X, INCX,A, IFOUND, IWOULD, ICOUNT)

```

## Search Ordered Vector for Element

OSRCHM

**Purpose** Given an ordered integer vector  $x$  of length  $n$ , this subprogram searches sequentially through the vector for the first element  $x_i$  which contains a specified group of bits that equals a given scalar  $a$  and returns the index  $i$  of that element. The index is such that

$$\text{AND}(\text{SHIFTR}(x_i, rshift), mask) = a$$

It also returns the number of elements of the vector that are equal to the scalar, and the index of the location within the vector where the scalar should fit in the array, whether it finds it or not.

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage**

SCILIB:

```
INTEGER*8 n,x(lenx),incx,a,mask,rshift,ifound,iwould,icount
CALL OSRCHM (n, x, incx, a, mask, rshift, ifound, iwould, icount)
```

**Input**

**n** Number of elements of vector  $x$  to be compared to  $a$ . If  $n \leq 0$ , the subprograms do not reference  $x$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ . The elements of  $x$  are assumed to be in ascending order.

**incx** Increment for the array  $x$ :

**incx**  $\geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**a** The scalar  $a$ .

**mask** Mask of 1-bits to extract desired group of bits from the shifted elements of  $x$  with a bitwise logical product operation. Refer to "Purpose."

**rshift** Number of bits by which to right shift each element of  $x$  so as to align the specified group of bits with  $a$ ,  $0 \leq \text{rshift} \leq 63$ . Refer to "Purpose."

**icount** Flag indicating whether to count the number of occurrences of  $a$  in  $x$ :

**icount**  $\neq 0$  do count the number of occurrences of  $a$  in  $x$ .

**icount** = 0 do not count the number of occurrences of  $a$  in  $x$ .

**Output**

**ifound** If  $n \leq 0$ , then **ifound** = 0. If  $n > 0$  and no element of  $x$  equals  $a$ , then **ifound** =  $n + 1$ . Otherwise, **ifound** is the index  $i$  of the first element  $x_i$  of  $x$  that equals  $a$ .

**iwould** If  $n \leq 0$ , then **iwould** = 0. If  $n > 0$ , then **iwould** is the index  $i$  of the first element  $x_i$  of  $x$  such that  $x_i \leq a \leq x_{i+1}$ . If  $a$  is found in  $x$ , then **iwould** = **ifound**.

**icount** If **icount**  $\neq$  0 on entry, the number of occurrences of *a* in *x*. Not used as output if **icount** = 0.

**Notes** No check is made to ensure that the specified group of bits of the elements of *x* are in ascending order. The output is undefined if the elements are unordered.

**FORTRAN  
Equivalent**

```

SUBROUTINE ORSCHM (N,X, INCX,A, MASK,RSHIFT, IFOUND, IWOULD, ICOUNT)
INTEGER*8 N,X(*), INCX,A, IFOUND, IWOULD, ICOUNT
IF (N .LE. 0) THEN
 IFOUND = 0
 IWOULD = 0
 IF (ICOUNT .NE. 0) ICOUNT = 0
 RETURN
END IF
IX = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
DO 20 I = 1, N
 IF (AND(SHIFTR(X(IX),RSHIFT),MASK) .GE. A) THEN
 IF (AND(SHIFTR(X(IX),RSHIFT),MASK) .EQ. A) THEN
 IFOUND = I
 IWOULD = I
 IF (ICOUNT .NE. 0) THEN
 ICOUNT = 1
 IX = IX + INCX
 DO 10 J = I+1, N
 IF (AND(SHIFTR(X(IX),RSHIFT),MASK) .EQ. A) THEN
 ICOUNT = ICOUNT + 1
 IX = IX + INCX
 ELSE
 RETURN
 END IF
 END IF
 CONTINUE
 END IF
 ELSE
 IFOUND = N+1
 IWOULD = I
 IF (ICOUNT .NE. 0) ICOUNT = 0
 END IF
 RETURN
 END IF
 IX = IX + INCX
20 CONTINUE
IFOUND = N+1
IWOULD = N+1
IF (ICOUNT .NE. 0) ICOUNT = 0
RETURN
END

```

**Example** Search for the first element of an INTEGER\*8 vector  $x$  such that the second group of eight bits from the right contain the value 13, where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 N,X(20),INCX,A,MASK,RSHIFT,IFOUND,IWOULD,ICOUNT
N = 10
INCX = 1
A = 13
MASK = 'FF'X
RSHIFT = 8
CALL OSRCHM (N,X,INCX,A,MASK,RSHIFT,IFOUND,IWOULD,ICOUNT)
```

**Purpose** Given a real or integer vector  $x$  of length  $n$ , SASUM computes the  $l_1$  norm of  $x$ , i.e., the sum of magnitudes of the elements of the vector

$$s = \|x\|_1 = \sum_{i=1}^n |x_i|.$$

Given a complex vector  $x$  of length  $n$ , SCASUM computes

$$s = \sum_{i=1}^n |Re(x_i)| + |Im(x_i)|$$

where  $Re(x_i)$  and  $Im(x_i)$  are the real and imaginary parts of  $x_i$ , respectively. The usual definition of sum of magnitudes of a complex vector is

$$t = \|x\|_1 = \sum_{i=1}^n \left\{ Re(x_i)^2 + Im(x_i)^2 \right\}^{1/2}.$$

$s$  is computed instead of  $t$  since it is faster because it does not require the  $n$  square roots. Since  $t \leq s \leq \sqrt{2}t$ ,  $s$  will often be an acceptable substitute for  $t$ .

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage****SCILIB:**

```
INTEGER*8 n,incx
REAL*8 s,SASUM,x(lenx)
s = SASUM (n, x, incx)
```

```
INTEGER*8 n,incx
REAL*8 s,SCASUM
COMPLEX*16 x(lenx)
s = SCASUM (n, x, incx)
```

**Input**

**n** Number of elements of vector  $x$  to be used in the sum of magnitudes. If  $n \leq 0$ , the subprograms do not reference  $x$ .

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ .  $x$  is stored forward in array  $x$  with increment  $|incx|$ , i.e.,  $x_i$  is stored in  $x((i-1) \times |incx| + 1)$ .

Use  $incx = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**s** If  $n \leq 0$ , then  $s = 0$ . Otherwise,  $s$  is the sum of magnitudes of the elements of  $x$ .

```
FORTRAN REAL*8 FUNCTION SASUM (N, X, INCX)
Equivalent INTEGER*8 N, INCX
 REAL*8 X(*)
 SASUM = 0.0
 IF (N .LE. 0) RETURN
 IX = 1
 INCXA = ABS (INCX)
 DO 10 I = 1, N
 SASUM = SASUM + ABS (X(IX))
 IX = IX + INCXA
10 CONTINUE
 RETURN
 END
```

**Example** Compute the sum of magnitudes of the elements of a REAL\*8 vector  $x$ , where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
 INTEGER*8 N, INCX
 REAL*8 S, DASUM, X(20)
 N = 10
 INCX = 1
 S = DASUM (N, X, INCX)
```

**Purpose** Given a real or complex scalar  $a$  and real or complex vectors  $x$  and  $y$  of length  $n$ , these subprograms perform the elementary vector operations

$$y = ax + y \quad \text{and} \quad y = a\bar{x} + y$$

where  $\bar{x}$  is the complex conjugate of  $x$ . The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx,incy
REAL*8 a,x(lenx),y(leny)
CALL SAXPY (n, a, x, incx, y, incy)
```

```
INTEGER*8 n,incx,incy
COMPLEX*16 a,x(lenx),y(leny)
CALL CAXPY (n, a, x, incx, y, incy)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used in the elementary vector operation. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**a** The scalar  $a$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .  $x$  is used in unconjugated form by the subprograms. Refer to "Purpose."

**incx** Increment for the array  $x$ :

**incx**  $\geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array  $y$ , **incy**  $\neq 0$ :

**incy**  $> 0$   $y$  is stored forward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy**  $< 0$   $y$  is stored backward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use **incy** = 1 if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**y** If  $n \leq 0$  or  $a = 0$ , then  $y$  is unchanged. Otherwise,  $ax + y$  overwrites the input.

**Notes**

If **incx** = 0, then  $x_i = x(1)$  for all  $i$ .

The result is unspecified if **incy** = 0 or if  $x$  and  $y$  overlap such that any element of  $x$  shares a memory location with any element of  $y$ .

```

FORTRAN SUBROUTINE SAXPY (N, A, X, INCX, Y, INCY)
Equivalent INTEGER*8 N, INCX, INCY
 REAL*8 X(*), Y(*), A
 IF (N .LE. 0) RETURN
 IF (A .EQ. 0.0) RETURN
 IX = 1
 IY = 1
 IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
 IF (INCY .LT. 0) IY = 1 - (N-1) * INCY
 DO 10 I = 1, N
 Y(IY) = A * X(IX) + Y(IY)
 IX = IX + INCX
 IY = IY + INCY
 10 CONTINUE
 RETURN
 END

```

**Example 1** Compute the REAL\*8 elementary vector operation

$$y = 2x + y,$$

where  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 A, X(20), Y(20)
N = 10
A = 2.0
INCX = 1
INCY = 1
CALL SAXPY (N, A, X, INCX, Y, INCY)

```

**Example 2** Subtract 3 times the 4th row of a 10 by 10 matrix from the 5th row. The matrix is stored in a two-dimensional array B of dimension 20 by 21.

```

INTEGER*8 N, INCX, INCY
REAL*8 A, B(20, 21)
N = 10
A = -3.0
INCX = 20
INCY = 20
CALL SAXPY (N, A, B(4, 1), INCX, B(5, 1), INCY)

```

**Purpose** Given a sparse vector  $x$  stored in compact form via a set of indices, this subprogram scatters those elements into the corresponding elements of a dense vector  $y$  stored in full storage form.

More precisely, let  $x$  be a sparse  $n$ -vector with  $m \leq n$  *interesting* (usually nonzero) elements, and let  $\{k_1, k_2, \dots, k_m\}$  be the indices of these elements. If  $x$  is represented by arrays  $\mathbf{x}$  and  $\mathbf{indx}$  such that  $\mathbf{indx}(i) = k_i$  and  $\mathbf{x}(i) = x_{k_i}$ , then

$$y_{k_i} = x_i, \quad i = 1, 2, \dots, m.$$

**Usage**

SCILIB:

```
INTEGER*8 m,indx(m)
REAL*8 y(n),x(m)
CALL SCATTER (m, y, indx, x)
```

**Input**

**m** Number of interesting elements,  $m \leq n$ , where  $n$  is the length of  $\mathbf{y}$ . If  $m \leq 0$ , the subprogram does not reference  $\mathbf{x}$ ,  $\mathbf{indx}$ , or  $\mathbf{y}$ .

**indx** Array containing the indices  $\{k_i\}$  of the interesting elements of  $x$ . The indices must satisfy

$$1 \leq \mathbf{indx}(i) \leq n, \quad i = 1, 2, \dots, m$$

and

$$\mathbf{indx}(i) \neq \mathbf{indx}(j), \quad 1 \leq i \neq j \leq m,$$

where  $n$  is the length of  $\mathbf{y}$ .

**x** Array of length  $m$  containing the interesting elements of  $x$ .  $\mathbf{x}(j) = x_i$  if  $\mathbf{indx}(j) = i$ .

**Output**

**y** Array containing the elements of  $y$ ,  $\mathbf{y}(i) = y_i$ . If  $m \leq 0$ , then  $\mathbf{y}$  is unchanged. Otherwise, only the elements of  $\mathbf{y}$  whose indices are included in  $\mathbf{indx}$  are changed.

**Notes**

Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

The result is unspecified if any element of  $\mathbf{indx}$  is out of range, if any two elements of  $\mathbf{indx}$  have the same value, or if  $\mathbf{x}$ ,  $\mathbf{indx}$ , and  $\mathbf{y}$  overlap such that any element of  $x$  or any index shares a memory location with any element of  $y$ .

**FORTTRAN  
Equivalent**

```
SUBROUTINE SCATTER (M, Y, INDX, X)
 INTEGER*8 M, INDX(*)
 REAL*8 Y(*), X(*)
 DO 10 I = 1, M
 Y(INDX(I)) = X(I)
 10 CONTINUE
 RETURN
 END
```

**Example** Scatter  $x$  into  $y$ , where  $x$  is a sparse vector with interesting elements  $x_1$ ,  $x_4$ ,  $x_5$ , and  $x_9$  stored in one-dimensional array  $X$ , and  $y$  is stored in a one-dimensional array  $Y$  of dimension 20.

```
INTEGER*8 M, INDX(4)
REAL*8 Y(20), X(4)
DATA INDX / 1, 4, 5, 9 /
M = 4
CALL SCATTER (M, Y, INDX, X)
```

**Purpose** Given real, integer, or complex vectors  $x$  and  $y$  of length  $n$ , these subprograms perform the vector copy operations

$$y = x \quad \text{and} \quad y = \bar{x}$$

where  $\bar{x}$  is the complex conjugate of  $x$ . The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays. Indexing through the arrays may be either forward or backward.

**Usage**

SCILIB:

```
INTEGER*8 n,incx,incy
REAL*8 x(lenx),y(leny)
CALL SCOPY (n, x, incx, y, incy)
```

```
INTEGER*8 n,incx,incy
COMPLEX*16 x(lenx),y(leny)
CALL CCOPY (n, x, incx, y, incy)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used in the copy operation. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .  $x$  is used in unconjugated form by the subprograms. Refer to "Purpose."

**incx** Increment for the array  $x$ :

```
incx ≥ 0 x is stored forward in array x, i.e.,
 xi is stored in x((i-1)×incx+1).
incx < 0 x is stored backward in array x, i.e.,
 xi is stored in x((i-n)×incx+1).
```

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "Notes" for use of  $\text{incx} = 0$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**incy** Increment for the array  $y$ ,  $\text{incy} \neq 0$ :

```
incy > 0 y is stored forward in array y, i.e.,
 yi is stored in y((i-1)×incy+1).
incy < 0 y is stored backward in array y, i.e.,
 yi is stored in y((i-n)×incy+1).
```

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ . If  $n \leq 0$ , then  $y$  is unchanged. Otherwise,  $y = x$ .

**Notes**

If  $\text{incx} = 0$ , then  $x_i = x(1)$  for all  $i$ . This can be used to initialize all elements of  $y$  to a constant. Refer to "Example 2."

The result is unspecified if  $x$  and  $y$  overlap such that any element of  $x$  shares a memory location with any element of  $y$ .

**FORTRAN**  
**Equivalent**

```

SUBROUTINE SCOPY (N, X, INCX, Y, INCY)
INTEGER*8 N, INCX, INCY
REAL*8 X(*), Y(*)
IF (N .LE. 0) RETURN
IX = 1
IY = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
IF (INCY .LT. 0) IY = 1 - (N-1) * INCY
DO 10 I = 1, N
 Y(IY) = X(IX)
 IX = IX + INCX
 IY = IY + INCY
10 CONTINUE
RETURN
END

```

**Example 1** Copy the REAL\*8 vector  $x$  into  $y$ , where  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 X(20), Y(20)
N = 10
INCX = 1
INCY = 1
CALL SCOPY (N, X, INCX, Y, INCY)

```

**Example 2** Initialize a one-dimensional array to zero.

```

INTEGER*8 N, INCX, INCY
REAL*8 Y(20)
N = 10
INCX = 0
INCY = 1
CALL SCOPY (N, 0.0, INCX, Y, INCY)

```

**Purpose** Given real or complex data vectors  $x$  and  $y$  of length  $n$ , these subprograms compute the dot products

$$s = \sum_{i=1}^n x_i y_i \quad \text{and} \quad s = \sum_{i=1}^n \bar{x}_i y_i$$

where  $\bar{x}$  is the complex conjugate of  $x$ . The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays. Indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,incx,incy
REAL*8 s,SDOT,x(lenx),y(leny)
s = SDOT (n, x, incx, y, incy)
```

```
INTEGER*8 n,incx,incy
COMPLEX*16 s,CDOTC,x(lenx),y(leny)
s = CDOTC (n, x, incx, y, incy)
```

```
INTEGER*8 n,incx,incy
COMPLEX*16 s,CDOTU,x(lenx),y(leny)
s = CDOTU (n, x, incx, y, incy)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used in the dot product. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .  $x$  is used in unconjugated form by the subprograms. Refer to "Purpose."

**incx** Increment for the array  $x$ :

```
incx ≥ 0 x is stored forward in array x, i.e.,
 xi is stored in x((i-1)×incx+1).
incx < 0 x is stored backward in array x, i.e.,
 xi is stored in x((i-n)×incx+1).
```

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array  $y$ :

```
incy ≥ 0 y is stored forward in array y, i.e.,
 yi is stored in y((i-1)×incy+1).
incy < 0 y is stored backward in array y, i.e.,
 yi is stored in y((i-n)×incy+1).
```

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Continued

SDOT/CDOTC/CDOTU

**Output**      **s**      The resulting value of the dot product. If  $n \leq 0$ , then  $s = 0$ . Otherwise,

$$s = \sum_{i=1}^n x_i y_i$$

unless the subprogram name is CDOTC, in which case

$$s = \sum_{i=1}^n \bar{x}_i y_i$$

**Notes**      If  $\text{incx} = 0$ , then  $x_i = x(1)$  for all  $i$ . If  $\text{incy} = 0$ , then  $y_i = y(1)$  for all  $i$ . In either of these cases, another SCILIB subprogram would be more efficient.

**FORTRAN  
Equivalent**

```

REAL*8 FUNCTION SDOT (N, X, INCX, Y, INCY)
INTEGER*8 N, INCX, INCY
REAL*8 X(*), Y(*)
SDOT = 0.0
IF (N .LE. 0) RETURN
IX = 1
IY = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
IF (INCY .LT. 0) IY = 1 - (N-1) * INCY
DO 10 I = 1, N
 SDOT = SDOT + X(IX) * Y(IY)
 IX = IX + INCX
 IY = IY + INCY
10 CONTINUE
RETURN
END

```

**Example 1**      Compute the REAL\*8 dot product

$$s = \sum_{i=1}^{10} x_i y_i,$$

where  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 S, SDOT, X(20), Y(20)
N = 10
INCX = 1
INCY = 1
S = SDOT (N, X, INCX, Y, INCY)

```

**Example 2** Compute the REAL\*8 dot product

$$s = \sum_{i=1}^{10} x_i y_i,$$

where  $x$  is the 4th row of a 10 by 10 matrix stored in a two-dimensional array X of dimension 20 by 21, and  $y$  is a vector 10 elements long stored in one-dimensional array Y of dimension 20.

```
INTEGER*8 N, INCX, INCY
REAL*8 S,SDOT,X(20,21),Y(20)
N = 10
S = SDOT (N,X(4,1),20,Y,1)
```

**Euclidean Norm****SNRM2/SCNRM2**

**Purpose** Given a real, integer, or complex vector  $x$  of length  $n$ , these subprograms compute the Euclidean (i.e.,  $l_2$ ) norm of the vector

$$s = \|x\|_2 = \left\{ \sum_{i=1}^n |x_i|^2 \right\}^{1/2}$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx
REAL*8 s,SNRM2,x(lenx)
s = SNRM2 (n, x, incx)
```

```
INTEGER*8 n,incx
REAL*8 s,SCNRM2
COMPLEX*16 x(lenx)
s = SCNRM2 (n, x, incx)
```

**Input**

**n** Number of elements of vector  $x$  to be used in the Euclidean norm. If  $n \leq 0$ , the subprograms do not reference  $x$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ .  $x$  is stored forward in array  $x$  with increment  $|\text{incx}|$ , i.e.,  $x_i$  is stored in  $x((i-1) \times |\text{incx}| + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**s** If  $n \leq 0$ , then  $s = 0$ . Otherwise,  $s$  is the Euclidean norm of  $x$ .

**FORTTRAN**  
**Equivalent**

```

REAL*8 FUNCTION SNRM2 (N, X, INCX)
REAL*8 T, X(*)
SNRM2 = 0.0
IF (N .LE. 0) RETURN
T = 0.0
IX = 1
INCXA = ABS (INCX)
disable overflow and underflow traps
DO 10 I = 1, N
 T = T + X(IX) ** 2
 IX = IX + INCXA
10 CONTINUE
IF (no overflow occurred) THEN
 IF (T .GT. N * 2.0 ** -104 .OR. no underflow occurred) THEN
 SNRM2 = SQRT (T)
 RETURN
 ELSE
 SCALE = 2.0 ** 72
 END IF
ELSE
 SCALE = 0.5 ** 72
END IF
T = 0.0
IX = 1
DO 20 I = 1, N
 T = T + (SCALE * X(IX)) ** 2
 IX = IX + INCXA
20 CONTINUE
reenable overflow trap if originally enabled
SNRM2 = SQRT (T) / SCALE
RETURN
END

```

**Example**

Compute the Euclidean norm of the REAL\*8 vector  $x$ , where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 N, INCX
REAL*8 S, SNRM2, X(20)
N = 10
INCX = 1
S = SNRM2 (N, X, INCX)

```

## Sparse Elementary Vector Operation

SPAXPY

**Purpose** Given a real scalar  $a$ , a sparse vector  $x$  stored in compact form via a set of indices, and a dense vector  $y$  stored in full storage form, this subprogram performs the elementary vector operation

$$y = ax + y.$$

More precisely, let  $x$  be a sparse  $n$ -vector with  $m \leq n$  interesting (usually nonzero) elements, and let  $\{k_1, k_2, \dots, k_m\}$  be the indices of these elements. All uninteresting elements of  $x$  are assumed to be zero. Let  $y$  be an ordinary  $n$ -vector. If  $x$  is represented by arrays  $\mathbf{x}$  and  $\mathbf{indx}$  such that  $\mathbf{indx}(i) = k_i$  and  $\mathbf{x}(i) = x_{k_i}$ , then these subprograms compute

$$y_{k_i} = ax_i + y_{k_i}, \quad i = 1, 2, \dots, m.$$

**Usage** SCILIB:

```
INTEGER*8 m,indx(m)
REAL*8 a,x(m),y(n)
CALL SPAXPY (m, a, x, y, indx)
```

**Input** **m** Number of interesting elements of  $x$ ,  $m \leq n$ , where  $n$  is the length of  $y$ . If  $m \leq 0$ , the subprogram does not reference  $x$ ,  $\mathbf{indx}$ , or  $y$ .

**a** The scalar  $a$ .

**x** Array of length  $m$  containing the interesting elements of  $x$ .  $\mathbf{x}(j) = x_i$  if  $\mathbf{indx}(j) = i$ .

**y** Array containing the elements of  $y$ ,  $\mathbf{y}(i) = y_i$ .

**indx** Array containing the indices  $\{k_i\}$  of the interesting elements of  $x$ . The indices must satisfy

$$1 \leq \mathbf{indx}(i) \leq n, \quad i = 1, 2, \dots, m$$

and

$$\mathbf{indx}(i) \neq \mathbf{indx}(j), \quad 1 \leq i \neq j \leq m,$$

where  $n$  is the length of  $y$ .

**Output** **y** If  $m \leq 0$  or  $a = 0$ , then  $\mathbf{y}$  is unchanged. Otherwise,  $ax + y$  overwrites the input. Only the elements of  $\mathbf{y}$  whose indices are included in  $\mathbf{indx}$  are changed.

**Notes** The result is unspecified if any element of  $\mathbf{indx}$  is out of range, if any two elements of  $\mathbf{indx}$  have the same value, or if  $\mathbf{x}$ ,  $\mathbf{indx}$ , and  $\mathbf{y}$  overlap such that any element of  $x$  or any index shares a memory location with any element of  $y$ .

```

FORTTRAN SUBROUTINE SPAXPY (M, A, X, Y, INDX)
Equivalent INTEGER*8 M,INDX(*)
 REAL*8 A,X(*),Y(*)
 IF (A .EQ. 0.0) RETURN
 DO 10 I = 1, M
 Y(INDX(I)) = A * X(I) + Y(INDX(I))
10 CONTINUE
 RETURN
 END

```

**Example**      Compute the REAL\*8 elementary vector operation

$$y - 2x + y,$$

where  $x$  is a sparse vector with interesting elements  $x_1, x_4, x_5,$  and  $x_9$  stored in one-dimensional array X, and  $y$  is stored in a one-dimensional array Y of dimension 20.

```

 INTEGER*8 M,INDX(4)
 REAL*8 A,X(4),Y(20)
 DATA INDX / 1, 4, 5, 9 /
 M = 4
 A = 2.0
 CALL SPAXPY (M,A,X,INDX,Y)

```

## Sparse Dot Product

## SPDOT

**Purpose** Given a real sparse vector  $x$  stored in compact form via an index vector, and a dense vector  $y$  stored in full storage form, this subprogram computes the sparse dot product

$$s = \sum_{i=1}^n x_i y_i$$

More precisely, let  $x$  be a sparse  $n$ -vector with  $m \leq n$  interesting (usually nonzero) elements, let  $\{k_1, k_2, \dots, k_m\}$  be the indices of these elements. (While some interesting elements of  $x$  may be zero, all *uninteresting* elements are assumed to be zero.) Let  $y$  be an ordinary  $n$ -vector. If  $x$  is represented by arrays  $\mathbf{x}$  and  $\mathbf{indx}$  such that  $\mathbf{indx}(i) = k_i$  and  $\mathbf{x}(i) = x_{k_i}$ , then these subprograms compute

$$s = \sum_{i=1}^m \mathbf{x}_i y_{k_i}$$

**Usage**

**SCILIB:**

```
INTEGER*8 m,indx(m)
REAL*8 s,SPDOT,y(n),x(m)
s = SPDOT (m, y, indx, x)
```

**Input**

**m** Number of interesting elements of  $x$ ,  $m \leq n$ . If  $m \leq 0$ , the subprogram does not reference  $\mathbf{x}$ ,  $\mathbf{indx}$ , or  $\mathbf{y}$ .

**y** Array containing the elements of  $y$ ,  $\mathbf{y}(i) = y_i$ .

**indx** Array containing the indices  $\{k_i\}$  of the interesting elements of  $x$ . The indices must satisfy

$$1 \leq \mathbf{indx}(i) \leq n, \quad i = 1, 2, \dots, m,$$

where  $n$  is the length of  $\mathbf{y}$ .

**x** Array of length  $m$  containing the interesting elements of  $x$ .

**Output**

**s** The resulting value of the dot product. If  $m \leq 0$ , then  $s = 0$ . Otherwise,

$$s = \sum_{i=1}^m \mathbf{x}(i) \times \mathbf{y}(\mathbf{indx}(i)).$$

**FORTTRAN  
Equivalent**

```
REAL*8 FUNCTION SPDOT (M, Y, INDX, X)
INTEGER*8 M, INDX(*)
REAL*8 Y(*), X(*)
SPDOT = 0.0
DO 10 I = 1, M
 SPDOT = SPDOT + X(I) * Y(INDX(I))
10 CONTINUE
RETURN
END
```

**Example**      Compute the REAL\*8 sparse dot product

$$s = \sum_{i=1}^{10} x_i y_i,$$

where  $x$  is a sparse vector with interesting elements  $x_1$ ,  $x_4$ ,  $x_5$ , and  $x_9$  stored in one-dimensional array  $X$ , and  $y$  is a vector 10 elements long stored in a one-dimensional array  $Y$  of dimension 20.

```
INTEGER*8 M,INDX(4)
REAL*8 S,SPDOT,Y(20),X(4)
DATA INDX / 1, 4, 5, 9 /
M = 4
S = SPDOT (M,Y,INDX,X)
```

## Apply Givens Rotation

## SROT/CROT

**Purpose** Given a real scalar  $c$ , a real or complex scalar,  $s$  and real or complex vectors  $x$  and  $y$  of length  $n$ , these subprograms apply the Givens rotation

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} c & s \\ -\bar{s} & c \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{for } i = 1, \dots, n$$

where  $\bar{s}$  is the complex conjugate of  $s$ ;  $\bar{s} = s$  if  $s$  is real. The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

Usually,  $c$  and  $s$  have been determined by the companion subprogram SROTG, or CROTG.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx,incy
REAL*8 x(lenx),y(leny),c,s
CALL SROT (n, x, incx, y, incy, c, s)
```

```
INTEGER*8 n,incx,incy
REAL*8 c
COMPLEX*16 x(lenx),y(leny),s
CALL CROT (n, x, incx, y, incy, c, s)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used in the Givens rotation. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array  $y$ ,  $\text{incy} \neq 0$ :

**incy** > 0  $y$  is stored forward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy** < 0  $y$  is stored backward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**c** The scalar  $c$ .

**s** The scalar  $s$ .

**Output**  $x$  and  $y$  If  $n \leq 0$  or if  $c = 1$  and  $s = 0$ , then  $x$  and  $y$  are unchanged. Otherwise, the result vectors overwrite the input.

**Notes** The result is unspecified if  $incx = 0$  or  $incy = 0$  or if  $x$  and  $y$  overlap such that any element of  $x$  shares a memory location with any element of  $y$ .

SCILIB also contains subprograms that construct and apply modified Givens rotations. They are documented elsewhere in this chapter. The modified Givens subprograms are a little more difficult to use, but are more efficient on the CONVEX supercomputer.

**FORTRAN  
Equivalent**

```

SUBROUTINE SROT (N, X, INCX, Y, INCY, C, S)
REAL*8 C, S, TEMP, X(*), Y(*)
INTEGER*8 N, INCX, INCY
IF (N .LE. 0) RETURN
IF (C .EQ. 1.0 .AND. S .EQ. 0.0) RETURN
IX = 1
IY = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
IF (INCY .LT. 0) IY = 1 - (N-1) * INCY
DO 10 I = 1, N
 TEMP = C * X(IX) + S * Y(IY)
 Y(IY) = C * Y(IY) - S * X(IX)
 X(IX) = TEMP
 IX = IX + INCX
 IY = IY + INCY
10 CONTINUE
RETURN
END

```

**Example 1** Apply a Givens rotation to  $x$  and  $y$ , vectors 10 elements long stored in one-dimensional arrays  $X$  and  $Y$  of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 X(20), Y(20), C, S
N = 10
INCX = 1
INCY = 1
CALL SROT (N, X, INCX, Y, INCY, C, S)

```

**Example 2** Reduce 10 by 10 matrix  $a$  stored in two-dimensional array  $A$  of dimension 20 by 21 to upper-triangular form via Givens rotations (compare with "Example 2" in the description of SROTM).

```

INTEGER*8 INCA, I, J, N
REAL*8 A(20,21), C, S
INCA = 20
DO 20 I = 1, 9
 N = 10 - I
 DO 10 J = I+1, 10
 CALL SROTG (A(I, I), A(J, I), C, S)
 CALL SROT (N, A(I, I+1), INCA, A(J, I+1), INCA, C, S)
10 CONTINUE
20 CONTINUE

```

**Construct Givens Rotation****SROTG/CROTG**

**Purpose** Given real or complex scalars  $a$  and  $b$ , these subprograms construct a Givens plane rotation matrix that annihilates  $b$ . Specifically, they determine scalars  $c$  and  $s$  such that

$$\begin{bmatrix} c & s \\ -\bar{s} & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

where  $c$  is real,  $r$  and  $s$  are of the same type as  $a$  and  $b$ , and  $\bar{s}$  is the complex conjugate of  $s$ .

Usually,  $c$  and  $s$  are passed to companion subprogram SROT, or CROT to apply the Givens rotation to a pair of vectors.

SROTG also determines a quantity  $z$  that permits the later stable reconstruction of  $c$  and  $s$  from a single quantity.

**Usage****SCILIB:**

```
REAL*8 a,b,c,s
CALL SROTG (a, b, c, s)
```

```
REAL*8 c
COMPLEX*16 a,b,s
CALL CROTG (a, b, c, s)
```

**Input**

**a** The scalar  $a$ .

**b** The scalar  $b$ .

**Output**

**a** The rotated result  $r$  overwrites  $a$ .

**b** Not used as output by CROTG. In SROTG, the reconstruction quantity  $z$  overwrites  $b$ . The reconstruction quantity  $z$  is useful if a matrix is being transformed by a sequence of Givens rotations that must be saved to be applied again. Since each  $z$  overwrites an element that has been reduced to zero, the transformations can be saved without using any additional storage.

The quantities  $c$  and  $s$  may be reconstructed from  $z$  as follows:

```
if |z| = 0, set c = 0 and s = 1.
if |z| < 0, set c = √(1-z2) and s = z.
if |z| > 0, set c = 1/z and s = √(1-c2).
```

**c** The rotation scalar  $c$ .

**s** The rotation scalar  $s$ .

**Notes**

SCILIB also contains subprograms that construct and apply modified Givens rotations. They are documented elsewhere in this chapter. The modified Givens subprograms are a little more difficult to use, but are more efficient on the CONVEX supercomputer.

**Example** Construct a Givens plane rotation that will rotate vectors  $x$  and  $y$  in such a way as to annihilate  $y_1$ .  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```
REAL*8 X(20),Y(20),C,S
CALL SROTG (X(1),Y(1),C,S)
```

X(1) is the rotated result and Y(1) is the reconstruction quantity, so these elements should not be rotated by a subsequent call to SROT.

**Apply Modified Givens Rotation****SROTM**

**Purpose** Given a modified Givens rotation matrix  $H = \{h_{ij}\}$  as constructed by SROTM, and real vectors  $x$  and  $y$  of length  $n$ , these subprograms apply the modified rotation

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{for } i = 1, \dots, n.$$

Refer to the description of the companion subprogram SROTMG for more details about the modified Givens rotation.

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,incx,incy
REAL*8 x(lenx),y(leny),param(5)
CALL SROTM (n, x, incx, y, incy, param)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,

$x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,

$x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array  $y$ ,  $\text{incy} \neq 0$ :

**incy**  $> 0$   $y$  is stored forward in array  $y$ , i.e.,

$y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy**  $< 0$   $y$  is stored backward in array  $y$ , i.e.,

$y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**param** Array containing the matrix elements of the modified Givens rotation matrix  $H$  and a flag indicating which form the rotation matrix takes, and therefore which of the elements of **param** are significant. **param** will usually have been set by the companion subprogram SROTMG; refer to the description of this companion subprogram for the specific contents of **param**.

**Output**     **x** and **y**    If  $n \leq 0$  or if **param**(1) = -2, **x** and **y** are unchanged. Otherwise, the result vectors overwrite the input.

**Notes**        The result is unspecified if **incx** = 0 or **incy** = 0 or if **x** and **y** overlap such that any element of **x** shares a memory location with any element of **y**.

SCILIB also contains subprograms that construct and apply regular Givens rotations. They are documented elsewhere in this chapter. The modified Givens subprograms are a little more difficult to use, but are more efficient on the CONVEX supercomputer.

**Example 1**     Apply a modified Givens rotation to **x** and **y**, vectors 10 elements long stored in one-dimensional arrays **X** and **Y** of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 X(20), Y(20), PARAM(5)
N = 10
INCX = 1
INCY = 1
CALL DROTM (N, X, INCX, Y, INCY, PARAM)

```

**Example 2**     Reduce 10 by 10 matrix **a** stored in two-dimensional array **A** of dimension 20 by 21 to upper-triangular form via modified Givens rotations (compare with "Example 2" in the description of SROT.)

```

INTEGER*8 INCA, I, J, N
REAL*8 A(20, 21), D(20), PARAM(5)
INCA = 20
DO 10 I = 1, 10
 D(I) = 1.0
10 CONTINUE
DO 30 I = 1, 9
 N = 10 - I
 DO 20 J = I+1, 10
 CALL SROTMG (D(I), D(J), A(I, I), A(J, I), PARAM)
 CALL SROTM (N, A(I, I+1), INCA, A(J, I+1), INCA, PARAM)
20 CONTINUE
30 CONTINUE
DO 40 I = 1, 10
 N = 11 - I
 CALL SSCAL (N, SQRT(D(I)), A(I, I), INCA)
40 CONTINUE

```

**Construct Modified Givens Rotation****SROTMG**

**Purpose** The Givens rotation,  $G$ , that annihilates  $z_1$ , if  $z_1 \neq 0$ , is

$$GW = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} w_1 & \cdots & w_n \\ z_1 & \cdots & z_n \end{bmatrix},$$

where  $c = w_1/r$ ,  $s = z_1/r$ , and  $r = \pm(w_1^2 + z_1^2)^{1/2}$ . Computing  $G$  and applying it to a pair of  $n$  vectors requires  $\sim 4n$  floating-point multiplications,  $\sim 2n$  floating-point additions, and one square root.

The modified Givens rotation is a device for reducing this operation count. Suppose that  $W$  above is available in factored form

$$W = D^{1/2}X = \begin{bmatrix} d_1^{1/2} & 0 \\ 0 & d_2^{1/2} \end{bmatrix} \cdot \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}.$$

These subprograms construct  $\bar{d}_1$ ,  $\bar{d}_2$ , and  $H$  such that  $GW$  is obtained in the same factored form in which  $W$  was given

$$GW = \begin{bmatrix} \bar{d}_1^{1/2} & 0 \\ 0 & \bar{d}_2^{1/2} \end{bmatrix} \cdot \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \cdot \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{bmatrix}.$$

$H$  is chosen to have the same numerical stability as the standard Givens rotation but better computational efficiency. Thus,  $H$  will usually have two elements equal to  $\pm 1$ . When this is true, computing  $H$  and applying it to a pair of  $n$ -vectors requires  $\sim 2n$  floating-point multiplications,  $\sim 2n$  floating-point additions, and no square roots. Companion SCILIB subprograms SROTM are provided to apply the modified Givens notation to a pair of vectors.

In most applications,  $d_1$  and  $d_2$  are initially set to 1, are manipulated by SROTMG as the modified Givens rotations are constructed, then are applied to the vectors as the final step in the computation. For example, the reduction of an  $n$  by  $n$  matrix to upper-triangular form via modified Givens rotations requires  $O(n)$  square roots compared to the  $O(n^2)$  required by ordinary Givens rotations. Refer to "Example 2" in the description of SROTM.

**Usage** SCILIB:

```
REAL*8 d1,d2,x1,y1,param(5)
CALL SROTMG (d1, d2, x1, y1, param)
```

**Input**

**d1** The scale factor for the "x" row.

**d2** The scale factor for the "y" row.

**x1** The first element of the "x" row.

**y1** The first element of the "y" row. This is the element that will be annihilated by the rotation.

**Output**

**d1** The updated scale factor for the "x" row.

- d2** The updated scale factor for the "y" row.
- x1** The rotated first element of the "x" row.
- param** Array containing the matrix elements of the modified Givens rotation matrix  $H$  and a flag indicating which form the rotation matrix  $H$  takes and, therefore, which elements of **param** are significant. **param** will usually be an argument to the companion subprogram SROTM.

**param(1)** specifies the form of the rotation matrix  $H$ , as follows:

$$\text{param}(1) = -2 \quad H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\text{param}(1) = -1 \quad H = \begin{bmatrix} \text{param}(2) & \text{param}(4) \\ \text{param}(3) & \text{param}(5) \end{bmatrix}$$

$$\text{param}(1) = 0 \quad H = \begin{bmatrix} 1 & \text{param}(4) \\ \text{param}(3) & 1 \end{bmatrix}$$

$$\text{param}(1) = 1 \quad H = \begin{bmatrix} \text{param}(2) & 1 \\ -1 & \text{param}(5) \end{bmatrix}$$

For each of the four values of **param(1)**, only the indicated values of **param(2)** through **param(5)** are defined. The 0, 1, and -1 elements are not stored in **param**.

**Notes** SCILIB also contains subprograms that construct and apply ordinary Givens rotations. They are documented elsewhere in this chapter. The modified Givens subprograms are a little more difficult to use, but are more efficient on the CONVEX supercomputer.

**Example** Construct a modified Givens plane rotation that will rotate vectors  $d_1x_i$  and  $d_2y_i$  in such a way as to annihilate  $d_2y_i$ .  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```
REAL*8 D1,D2,X(20),Y(20),PARAM(5)
CALL SROTMG (D1,D2,X(1),Y(1),PARAM)
```

X(1) is the rotated result, so it should not be rotated by a subsequent call to SROTM.

## Scale Vector

## SSCAL/CSCAL/CSSCAL

**Purpose** Given a real or complex scalar  $a$  and a real or complex vector  $x$  of length  $n$ , these subprograms perform the vector scaling operations

$$x \leftarrow ax \text{ and } x \leftarrow a\bar{x}$$

where  $\bar{x}$  is the complex conjugate of  $x$ . The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage****SCILIB:**

```
INTEGER*8 n,incx
REAL*8 a,x(lenx)
CALL SSCAL (n, a, x, incx)
```

```
INTEGER*8 n,incx
COMPLEX*16 a,x(lenx)
CALL CSCAL (n, a, x, incx)
```

```
INTEGER*8 n,incx
REAL*8 a
COMPLEX*16 x(lenx)
CALL CSSCAL (n, a, x, incx)
```

**Input**

**n** Number of elements of vector  $x$  to be used in the scaling operation. If  $n \leq 0$ , the subprograms do not reference  $x$ .

**a** The scalar  $a$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .  $x$  is used in unconjugated form by the subprograms. Refer to "Purpose."

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ .  $x$  is stored forward in array  $x$  with increment  $|\text{incx}|$ , i.e.,  $x_i$  is stored in  $x((i-1) \times |\text{incx}| + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**x** If  $n \leq 0$ , then  $x$  is unchanged. Otherwise,  $ax$  replaces the input.

**Notes**

The result is unspecified if  $\text{incx} = 0$ .

**FORTRAN  
Equivalent**

```
SUBROUTINE SSCAL (N,A, X, INCX)
REAL*8 A, X(*)
INTEGER*8 N, INCX
IF (N .LE. 0) RETURN
IX = 1
INCXA = ABS (INCX)
DO 10 I = 1, N
 X(IX) = A * X(IX)
 IX = IX + INCXA
10 CONTINUE
RETURN
END
```

**Example**      Scale the REAL\*8 vector  $x$  by 2, where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 N, INCX
REAL*8 A, X(20)
N = 10
INCX = 1
A = 2.0
CALL SSCAL (N, A, X, INCX)
```

## Vector Sum

SSUM/CSUM

**Purpose** Given a real, integer, or complex vector  $x$  of length  $n$ , these subprograms compute the sum of the elements of the vector

$$s = \sum_{i=1}^n x_i.$$

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage**

SCILIB:

```
INTEGER*8 n,incx
REAL*8 s,SSUM,x(lenx)
s = SSUM (n, x, incx)
```

```
INTEGER*8 n,incx
COMPLEX*16 s,CSUM,x(lenx)
s = CSUM (n, x, incx)
```

**Input**

**n** Number of elements of vector  $x$  to be used in the sum. If  $n \leq 0$ , the subprograms do not reference  $x$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ .  $x$  is stored forward in array  $x$  with increment  $|\text{incx}|$ , i.e.,  $x_i$  is stored in  $x((i-1) \times |\text{incx}| + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**s** If  $n \leq 0$ , then  $s = 0$ . Otherwise,  $s$  is the sum of the elements of  $x$ .

**FORTRAN  
Equivalent**

```
REAL*8 FUNCTION SSUM (N, X, INCX)
 INTEGER*8 N, INCX
 REAL*8 X(*)
 SSUM = 0.0
 IF (N .LE. 0) RETURN
 IX = 1
 INCXA = ABS (INCX)
 DO 10 I = 1, N
 SSUM = SSUM + X(IX)
 IX = IX + INCXA
 10 CONTINUE
 RETURN
 END
```

**Example**

Compute the sum of the elements of a REAL\*8 vector  $x$ , where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```
INTEGER*8 N, INCX
REAL*8 S, X(20)
N = 10
INCX = 1
S = DSUM (N, X, INCX)
```

**Purpose** Given real, integer, or complex vectors  $x$  and  $y$  of length  $n$ , these subprograms perform the vector interchange operation

$$x \leftrightarrow y.$$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx,incy
REAL*8 x(lenx),y(leny)
CALL SSWAP (n, x, incx, y, incy)
```

```
INTEGER*8 n,incx,incy
COMPLEX*16 x(lenx),y(leny)
CALL CSWAP (n, x, incx, y, incy)
```

**Input**

**n** Number of elements of vectors  $x$  and  $y$  to be used in the swap operation. If  $n \leq 0$ , the subprograms do not reference  $x$  or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array  $y$ ,  $\text{incy} \neq 0$ :

**incy**  $> 0$   $y$  is stored forward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy**  $< 0$   $y$  is stored backward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**Output**

**x and y** If  $n \leq 0$ , then  $x$  and  $y$  are unchanged. Otherwise,  $x$  and  $y$  are interchanged in  $x$  and  $y$ .

**Notes**

The result is unspecified if  $\text{incx} = 0$  or  $\text{incy} = 0$  or if  $x$  and  $y$  overlap such that any element of  $x$  shares a memory location with any element of  $y$ .

## Continued

**FORTRAN**  
**Equivalent**

```

SUBROUTINE SSWAP (N, X, INCX, Y, INCY)
REAL*8 TEMP, X(*), Y(*)
INTEGER*8 N, INCX, INCY
IF (N .LE. 0) RETURN
IX = 1
IY = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
IF (INCY .LT. 0) IY = 1 - (N-1) * INCY
DO 10 I = 1, N
 TEMP = X(IX)
 X(IX) = Y(IY)
 Y(IY) = TEMP
 IX = IX + INCX
 IY = IY + INCY
10 CONTINUE
RETURN
END

```

**Example 1** Interchange REAL\*8 vectors  $x$  and  $y$ , where  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays X and Y of dimension 20.

```

INTEGER*8 N, INCX, INCY
REAL*8 X(20), Y(20)
N = 10
INCX = 1
INCY = 1
CALL SSWAP (N, X, INCX, Y, INCY)

```

**Example 2** Interchange rows 3 and 6 of a 10 by 10 matrix  $a$  stored in two-dimensional array A of dimension 20 by 21.

```

INTEGER*8 N, INCA
REAL*8 A(20, 21)
N = 10
INCA = 20
CALL SSWAP (N, A(3, 1), INCA, A(6, 1), INCA)

```

**Purpose** Given a real or integer vector  $x$  of length  $n$ , these subprograms search sequentially through the vector and fill an array with a list of the indices  $i$  for which the elements  $x_i$  satisfy a specified relationship with a given scalar  $a$ .

The last two characters of the subprogram name specify the relationship of interest between the elements of the vector and the scalar. These characters and the corresponding list contents may be

| <u>xx</u> | <u>List contents</u> |
|-----------|----------------------|
| EQ        | { $i : x_i = a$ }    |
| GE        | { $i : x_i \geq a$ } |
| GT        | { $i : x_i > a$ }    |
| LE        | { $i : x_i \leq a$ } |
| LT        | { $i : x_i < a$ }    |
| NE        | { $i : x_i \neq a$ } |

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage****SCILIB:**

```
INTEGER*8 n,incx,indx(n),nindx
REAL*8 x(lenx),a
CALL WHENEQ (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(n),nindx
CALL WHENEQ (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,incx,indx(n),nindx
REAL*8 x(lenx),a
CALL WHENNE (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(n),nindx
CALL WHENNE (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,incx,indx(n),nindx
REAL*8 x(lenx),a
CALL WHENFxx (n, x, incx, a, indx, nindx)
```

```
INTEGER*8 n,x(lenx),incx,a,indx(n),nindx
CALL WHENIxx (n, x, incx, a, indx, nindx)
```

**Input** **n** Number of elements of vector  $x$  to be compared to  $a$ . If  $n \leq 0$ , the subprograms do not reference  $x$  or  $indx$ .

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ :

**incx**  $\geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

Continued

WHENEQ/WHENNE/.../WHENILT

**a** The scalar  $a$ .

**Output**

**indx** Array filled with the list of indices  $i$  of the elements  $x_i$  of  $x$  that satisfy the relationship with  $a$  specified by the subprogram name. Only the first **nindx** elements of **indx** are changed.

**nindx** If  $n \leq 0$ , then **nindx** = 0. Otherwise, **nindx** is the number of elements of  $x$  that satisfy the relationship with  $a$  specified by the subprogram name.

**Notes** These subprograms are sometimes useful for optimizing a loop containing an **IF** statement. Refer to "Example 2."

**FORTRAN  
Equivalent**

```

SUBROUTINE WHENEQ (N,X,INCX,A,INDX,NINDX)
INTEGER*8 N,X(*),INCX,A,INDX(*),NINDX
IX = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
NINDX = 0
DO 10 I = 1, N
 IF (X(IX) .EQ. A) THEN
 NINDX = NINDX + 1
 INDX(NINDX) = I
 END IF
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example 1** Find the zero elements of a REAL\*8 vector  $x$ , where  $x$  is a vector 10 elements long stored in a one-dimensional array X of dimension 20.

```

INTEGER*8 N,INCX,INDX(20),NINDX
REAL*8 A,X(20)
N = 10
INCX = 1
A = 0.0
CALL WHENEQ (N,X,INCX,A,INDX,NINDX)

```

**Example 2** Optimize the following program segment, where the **THEN** clause of the **IF** statement is much more likely than the **ELSE** clause.

```

INTEGER*4 I,N
REAL*8 A,B,D,DLIM,R
REAL*8 F(20000),X(20000),Y(20000),Z(20000)
N = 20000
DO 10 I = 1, N
 D = SQRT(X(I)**2 + Y(I)**2 + Z(I)**2) - R
 IF (D .GT. DLIM) THEN
 F(I) = A * EXP(B * D)
 ELSE
 CALL FORCE (D,F(I))
 END IF
10 CONTINUE

```

Change D to an array and introduce array **INDX** to hold the indices corresponding to the **ELSE** clause. Split the body of the **DO** loop into two parts. The first part corresponds to the body of the loop before the **IF** statement and the **THEN** clause. It fully vectorizes, so even though it computes a few more exponentials than the original code, it is still considerably faster. **WHENFLE** is then called to determine the indices for which the **ELSE** clause must be executed, and the second **DO** loop executes the **ELSE** clause for those indices. The resulting program segment is

```

INTEGER*4 I, J, N, INDX(20000), NINDX
REAL*8 A, B, DLIM, R
REAL*8 D(20000), F(20000), X(20000), Y(20000), Z(20000)
N = 20000
DO 10 I = 1, N
 D(I) = SQRT(X(I)**2 + Y(I)**2 + Z(I)**2) - R
 F(I) = A * EXP(B * D(I))
10 CONTINUE
CALL WHENFLE (N,D,1,DLIM,INDX,NINDX)
DO 20 J = 1, NINDX
 I = INDX(J)
 CALL FORCE (D(I),F(I))
20 CONTINUE

```

Find Selected Vector Elements WHENMEQ/WHENMGE/.../WHENMNE

**Purpose** Given a vector  $x$  of length  $n$ , these subprograms search sequentially through the vector and fill an array with a list of the indices of the elements  $x_i$  which contain a specified group of bits that satisfy a specified relationship with a given scalar  $a$ .

The last two characters of the subprogram name specify the relationship of interest between the elements of the vector and the scalar. These characters and the corresponding list contents may be

| xx | List contents                                                               |
|----|-----------------------------------------------------------------------------|
| EQ | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) = a\}$    |
| GE | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) \geq a\}$ |
| GT | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) > a\}$    |
| LE | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) \leq a\}$ |
| LT | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) < a\}$    |
| NE | $\{i : \text{AND}(\text{SHIFTR}(x_i, \text{rshift}), \text{mask}) \neq a\}$ |

The vector may be stored in a one-dimensional array or either in a row or a column of a two-dimensional array.

**Usage****SCILIB:**

```
INTEGER*8 n,x(lenx),incx,a,indx(n),nind,mask,rshift
CALL WHENMxx (n, x, incx, a, indx, nindx, mask, rshift)
```

**Input**

**n** Number of elements of vector  $x$  to be compared to  $a$ . If  $n \leq 0$ , the subprograms do not reference  $x$  or  $indx$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ :

**incx**  $\geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .  
**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to this chapter.

**a** The scalar  $a$ .

**mask** Mask of 1-bits to extract desired group of bits from the shifted elements of  $x$  with a bitwise logical product operation. Refer to "Purpose."

**rshift** Number of bits by which to right shift each element of  $x$  so as to align the specified group of bits with  $a$ ,  $0 \leq \text{rshift} \leq 63$ . Refer to "Purpose."

**Output**

**indx** Array filled with the list of indices  $i$  of the elements  $x_i$  of  $x$  that satisfy the relationship with  $a$  specified by the subprogram name. Only the first **nindx** elements of **indx** are changed.

**nindx** If  $n \leq 0$ , then **nindx** = 0. Otherwise, **nindx** is the number of elements of  $x$  that satisfy the relationship with  $a$  specified by the subprogram name.

**FORTRAN  
Equivalent**

```

SUBROUTINE WHENMEQ (N,X,INCX,A,INDX,NINDX,MASK,RSHIFT)
INTEGER*8 N,X(*),INCX,A,INDX(*),NINDX,MASK,RSHIFT
IX = 1
IF (INCX .LT. 0) IX = 1 - (N-1) * INCX
NINDX = 0
DO 10 I = 1, N
 IF (AND(SHIFTR(X(IX),RSHIFT),MASK) .EQ. A) THEN
 NINDX = NINDX + 1
 INDX(NINDX) = I
 END IF
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example**

Find the odd elements of an INTEGER\*8 vector  $x$ , where  $x$  is a vector 10 elements long stored in a one-dimensional array  $X$  of dimension 20.

```

INTEGER*8 N,X(20),INCX,A,INDX(20),NINDX,MASK,RSHIFT
N = 10
INCX = 1
A = 1
MASK = 1
RSHIFT = 0
CALL WHENMEQ (N,X,INCX,A,INDX,NINDX,MASK,RSHIFT)

```

# Basic Matrix Operations

## Overview

This chapter describes the subprograms in the Level 2 (two-loop) BLAS and the Level 3 (three-loop) BLAS. Collectively, these two sets of subprograms are called the Extended BLAS. The most important of these subprograms have been coded in highly tuned CONVEX assembly language. The remaining Level 2 and Level 3 BLAS routines are coded in FORTRAN, but they will be recoded in assembly language for a future release of SCILIB. You should use these routines to take advantage of the optimized performance that is now available or will soon be available.

This chapter explains how to use the SCILIB matrix subprograms, which perform common computationally-intensive linear algebra operations. The operations covered are:

- basic matrix/vector operations
- basic matrix/matrix operations

Chapter 4 discusses matrix inverse operations.

## Chapter Objectives

After reading this chapter you will:

- be familiar with the Extended BLAS subroutine naming convention
- know what operations the Extended BLAS performs
- know how to use the described subprograms

## What You Need to Know to Use These Subprograms

### Subroutine Naming Convention

The Extended BLAS uses a subroutine naming convention that encodes the function of each subroutine into its name. Extended BLAS subprogram names consist of four, five, or six characters in the form TXXY, TXXYY, or TXXYYY.

The first letter in the naming convention indicates one of the four FORTRAN data types, as shown in Table 3-1:

**Table 3-1: Extended BLAS Naming Convention — Data Type**

| <b>T</b> | <b>Data Type</b>         |
|----------|--------------------------|
| S        | Single Precision REAL    |
| C        | Single Precision COMPLEX |

The next two letters in the naming convention indicate the form of the matrix, as presented in Table 3-2:

**Table 3-2: Extended BLAS Naming Convention — Matrix Form**

| <b>XX</b> | <b>Form of Matrix</b> |
|-----------|-----------------------|
| GE        | General               |
| GB        | General band          |
| HE        | Hermitian             |
| HB        | Hermitian band        |
| HP        | Hermitian packed      |
| SY        | Symmetric             |
| SB        | Symmetric band        |
| SP        | Symmetric packed      |
| TR        | Triangular            |
| TB        | Triangular band       |
| TP        | Triangular packed     |

Table 3-3 lists the final one, two, or three characters in the naming convention, indicating the computation of a particular subroutine:

**Table 3-3: Extended BLAS Naming Convention — Computation**

| <b>YY</b> | <b>Subroutine Computation</b>              |
|-----------|--------------------------------------------|
| MM        | Matrix-Matrix multiply                     |
| MV        | Matrix-Vector multiply                     |
| R         | Rank-1 update                              |
| R2        | Rank-2 update                              |
| RK        | Rank-k update                              |
| R2K       | Rank-2k update                             |
| SM        | Solve multiple systems of linear equations |
| SV        | Solve a system of linear equations         |

For example, SGBMV multiplies a vector (MV) by a general band matrix (GB) using the single precision REAL data type (S). CTRSM solves a system of linear equations with one triangular coefficient matrix and a matrix of right-hand sides, using the single precision COMPLEX data type.

Table 3-4 shows the valid combinations of T, XX, and Y, YY, or YYY. Each line indicates the allowable T prefixes and Y, YY, or YYY suffixes for a particular root name XX.

**Table 3-4: Extended BLAS Naming Convention — Subprogram Names**

| Valid T | XX | Valid Y, YY, or YYY |    |    |   |    |    |     |
|---------|----|---------------------|----|----|---|----|----|-----|
| S       | GE | MM                  | MV | R  |   |    |    |     |
|         | C  | GE                  | MM | MV |   |    | RC | RU  |
| S       | C  | GB                  |    | MV |   |    |    |     |
|         | C  | HE                  | MM | MV | R | R2 | RK | R2K |
|         | C  | HB                  |    | MV |   |    |    |     |
|         | C  | HP                  |    | MV | R | R2 |    |     |
| S       |    | SY                  | MM | MV | R | R2 | RK | R2K |
|         | C  | SY                  | MM |    |   |    | RK | R2K |
| S       |    | SB                  |    | MV |   |    |    |     |
| S       |    | SP                  |    | MV | R | R2 |    |     |
| S       | C  | TR                  | MM | MV |   |    | SM | SV  |
| S       | C  | TB                  |    | MV |   |    |    | SV  |
| S       | C  | TP                  |    | MV |   |    |    | SV  |

## Supplemental Reading

Dongarra, J.J., J. DuCroz, S. Hammarling, and R. Hanson. March, 1988. "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, Vol. 14, No. 1.

Dongarra, J.J., J. DuCroz, S. Hammarling, and I. Duff. March, 1990. "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, Vol. 16, No. 1.

## Subprogram Descriptions

|                                                                              |      |
|------------------------------------------------------------------------------|------|
| Specialized Matrix-Matrix Multiply<br>MXM .....                              | 3-5  |
| Generalized Matrix-Matrix Multiply<br>MXMA .....                             | 3-7  |
| Specialized Matrix-Vector Multiply<br>MXV .....                              | 3-11 |
| Generalized Matrix-Vector Multiply<br>MXVA .....                             | 3-13 |
| General Band Matrix-Vector Multiply<br>SGBMV, CGBMV .....                    | 3-16 |
| General Matrix-Matrix Multiply<br>SGEMM, CGEMM .....                         | 3-20 |
| General Matrix-Matrix Multiply via Strassen's Method<br>SGEMMS, CGEMMS ..... | 3-23 |
| General Matrix-Vector Multiply<br>SGEMV, CGEMV .....                         | 3-27 |
| General Rank-1 Update<br>SGER, CGERC, CGERU .....                            | 3-30 |

## Basic Matrix Operations

|                                                                                   |      |
|-----------------------------------------------------------------------------------|------|
| Matrix-Vector Multiply and Add<br>SMXPY .....                                     | 3-32 |
| Symmetric or Hermitian Band Matrix-Vector Multiply<br>SSBMV, CHBMV .....          | 3-34 |
| Symmetric or Hermitian Matrix-Vector Multiply<br>SSPMV, CHPMV .....               | 3-38 |
| Symmetric or Hermitian Rank-1 Update<br>SSPR, CHPR .....                          | 3-41 |
| Symmetric or Hermitian Rank-2 Update<br>SSPR2, CHPR2 .....                        | 3-44 |
| Symmetric or Hermitian Matrix-Matrix Multiply<br>SSYMM,CHEMM, CSYMM, ZHEMM, ..... | 3-47 |
| Symmetric or Hermitian Matrix-Vector Multiply<br>SSYMV, CHEMV .....               | 3-50 |
| Symmetric or Hermitian Rank-1 Update<br>SSYR, CHER .....                          | 3-53 |
| Symmetric or Hermitian Rank-2 Update<br>SSYR2, CHER2 .....                        | 3-55 |
| Symmetric or Hermitian Rank-2k Update<br>SSYR2K, CHER2K, CSYR2K .....             | 3-58 |
| Symmetric or Hermitian Rank-k Update<br>SSYRK, CHERK, CSYRK .....                 | 3-61 |
| Triangular Band Matrix-Vector Multiply<br>STBMV, CTBMV .....                      | 3-64 |
| Solve Triangular Band System<br>STBSV, CTBSV .....                                | 3-68 |
| Triangular Matrix-Vector Multiply<br>STPMV, CTPMV .....                           | 3-72 |
| Solve One Triangular System<br>STPSV, CTPSV .....                                 | 3-75 |
| Triangular Matrix-Matrix Multiply<br>STRMM, CTRMM .....                           | 3-78 |
| Triangular Matrix-Vector Multiply<br>STRMV, CTRMV .....                           | 3-81 |
| Solve Simultaneous Triangular Systems<br>STRSM, CTRSM .....                       | 3-83 |
| Solve One Triangular System<br>STRSV, CTRSV .....                                 | 3-86 |
| Matrix-Vector Multiply and Add<br>SXMPY .....                                     | 3-89 |
| BLAS Error Handler<br>XERBLA .....                                                | 3-91 |

## Specialized Matrix-Matrix Multiply

MXM

- Purpose** This subprogram computes the matrix-matrix product  $C = AB$ , where  $A$  is an  $m$  by  $k$  matrix,  $B$  is a  $k$  by  $n$  matrix, and  $C$  is an  $m$  by  $n$  matrix. The elements of the matrices must be stored in consecutive memory locations in two-dimensional arrays of size  $m$  by  $k$ ,  $k$  by  $n$ , and  $m$  by  $n$ , respectively. SCILIB subprograms SGEMM, SGEMMS, and MXMA allow more general matrix storage and also admit the transposes of  $A$ ,  $B$ , and, in the case of MXMA,  $C$ .
- Usage** **SCILIB:**  
**INTEGER\*8** m,k,n  
**REAL\*8** a(m,k),b(k,n),c(m,n)  
**CALL** MXM (a,m,b,k,c,n)
- Input**
- a** Array containing the  $m$  by  $k$  matrix  $A$ .
  - m** Number of rows in matrices  $A$  and  $C$ ,  $m \geq 0$ . If  $m = 0$ , the subprogram does not reference **a**, **b**, or **c**.
  - b** Array containing the  $k$  by  $n$  matrix  $B$ .
  - k** Number of columns in matrix  $A$  and number of rows in matrix  $B$ ,  $k \geq 0$ . If  $k = 0$ , the subprogram computes  $C = 0$  without referencing **a** or **b**.
  - n** Number of columns in matrices  $B$  and  $C$ ,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference **a**, **b**, or **c**.
- Output**
- c** The result  $C$  matrix.
- Notes** Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.
- If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

**m** < 0,  
**n** < 0, and  
**k** < 0.

**FORTTRAN Equivalent** Except for the argument error checking, the following FORTRAN subroutine is equivalent to MXMA, and illustrates the meanings of the six increment arguments.

```

SUBROUTINE MXM (A, M, B, K, C, N)
 INTEGER*8 M, K, N
 REAL*8 A(M, K), B(K, N), C(M, N)
 DO 110 J = 1, N
 DO 120 I = 1, M
 C(I, J) = 0.0
110 CONTINUE
120 CONTINUE
 DO 150 J = 1, N
 DO 140 L = 1, K
 DO 130 I = 1, M
 C(I, J) = C(I, J) + A(I, L) * B(L, J)
130 CONTINUE
140 CONTINUE
150 CONTINUE
 RETURN
 END

```

**Example** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  whose dimensions are 9 by 6,  $B$  is a 6 by 8 real matrix stored in an array  $B$  of dimension 6 by 8, and  $C$  is a 9 by 8 real matrix stored in an array  $C$ , also of dimension 9 by 8.

```

INTEGER*8 M, K, N
REAL*8 A(9, 6), B(6, 8), C(9, 8)
M = 9
N = 8
K = 6
CALL MXM (A, M, B, K, C, N)

```

## Generalized Matrix-Matrix Multiply

MXMA

**Purpose** This subprogram computes the matrix-matrix product  $C = AB$ , where  $A$  is an  $m$  by  $k$  matrix,  $B$  is a  $k$  by  $n$  matrix, and  $C$  is an  $m$  by  $n$  matrix. The rows and columns of the matrices may be stored with unit or non-unit strides, effectively allowing  $A$ ,  $B$ , and  $C$ , or their transposes, to be stored in two-dimensional arrays via the storage-association rules of FORTRAN.

SCILIB subprograms SGEMM and SGEMMS allow matrix and transposed matrix storage without resorting to storage association, also admitting the ability to add or subtract the product matrix from the original contents of the result matrix.

**Usage**

SCILIB:

```
INTEGER*8 ia,ja,ib,jb,ic,jc,m,k,n
REAL*8 a(lena),b(lenb),c(lenc)
CALL MXMA (a,ia,ja, b,ib,jb, c,ic,jc, m,k,n)
```

**Input**

**a** Array containing the  $m$  by  $k$  matrix  $A$ . Typically, **a** will be a two-dimensional array with the rows and columns of  $A$  comprising one-dimensional array sections of **a**. Refer to "Notes" for suggested usages. Treating **a** as a one-dimensional array results in

$$\text{lena} = (m-1) \times |\text{ia}| + (k-1) \times |\text{ja}| + 1.$$

$A_{ij}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq k$ , is stored in

$$\mathbf{a}((i-1) \times \text{ia} + (j-1) \times \text{ja} + 1).$$

Note that negative **ia** or **ja** will result in subscript values that lie outside the **a** array as declared above. This need not be an error; see "Example 3" for details.

**ia** Storage increment between successive elements in the same column of matrix  $A$  in array **a**. Refer to "Notes" for suggested values.

**ja** Storage increment between successive elements in the same row of matrix  $A$  in array **a**. Refer to "Notes" for suggested values.

**b** Array containing the  $k$  by  $n$  matrix  $B$ . Typically, **b** will be a two-dimensional array with the rows and columns of  $B$  comprising one-dimensional array sections of **b**. Refer to "Notes" for suggested usages. Treating **b** as a one-dimensional array results in

$$\text{lenb} = (k-1) \times |\text{ib}| + (n-1) \times |\text{jb}| + 1.$$

$B_{ij}$ ,  $1 \leq i \leq k$ ,  $1 \leq j \leq n$ , is stored in

$$\mathbf{b}((i-1) \times \text{ib} + (j-1) \times \text{jb} + 1).$$

Note that negative **ib** or **jb** will result in subscript values that lie outside the **b** array as declared above. This need not be an error; see "Example 3" for details.

**ib** Storage increment between successive elements in the same column of matrix  $B$  in array **b**. Refer to "Notes" for suggested values.

**jb** Storage increment between successive elements in the same row of matrix  $B$  in array **b**. Refer to "Notes" for suggested values.

- ic** Storage increment between successive elements in the same column of matrix  $C$  in array  $c$ . Refer to "Notes" for suggested values.
- jc** Storage increment between successive elements in the same row of matrix  $C$  in array  $c$ . Refer to "Notes" for suggested values.
- m** Number of rows in matrices  $A$  and  $C$ ,  $m \geq 0$ . If  $m = 0$ , the subprogram does not reference  $a$ ,  $b$ , or  $c$ .
- k** Number of columns in matrix  $A$  and number of rows in matrix  $B$ ,  $k \geq 0$ . If  $k = 0$ , the subprogram computes  $C = 0$  without referencing  $a$  or  $b$ .
- n** Number of columns in matrices  $B$  and  $C$ ,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $b$ , or  $c$ .
- Output**
- c** The result  $C$  matrix. Typically,  $c$  will be a two-dimensional array with the rows and columns of  $C$  comprising one-dimensional array sections of  $c$ . Refer to "Notes" for suggested usages. Treating  $c$  as a one-dimensional array results in

$$\text{lenc} = (m-1) \times |\text{ic}| + (n-1) \times |\text{jc}| + 1.$$

$C_{ij}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , is stored in

$$c((i-1) \times \text{ic} + (j-1) \times \text{jc} + 1).$$

Note that negative  $\text{ic}$  or  $\text{jc}$  will result in subscript values that lie outside the  $c$  array as declared above. This need not be an error; see "Example 3" for details.

**Notes** Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

Typically,  $a$ ,  $b$ , and  $c$  will be two-dimensional arrays with the rows and columns comprising one-dimensional sections of the arrays, i.e., one subscript will vary within a row or column of the matrix, and the other will be constant.

If  $a$ , for example, is a two-dimensional array of dimension  $\text{lda}$  by  $\text{mda}$ , and  $A$  is stored in untransposed form in  $a$ , then  $\text{ia} = 1$  and  $\text{ja} = \text{lda}$ , while if  $A$  is stored in transposed form ( $A^T$  is stored), then  $\text{ia} = \text{lda}$  and  $\text{ja} = 1$ .

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

m < 0,
n < 0,
k < 0,
ia = 0,
ja = 0,
ib = 0,
jb = 0,
ic = 0, and
jc = 0.

```

## Continued

## MXMA

**FORTTRAN Equivalent** Except for the argument error checking, the following FORTRAN subroutine is equivalent to MXMA, and illustrates the meanings of the six increment arguments.

```

 SUBROUTINE MXMA (A, IA, JA, B, IB, JB, C, IC, JC, M, K, N)
 INTEGER*8 IA, JA, IB, JB, IC, JC, M, K, N
 REAL*8 A(*), B(*), C(*)
 DO 120 J = 1, N
 DO 110 I = 1, M
 C((I-1)*IC+(J-1)*JC+1) = 0.0 ! C(I, J) = 0.0
110 CONTINUE
120 CONTINUE
 DO 150 J = 1, N
 DO 140 L = 1, K
 DO 130 I = 1, M
 C((I-1)*IC+(J-1)*JC+1) = ! C(I, J) =
1 C((I-1)*IC+(J-1)*JC+1) + ! C(I, J) +
2 A((I-1)*IA+(L-1)*JA+1) * ! A(I, L) *
3 B((L-1)*IB+(J-1)*JB+1) ! B(L, J)
130 CONTINUE
140 CONTINUE
150 CONTINUE
 RETURN
 END

```

**Example 1** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  of dimension 10 by 11,  $B$  is a 6 by 8 real matrix stored in an array  $B$  of dimension 12 by 13, and  $C$  is a 9 by 8 real matrix stored in an array  $C$ , of dimension 14 by 15.

```

 INTEGER*8 IA, JA, IB, JB, IC, JC, M, K, N
 REAL*8 A(10,11), B(12,13), C(14,15)
 IA = 1
 JA = 10
 IB = 1
 JB = 12
 IC = 1
 JC = 14
 M = 9
 N = 8
 K = 6
 CALL MXMA (A, IA, JA, B, IB, JB, C, IC, JC, M, K, N)

```

**Example 2** Form the REAL\*8 matrix product  $C = A^T B$ , where  $A$  is a 6 by 9 real matrix stored in an array A of dimension 10 by 11,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 12 by 13, and  $C$  is a 9 by 8 real matrix stored in an array C, of dimension 14 by 15.

```

INTEGER*8 IA, JA, IB, JB, IC, JC, M, K, N
REAL*8 A(10,11), B(12,13), C(14,15)
IA = 10
JA = 1
IB = 1
JB = 12
IC = 1
JC = 14
M = 9
N = 8
K = 6
CALL MXMA (A, IA, JA, B, IB, JB, C, IC, JC, M, K, N)

```

**Example 3** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 9 by 6 real matrix stored "upside-down and backwards", i.e., with the row and column subscripts decreasing to the right and bottom, in an array A of dimension 10 by 11,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 12 by 13, and  $C$  is a 9 by 8 real matrix stored in an array C, of dimension 14 by 15.

```

INTEGER*8 IA, JA, IB, JB, IC, JC, M, K, N
REAL*8 A(10,11), B(12,13), C(14,15)
IA = -1
JA = -10
IB = 1
JB = 12
IC = 1
JC = 14
M = 9
N = 8
K = 6
CALL MXMA (A(M,K), IA, JA, B, IB, JB, C, IC, JC, M, K, N)

```

## Specialized Matrix-Vector Multiply

MXV

**Purpose** This subprogram computes the matrix-vector product  $y = Ax$ , where  $A$  is an  $m$  by  $n$  matrix,  $x$  is an  $n$ -vector, and  $y$  is an  $m$ -vector. The elements of  $A$  must be stored in consecutive memory locations in a two-dimensional array of size  $m$  by  $n$ , and the elements of the  $x$  and  $y$  must be stored in consecutive memory locations in one-dimensional arrays of size  $n$  and  $m$ , respectively. SCILIB subprograms SGEMV and MXVA allow more general storage and also admit the transpose of  $A$ . SGEMV also admits the ability to add or subtract the product from the original contents of the result vector.

**Usage** SCILIB:  
 INTEGER\*8 m,n  
 REAL\*8 a(m,n),x(n),y(m)  
 CALL MXV (a,m,x,n,y)

**Input**

**a** Array containing the  $m$  by  $n$  matrix  $A$ .

**m** Number of rows in matrix  $A$  and length of vector  $y$ ,  $m > 0$ .

**x** Array containing the  $n$ -vector  $x$ .

**n** Number of columns in matrix  $A$  and length of vector  $x$ ,  $n > 0$ . **a** or **x**.

**Output** **y** The result  $y$  vector.

**Notes** Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m \leq 0$ , and  
 $n \leq 0$ .

**FORTRAN Equivalent** Except for the argument error checking, the following FORTRAN subroutine is equivalent to MXVA, and illustrates the meanings of the four increment arguments.

```

SUBROUTINE MXV (A,M,X,N,Y)
 INTEGER*8 M,N
 REAL*8 A(M,N),X(N),Y(M)
 DO 110 I = 1, M
 Y(I) = 0.0
110 CONTINUE
 DO 130 J = 1, N
 DO 120 I = 1, M
 Y(I) = Y(I) + A(I,J) * X(J)
120 CONTINUE
130 CONTINUE
 RETURN
 END

```

**Example** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  whose dimensions are 9 by 6,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 6, and  $y$  is a real vector 9 elements long stored in an array  $Y$  of dimension 9.

```
INTEGER*8 M,K,N
REAL*8 A(9,6),X(6),Y(9)
M = 9
N = 6
CALL MXV (A,M,X,N,Y)
```

## Generalized Matrix-Vector Multiply

MXVA

**Purpose** This subprogram computes the matrix-vector product  $y = Ax$ , where  $A$  is an  $m$  by  $n$  matrix,  $x$  is a  $n$ -vector, and  $y$  is an  $m$ -vector. The rows and columns of  $A$  may be stored with unit or non-unit stride, effectively allowing  $A$  or its transpose to be stored in a two-dimensional array via the storage-association rules of FORTRAN.

SCILIB subprogram SGEMV allows matrix and transposed matrix storage without resorting to storage association, also admitting the ability to add or subtract the product from the original contents of the result vector.

**Usage****SCILIB:**

```
INTEGER*8 ia,ja,incx,incy,m,n
REAL*8 a(lena),x(lenx),y(leny)
CALL MXVA (a, ia, ja, x, incx, y, incy, m, n)
```

**Input**

**a** Array containing the  $m$  by  $n$  matrix  $A$ . Typically, **a** will be a two-dimensional array with the rows and columns of  $A$  comprising one-dimensional array sections of **a**. Refer to "Notes" for suggested usages. Treating **a** as a one-dimensional array results in

$$\text{lena} = (m-1) \times |\text{ia}| + (n-1) \times |\text{ja}| + 1.$$

$A_{ij}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , is stored in

$$\mathbf{a}((i-1) \times \text{ia} + (j-1) \times \text{ja} + 1).$$

Note that negative **ia** or **ja** will result in subscript values that lie outside the **a** array as declared above. This need not be an error; refer to "Example 3" for details.

**ia** Storage increment between successive elements in the same column of matrix  $A$  in array **a**. Refer to "Notes" for suggested values.

**ja** Storage increment between successive elements in the same row of matrix  $A$  in array **a**. Refer to "Notes" for suggested values.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Storage increment between successive elements of vector  $x$  in array **x**.  $x_i$  is stored in  $\mathbf{x}((i-1) \times \text{incx} + 1)$ . Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $\mathbf{x}(i)$ .

Note that negative **incx** will result in subscript values that lie outside the **x** array as declared above. This need not be an error; refer to "Example 3" for details.

**incy** Storage increment between successive elements of vector  $y$  in array **y**.  $y_i$  is stored in  $\mathbf{y}((i-1) \times \text{incy} + 1)$ . Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array **y**, i.e., if  $y_i$  is stored in  $\mathbf{y}(i)$ .

Note that a negative **incy** will result in subscript values that lie outside the **y** array as declared above. This need not be an error; refer to "Example 3" for details.

**m** Number of rows in matrix  $A$  and vector  $y$ ,  $\mathbf{m} > 0$ .

**n**      Number of columns in matrix  $A$  and length of vector  $x$ ,  $n > 0$ .  $a$  or  $x$ .

**Output**    **y**      Array of length  $leny = (m-1) \times |incy| + 1$  containing the resulting  $y$  vector.

**Notes**      Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

Typically,  $a$  will be a two-dimensional array with rows and columns comprising one-dimensional sections of the array, i.e., one subscript will vary within a row or column of the matrix, and the other will be constant.

If  $a$  is a two-dimensional array of dimension  $lda$  by  $mda$ , and  $A$  is stored in untransposed form in  $a$ , then  $ia = 1$  and  $ja = lda$ , while if  $A$  is stored in transposed form ( $A^T$  is stored), then  $ia = lda$  and  $ja = 1$ .

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

m ≤ 0,
n ≤ 0,
ia = 0,
ja = 0,
incx = 0, and
incy = 0.

```

**FORTRAN Equivalent**    Except for the argument error checking, the following FORTRAN subroutine is equivalent to MXVA, and illustrates the meanings of the four increment arguments.

```

SUBROUTINE MXVA (A, IA, JA, X, INCX, Y, INCY, M, N)
INTEGER*8 IA, JA, INCX, INCY, M, N
REAL*8 A(*), X(*), Y(*)
DO 110 I = 1, M
 Y((I-1)*INCY+1) = 0.0 ! Y(I) = 0.0
110 CONTINUE
DO 130 J = 1, N
 DO 120 I = 1, M
 Y((I-1)*INCY+1) = ! Y(I) =
1 Y((I-1)*INCY+1) + ! Y(I) +
2 A((I-1)*IA+(J-1)*JA+1) * ! A(I, J) *
3 X((J-1)*INCX+1) ! X(J)
120 CONTINUE
130 CONTINUE
RETURN
END

```

**Example 1** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  of dimension 10 by 11,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 12, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , of dimension 13.

```

INTEGER*8 IA, JA, INCX, INCY, M, N
REAL*8 A(10,11), B(12), Y(13)
IA = 1
JA = 10
INCX = 1
INCY = 1
M = 9
N = 6
CALL MXVA (A, IA, JA, X, INCX, Y, INCY, M, N)

```

**Example 2** Form the REAL\*8 matrix-vector product  $y = A^T x$ , where  $A$  is a 6 by 9 real matrix stored in an array  $A$  of dimension 10 by 11,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 12, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , of dimension 13.

```

INTEGER*8 IA, JA, INCX, INCY, M, N
REAL*8 A(10,11), X(12), Y(13)
IA = 10
JA = 1
INCX = 1
INCY = 1
M = 9
N = 6
CALL MXVA (A, IA, JA, X, INCX, Y, INCY, M, N)

```

**Example 3** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real matrix stored "upside-down and backwards", i.e., with the row and column subscripts decreasing to the right and bottom, in an array  $A$  of dimension 10 by 11,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 12, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , of dimension 13.

```

INTEGER*8 IA, JA, INCX, INCY, M, N
REAL*8 A(10,11), X(12), Y(13)
IA = -1
JA = -10
INCX = 1
INCY = 1
M = 9
N = 6
CALL MXVA (A(M,K), IA, JA, X, INCX, Y, INCY, M, N)

```

**Purpose** These subprograms compute the matrix-vector products  $Ax$ ,  $A^T x$ , and  $A^*x$ , where  $A$  is an  $m$  by  $n$  band matrix stored in a two-dimensional array,  $A^T$  is the transpose of  $A$ , and  $A^*$  is the conjugate transpose of  $A$ .

A band matrix is a matrix whose nonzero elements all are near the principal diagonal. Specifically,  $a_{ij} = 0$  if  $i-j > ml$  or  $j-i > mu$  for some integers  $ml$  and  $mu$ . The smallest such  $ml$  and  $mu$  for a given matrix are called the lower and upper bandwidths, respectively, and  $m = ml + mu + 1$  is the total bandwidth. The subprograms for general band matrices use less storage than the subprograms for general full matrices if  $ml + mu < n$ .

The product may be stored in the result array, or optionally added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix-vector product and the result vector. Specifically, these subprograms compute matrix-vector products of the forms

$$y - \alpha Ax + \beta y, \quad y - \alpha A^T x + \beta y, \quad \text{and} \quad y - \alpha A^*x + \beta y.$$

**Storage** The following example illustrates the storage of general band matrices. If the matrix is

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 14 | 0  | 0  | 0  | 0  |
| 21 | 22 | 23 | 24 | 25 | 0  | 0  | 0  |
| 31 | 32 | 33 | 34 | 35 | 36 | 0  | 0  |
| 0  | 42 | 43 | 44 | 45 | 46 | 47 | 0  |
| 0  | 0  | 53 | 54 | 55 | 56 | 57 | 58 |
| 0  | 0  | 0  | 64 | 65 | 66 | 67 | 68 |
| 0  | 0  | 0  | 0  | 75 | 76 | 77 | 78 |
| 0  | 0  | 0  | 0  | 0  | 86 | 87 | 88 |
| 0  | 0  | 0  | 0  | 0  | 0  | 97 | 98 |

then  $m = 9$ ,  $n = 8$ , and its lower and upper bandwidths are  $ml = 2$  and  $mu = 3$ . Thus,  $A$  is given in an array  $\mathbf{a}$  with at least  $ml + mu + 1 = 6$  rows and  $n = 8$  columns as follows:

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| *  | *  | *  | 14 | 25 | 36 | 47 | 58 |
| *  | *  | 13 | 24 | 35 | 46 | 57 | 68 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 | 78 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 |
| 21 | 32 | 43 | 54 | 65 | 76 | 87 | 98 |
| 31 | 42 | 53 | 64 | 75 | 86 | 97 | *  |

where asterisks in the  $mu$  by  $mu$  triangle at the upper-left corner and in the  $n - m + ml$  by  $n - m + ml$  triangle at the lower-right corner represent elements of  $\mathbf{a}$  that are not referenced. Thus, if  $a_{ij}$  is an element within the band of  $A$ , then it is stored in  $\mathbf{a}(i-j+mu+1, j)$ . Therefore, the columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in the rows of  $\mathbf{a}$ , such that the principal diagonal is stored in row  $mu + 1$  of  $\mathbf{a}$ .

Continued

Usage      SCILIB:

```

CHARACTER*1 trans
INTEGER*8 m,n,ml,mu,lda,incx,incy
REAL*8 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL SGBMV (trans,m,n,ml,mu,alpha,a,lda,x,incx,beta,y,incy)

CHARACTER*1 trans
INTEGER*8 m,n,ml,mu,lda,incx,incy
COMPLEX*16 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL CGBMV (trans,m,n,ml,mu,alpha,a,lda,x,incx,beta,y,incy)

```

Input      **trans**      Transposition option for  $A$ :

             'N' or 'n'      Compute  $y - \alpha Ax + \beta y$   
             'T' or 't'      Compute  $y - \alpha A^T x + \beta y$   
             'C' or 'c'      Compute  $y - \alpha A^* x + \beta y$

             where  $A^T$  is the transpose of  $A$  and  $A^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**m**          Number of rows in matrix  $A$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**n**          Number of columns in matrix  $A$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**ml**        The lower bandwidth of  $A$ , i.e., the number of nonzero diagonals below the principal diagonal in the band,  $0 \leq ml < n$ .

**mu**        The upper bandwidth of  $A$ , i.e., the number of nonzero diagonals above the principal diagonal in the band,  $0 \leq mu < n$ .

**alpha**      The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $y - \beta y$  without referencing  $a$  or  $x$ .

**a**          Array containing the  $m$  by  $n$  band matrix  $A$  in the compressed form described above. If  $a_{ij}$  is in the band, it is stored in  $a(i-j+mu+1,j)$ . The columns of  $A$  are stored in the columns of  $a$ , and the diagonals of  $A$  are stored in rows 1 through  $ml+mu+1$ .

**lda**        The leading dimension of array  $a$  as declared in the calling program unit, with  $lda \geq ml+mu+1$ .

**x**          Array containing the vector  $x$ . The number of elements of  $x$  and the value of **lenx**, the dimension of the array  $x$ , depend on **trans**:

|            |                      |                                  |
|------------|----------------------|----------------------------------|
| 'N' or 'n' | $x$ has $n$ elements | $lenx = (n-1) \times  incx  + 1$ |
| otherwise  | $x$ has $m$ elements | $lenx = (m-1) \times  incx  + 1$ |

**incx**      Increment for the array  $x$ , **incx**  $\neq 0$ :

|                 |                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>incx</b> > 0 | $x$ is stored forward in array $x$ , i.e., $x_i$ is stored in $x((i-1) \times incx + 1)$ .                                                                                                |
| <b>incx</b> < 0 | $x$ is stored backward in array $x$ , i.e., if <b>trans</b> = 'N' or 'n', then $x_i$ is stored in $x((i-n) \times incx + 1)$ ; otherwise, $x_i$ is stored in $x((i-m) \times incx + 1)$ . |

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**beta** The scalar  $\beta$ .

**y** Array containing the vector  $y$ . The number of elements of  $y$  and the value of **leny**, the dimension of the array  $y$ , depend on **trans**:

|            |                      |                                                |
|------------|----------------------|------------------------------------------------|
| 'N' or 'n' | $y$ has $m$ elements | <b>leny</b> = $(m-1) \times  \text{incy}  + 1$ |
| otherwise  | $y$ has $n$ elements | <b>leny</b> = $(n-1) \times  \text{incy}  + 1$ |

Not used as input if **beta** = 0.

**incy** Increment for the array  $y$ , **incy**  $\neq$  0:

|                 |                                                                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>incy</b> > 0 | $y$ is stored forward in array $y$ , i.e.,<br>$y_i$ is stored in $y((i-1) \times \text{incy} + 1)$ .                                                                                                          |
| <b>incy</b> < 0 | $y$ is stored backward in array $y$ , i.e.,<br>if <b>trans</b> = 'N' or 'n', then $y_i$ is stored in $y((i-m) \times \text{incy} + 1)$ ;<br>otherwise, $y_i$ is stored in $y((i-n) \times \text{incy} + 1)$ . |

Use **incy** = 1 if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **y** The updated  $y$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

trans \neq 'N' or 'n' or 'T' or 't' or 'C' or 'c',
m < 0,
n < 0,
ml < 0,
mu < 0,
lda < ml+mu+1,
incx = 0, and
incy = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

## Continued

**Example 1** Form the REAL\*4 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real band matrix whose lower bandwidth is 2 and whose upper bandwidth is 3.  $A$  is stored in an array A whose dimensions are 10 by 10,  $x$  is a real vector 6 elements long stored in an array X of dimension 10, and  $y$  is a real vector 9 elements long stored in an array Y, also of dimension 10.

```

CHARACTER*1 TRANS
INTEGER*4 M,N,ML,MU,LDA,INCX,INCY
REAL*4 ALPHA,BETA,A(10,10),X(10),Y(10)
TRANS = 'N'
M = 9
N = 6
ML = 2
MU = 3
ALPHA = 1.0
BETA = 0.0
LDA = 10
INCX = 1
INCY = 1
CALL SGBMV (TRANS,M,N,ML,MU,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

```

**Example 2** Form the REAL\*8 matrix-vector product  $y = \frac{1}{2}y - \rho A^T x$ , where  $\rho$  is a real scalar,  $A$  is a 6 by 9 real band matrix whose lower bandwidth is 1 and whose upper bandwidth is 2.  $A$  is stored in an array A whose dimensions are 10 by 10,  $x$  is a real vector 6 elements long stored in an array X of dimension 10, and  $y$  is a real vector 9 elements long stored in an array Y, also of dimension 10.

```

INTEGER*8 M,N,ML,MU,LDA
REAL*8 RHO,A(10,10),X(10),Y(10)
M = 9
N = 6
ML = 1
MU = 2
LDA = 10
CALL SGBMV ('TRANSPOSE',M,N,ML,MU,-RHO,A,LDA,X,1,0.5,Y,1)

```

**Purpose** These subprograms compute the matrix-matrix product  $AB$ , where  $A$  is an  $m$  by  $k$  matrix, and  $B$  is a  $k$  by  $n$  matrix. Optionally,  $A$  may be replaced by  $A^T$  or  $A^*$ , where  $A$  is a  $k$  by  $m$  matrix, and  $B$  may be replaced by  $B^T$  or  $B^*$ , where  $B$  is an  $n$  by  $k$  matrix. Here,  $A^T$  and  $B^T$  are the transposes and  $A^*$  and  $B^*$  are the conjugate-transposes of  $A$  and  $B$ , respectively. The product may be stored in the result matrix (which is always of size  $m$  by  $n$ ) or optionally may be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix product and the result matrix. Specifically, these subprograms compute matrix products of the forms

$$\begin{array}{lll} C - \alpha AB + \beta C, & C - \alpha A^T B + \beta C, & C - \alpha A^* B + \beta C, \\ C - \alpha AB^T + \beta C, & C - \alpha A^T B^T + \beta C, & C - \alpha A^* B^T + \beta C, \\ C - \alpha AB^* + \beta C, & C - \alpha A^T B^* + \beta C, & C - \alpha A^* B^* + \beta C. \end{array}$$

**Usage**

**SCILIB:**

```
CHARACTER*1 transa,transb
INTEGER*8 m,n,k,lda,ldb,ldc
REAL*8 alpha,beta,a(lda,*),b(ldb,*),c(ldc,n)
CALL SGEMM (transa,transb,m,n,k,alpha,a,lda,b,ldb,beta,c,ldc)
```

```
CHARACTER*1 transa,transb
INTEGER*8 m,n,k,lda,ldb,ldc
COMPLEX*16 alpha,beta,a(lda,*),b(ldb,*),c(ldc,n)
CALL CGEMM (transa,transb,m,n,k,alpha,a,lda,b,ldb,beta,c,ldc)
```

**Input**

**transa** Transposition option for  $A$ :

'N' or 'n' Use  $m$  by  $k$  matrix  $A$   
 'T' or 't' Use  $A^T$  where  $A$  is a  $k$  by  $m$  matrix  
 'C' or 'c' Use  $A^*$  where  $A$  is a  $k$  by  $m$  matrix

where  $A^T$  is the transpose of  $A$  and  $A^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**transb** Transposition option for  $B$ :

'N' or 'n' Use  $k$  by  $n$  matrix  $B$   
 'T' or 't' Use  $B^T$  where  $B$  is an  $n$  by  $k$  matrix  
 'C' or 'c' Use  $B^*$  where  $B$  is an  $n$  by  $k$  matrix

where  $B^T$  is the transpose of  $B$  and  $B^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**m** Number of rows in matrix  $C$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference **a**, **b**, or **c**.

**n** Number of columns in matrix  $C$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **a**, **b**, or **c**.

**k** The *middle* dimension of the matrix multiply,  $k \geq 0$ . If  $k = 0$ , the subprograms compute  $C - \beta C$  without referencing **a** or **b**.

**alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $C - \beta C$  without referencing **a** or **b**.

## Continued

|               |             |                                                                                                                                                                    |
|---------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | <b>a</b>    | Array containing the matrix $A$ , whose size is indicated by <b>transa</b> :<br><br>'N' or 'n' $A$ is an $m$ by $k$ matrix<br>otherwise $A$ is a $k$ by $m$ matrix |
|               | <b>lda</b>  | The leading dimension of array <b>a</b> as declared in the calling program unit, with $\text{lda} \geq \max(\text{the number of rows of } A, 1)$ .                 |
|               | <b>b</b>    | Array containing the matrix $B$ , whose size is indicated by <b>transb</b> :<br><br>'N' or 'n' $B$ is a $k$ by $n$ matrix<br>otherwise $B$ is an $n$ by $k$ matrix |
|               | <b>ldb</b>  | The leading dimension of array <b>b</b> as declared in the calling program unit, with $\text{ldb} \geq \max(\text{the number of rows of } B, 1)$ .                 |
|               | <b>beta</b> | The scalar $\beta$ .                                                                                                                                               |
|               | <b>c</b>    | Array containing the $m$ by $n$ matrix $C$ . Not used as input if $\text{beta} = 0$ .                                                                              |
|               | <b>ldc</b>  | The leading dimension of array <b>c</b> as declared in the calling program unit, with $\text{ldc} \geq \max(\mathbf{m}, 1)$ .                                      |
| <b>Output</b> | <b>c</b>    | The updated $C$ matrix replaces the input.                                                                                                                         |

**Notes**            These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**transa**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**transb**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**m**  $< 0$ ,  
**n**  $< 0$ ,  
**k**  $< 0$ ,  
**lda** too small,  
**ldb** too small, and  
**ldc**  $< \max(\mathbf{m}, 1)$ .

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved, for example, by coding the **transa** and **transb** arguments as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 9 by 6 real matrix stored in an array A whose dimensions are 10 by 10,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10, and  $C$  is a 9 by 8 real matrix stored in an array C, also of dimension 10 by 10.

```

CHARACTER*1 TRANSA, TRANSB
INTEGER*8 M, N, K, LDA, LDB, LDC
REAL*8 ALPHA, BETA, A(10,10), B(10,10), C(10,10)
TRANSA = 'N'
TRANSB = 'N'
M = 9
N = 8
K = 6
ALPHA = 1.0
BETA = 0.0
LDA = 10
LDB = 10
LDC = 10
CALL SGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

```

**Example 2** Form the REAL\*8 matrix product  $C = \frac{1}{2}C + \rho A^T B$ , where  $\rho$  is a real scalar,  $A$  is a 6 by 9 real matrix stored in an array A whose dimensions are 10 by 10,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10, and  $C$  is a 9 by 8 real matrix stored in an array C, also of dimension 10 by 10.

```

INTEGER*8 M, N, K, LDA, LDB, LDC
REAL*8 RHO, A(10,10), B(10,10), C(10,10)
M = 9
N = 8
K = 6
LDA = 10
LDB = 10
LDC = 10
CALL SGEMM ('TRAN', 'NONTRAN', M, N, K, RHO, A, LDA, B, LDB, 0.5, C, LDC)

```

## Strassen Matrix-Matrix Multiply

## SGEMMS/CGEMMS

**Purpose** These subprograms use Strassen's method to compute the matrix-matrix product  $AB$ , where  $A$  is an  $m$  by  $k$  matrix, and  $B$  is a  $k$  by  $n$  matrix. Strassen's method is an algorithm for matrix multiplication which, under certain circumstances, uses fewer than  $mnk$  multiplications and additions. These subprograms are functionally equivalent to the SCILIB Level 3 BLAS subprograms SGEMM and CGEMM, and differ in usage only by the extra character in the subprogram name and the additional argument, **work**. By using Strassen's method, these subprograms may be considerably faster than their SCILIB counterparts. Refer to "Notes" for details.

In addition to computing the matrix-matrix product  $AB$ ,  $A$  may be replaced by  $A^T$  or  $A^*$ , where  $A$  is a  $k$  by  $m$  matrix, and  $B$  may be replaced by  $B^T$  or  $B^*$ , where  $B$  is an  $n$  by  $k$  matrix. Here,  $A^T$  and  $B^T$  are the transposes and  $A^*$  and  $B^*$  are the conjugate-transposes of  $A$  and  $B$ , respectively. The product may be stored in the result matrix (which is always of size  $m$  by  $n$ ) or optionally may be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix product and the result matrix. Specifically, these subprograms compute matrix products of the forms

$$\begin{array}{lll} C - \alpha AB + \beta C, & C - \alpha A^T B + \beta C, & C - \alpha A^* B + \beta C, \\ C - \alpha AB^T + \beta C, & C - \alpha A^T B^T + \beta C, & C - \alpha A^* B^T + \beta C, \\ C - \alpha AB^* + \beta C, & C - \alpha A^T B^* + \beta C, & C - \alpha A^* B^* + \beta C. \end{array}$$

**Usage** SCILIB:

```

CHARACTER*1 transa, transb
INTEGER*8 m, n, k, lda, ldb, ldc
REAL*8 alpha, beta, a(lda,*), b(ldb,*), c(ldc,n), work(lwork)
CALL SGEMMS (transa, transb, m, n, k, alpha, a, lda, b, ldb,
 beta, c, ldc, work)

CHARACTER*1 transa, transb
INTEGER*8 m, n, k, lda, ldb, ldc
COMPLEX*16 alpha, beta, a(lda,*), b(ldb,*), c(ldc,n), work(lwork)
CALL CGEMMS (transa, transb, m, n, k, alpha, a, lda, b, ldb,
 beta, c, ldc, work)

```

**Input** **transa** Transposition option for  $A$ :

'N' or 'n' Use  $m$  by  $k$  matrix  $A$   
 'T' or 't' Use  $A^T$  where  $A$  is a  $k$  by  $m$  matrix  
 'C' or 'c' Use  $A^*$  where  $A$  is a  $k$  by  $m$  matrix

where  $A^T$  is the transpose of  $A$  and  $A^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**transb** Transposition option for  $B$ :

'N' or 'n' Use  $k$  by  $n$  matrix  $B$   
 'T' or 't' Use  $B^T$  where  $B$  is an  $n$  by  $k$  matrix  
 'C' or 'c' Use  $B^*$  where  $B$  is an  $n$  by  $k$  matrix

where  $B^T$  is the transpose of  $B$  and  $B^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

|                        |                                                                                                                                                                    |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>m</b>               | Number of rows in matrix $C$ , $m \geq 0$ . If $m = 0$ , the subprograms do not reference $a$ , $b$ , or $c$ .                                                     |
| <b>n</b>               | Number of columns in matrix $C$ , $n \geq 0$ . If $n = 0$ , the subprograms do not reference $a$ , $b$ , or $c$ .                                                  |
| <b>k</b>               | The <i>middle</i> dimension of the matrix multiply, $k \geq 0$ . If $k = 0$ , the subprograms compute $C - \beta C$ without referencing $a$ or $b$ .               |
| <b>alpha</b>           | The scalar $\alpha$ . If <b>alpha</b> = 0, the subprograms compute $C - \beta C$ without referencing $a$ or $b$ .                                                  |
| <b>a</b>               | Array containing the matrix $A$ , whose size is indicated by <b>transa</b> :<br><br>'N' or 'n' $A$ is an $m$ by $k$ matrix<br>otherwise $A$ is a $k$ by $m$ matrix |
| <b>lda</b>             | The leading dimension of array <b>a</b> as declared in the calling program unit, with <b>lda</b> $\geq$ max(the number of rows of $A$ ,1).                         |
| <b>b</b>               | Array containing the matrix $B$ , whose size is indicated by <b>transb</b> :<br><br>'N' or 'n' $B$ is a $k$ by $n$ matrix<br>otherwise $B$ is an $n$ by $k$ matrix |
| <b>ldb</b>             | The leading dimension of array <b>b</b> as declared in the calling program unit, with <b>ldb</b> $\geq$ max(the number of rows of $B$ ,1).                         |
| <b>beta</b>            | The scalar $\beta$ .                                                                                                                                               |
| <b>c</b>               | Array containing the $m$ by $n$ matrix $C$ . Not used as input if <b>beta</b> = 0.                                                                                 |
| <b>ldc</b>             | The leading dimension of array <b>c</b> as declared in the calling program unit, with <b>ldc</b> $\geq$ max( <b>m</b> ,1).                                         |
| <b>Working Storage</b> | <b>work</b> An array of size <b>lwork</b> = $2.34 \times \max(m,k) \times \max(n,k)$ , used for work space.                                                        |
| <b>Output</b>          | <b>c</b> The updated $C$ matrix replaces the input.                                                                                                                |

**Notes**        Except for the extra character in the subprogram name and the additional working storage argument, these subprograms conform to specifications of the Level 3 BLAS subprograms SGEMM and CGEMM.

Because of their use of Strassen's method, CGEMMS and SGEMMS are asymptotically faster than standard matrix multiply methods such as those employed in the SCILIB routines CGEMM and SGEMM. In practice these particular implementations are faster than their standard counterparts in the following cases:

- If there is a significant bank conflict problem stemming from the combination of TRANSA, TRANSB, LDA, and LDB. (A pleasant side-effect of the data motion supporting Strassen's method is the alleviation of that problem.)
- If  $\min(m,n,k) > 200$  for CGEMMS and  $\min(m,n,k) > 512$  for SGEMMS. The speedup in the complex case is much more pronounced. That is due in large part to the complex bilinear reduction technique (implemented underneath Strassen's method) that allows two complex matrices to be multiplied using only 3/4 of the multiplications required by the traditional method. Also, the relative cost of data

motion is lower in the complex case. In this first release, the gains in the real case are marginal.

In the operator norm, Strassen's method is slightly less stable than traditional matrix multiplication, and the computation of individual elements is unstable. The emerging consensus seems to be that Strassen's method is sufficiently stable for most applications.

For a good overview and bibliography of this subject, see (Higham).

If an error in the arguments is detected, the subprograms call the BLAS error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter) may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

transa ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
transb ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
m < 0,
n < 0,
k < 0,
lda too small,
ldb too small, and
ldc < max(m,1).

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved, for example, by coding the **transa** and **transb** arguments as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 900 by 600 real matrix stored in an array A whose dimensions are 1000 by 1000,  $B$  is a 600 by 800 real matrix stored in an array B of dimension 1000 by 1000, and  $C$  is a 900 by 800 real matrix stored in an array C, also of dimension 1000 by 1000. WORK is declared large enough to handle all matrices that will fit in the arrays.

```

CHARACTER*1 TRANSA, TRANSB
INTEGER*8 M, N, K, LDA, LDB, LDC
REAL*8 ALPHA, BETA, A(1000,1000), B(1000,1000),
1 C(1000,1000), WORK(2340000)
TRANSA = 'N'
TRANSB = 'N'
M = 900
N = 800
K = 600
ALPHA = 1.0
BETA = 0.0
LDA = 1000
LDB = 1000
LDC = 1000
CALL SGEMMS (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA,
1 C, LDC, WORK)

```

**Example 2** Form the COMPLEX\*16 matrix product  $C = \frac{1}{2}C + \rho A*B$ , where  $\rho$  is a complex scalar,  $A$  is a 600 by 900 complex matrix stored in an array A whose dimensions are 1000 by 1000,  $B$  is a 600 by 800 complex matrix stored in an array B of dimension 1000 by 1000, and  $C$  is a 900 by 800 complex matrix stored in an array C, also of dimension 1000 by 1000. WORK is declared large enough to handle all matrices that will fit in the arrays.

```
INTEGER*8 M,N,K,LDA,LDB,LDC
COMPLEX*16 RHO,A(1000,1000),B(1000,1000),C(1000,1000),
1 WORK(2340000)
M = 900
N = 800
K = 600
LDA = 1000
LDB = 1000
LDC = 1000
CALL CGEMMS ('CONJ', 'NORMAL', M,N,K,RHO,A,LDA,B,LDB,(0.5,0.0),
1 C,LDC,WORK)
```

## Matrix-Vector Multiply

## SGEMV/CGEMV

**Purpose** These subprograms compute the matrix-vector products  $Ax$ ,  $A^T x$ , and  $A^*x$ , where  $A$  is an  $m$  by  $n$  matrix,  $A^T$  is the transpose of  $A$ , and  $A^*$  is the conjugate transpose of  $A$ . The product may be stored in the result array, or optionally added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix-vector product and the result vector. Specifically, these subprograms compute matrix-vector products of the forms

$$y - \alpha Ax + \beta y, \quad y - \alpha A^T x + \beta y, \quad \text{and} \quad y - \alpha A^*x + \beta y.$$

**Usage****SCILIB:**

```
CHARACTER*1 trans
INTEGER*8 m,n,lda,incx,incy
REAL*8 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL SGEMV (trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
```

```
CHARACTER*1 trans
INTEGER*8 m,n,lda,incx,incy
COMPLEX*16 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL CGEMV (trans, m, n, alpha, a, lda, x, incx, beta, y, incy)
```

**Input**

**trans** Transposition option for  $A$ :

'N' or 'n' Compute  $y - \alpha Ax + \beta y$   
 'T' or 't' Compute  $y - \alpha A^T x + \beta y$   
 'C' or 'c' Compute  $y - \alpha A^*x + \beta y$

where  $A^T$  is the transpose of  $A$  and  $A^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**m** Number of rows in matrix  $A$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**n** Number of columns in matrix  $A$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**alpha** The scalar  $\alpha$ . If  $\alpha = 0$ , the subprograms compute  $y - \beta y$  without referencing  $A$  or  $x$ .

**a** Array containing the  $m$  by  $n$  matrix  $A$ .

**lda** The leading dimension of array  $a$  as declared in the calling program unit, with  $lda \geq \max(m, 1)$ .

**x** Array containing the vector  $x$ . The number of elements of  $x$  and the value of **lenx**, the dimension of the array  $x$ , depend on **trans**:

|            |                      |                                                |
|------------|----------------------|------------------------------------------------|
| 'N' or 'n' | $x$ has $m$ elements | $\text{lenx} = (m-1) \times  \text{incx}  + 1$ |
| otherwise  | $x$ has $n$ elements | $\text{lenx} = (n-1) \times  \text{incx}  + 1$ |

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 if **trans** = 'N' or 'n', then  $x_i$  is stored in  $x((i-m) \times \text{incx} + 1)$ ;  
 otherwise,  $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**beta** The scalar  $\beta$ .

**y** Array containing the vector  $y$ . The number of elements of  $y$  and the value of **leny**, the dimension of the array  $y$ , depend on **trans**:

|            |                      |                                                |
|------------|----------------------|------------------------------------------------|
| 'N' or 'n' | $y$ has $n$ elements | <b>leny</b> = $(n-1) \times  \text{incy}  + 1$ |
| otherwise  | $y$ has $m$ elements | <b>leny</b> = $(m-1) \times  \text{incy}  + 1$ |

Not used as input if **beta** = 0.

**incy** Increment for the array  $y$ , **incy**  $\neq$  0:

|                 |                                                                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>incy</b> > 0 | $y$ is stored forward in array $y$ , i.e.,<br>$y_i$ is stored in $y((i-1) \times \text{incy} + 1)$ .                                                                                                          |
| <b>incy</b> < 0 | $y$ is stored backward in array $y$ , i.e.,<br>if <b>trans</b> = 'N' or 'n', then $y_i$ is stored in $y((i-n) \times \text{incy} + 1)$ ;<br>otherwise, $y_i$ is stored in $y((i-m) \times \text{incy} + 1)$ . |

Use **incy** = 1 if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **y** The updated  $y$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**m** < 0,  
**n** < 0,  
**lda** < max(**m**,1),  
**incx** = 0, and  
**incy** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 10, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , also of dimension 10.

```

CHARACTER*1 TRANS
INTEGER*8 M,N,LDA,INCX,INCY
REAL*8 ALPHA,BETA,A(10,10),X(10),Y(10)
TRANS = 'N'
M = 9
N = 6
ALPHA = 1.0
BETA = 0.0
LDA = 10
INCX = 1
INCY = 1
CALL SGEMV (TRANS,M,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

```

**Example 2** Form the REAL\*8 matrix-vector product  $y = \frac{1}{2}y - \rho A^T x$ , where  $\rho$  is a real scalar,  $A$  is a 6 by 9 real matrix stored in an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 10, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , also of dimension 10.

```

INTEGER*8 M,N,LDA
REAL*8 RHO,A(10,10),X(10),Y(10)
M = 9
N = 6
LDA = 10
CALL SGEMV ('TRANSPOSE',M,N,-RHO,A,LDA,X,1,0.5,Y,1)

```

**Purpose** These subprograms compute the rank-1 updates

$$A - \alpha xy^T + A \quad \text{and} \quad A - \alpha xy^* + A,$$

where  $A$  is an  $m$  by  $n$  matrix,  $\alpha$  is a scalar,  $x$  is an  $m$ -vector,  $y$  is an  $n$ -vector, and  $y^T$  and  $y^*$  are the transpose and conjugate transpose of  $y$ , respectively.

**Usage**

**SCILIB:**

```
INTEGER*8 m,n,lda,incx,incy
REAL*8 alpha,a(lda,n),x(lenx),y(leny)
CALL SGER (m, n, alpha, x, incx, y, incy, a, lda)
```

```
INTEGER*8 m,n,lda,incx,incy
COMPLEX*16 alpha,a(lda,n),x(lenx),y(leny)
CALL CGERC (m, n, alpha, x, incx, y, incy, a, lda)
```

```
INTEGER*8 m,n,lda,incx,incy
COMPLEX*16 alpha,a(lda,n),x(lenx),y(leny)
CALL CGERU (m, n, alpha, x, incx, y, incy, a, lda)
```

**Input**

**m** Number of rows in matrix  $A$  and elements of vector  $x$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**n** Number of columns in matrix  $A$  and elements of vector  $y$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**alpha** The scalar  $\alpha$ . If  $\alpha = 0$ , the subprograms do not reference  $A$ ,  $x$ , or  $y$ .

**x** Array of length  $\text{lenx} = (m-1) \times |\text{incx}| + 1$  containing the  $m$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-m) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .  $y$  is used in conjugated form by CGERC, and in unconjugated form by the other subprograms. Refer to "Purpose."

**incy** Increment for the array  $y$ ,  $\text{incy} \neq 0$ :

**incy** > 0  $y$  is stored forward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy** < 0  $y$  is stored backward in array  $y$ , i.e.,  
 $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use  $\text{incy} = 1$  if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

|               |            |                                                                                                                               |
|---------------|------------|-------------------------------------------------------------------------------------------------------------------------------|
|               | <b>a</b>   | Array containing the <b>m</b> by <b>n</b> matrix <i>A</i> .                                                                   |
|               | <b>lda</b> | The leading dimension of array <b>a</b> as declared in the calling program unit, with $\text{lda} \geq \max(\mathbf{m}, 1)$ . |
| <b>Output</b> | <b>a</b>   | The updated <i>A</i> matrix replaces the input.                                                                               |
| <b>Notes</b>  |            | These subprograms conform to specifications of the Level 2 BLAS.                                                              |

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

m < 0,
n < 0,
lda < max(m,1),
incx = 0, and
incy = 0.

```

**Example 1** Apply a REAL\*8 rank-1 update  $xy^T$  to *A*, where *A* is a 6 by 9 real matrix stored in an array A whose dimensions are 10 by 10, *x* is a real vector 6 elements long stored in an array X of dimension 10, and *y* is a real vector 9 elements long stored in an array Y, also of dimension 10.

```

INTEGER*8 M,N,LDA,INCX,INCY
REAL*8 ALPHA,A(10,10),X(10),Y(10)
M = 6
N = 9
ALPHA = 1.0
LDA = 10
INCX = 1
INCY = 1
CALL SGER (M,N,ALPHA,X,INCX,Y,INCY,A,LDA)

```

**Example 2** Apply a COMPLEX\*16 conjugated rank-1 update  $-2xy^*$  to *A*, where *A* is a 6 by 9 complex matrix stored in an array A whose dimensions are 10 by 10, *x* is a complex vector 6 elements long stored in an array X of dimension 10, and *y* is a complex vector 9 elements long stored in an array Y, also of dimension 10.

```

INTEGER*8 M,N,LDA
COMPLEX*16 A(10,10),X(10),Y(10)
M = 6
N = 9
LDA = 10
CALL CGERC (M,N,(-2.0EO,0.0EO),X,1,Y,1,A,LDA)

```

**Purpose** This subprogram computes the matrix-vector product  $Ax$ , and adds the result to another vector  $y$ , where  $A$  is an  $m$  by  $n$  matrix,  $x$  is an  $n$ -vector, and  $y$  is an  $m$ -vector. SCILIB subprogram SGEMV allows more general storage of  $x$  and  $y$  and also admits scaling, subtraction, and transposing  $A$ .

**Usage** SCILIB:  
 INTEGER\*8 m,n,lda  
 REAL\*8 a(lda,n),x(n),y(m)  
 CALL SMXPY (m,y,n,lda,x,a)

**Input** **m** Number of rows in matrix  $A$  and length of vector  $y$ ,  $m \geq 0$ . If  $m = 0$ , the subprogram does not reference  $a$ ,  $x$ , or  $y$ .

**y** Array containing the vector  $y$ .

**n** Number of columns in matrix  $A$  and length of vector  $x$ ,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $x$ , or  $y$ .

**lda** The leading dimension of array  $a$  as declared in the calling program unit.

**x** Array containing the  $n$ -vector  $x$ .

**a** Array containing the  $m$  by  $n$  matrix  $A$ .

**Output** **y** The updated  $y$  vector replaces the input.

**Notes** Cray Research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0$ ,  
 $n < 0$ , and  
 $lda < m$ .

**FORTRAN Equivalent** Except for the argument error checking, the following FORTRAN subroutine is equivalent to SMXPY.

```

SUBROUTINE SMXPY (M,Y,N,LDA,X,A)
 INTEGER*8 M,N,LDA
 REAL*8 A(LDA,N),X(N),Y(M)
 DO 120 J = 1, N
 DO 110 I = 1, M
 Y(I) = A(I,J) * X(J) + Y(I)
 110 CONTINUE
 120 CONTINUE
 RETURN
 END
```

**Example** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 6 elements long stored in an array  $X$  of dimension 10, and  $y$  is a real vector 9 elements long stored in an array  $Y$  of dimension 10.

```
INTEGER*8 M,N,LDA
REAL*8 A(10,10),X(10),Y(10)
M = 9
N = 6
LDA = 10
CALL SMXPY (M,Y,N,LDA,X,A)
```

**Purpose** These subprograms compute the matrix-vector product  $Ax$  where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian band matrix and  $x$  is a real or complex  $n$ -vector. The product may be stored in the result array, or, optionally, be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix-vector product and the result vector. Specifically, these subprograms compute the matrix-vector product of the form

$$y = \alpha Ax + \beta y.$$

The structure of  $A$  is indicated by the name of the subprogram used:

SSBMV  $A$  is a real symmetric band matrix

CHBMV  $A$  is a complex Hermitian band matrix

A symmetric or Hermitian band matrix is a symmetric or Hermitian matrix whose nonzero elements all are on or fairly near the principal diagonal. Specifically,  $a_{ij} \neq 0$  only if  $|i-j| \leq m$  for some integer  $m$ .

**Storage** Because either triangle of  $A$  may be obtained from the other, these subprograms allow you to save memory by supplying the band of only one triangle of the band of  $A$ . You may choose to supply either the upper or the lower triangle of the band of  $A$ . The following examples illustrate the storage of symmetric band matrices.

**Upper triangular storage.** Suppose that  $A$  is a 9 by 9 symmetric band matrix with bandwidth  $m = 3$ . If the upper triangle of  $A$  is

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 14 | 0  | 0  | 0  | 0  | 0  |
|    | 22 | 23 | 24 | 25 | 0  | 0  | 0  | 0  |
|    |    | 33 | 34 | 35 | 36 | 0  | 0  | 0  |
|    |    |    | 44 | 45 | 46 | 47 | 0  | 0  |
|    |    |    |    | 55 | 56 | 57 | 58 | 0  |
|    |    |    |    |    | 66 | 67 | 68 | 69 |
|    |    |    |    |    |    | 77 | 78 | 79 |
|    |    |    |    |    |    |    | 88 | 89 |
|    |    |    |    |    |    |    |    | 99 |

then the upper triangular band part of  $A$  is stored as follows in an array  $\mathbf{a}$  with at least  $m+1 = 4$  rows and 9 columns:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| *  | *  | *  | 14 | 25 | 36 | 47 | 58 | 69 |
| *  | *  | 13 | 24 | 35 | 46 | 57 | 68 | 79 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |

where asterisks represent elements in the  $m$  by  $m$  triangle at the upper-left corner of  $\mathbf{a}$  that are not referenced. Thus, if  $a_{ij}$  is an element within the band of  $A$ , it is stored in  $\mathbf{a}(i-j+m+1, j)$ . In words, the columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in the rows of  $\mathbf{a}$ , with the principal diagonal in row  $m+1$ , the first superdiagonal starting in the second position in row  $m$ , and so on.

**Lower triangular storage.** Suppose that  $A$  is a 9 by 9 symmetric band matrix with bandwidth  $m = 3$ . If the lower triangle of  $A$  is

```

11
21 22
31 32 33
41 42 43 44
 0 52 53 54 55
 0 0 63 64 65 66
 0 0 0 74 75 76 77
 0 0 0 0 85 86 87 88
 0 0 0 0 0 96 97 98 99

```

then the lower triangular band part of  $A$  is stored as follows in an array  $\mathbf{a}$  with at least  $m + 1 = 4$  rows and 9 columns:

```

11 22 33 44 55 66 77 88 99
21 32 43 54 65 76 87 98 *
31 42 53 64 75 86 97 * *
41 52 63 74 85 96 * * *

```

where asterisks represent elements in the  $m$  by  $m$  triangle at the lower-right corner of  $\mathbf{a}$  that are not referenced. Thus, if  $a_{ij}$  is an element within the band of  $A$ , it is stored in  $\mathbf{a}(i-j+1,j)$ . In words, the columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in the rows of  $\mathbf{a}$ , with the principal diagonal in the first row, the first subdiagonal in the second row, and so on.

**Usage****SCILIB:**

```

CHARACTER*1 uplo
INTEGER*8 n,m,lda,incx,incy
REAL*8 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL SSBMV (uplo, n, m, alpha, a, lda, x, incx, beta, y, incy)
CHARACTER*1 uplo
INTEGER*8 n,m,lda,incx,incy
COMPLEX*16 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL CHBMV (uplo, n, m, alpha, a, lda, x, incx, beta, y, incy)

```

**Input**

**uplo** Upper/lower triangular option for  $A$ :

‘L’ or ‘l’ The lower triangle of  $A$  is stored.  
‘U’ or ‘u’ The upper triangle of  $A$  is stored.

**n** Number of rows and columns in matrix  $A$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $\mathbf{a}$  or  $\mathbf{x}$ .

**m** The number of nonzero diagonals above or below the principal diagonal.

**alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $y - \beta y$  without referencing  $\mathbf{a}$  or  $\mathbf{x}$ .

**a** Array containing the  $n$  by  $n$  symmetric band matrix  $A$  in the compressed form described above. The columns of the band of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of the band of  $A$  are stored in the rows of  $\mathbf{a}$ .

- lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq m+1$ .
- x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the input vector  $x$ .
- incx** Increment for the array **x**,  $incx \neq 0$ :
- incx** > 0  $x$  is stored forward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .
- incx** < 0  $x$  is stored backward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .
- Use **incx** = 1 if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.
- beta** The scalar  $\beta$ .
- y** Array of length  $leny = (n-1) \times |incy| + 1$  containing the  $n$ -vector  $y$ . Not used as input if **beta** = 0.
- incy** Increment for the array **y**,  $incy \neq 0$ :
- incy** > 0  $y$  is stored forward in array **y**, i.e.,  
 $y_i$  is stored in  $y((i-1) \times incy + 1)$ .
- incy** < 0  $y$  is stored backward in array **y**, i.e.,  
 $y_i$  is stored in  $y((i-n) \times incy + 1)$ .
- Use **incy** = 1 if the vector  $y$  is stored contiguously in array **y**, i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **y** The updated  $y$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**n** < 0,  
**m** < 0,  
**lda** < **m**+1,  
**incx** = 0, and  
**incy** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $Ax$ , where  $A$  is a 75 by 75 unit-diagonal, lower-triangular real band matrix with bandwidth 15 that is stored in an array T whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

CHARACTER*1 UPLO,TRANS,DIAG
INTEGER*8 N,M,LDA,INCX
REAL*8 T(25,100),X(100)
UPLO = 'L'
N = 75
M = 15
LDA = 25
INCX = 1
CALL SSBMV (UPLO, N, M, 1.0, A, LDA, X, INCX, 0.0)

```

**Example 2** Form the REAL\*8 matrix-vector product  $A^T x$ , where  $A$  is a 75 by 75 nonunit-diagonal, upper-triangular real band matrix with bandwidth 15 that is stored in an array T whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

INTEGER*8 N,M,LDA
REAL*8 T(25,100),X(100),Y(100)
N = 75
M = 15
LDA = 25
CALL SSBMV ('UPPER', N, M, 1.0, A, LDA, X, 1, 1.0, Y, 1)

```

**Purpose** These subprograms compute the matrix-vector product  $Ax$  where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix stored in packed form as described in "Usage," and  $x$  is a real or complex  $n$ -vector. The product may be stored in the result array, or, optionally, be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix-vector product and the result vector. Specifically, these subprograms compute the matrix-vector product of the form

$$y = \alpha Ax + \beta y.$$

The structure of  $A$  is indicated by the name of the subprogram used:

SSPMV or  $A$  is a real symmetric matrix  
 CHPMV or  $A$  is a complex Hermitian matrix

**Storage** Because either triangle of  $A$  may be obtained from the other, these subprograms allow you to reduce memory usage by supplying only one triangle of  $A$ , stored column by column in packed form in a 1-dimensional array. You may choose to supply either the upper or the lower triangle of  $A$ . The following examples illustrate the packed storage of symmetric or Hermitian matrices.

**Upper triangular storage.** If the upper triangle of  $A$  is

|    |    |    |    |
|----|----|----|----|
| 11 | 12 | 13 | 14 |
|    | 22 | 23 | 24 |
|    |    | 33 | 34 |
|    |    |    | 44 |

then  $A$  is packed column by column into an array **ap** as follows:

|       |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|
| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| ap(i) | 11 | 12 | 22 | 13 | 23 | 33 | 14 | 24 | 34 | 44 |

Upper-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (j-1)/2 + i$ ).

**Lower triangular storage.** If the lower triangle of  $A$  is

|    |    |    |    |  |  |  |  |  |  |
|----|----|----|----|--|--|--|--|--|--|
| 11 |    |    |    |  |  |  |  |  |  |
| 21 | 22 |    |    |  |  |  |  |  |  |
| 31 | 32 | 33 |    |  |  |  |  |  |  |
| 41 | 42 | 43 | 44 |  |  |  |  |  |  |

then  $A$  is packed column by column into an array **ap** as follows:

|       |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|
| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| ap(i) | 11 | 21 | 31 | 41 | 22 | 32 | 42 | 33 | 43 | 44 |

Lower-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (2n - j - 1)/2 + i$ ).

Usage      **SCILIB:**

```

CHARACTER*1 uplo
INTEGER*8 n,incx,incy
REAL*8 alpha,beta,ap(lenap),x(lenx),y(leny)
CALL SSPMV (uplo, n, alpha, ap, x, incx, beta, y, incy)

CHARACTER*1 uplo
INTEGER*8 n,incx,incy
COMPLEX*16 alpha,beta,ap(lenap),x(lenx),y(leny)
CALL CHPMV (uplo, n, alpha, ap, x, incx, beta, y, incy)

```

Input      **uplo**   Upper/lower triangular option for  $A$ :

          'L' or 'l'   The lower triangle of  $A$  is stored in the packed array.  
          'U' or 'u'   The upper triangle of  $A$  is stored in the packed array.

**n**       Number of rows and columns in matrix  $A$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **ap**, **x**, or **y**.

**alpha**   The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $y - \beta y$  without referencing **ap** or **x**.

**ap**       Array of length **lenap** =  $n \times (n+1)/2$  containing the upper or lower triangle, as specified by **uplo**, of an  $n$  by  $n$  real symmetric or complex Hermitian matrix  $A$ , stored by columns in the packed form described above.

**x**        Array of length **lenx** =  $(n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx**    Increment for the array **x**, **incx**  $\neq 0$ :

**incx** > 0   **x** is stored forward in array **x**, i.e.,  
                           $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .  
          **incx** < 0   **x** is stored backward in array **x**, i.e.,  
                          if **trans** = 'N' or 'n', then  $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ ;  
                          otherwise,  $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

          Use **incx** = 1 if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**beta**    The scalar  $\beta$ .

**y**        Array of length **leny** =  $(n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ . Not used as input if **beta** = 0.

**incy**    Increment for the array **y**, **incy**  $\neq 0$ :

**incy** > 0   **y** is stored forward in array **y**, i.e.,  
                           $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .  
          **incy** < 0   **y** is stored backward in array **y**, i.e.,  
                          if **trans** = 'N' or 'n', then  $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ ;  
                          otherwise,  $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

          Use **incy** = 1 if the vector  $y$  is stored contiguously in array **y**, i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**     **y**           The updated *y* vector replaces the input.

**Notes**        These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
incx = 0, and
incy = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1**     Form the REAL\*8 matrix-vector product  $y = Ax$ , where *A* is a 9 by 9 real symmetric matrix whose upper triangle is stored in packed form in an array AP of dimension 55, *x* is a real vector 9 elements long stored in an array X of dimension 10, and *y* is a real vector 9 elements long stored in an array Y, also of dimension 10.

```

CHARACTER*1 UPLO
INTEGER*8 N, INCX, INCY
REAL*8 ALPHA, BETA, AP(55), X(10), Y(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
BETA = 0.0
INCX = 1
INCY = 1
CALL SSPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)

```

**Example 2**     Form the COMPLEX\*16 matrix-vector product  $y = \frac{1}{2}y - \rho Ax$ , where  $\rho$  is a complex scalar, *A* is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in packed form in an array AP of dimension 55, *x* is a complex vector 9 elements long stored in an array X of dimension 10, and *y* is a complex vector 9 elements long stored in an array Y, also of dimension 10.

```

INTEGER*8 N
COMPLEX*16 RHO, AP(55), X(10), Y(10)
N = 9
CALL CHPMV ('LOWER', N, -RHO, AP, X, 1, (0.5, 0.0), Y, 1)

```

## Rank-1 Update

SSPR/CHPR

**Purpose** These subprograms compute the real symmetric or complex Hermitian rank-1 update

$$A - \alpha x x^* + A,$$

where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix stored in packed form as described in "Usage,"  $\alpha$  is a real scalar,  $x$  is a real or complex  $n$ -vector, and  $x^*$  is the conjugate transpose of  $x$ . (The conjugate transpose of a real vector is simply the transpose.)

The structure of  $A$  is indicated by the name of the subprogram used:

SSPR  $A$  is a real symmetric matrix

CHPR  $A$  is a complex Hermitian matrix

**Storage** Because either triangle of  $A$  may be obtained from the other, these subprograms allow you to save memory by supplying only one triangle of  $A$ , stored column by column in packed form in a 1-dimensional array. You may choose to supply either the upper or the lower triangle of  $A$ . The following examples illustrate the packed storage of symmetric or Hermitian matrices.

**Upper triangular storage.** If the upper triangle of  $A$  is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then  $A$  is packed column by column into an array **ap** as follows:

|       |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|
| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| ap(i) | 11 | 12 | 22 | 13 | 23 | 33 | 14 | 24 | 34 | 44 |

Upper-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (j-1)/2 + i$ ).

**Lower triangular storage.** If the lower triangle of  $A$  is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then  $A$  is packed column by column into an array **ap** as follows:

|       |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|
| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| ap(i) | 11 | 21 | 31 | 41 | 22 | 32 | 42 | 33 | 43 | 44 |

Lower-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (2n - j - 1)/2 + i$ ).

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Usage  | <p>SCILIB:</p> <pre> CHARACTER*1 uplo INTEGER*8    n,incx REAL*8      alpha,ap(lenap),x(lenx) CALL SSPR (uplo, n, alpha, x, incx, ap)  CHARACTER*1 uplo INTEGER*8    n,incx REAL*8      alpha COMPLEX*16  ap(lenap),x(lenx) CALL CHPR (uplo, n, alpha, x, incx, ap) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Input  | <p><b>uplo</b> Upper/lower triangular option for <math>A</math>:</p> <p>‘L’ or ‘l’ The lower triangle of <math>A</math> is stored in the packed array.<br/> ‘U’ or ‘u’ The upper triangle of <math>A</math> is stored in the packed array.</p> <p><b>n</b> Number of rows and columns in matrix <math>A</math> and elements of vector <math>x</math>, <math>n \geq 0</math>.<br/> If <math>n = 0</math>, the subprograms do not reference <b>ap</b> or <b>x</b>.</p> <p><b>alpha</b> The scalar <math>\alpha</math>. If <b>alpha</b> = 0, the subprograms do not reference <b>ap</b> or <b>x</b>.</p> <p><b>x</b> Array of length <math>\text{lenx} = (n-1) \times  \text{incx}  + 1</math> containing the <math>n</math>-vector <math>x</math>.</p> <p><b>incx</b> Increment for the array <b>x</b>, <b>incx</b> <math>\neq 0</math>:</p> <p><b>incx</b> &gt; 0 <math>x</math> is stored forward in array <b>x</b>, i.e.,<br/> <math>x_i</math> is stored in <math>\text{x}((i-1) \times \text{incx} + 1)</math>.</p> <p><b>incx</b> &lt; 0 <math>x</math> is stored backward in array <b>x</b>, i.e.,<br/> <math>x_i</math> is stored in <math>\text{x}((i-n) \times \text{incx} + 1)</math>.</p> <p>Use <b>incx</b> = 1 if the vector <math>x</math> is stored contiguously in array <b>x</b>, i.e., if <math>x_i</math> is stored in <math>\text{x}(i)</math>. Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.</p> <p><b>ap</b> Array of length <math>\text{lenap} = n \times (n+1) / 2</math> containing the upper or lower triangle, as specified by <b>uplo</b>, of an <math>n</math> by <math>n</math> real symmetric or complex Hermitian matrix <math>A</math>, stored by columns in the packed form described above.</p> |
| Output | <p><b>ap</b> The upper or lower triangle of the updated <math>A</math> matrix, as specified by <b>uplo</b>, replaces the input.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Notes  | <p>These subprograms conform to specifications of the Level 2 BLAS.</p> <p>If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are</p> <p style="text-align: center;"><b>uplo</b> <math>\neq</math> ‘L’ or ‘l’ or ‘U’ or ‘u’,<br/> <b>n</b> &lt; 0, and<br/> <b>incx</b> = 0.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1** Apply a REAL\*8 symmetric rank-1 update  $xx^T$  to  $A$ , where  $A$  is a 9 by 9 real symmetric matrix whose upper triangle is stored in packed form in an array **AP** of dimension 55, and  $x$  is a real vector 9 elements long stored in an array **X** of dimension 10.

```

CHARACTER*1 UPLO
INTEGER*8 N, INCX
REAL*8 ALPHA, AP(55), X(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
INCX = 1
CALL SSPR (UPLO, N, ALPHA, X, INCX, AP)

```

**Example 2** Apply a COMPLEX\*16 Hermitian rank-1 update  $-2xx^*$  to  $A$ , where  $A$  is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in packed form in an array **AP** of dimension 55, and  $x$  is a complex vector 9 elements long stored in an array **X** of dimension 10.

```

INTEGER*8 N
REAL*8 ALPHA
COMPLEX*16 AP(55), X(10)
N = 9
ALPHA = -2.0
CALL CHPR ('LOWER', N, ALPHA, X, 1, AP)

```

**Purpose** These subprograms compute the real symmetric or complex Hermitian rank-2 update

$$A - \alpha xy^* + \bar{\alpha} yx^* + A,$$

where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix stored in packed form as described in "Usage,"  $\alpha$  is a complex scalar,  $\bar{\alpha}$  is the complex conjugate of  $\alpha$ ,  $x$  and  $y$  are real or complex  $n$ -vectors, and  $x^*$  and  $y^*$  are the conjugate transposes of  $x$  and  $y$ , respectively. (The conjugate of a real scalar is just the scalar, and the conjugate transpose of a real vector is simply the transpose.)

The structure of  $A$  is indicated by the name of the subprogram used:

SSPR2  $A$  is a real symmetric matrix

CHPR2  $A$  is a complex Hermitian matrix

**Storage**

Because either triangle of  $A$  may be obtained from the other, these subprograms allow you to reduce memory usage by supplying only one triangle of  $A$ , stored column by column in packed form in a 1-dimensional array. You may choose to supply either the upper or the lower triangle of  $A$ . The following examples illustrate the packed storage of symmetric or Hermitian matrices.

**Upper triangular storage.** If the upper triangle of  $A$  is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ & 22 & 23 & 24 \\ & & 33 & 34 \\ & & & 44 \end{array}$$

then  $A$  is packed column by column into an array **ap** as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| ap(i) | 11 | 12 | 22 | 13 | 23 | 33 | 14 | 24 | 34 | 44 |

Upper-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (j-1)/2 + i$ ).

**Lower triangular storage.** If the lower triangle of  $A$  is

$$\begin{array}{cccc} 11 & & & \\ 21 & 22 & & \\ 31 & 32 & 33 & \\ 41 & 42 & 43 & 44 \end{array}$$

then  $A$  is packed column by column into an array **ap** as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| ap(i) | 11 | 21 | 31 | 41 | 22 | 32 | 42 | 33 | 43 | 44 |

Lower-triangular matrix element  $a_{ij}$  is stored in array element **ap**( $j \times (2n - j - 1)/2 + i$ ).

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Usage  | <p>SCILIB:</p> <pre> CHARACTER*1 uplo INTEGER*8    n,incx,incy REAL*8       alpha,ap(lenap),x(lenx),y(leny) CALL SSPR2 (uplo, n, alpha, x, incx, y, incy, ap)  CHARACTER*1 uplo INTEGER*8    n,incx,incy COMPLEX*16  alpha,ap(lenap),x(lenx),y(leny) CALL CHPR2 (uplo, n, alpha, x, incx, y, incy, ap) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Input  | <p><b>uplo</b> Upper/lower triangular option for <math>A</math>:</p> <p>‘L’ or ‘l’ The lower triangle of <math>A</math> is stored in the packed array.<br/> ‘U’ or ‘u’ The upper triangle of <math>A</math> is stored in the packed array.</p> <p><b>n</b> Number of rows and columns in matrix <math>A</math> and elements of vectors <math>x</math> and <math>y</math>, <math>n \geq 0</math>. If <math>n = 0</math>, the subprograms do not reference <b>ap</b>, <b>x</b>, or <b>y</b>.</p> <p><b>alpha</b> The scalar <math>\alpha</math>. If <b>alpha</b> = 0, the subprograms do not reference <b>ap</b>, <b>x</b>, or <b>y</b>.</p> <p><b>x</b> Array of length <math>\text{lenx} = (n-1) \times  \text{incx}  + 1</math> containing the <math>n</math>-vector <math>x</math>.</p> <p><b>incx</b> Increment for the array <b>x</b>, <b>incx</b> <math>\neq 0</math>:</p> <p><b>incx</b> &gt; 0 <math>x</math> is stored forward in array <b>x</b>, i.e.,<br/> <math>x_i</math> is stored in <math>\mathbf{x}((i-1) \times \text{incx} + 1)</math>.</p> <p><b>incx</b> &lt; 0 <math>x</math> is stored backward in array <b>x</b>, i.e.,<br/> <math>x_i</math> is stored in <math>\mathbf{x}((i-n) \times \text{incx} + 1)</math>.</p> <p>Use <b>incx</b> = 1 if the vector <math>x</math> is stored contiguously in array <b>x</b>, i.e., if <math>x_i</math> is stored in <math>\mathbf{x}(i)</math>. Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.</p> <p><b>y</b> Array of length <math>\text{leny} = (n-1) \times  \text{incy}  + 1</math> containing the <math>n</math>-vector <math>y</math>.</p> <p><b>incy</b> Increment for the array <b>y</b>, <b>incy</b> <math>\neq 0</math>:</p> <p><b>incy</b> &gt; 0 <math>y</math> is stored forward in array <b>y</b>, i.e.,<br/> <math>y_i</math> is stored in <math>\mathbf{y}((i-1) \times \text{incy} + 1)</math>.</p> <p><b>incy</b> &lt; 0 <math>y</math> is stored backward in array <b>y</b>, i.e.,<br/> <math>y_i</math> is stored in <math>\mathbf{y}((i-n) \times \text{incy} + 1)</math>.</p> <p>Use <b>incy</b> = 1 if the vector <math>y</math> is stored contiguously in array <b>y</b>, i.e., if <math>y_i</math> is stored in <math>\mathbf{y}(i)</math>. Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.</p> <p><b>ap</b> Array of length <math>\text{lenap} = n \times (n+1) / 2</math> containing the upper or lower triangle, as specified by <b>uplo</b>, of an <math>n</math> by <math>n</math> real symmetric or complex Hermitian matrix <math>A</math>, stored by columns in the packed form described above.</p> |
| Output | <p><b>ap</b> The upper or lower triangle of the updated <math>A</math> matrix, as specified by <b>uplo</b>, replaces the input.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
incx = 0, and
incy = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the uplo argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1** Apply a REAL\*8 symmetric rank-2 update  $xy^T + x^T y$  to  $A$ , where  $A$  is a 9 by 9 real symmetric matrix whose upper triangle is stored in packed form in an array AP of dimension 55,  $x$  is a real vector 9 elements long stored in an array X of dimension 10, and  $y$  is a real vector 9 elements long stored in an array Y also of dimension 10.

```

CHARACTER*1 UPLO
INTEGER*8 N, INCX, INCY
REAL*8 ALPHA, AP(55), X(10), Y(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
INCX = 1
INCY = 1
CALL SSPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)

```

**Example 2** Apply a COMPLEX\*16 Hermitian rank-2 update  $\alpha xy^* + \bar{\alpha} yx^*$  to  $A$ , where  $A$  is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in packed form in an array AP of dimension 55,  $\alpha$  is a complex scalar,  $x$  is a complex vector 9 elements long stored in an array X of dimension 10, and  $y$  is a complex vector 9 elements long stored in an array Y of dimension 10.

```

INTEGER*8 N
COMPLEX*16 ALPHA, AP(55), X(10), Y(10)
N = 9
CALL CHPR2 ('LOWER', N, ALPHA, X, 1, Y, 1, AP)

```

**Matrix-Matrix Multiply****SSYMM/CHEMM/CSYMM/ZHEMM**

**Purpose** These subprograms compute the matrix-matrix products  $AB$  and  $BA$ , where  $A$  is a real symmetric, complex symmetric, or complex Hermitian matrix and  $B$  is an  $m$  by  $n$  matrix. The size of  $A$ , either  $m$  by  $m$  or  $n$  by  $n$ , depends on which matrix product is requested. The product may be stored in the result matrix (which is always of size  $m$  by  $n$ ) or, optionally, may be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix product and the result matrix. Specifically, these subprograms compute matrix products of the forms

$$C = \alpha AB + \beta C \quad \text{and} \quad C = \alpha BA + \beta C.$$

The structure of  $A$  is indicated by the name of the subprogram used:

SSYMM or                     $A$  is a real symmetric matrix  
 CHEMM or ZHEMM          $A$  is a complex Hermitian matrix  
 CSYMM or                    $A$  is a complex symmetric matrix

Because either triangle of  $A$  may be obtained from the other, these subprograms reference only one triangle of  $A$ , as you specify. That triangle of  $A$  is stored in a 2-dimensional array.

**Usage****SCILIB:**

```
CHARACTER*1 side,uplo
INTEGER*8 m,n,lda,ldb,ldc
REAL*8 alpha,beta,a(lda,*),b(ldb,*),c(ldc,*)
CALL SSYMM (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
CHARACTER*1 side,uplo
INTEGER*8 m,n,lda,ldb,ldc
COMPLEX*16 alpha,beta,a(lda,*),b(ldb,*),c(ldc,*)
CALL CHEMM (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
```

```
CHARACTER*1 side,uplo
INTEGER*8 m,n,lda,ldb,ldc
COMPLEX*16 alpha,beta,a(lda,*),b(ldb,*),c(ldc,*)
CALL CSYMM (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
```

**Input**

**side** Specifies whether symmetric or Hermitian matrix  $A$  is the left or right matrix operand:

'L' or 'l'  $A$  is the left matrix operand, i.e. compute  $C = \alpha AB + \beta C$

'R' or 'r'  $A$  is the right matrix operand, i.e. compute  $C = \alpha BA + \beta C$

**uplo** Upper/lower triangular storage option for  $A$ :

'L' or 'l' Reference only the lower triangle of  $A$

'U' or 'u' Reference only the upper triangle of  $A$

**m** Number of rows in matrix  $C$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference  $a$ ,  $b$ , or  $c$ .

**n** Number of columns in matrix  $B$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $b$ , or  $c$ .

**alpha** The scalar  $\alpha$ . If  $\alpha = 0$ , the subprograms compute  $C = \beta C$  without referencing  $a$  or  $b$ .

**a** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of the matrix  $A$ . The other triangle of **a** is not referenced. The size of  $A$  is indicated by **side**:

'L' or 'l'       $A$  is  $m$  by  $m$   
 'R' or 'r'       $A$  is  $n$  by  $n$

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $\text{lda} \geq \max(\text{the number of rows of } A, 1)$ .

**b** Array containing the  $m$  by  $n$  matrix  $B$ .

**ldb** The leading dimension of array **b** as declared in the calling program unit, with  $\text{ldb} \geq \max(m, 1)$ .

**beta** The scalar  $\beta$ .

**c** Array containing the  $m$  by  $n$  matrix  $C$ . Not used as input if  $\text{beta} = 0$ .

**ldc** The leading dimension of array **c** as declared in the calling program unit, with  $\text{ldc} \geq \max(m, 1)$ .

**Output**      **c**      The updated  $C$  matrix replaces the input.

**Notes**      These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**side**  $\neq$  'L' or 'l' or 'R' or 'r',  
**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**m**  $< 0$ ,  
**n**  $< 0$ ,  
**lda** too small,  
**ldb**  $< \max(m, 1)$ , and  
**ldc**  $< \max(m, 1)$ .

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved, for example, by coding the **side** argument as 'LEFT' for 'L' or 'RIGHT' for 'R'. Refer to "Example 2."

Continued

SSYMM/CHEMM/CSYMM/ZHEMM

**Example 1** Form the REAL\*8 matrix product  $C = AB$ , where  $A$  is a 6 by 6 real symmetric real matrix whose upper triangle is stored in the upper triangle of an array  $A$  of dimension 10 by 10,  $B$  is a 6 by 8 real matrix stored in an array  $B$  of dimension 10 by 10, and  $C$  is a 6 by 8 real matrix stored in an array  $C$ , also of dimension 10 by 10.

```

CHARACTER*1 SIDE,UPLO
INTEGER*8 M,N,LDA,LDB,LDC
REAL*8 ALPHA,BETA,A(10,10),B(10,10),C(10,10)
SIDE = 'L'
UPLO = 'U'
M = 6
N = 8
ALPHA = 1.0
BETA = 0.0
LDA = 10
LDB = 10
LDC = 10
CALL SSYMM (SIDE,UPLO,M,N,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

```

**Example 2** Form the COMPLEX\*16 matrix-matrix product  $C = \frac{1}{2}BA - \rho C$ , where  $\rho$  is a scalar,  $A$  is an 8 by 8 complex Hermitian matrix whose lower triangle is stored in the lower triangle of an array  $A$  of dimension 10 by 10,  $B$  is a 6 by 8 complex matrix stored in an array whose dimensions are 10 by 10, and  $C$  is a 6 by 8 complex matrix stored in an array  $C$ , also of dimension 10 by 10.

```

INTEGER*8 M,N,LDA,LDB,LDC
COMPLEX*16 HALF,RHO,A(10,10),B(10,10),C(10,10)
M = 6
N = 8
LDA = 10
LDB = 10
LDC = 10
HALF = (0.5,0.0)
CALL CHEMM ('RIGHT','LOWER',M,N,-RHO,A,LDA,B,LDB,HALF,C,LDC)

```

**Purpose** These subprograms compute the matrix-vector product  $Ax$  where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix and  $x$  is a real or complex  $n$ -vector. The product may be stored in the result array, or, optionally, be added to or subtracted from it. This is handled in a convenient, but general, way by two scalar arguments,  $\alpha$  and  $\beta$ , which are used as multipliers of the matrix-vector product and the result vector. Specifically, these subprograms compute the matrix-vector product of the form

$$y = \alpha Ax + \beta y.$$

The structure of  $A$  is indicated by the name of the subprogram used:

SSYMV  $A$  is a real symmetric matrix  
 CHEMV  $A$  is a complex Hermitian matrix

Because either triangle of  $A$  may be obtained from the other, these subprograms reference only one triangle of  $A$ , as you specify. That triangle of  $A$  is stored in a 2-dimensional array.

**Usage****SCILIB:**

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx,incy
REAL*8 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL SSYMV (uplo, n, alpha, a, lda, x, incx, beta, y, incy)
```

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx,incy
COMPLEX*16 alpha,beta,a(lda,n),x(lenx),y(leny)
CALL CHEMV (uplo, n, alpha, a, lda, x, incx, beta, y, incy)
```

**Input**

**uplo** Upper/lower triangular option for  $A$ :

'L' or 'l' Reference only the lower triangle of  $A$ .  
 'U' or 'u' Reference only the upper triangle of  $A$ .

**n** Number of rows and columns in matrix  $A$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**alpha** The scalar  $\alpha$ . If  $\alpha = 0$ , the subprograms compute  $y = \beta y$  without referencing  $a$  or  $x$ .

**a** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of an  $n$  by  $n$  real symmetric or complex Hermitian matrix  $A$ . The other triangle of  $a$  is not referenced.

**lda** The leading dimension of array  $a$  as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $incx \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .  
**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is

stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**beta** The scalar  $\beta$ .

**y** Array of length  $leny = (n-1) \times |incy| + 1$  containing the  $n$ -vector  $y$ . Not used as input if **beta** = 0.

**incy** Increment for the array **y**, **incy**  $\neq$  0:

**incy** > 0  $y$  is stored forward in array **y**, i.e.,

$y_i$  is stored in  $y((i-1) \times incy + 1)$ .

**incy** < 0  $y$  is stored backward in array **y**, i.e.,

$y_i$  is stored in  $y((i-n) \times incy + 1)$ .

Use **incy** = 1 if the vector  $y$  is stored contiguously in array **y**, i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **y** The updated  $y$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo \neq 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(n,1),
incx = 0, and
incy = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $y = Ax$ , where  $A$  is a 9 by 9 real symmetric matrix whose upper triangle is stored in the upper triangle of an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 9 elements long stored in an array  $X$  of dimension 10, and  $y$  is a real vector 9 elements long stored in an array  $Y$ , also of dimension 10.

```

CHARACTER*1 UPLO
INTEGER*8 N,LDA,INCX,INCY
REAL*8 ALPHA,BETA,A(10,10),X(10),Y(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
BETA = 0.0
LDA = 10
INCX = 1
INCY = 1
CALL SSYMV (UPLO,N,ALPHA,A,LDA,X,INCX,BETA,Y,INCY)

```

**Example 2** Form the COMPLEX\*16 matrix-vector product  $y = \frac{1}{2}y - \rho Ax$ , where  $\rho$  is a complex scalar,  $A$  is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in the lower triangle of an array  $A$  whose dimensions are 10 by 10,  $x$  is a complex vector 9 elements long stored in an array  $X$  of dimension 10, and  $y$  is a complex vector 9 elements long stored in an array  $Y$ , also of dimension 10.

```
INTEGER*8 N,LDA
COMPLEX*16 RHO,A(10,10),X(10),Y(10)
N = 9
LDA = 10
CALL CHEMV ('LOWER',N,-RHO,A,LDA,X,1,(0.5,0.0),Y,1)
```

## Rank-1 Update

## SSYR/CHER

**Purpose** These subprograms compute the real symmetric or complex Hermitian rank-1 update

$$A - \alpha x x^* + A,$$

where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix,  $\alpha$  is a real scalar,  $x$  is a real or complex  $n$ -vector, and  $x^*$  is the conjugate transpose of  $x$ . (The conjugate transpose of a real vector is simply the transpose.)

The structure of  $A$  is indicated by the name of the subprogram used:

SSYR  $A$  is a real symmetric matrix

CHER  $A$  is a complex Hermitian matrix

Because either triangle of  $A$  may be obtained from the other, these subprograms reference and apply the update to only one triangle of  $A$ , as you specify. That triangle of  $A$  is stored in a 2-dimensional array.

**Usage****SCILIB:**

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx
REAL*8 alpha,a(lda,n),x(lenx)
CALL SSYR (uplo, n, alpha, x, incx, a, lda)
```

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx
REAL*8 alpha
COMPLEX*16 a(lda,n),x(lenx)
CALL CHER (uplo, n, alpha, x, incx, a, lda)
```

**Input**

**uplo** Upper/lower triangular option for  $A$ :

'L' or 'l' Reference and update only the lower triangle of  $A$ .  
'U' or 'u' Reference and update only the upper triangle of  $A$ .

**n** Number of rows and columns in matrix  $A$  and elements of vector  $x$ ,  $n \geq 0$ .  
If  $n = 0$ , the subprograms do not reference  $a$  or  $x$ .

**alpha** The scalar  $\alpha$ . If  $\alpha = 0$ , the subprograms do not reference  $a$  or  $x$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**a** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of an  $n$  by  $n$  real symmetric or complex Hermitian matrix  $A$ . The other triangle of  $a$  is not referenced.

- lda**      The leading dimension of array **a** as declared in the calling program unit, with  $\text{lda} \geq \max(\mathbf{n}, 1)$ .
- Output**    **a**      The upper or lower triangle of the updated *A* matrix, as specified by **uplo**, replaces the upper or lower triangle of the input, respectively. The other triangle of **a** is unchanged.
- Notes**      These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo ≠ 'L' or 'l' or 'U' or 'u',
n < 0,
lda < max(n,1), and
incx = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

- Example 1**    Apply a REAL\*8 symmetric rank-1 update  $xx^T$  to *A*, where *A* is a 9 by 9 real symmetric matrix whose upper triangle is stored in the upper triangle of an array *A* whose dimensions are 10 by 10, and *x* is a real vector 9 elements long stored in an array *X* of dimension 10.

```

CHARACTER*1 UPLO
INTEGER*8 N,LDA, INCX
REAL*8 ALPHA, A(10,10), X(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
LDA = 10
INCX = 1
CALL SSYR (UPLO, N, ALPHA, X, INCX, A, LDA)

```

- Example 2**    Apply a COMPLEX\*16 Hermitian rank-1 update  $-2xx^*$  to *A*, where *A* is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in the lower triangle of an array *A* whose dimensions are 10 by 10, and *x* is a complex vector 9 elements long stored in an array *X* of dimension 10.

```

INTEGER*8 N,LDA
REAL*8 ALPHA
COMPLEX*16 A(10,10), X(10)
N = 9
ALPHA = -2.0
LDA = 10
CALL CHER ('LOWER', N, ALPHA, X, 1, A, LDA)

```

## Rank-2 Update

## SSYR2/CHER2

**Purpose** These subprograms compute the real symmetric or complex Hermitian rank-2 update

$$A - \alpha xy^* + \bar{\alpha} yx^* + A,$$

where  $A$  is an  $n$  by  $n$  real symmetric or complex Hermitian matrix,  $\alpha$  is a complex scalar,  $\bar{\alpha}$  is the complex conjugate of  $\alpha$ ,  $x$  and  $y$  are real or complex  $n$ -vectors, and  $x^*$  and  $y^*$  are the conjugate transposes of  $x$  and  $y$ , respectively. (The conjugate of a real scalar is just the scalar, and the conjugate transpose of a real vector is simply the transpose.)

The structure of  $A$  is indicated by the name of the subprogram used:

SSYR2 or  $A$  is a real symmetric matrix  
 CHER2 or  $A$  is a complex Hermitian matrix

Because either triangle of  $A$  may be obtained from the other, these subprograms reference and apply the update to only one triangle of  $A$ , as you specify. That triangle of  $A$  is stored in a 2-dimensional array.

**Usage**

SCILIB:

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx,incy
REAL*8 alpha,a(lda,n),x(lenx),y(leny)
CALL SSYR2 (uplo, n, alpha, x, incx, y, incy, a, lda)
```

```
CHARACTER*1 uplo
INTEGER*8 n,lda,incx,incy
COMPLEX*16 alpha,a(lda,n),x(lenx),y(leny)
CALL CHER2 (uplo, n, alpha, x, incx, y, incy, a, lda)
```

**Input**

**uplo** Upper/lower triangular option for  $A$ :

'L' or 'l' Reference and update only the lower triangle of  $A$ .  
 'U' or 'u' Reference and update only the upper triangle of  $A$ .

**n** Number of rows and columns in matrix  $A$  and elements of vectors  $x$  and  $y$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ , **incx**  $\neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .  
**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector  $y$ .

**incy** Increment for the array **y**, **incy**  $\neq$  0:

**incy**  $>$  0  $y$  is stored forward in array **y**, i.e.,

$y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy**  $<$  0  $y$  is stored backward in array **y**, i.e.,

$y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use **incy** = 1 if the vector  $y$  is stored contiguously in array **y**, i.e., if  $y_i$  is stored in **y**(*i*). Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**a** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of an  $n$  by  $n$  real symmetric or complex Hermitian matrix  $A$ . The other triangle of **a** is not referenced.

**lda** The leading dimension of array **a** as declared in the calling program unit, with **lda**  $\geq$   $\max(n, 1)$ .

#### Output

**a** The upper or lower triangle of the updated  $A$  matrix, as specified by **uplo**, replaces the upper or lower triangle of the input, respectively. The other triangle of **a** is unchanged.

#### Notes

These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',

**n**  $<$  0,

**lda**  $<$   $\max(n, 1)$ ,

**incx** = 0, and

**incy** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'.

#### Example 1

Apply a REAL\*8 symmetric rank-2 update  $xy^T + x^T y$  to  $A$ , where  $A$  is a 9 by 9 real symmetric matrix whose upper triangle is stored in the upper triangle of an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 9 elements long stored in an array  $X$  of dimension 10, and  $y$  is a real vector 9 elements long stored in an array  $Y$  also of dimension 10.

```
CHARACTER*1 UPLO
INTEGER*8 N, LDA, INCX, INCY
REAL*8 ALPHA, A(10,10), X(10), Y(10)
UPLO = 'U'
N = 9
ALPHA = 1.0
LDA = 10
INCX = 1
INCY = 1
CALL SSYR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)
```

**Example 2** Apply a COMPLEX\*16 Hermitian rank-2 update  $\alpha xy^* + \bar{\alpha} yx^*$  to  $A$ , where  $A$  is a 9 by 9 complex Hermitian matrix whose lower triangle is stored in the lower triangle of an array  $A$  whose dimensions are 10 by 10,  $\alpha$  is a complex scalar,  $x$  is a complex vector 9 elements long stored in an array  $X$  of dimension 10, and  $y$  is a complex vector 9 elements long stored in an array  $Y$  of dimension 10.

```
INTEGER*8 N,LDA,INCX,INCY
COMPLEX*16 ALPHA,A(10,10),X(10),Y(10)
N = 9
LDA = 10
INCX = 1
INCY = 1
CALL CHER2 ('LOWER',N,ALPHA,X,INCX,Y,INCY,A,LDA)
```

**Purpose** These subprograms apply a symmetric or Hermitian rank-2k update to a real symmetric, complex symmetric, or complex Hermitian matrix; specifically they compute the following operations:

for symmetric  $C$ :  $C - \alpha AB^T + \bar{\alpha} BA^T + \beta C$  and  $C - \alpha A^T B + \bar{\alpha} B^T A + \beta C$

for Hermitian  $C$ :  $C - \alpha AB^* + \bar{\alpha} BA^* + \beta C$  and  $C - \alpha A^* B + \bar{\alpha} B^* A + \beta C$

where  $\alpha$  and  $\beta$  are scalars,  $\bar{\alpha}$  is the complex conjugate of  $\alpha$ ,  $C$  is an  $n$  by  $n$  real symmetric, complex symmetric, or complex Hermitian matrix, and  $A$  and  $B$  are matrices whose size, either  $n$  by  $k$  or  $k$  by  $n$ , depends on which form of the update is requested. Here,  $A^T$  and  $B^T$  are the transposes and  $A^*$  and  $B^*$  are the conjugate transposes of  $A$  and  $B$ , respectively. (The conjugate of a real scalar is just the scalar, and the conjugate transpose of a real matrix is simply the transpose.)

The structure of  $C$  is indicated by the name of the subprogram used:

Because either triangle of  $C$  may be obtained from the other, these subprograms reference and apply the update to only one triangle of  $C$ , as you specify. That triangle of  $C$  is stored in a 2-dimensional array.

**Usage****SCILIB:**

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldb,ldc
REAL*8 alpha,beta,a(lda,*),b(ldb,*),c(ldc,*)
CALL SSYR2K (uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
```

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldb,ldc
REAL*8 alpha,beta
COMPLEX*16 a(lda,*),b(ldb,*),c(ldc,*)
CALL CHER2K (uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
```

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldb,ldc
COMPLEX*16 alpha,beta,a(lda,*),b(ldb,*),c(ldc,*)
CALL CSYR2K (uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
```

**Input**

**uplo** Upper/lower triangular storage option for  $C$ :

'L' or 'l' Reference and update only the lower triangle of  $C$   
 'U' or 'u' Reference and update only the upper triangle of  $C$

**trans** Specifies the operation to be performed:

'N' or 'n' Compute  $C - \alpha AB^T + \alpha BA^T + \beta C$   
 'T' or 't' Compute  $C - \alpha A^T B + \alpha B^T A + \beta C$   
 'C' or 'c' Compute  $C - \alpha A^* B + \alpha B^* A + \beta C$

'T' and 't' are invalid in subprogram CHER2K, and 'C' and 'c' are invalid in subprogram CSYR2K. In subprogram SSYR2K, 'C' and 'c' have the same meaning as 'T' and 't'.

- n** Number of rows and columns in matrix  $C$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $\mathbf{a}$ ,  $\mathbf{b}$ , or  $\mathbf{c}$ .
- k** Number of rows or columns in matrices  $A$  and  $B$ , depending on **trans**; refer to the description of  $\mathbf{a}$  for details.  $k \geq 0$ ; if  $k = 0$ , the subprograms do not reference  $\mathbf{a}$  or  $\mathbf{b}$ .
- alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $C + \beta C$  without referencing  $\mathbf{a}$  or  $\mathbf{b}$ .
- a** Array containing the matrix  $A$ , whose size is indicated by **trans**:
- |            |                   |
|------------|-------------------|
| 'N' or 'n' | $A$ is $n$ by $k$ |
| otherwise  | $A$ is $k$ by $n$ |
- lda** The leading dimension of array  $\mathbf{a}$  as declared in the calling program unit, with  $\text{lda} \geq \max(\text{the number of rows of } A, 1)$ .
- b** Array containing matrix  $B$ , which is the same size as matrix  $A$ . Refer to the description of  $\mathbf{a}$  above for details.
- ldb** The leading dimension of array  $\mathbf{b}$  as declared in the calling program unit, with  $\text{ldb} \geq \max(\text{the number of rows of } B, 1)$ .
- beta** The scalar  $\beta$ .
- c** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of the  $n$  by  $n$  symmetric or Hermitian matrix  $C$ . Not used as input if **beta** = 0.
- ldc** The leading dimension of array  $\mathbf{c}$  as declared in the calling program unit, with  $\text{ldc} \geq \max(n, 1)$ .
- Output**
- c** The upper or lower triangle of the updated matrix  $C$ , as specified by **uplo**, replaces the upper or lower triangle of the input, respectively. The other triangle of  $\mathbf{c}$  is unchanged.

**Notes** These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
 $n < 0$ ,  
 $k < 0$ ,  
**lda** too small,  
**ldb** too small, and  
 $\text{ldc} < \max(m, 1)$ .

Also, note that some of the values of **trans** listed above are invalid in subprograms CHER2K, and CSYR2K.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved, for example, by coding the **uplo** argument as 'LOWER' for 'L' or 'UPPER' for 'U'. Refer to "Example 2."

**Example 1** Apply a REAL\*8 rank-6 update  $AB^T + BA^T$  to an 8 by 8 real symmetric matrix  $C$  whose upper triangle is stored in the upper triangle of an array  $C$  of dimension 10 by 10, where  $A$  is an 8 by 3 real matrix stored in an array  $A$ , also of dimension 10 by 10.

```

CHARACTER*1 UPLO,TRANS
INTEGER*8 N,K,LDA,LDB,LDC
REAL*8 ALPHA,BETA,A(10,10),B(10,10),C(10,10)
UPLO = 'U'
TRANS = 'N'
N = 8
K = 3
ALPHA = 1.0
BETA = 1.0
LDA = 10
LDB = 10
LDC = 10
CALL SSYR2K (UPLO,TRANS,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

```

**Example 2** Apply a COMPLEX\*16 Hermitian rank-4 update  $-2AB^* - 2BA^*$  to a 9 by 9 complex Hermitian matrix  $C$  whose lower triangle is stored in the lower triangle of an array  $C$  of dimension 10 by 10, where  $A$  is a 9 by 2 complex matrix stored in an array  $A$  of dimension 10 by 10.

```

INTEGER*8 N,K,LDA,LDB,LDC
REAL*8 ALPHA,BETA
COMPLEX*16 A(10,10),B(10,10),C(10,10)
N = 9
K = 2
ALPHA = -2.0
BETA = 1.0
LDA = 10
LDB = 10
LDC = 10
CALL CHER2K ('LOWER', 'NONTRANS', N, K, ALPHA, A, LDA, B,
> LDB, BETA, C, LDC)

```

## Rank-k Update

## SSYRK/CHERK

**Purpose** These subprograms apply a rank- $k$  update to a real symmetric, complex symmetric, or complex Hermitian matrix; specifically they compute:

$$\begin{aligned} C - \alpha AA^T + \beta C, & \quad C - \alpha A^T A + \beta C, \\ C - \alpha AA^* + \beta C, & \quad C - \alpha A^* A + \beta C, \end{aligned}$$

where  $\alpha$  and  $\beta$  are scalars,  $C$  is an  $n$  by  $n$  real symmetric, complex symmetric, or complex Hermitian matrix, and  $A$  is a matrix whose size, either  $n$  by  $k$  or  $k$  by  $n$ , depends on which form of the update is requested. Here,  $A^T$  and  $A^*$  are the transpose and conjugate transpose of  $A$ , respectively.

The structure of  $C$  is indicated by the name of the subprogram used:

SSYRK  $C$  is a real symmetric matrix  
 CHERK  $C$  is a complex Hermitian matrix  
 CSYRK  $C$  is a complex symmetric matrix

Because either triangle of  $C$  may be obtained from the other, these subprograms reference and apply the update to only one triangle of  $C$ , as you specify. That triangle of  $C$  is stored in a 2-dimensional array.

**Usage****SCILIB:**

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldc
REAL*8 alpha,beta,a(lda,*),c(ldc,*)
CALL SSYRK (uplo, trans, n, k, alpha, a, lda, beta, c, ldc)
```

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldc
REAL*8 alpha,beta
COMPLEX*16 a(lda,*),c(ldc,*)
CALL CHERK (uplo, trans, n, k, alpha, a, lda, beta, c, ldc)
```

```
CHARACTER*1 uplo,trans
INTEGER*8 n,k,lda,ldc
COMPLEX*16 alpha,beta,a(lda,*),c(ldc,*)
CALL CSYRK (uplo, trans, n, k, alpha, a, lda, beta, c, ldc)
```

**Input**

**uplo** Upper/lower triangular storage option for  $C$ :

'L' or 'l' Reference and update only the lower triangle of  $C$   
 'U' or 'u' Reference and update only the upper triangle of  $C$

**trans** Specifies the operation to be performed:

'N' or 'n' Compute  $C - \alpha AA^T + \beta C$   
 'T' or 't' Compute  $C - \alpha A^T A + \beta C$   
 'C' or 'c' Compute  $C - \alpha A^* A + \beta C$

'T' and 't' are invalid in subprogram CHER2K, and 'C' and 'c' are invalid in subprogram CSYR2K. In subprograms SSYRK and , 'C' and 'c' have the same meaning as 'T' and 't', respectively.

- n** Number of rows and columns in matrix  $C$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **a** or **c**.
- k** Number of rows or columns in matrix  $A$ ,  $k \geq 0$ , depending on **trans**; refer to description of **A** for details. If  $k = 0$ , the subprograms do not reference **a**.
- alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $C - \beta C$  without referencing **a**.
- a** Array containing the matrix  $A$ , whose size is indicated by **trans**:
- ‘N’ or ‘n’     $A$  is  $n$  by  $k$   
otherwise     $A$  is  $k$  by  $n$
- lda** The leading dimension of array **a** as declared in the calling program unit, with **lda**  $\geq$  max (the number of rows of  $A, 1$ ).
- beta** The scalar  $\beta$ .
- c** Array whose upper or lower triangle, as specified by **uplo**, contains the upper or lower triangle of the  $n$  by  $n$  symmetric or Hermitian matrix  $C$ . Not used as input if **beta** = 0.
- ldc** The leading dimension of array **c** as declared in the calling program unit, with **ldc**  $\geq$  max(**n**,1).
- Output**
- c** The upper or lower triangle of the updated  $C$  matrix, as specified by **uplo**, replaces the upper or lower triangle of the input, respectively. The other triangle of **c** is unchanged.
- Notes** These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  ‘L’ or ‘l’ or ‘U’ or ‘u’,  
**trans**  $\neq$  ‘N’ or ‘n’ or ‘T’ or ‘t’ or ‘C’ or ‘c’,  
**n**  $< 0$ ,  
**k**  $< 0$ ,  
**lda** too small, and  
**ldc**  $< \max(\mathbf{n}, 1)$ .

Also, some values of **trans** listed above are invalid in subprograms CHERK and CSYRK.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved, for example, by coding the **uplo** argument as ‘LOWER’ for ‘L’ or ‘UPPER’ for ‘U’. Refer to “Example 2.”

**Example 1** Apply a REAL\*8 rank-6 update  $AA^T$  to an 8 by 8 real symmetric matrix  $C$  whose upper triangle is stored in the upper triangle of an array  $C$  of dimension 10 by 10, where  $A$  is an 8 by 6 real matrix stored in an array  $A$ , also of dimension 10 by 10.

```

CHARACTER*1 UPLO, TRANS
INTEGER*8 N, K, LDA, LDC
REAL*8 ALPHA, BETA, A(10,10), C(10,10)
UPLO = 'U'
TRANS = 'N'
N = 8
K = 6
ALPHA = 1.0
BETA = 1.0
LDA = 10
LDC = 10
CALL SSYRK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)

```

**Example 2** Apply a COMPLEX\*16 Hermitian rank-2 update  $-2AA^*$  to a 9 by 9 complex Hermitian matrix  $C$  whose lower triangle is stored in the lower triangle of an array  $C$  of dimension 10 by 10, where  $A$  is a 9 by 2 complex matrix stored in an array  $A$  of dimension 10 by 10.

```

INTEGER*8 N, K, LDA, LDC
REAL*8 ALPHA, BETA
COMPLEX*16 A(10,10), C(10,10)
N = 9
K = 2
ALPHA = -2.0
BETA = 1.0
LDA = 10
LDC = 10
CALL CHERK ('LOWER', 'NONTRANS', N, K, ALPHA, A, LDA, BETA, C, LDC)

```

**Purpose** Given an  $n$  by  $n$  upper- or lower-triangular band matrix  $T$ , and an  $n$ -vector  $x$ , these subprograms compute the matrix-vector products  $Tx$ ,  $T^T x$ , and  $T^*x$ , where  $T^T$  is the transpose of  $T$ , and  $T^*$  is the conjugate transpose. Specifically, these subprograms compute matrix-vector products of the forms

$$x - Tx, \quad x - T^T x, \quad \text{and} \quad x - T^*x.$$

A lower-triangular band matrix is a matrix whose strict upper triangle is zero, and whose nonzero lower-triangular elements all are on or fairly near the principal diagonal. Specifically,  $t_{ij} \neq 0$  only if  $0 \leq i-j \leq m$  for some integer  $m$ . In contrast, an upper-triangular band matrix is a matrix whose strict lower triangle is zero, and whose nonzero upper-triangular elements all are on or fairly near the principal diagonal, i.e., with  $t_{ij} \neq 0$  only if  $0 \leq j-i \leq m$ .

**Storage** Triangular band matrices are stored in a compressed form that takes advantage of knowing the positions of the only elements that may be nonzero. The following example illustrates the storage of lower-triangular band matrices. If  $T$  is a 9 by 9 lower-triangular band matrix with bandwidth  $m = 3$ , for example,

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 21 | 22 | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 31 | 32 | 33 | 0  | 0  | 0  | 0  | 0  | 0  |
| 41 | 42 | 43 | 44 | 0  | 0  | 0  | 0  | 0  |
| 0  | 52 | 53 | 54 | 55 | 0  | 0  | 0  | 0  |
| 0  | 0  | 63 | 64 | 65 | 66 | 0  | 0  | 0  |
| 0  | 0  | 0  | 74 | 75 | 76 | 77 | 0  | 0  |
| 0  | 0  | 0  | 0  | 85 | 86 | 87 | 88 | 0  |
| 0  | 0  | 0  | 0  | 0  | 96 | 97 | 98 | 99 |

the lower triangular band part of  $T$  is stored in an array  $\mathbf{t}$  with at least  $m+1 = 4$  rows and 9 columns:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| 21 | 32 | 43 | 54 | 65 | 76 | 87 | 98 | *  |
| 31 | 42 | 53 | 64 | 75 | 86 | 97 | *  | *  |
| 41 | 52 | 63 | 74 | 85 | 96 | *  | *  | *  |

where asterisks represent elements in the  $m$  by  $m$  triangle at the lower-right corner of  $\mathbf{t}$  that are not referenced. Thus, if  $t_{ij}$  is an element within the band of  $T$ , it is stored in  $\mathbf{t}(i-j+1, j)$ . In words, the columns of  $T$  are stored in the columns of  $\mathbf{t}$ , and the diagonals of  $T$  are stored in the rows of  $\mathbf{t}$ , with the principal diagonal in the first row, the first subdiagonal in the second row, and so on.

Similarly, the following example illustrates the storage of upper-triangular band matrices. If  $T$  is a 9 by 9 upper-triangular band matrix with bandwidth  $m = 3$ , for example,

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 14 | 0  | 0  | 0  | 0  | 0  |
| 0  | 22 | 23 | 24 | 25 | 0  | 0  | 0  | 0  |
| 0  | 0  | 33 | 34 | 35 | 36 | 0  | 0  | 0  |
| 0  | 0  | 0  | 44 | 45 | 46 | 47 | 0  | 0  |
| 0  | 0  | 0  | 0  | 55 | 56 | 57 | 58 | 0  |
| 0  | 0  | 0  | 0  | 0  | 66 | 67 | 68 | 69 |
| 0  | 0  | 0  | 0  | 0  | 0  | 77 | 78 | 79 |
| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 88 | 89 |
| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 99 |

the upper triangular band part of  $T$  is stored in an array  $t$  with at least  $m + 1 = 4$  rows and 9 columns:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| *  | *  | *  | 14 | 25 | 36 | 47 | 58 | 69 |
| *  | *  | 13 | 24 | 35 | 46 | 57 | 68 | 79 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |

where asterisks represent elements in the  $m$  by  $m$  triangle at the upper-left corner of  $t$  that are not referenced. Thus, if  $t_{ij}$  is an element within the band of  $T$ , it is stored in  $t(i-j+m+1,j)$ . In words, the columns of  $T$  are stored in the columns of  $t$ , and the diagonals of  $T$  are stored in the rows of  $t$ , with the principal diagonal in row  $m + 1$ , the first superdiagonal starting in the second position in row  $m$ , and so on.

**Usage****SCILIB:**

```

CHARACTER*1 uplo,trans,diag
INTEGER*8 n,m,ldt,incx
REAL*8 t(ldt,n),x(lenx)
CALL STBMV (uplo, trans, diag, n, m, t, ldt, x, incx)
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,m,ldt,incx
COMPLEX*16 t(ldt,n),x(lenx)
CALL CTBMV (uplo, trans, diag, n, m, t, ldt, x, incx)

```

**Input**

**uplo** Upper/lower triangular option for  $T$ :

'L' or 'l'  $T$  is lower triangular  
 'U' or 'u'  $T$  is upper triangular

**trans** Transposition option for  $T$ :

'N' or 'n' Compute  $x - Tx$   
 'T' or 't' Compute  $x - T^T x$   
 'C' or 'c' Compute  $x - T^* x$

where  $T^T$  is the transpose of  $T$ , and  $T^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag** Specifies whether the matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

'N' or 'n' The diagonal of  $T$  is stored in the array  
 'U' or 'u' The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as 'U' or 'u', diagonal elements of  $T$  are not referenced, but space must be reserved for them.

- n** Number of rows and columns in matrix  $T$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **t** or **x**.
- m** The number of nonzero diagonals above or below the principal diagonal. If **uplo** is supplied as 'U' or 'u', **m** specifies the number of nonzero diagonals above the principal diagonal. If **uplo** is supplied as 'L' or 'l', **m** specifies the number of nonzero diagonals below the principal diagonal.
- t** Array containing the  $n$  by  $n$  triangular band matrix  $T$  in the compressed form described above. The columns of the band of  $T$  are stored in the columns of **t**, and the diagonals of the band of  $T$  are stored in the rows of **t**.
- ldt** The leading dimension of array **t** as declared in the calling program unit, with  $\text{ldt} \geq m+1$ .
- x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the input vector  $x$ .
- incx** Increment for the array **x**,  $\text{incx} \neq 0$ :
- incx** > 0  $x$  is stored forward in array **x**, i.e.,  
 $x_i$  is stored in  $\mathbf{x}((i-1) \times \text{incx} + 1)$ .
- incx** < 0  $x$  is stored backward in array **x**, i.e.,  
 $x_i$  is stored in  $\mathbf{x}((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $\mathbf{x}(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **x** The updated  $x$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**n** < 0,  
**m** < 0,  
**ldt** <  $m+1$ , and  
**incx** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $Tx$ , where  $T$  is a 75 by 75 unit-diagonal, lower-triangular real band matrix with bandwidth 15 that is stored in an array T whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

CHARACTER*1 UPLO,TRANS,DIAG
INTEGER*8 N,M,LDT,INCX
REAL*8 T(25,100),X(100)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 75
M = 15
LDT = 25
INCX = 1
CALL STBMV (UPLO,TRANS,DIAG,N,M,T,LDT,X,INCX)

```

**Example 2** Form the REAL\*8 matrix-vector product  $T^T x$ , where  $T$  is a 75 by 75 nonunit-diagonal, upper-triangular real band matrix with bandwidth 15 that is stored in an array T whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

INTEGER*8 N,M,LDT,INCX
REAL*8 T(25,100),X(100)
N = 75
M = 15
LDT = 25
INCX = 1
CALL STBMV ('UPPER','TRANSPPOSE','NONUNIT',N,M,T,LDT,X,INCX)

```

**Purpose** Given an  $n$  by  $n$  upper- or lower-triangular band matrix  $T$ , and an  $n$ -vector  $x$ , these subprograms overwrite  $x$  with the solution  $y$  to the system of linear equations  $Ty = x$ . This is the forward elimination or back substitution step of Gaussian elimination for band matrices. Optionally,  $T$  may be replaced by  $T^T$ , the transpose of  $T$ , or by  $T^*$ , the conjugate transpose of  $T$ .

A lower-triangular band matrix is a matrix whose strict upper triangle is zero, and whose nonzero lower-triangular elements all are on or fairly near the principal diagonal. Specifically,  $t_{ij} \neq 0$  only if  $0 \leq i-j \leq m$  for some integer  $m$ . In contrast, an upper-triangular band matrix is a matrix whose strict lower triangle is zero, and whose nonzero upper-triangular elements all are on or fairly near the principal diagonal, but with  $t_{ij} \neq 0$  only if  $0 \leq j-i \leq m$ .

Specifically, these subprograms compute

$$x - T^{-1}x, \quad x - T^{-T}x, \quad \text{and} \quad x - T^{-*}x$$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose of  $T$ .

These subprograms are more primitive than the LINPACK band equation solvers. As such, they are intended to supplement the equation solvers, but not replace them, serving instead as building blocks in constructing optimized linear algebra software. In fact, many of the LINPACK subprograms have been recoded to call these routines.

**Storage** Triangular band matrices are stored in a compressed form that takes advantage of knowing the positions of the only elements that may be nonzero. The following example illustrates the storage of lower-triangular band matrices. If  $T$  is a 9 by 9 lower-triangular band matrix with bandwidth  $m = 3$ , for example,

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 21 | 22 | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| 31 | 32 | 33 | 0  | 0  | 0  | 0  | 0  | 0  |
| 41 | 42 | 43 | 44 | 0  | 0  | 0  | 0  | 0  |
| 0  | 52 | 53 | 54 | 55 | 0  | 0  | 0  | 0  |
| 0  | 0  | 63 | 64 | 65 | 66 | 0  | 0  | 0  |
| 0  | 0  | 0  | 74 | 75 | 76 | 77 | 0  | 0  |
| 0  | 0  | 0  | 0  | 85 | 86 | 87 | 88 | 0  |
| 0  | 0  | 0  | 0  | 0  | 96 | 97 | 98 | 99 |

the lower triangular band part of  $T$  is stored in an array  $t$  with at least  $m+1 = 4$  rows and 9 columns:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| 21 | 32 | 43 | 54 | 65 | 76 | 87 | 98 | *  |
| 31 | 42 | 53 | 64 | 75 | 86 | 97 | *  | *  |
| 41 | 52 | 63 | 74 | 85 | 96 | *  | *  | *  |

where asterisks represent elements in the  $m$  by  $m$  triangle at the lower-right corner of  $t$  that are not referenced. Thus, if  $t_{ij}$  is an element within the band of  $T$ , it is stored in  $t(i-j+1,j)$ . In words, the columns of  $T$  are stored in the columns of  $t$ , and the diagonals of  $T$  are stored in the rows of  $t$ , with the principal diagonal in the first row, the first subdiagonal in the second row, and so on.

Continued

STBSV/CTBSV

Similarly, the following example illustrates the storage of upper-triangular band matrices. If  $T$  is a 9 by 9 upper-triangular band matrix with bandwidth  $m = 3$ , for example,

```

11 12 13 14 0 0 0 0 0
 0 22 23 24 25 0 0 0 0
 0 0 33 34 35 36 0 0 0
 0 0 0 44 45 46 47 0 0
 0 0 0 0 55 56 57 58 0
 0 0 0 0 0 66 67 68 69
 0 0 0 0 0 0 77 78 79
 0 0 0 0 0 0 0 88 89
 0 0 0 0 0 0 0 0 99

```

the upper triangular band part of  $T$  is stored in an array  $t$  with at least  $m + 1 = 4$  rows and 9 columns:

```

* * * 14 25 36 47 58 69
* * 13 24 35 46 57 68 79
* 12 23 34 45 56 67 78 89
11 22 33 44 55 66 77 88 99

```

where asterisks represent elements in the  $m$  by  $m$  triangle at the upper-left corner of  $t$  that are not referenced. Thus, if  $t_{ij}$  is an element within the band of  $T$ , it is stored in  $t(i-j+m+1,j)$ . In words, the columns of  $T$  are stored in the columns of  $t$ , and the diagonals of  $T$  are stored in the rows of  $t$ , with the principal diagonal in row  $m + 1$ , the first superdiagonal starting in the second position in row  $m$ , and so on.

**Usage****SCILIB:**

```

CHARACTER*1 uplo,trans,diag
INTEGER*8 n,m,ldt,incx
REAL*8 t(ldt,n),x(lenx)
CALL STBSV (uplo, trans, diag, n, m, t, ldt, x, incx)
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,m,ldt,incx
COMPLEX*16 t(ldt,n),x(lenx)
CALL CTBSV (uplo, trans, diag, n, m, t, ldt, x, incx)

```

**Input****uplo**Upper/lower triangular option for  $T$ :

'L' or 'l' Solve lower-triangular band system (forward elimination)  
 'U' or 'u' Solve upper-triangular band system (back substitution)

**trans**Transposition option for  $T$ :

'N' or 'n' Compute  $x - T^{-1}x$   
 'T' or 't' Compute  $x - T^{-T}x$   
 'C' or 'c' Compute  $x - T^{-*}x$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag**Specifies whether the matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

'N' or 'n' The diagonal of  $T$  is stored in the array  
 'U' or 'u' The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as 'U' or 'u', diagonal elements of  $T$  are not referenced, but space must be reserved for them.

- n** Number of rows and columns in matrix  $T$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **t** or **x**.
- m** The number of nonzero diagonals above or below the principal diagonal. If **uplo** is supplied as 'U' or 'u', **m** specifies the number of nonzero diagonals above the principal diagonal. If **uplo** is supplied as 'L' or 'l', **m** specifies the number of nonzero diagonals below the principal diagonal.
- t** Array containing the  $n$  by  $n$  triangular band matrix  $T$  in the compressed form described above. The columns of the band of  $T$  are stored in the columns of **t**, and the diagonals of the band of  $T$  are stored in the rows of **t**.
- ldt** The leading dimension of array **t** as declared in the calling program unit, with  $\text{ldt} \geq m+1$ .
- x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the right-hand-side  $n$ -vector  $x$ .
- incx** Increment for the array **x**,  $\text{incx} \neq 0$ :
- incx** > 0  $x$  is stored forward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .
- incx** < 0  $x$  is stored backward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **x** The solution vector of the triangular band system replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

The subprograms do not check for singularity of matrix  $T$ .  $T$  is singular if **diag** = 'N' or 'n' and some  $t_{ii} = 0$ . This condition causes a division by zero to occur. Therefore, the program must detect singularity and take appropriate action to avoid a problem before calling any of these subprograms.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**n** < 0,  
**m** < 0,  
**ldt** < **m**+1, and  
**incx** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Perform REAL\*8 forward elimination using the 75 by 75 unit-diagonal lower-triangular real band matrix with bandwidth 15 that is stored in an array A whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

CHARACTER*1 UPLO,TRANS,DIAG
INTEGER*8 N,M,LDA,INCX
REAL*8 A(25,100),X(100)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 75
M = 15
LDA = 25
INCX = 1
CALL STBSV (UPLO,TRANS,DIAG,N,M,A,LDA,X,INCX)

```

**Example 2** Perform REAL\*8 back substitution using the 75 by 75 nonunit-diagonal, upper-triangular real band matrix with bandwidth 15 that is stored in an array A whose dimensions are 25 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

INTEGER*8 N,M,LDA,INCX
REAL*8 A(25,100),X(100)
N = 75
M = 15
LDA = 25
INCX = 1
CALL STBSV ('UPPER', 'NONTRANS', 'NONUNIT', N,M,A,LDA,X,INCX)

```

**Purpose** Given an  $n$  by  $n$  upper- or lower-triangular matrix  $T$  stored in packed form as described in "Usage," and an  $n$ -vector  $x$ , these subprograms compute the matrix-vector products  $Tx$ ,  $T^T x$ , and  $T^*x$ , where  $T^T$  is the transpose of  $T$ , and  $T^*$  is the conjugate transpose of  $T$ . Specifically, these subprograms compute matrix-vector products of the forms

$$x - Tx, \quad x - T^T x, \quad \text{and} \quad x - T^*x.$$

**Storage** These subprograms allow you to reduce memory usage by supplying the matrix  $T$  in packed form, i.e., stored column by column in a 1-dimensional array. The following examples illustrate the packed storage of a triangular matrix.

**Upper triangular matrix.** If the upper triangular matrix  $T$  is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ 0 & 22 & 23 & 24 \\ 0 & 0 & 33 & 34 \\ 0 & 0 & 0 & 44 \end{array}$$

then the upper triangle of  $T$  is packed column by column into an array  $tp$  as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| tp(i) | 11 | 12 | 22 | 13 | 23 | 33 | 14 | 24 | 34 | 44 |

Upper-triangular matrix element  $t_{ij}$  is stored in array element  $tp(j \times (j-1)/2 + i)$ .

**Lower triangular matrix.** If the lower triangular matrix  $T$  is

$$\begin{array}{cccc} 11 & 0 & 0 & 0 \\ 21 & 22 & 0 & 0 \\ 31 & 32 & 33 & 0 \\ 41 & 42 & 43 & 44 \end{array}$$

then the lower triangle of  $T$  is packed column by column into an array  $tp$  as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| tp(i) | 11 | 21 | 31 | 41 | 22 | 32 | 42 | 33 | 43 | 44 |

Lower-triangular matrix element  $t_{ij}$  is stored in array element  $tp(j \times (2n - j - 1)/2 + i)$ .

**Usage**

**SCILIB:**

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,incx
REAL*8 tp(lentp),x(lenx)
CALL STPMV (uplo, trans, diag, n, tp, x, incx)
```

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,incx
COMPLEX*16 tp(lentp),x(lenx)
CALL CTPMV (uplo, trans, diag, n, tp, x, incx)
```

## Continued

|               |              |                                                                                                                                                                                                                                                                                                                              |
|---------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>  | <b>uplo</b>  | Upper/lower triangular option for $T$ :<br><br>'L' or 'l' $T$ is lower triangular<br>'U' or 'u' $T$ is upper triangular                                                                                                                                                                                                      |
|               | <b>trans</b> | Transposition option for $T$ :<br><br>'N' or 'n'    Compute $x \leftarrow Tx$<br>'T' or 't'    Compute $x \leftarrow T^T x$<br>'C' or 'c'    Compute $x \leftarrow T^* x$                                                                                                                                                    |
|               |              | where $T^T$ is the transpose of $T$ and $T^*$ is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.                                                                                                                                                                         |
|               | <b>diag</b>  | Specifies whether the matrix is unit triangular, i.e., $t_{ii} = 1$ , or not:<br><br>'N' or 'n'    The diagonal of $T$ is stored in the array<br>'U' or 'u'    The diagonal of $T$ consists of unstored ones                                                                                                                 |
|               |              | When <b>diag</b> is supplied as 'U' or 'u', the diagonal elements are not referenced.                                                                                                                                                                                                                                        |
|               | <b>n</b>     | Number of rows and columns in matrix $T$ , $n \geq 0$ . If $n = 0$ , the subprograms do not reference <b>tp</b> or <b>x</b> .                                                                                                                                                                                                |
|               | <b>tp</b>    | Array of length $\text{len}tp = n \times (n+1)/2$ containing the $n$ by $n$ triangular matrix $T$ , stored by columns in the packed form described above. Space must be left for the diagonal elements of $T$ even when <b>diag</b> is supplied as 'U' or 'u'.                                                               |
|               | <b>x</b>     | Array of length $\text{len}x = (n-1) \times  \text{incx}  + 1$ containing the input vector $x$ .                                                                                                                                                                                                                             |
|               | <b>incx</b>  | Increment for the array <b>x</b> , $\text{incx} \neq 0$ :<br><br>$\text{incx} > 0$ $x$ is stored forward in array <b>x</b> , i.e.,<br>$x_i$ is stored in $x((i-1) \times \text{incx} + 1)$ .<br>$\text{incx} < 0$ $x$ is stored backward in array <b>x</b> , i.e.,<br>$x_i$ is stored in $x((i-n) \times \text{incx} + 1)$ . |
|               |              | Use $\text{incx} = 1$ if the vector $x$ is stored contiguously in array <b>x</b> , i.e., if $x_i$ is stored in $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.                                                                                                                               |
| <b>Output</b> | <b>x</b>     | The updated $x$ vector replaces the input.                                                                                                                                                                                                                                                                                   |

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo ≠ 'L' or 'l' or 'U' or 'u',
trans ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag ≠ 'N' or 'n' or 'U' or 'u',
n < 0, and
incx = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $Tx$ , where  $T$  is a 9 by 9 unit-diagonal, lower-triangular real matrix stored in packed form in an array TP of dimension 55 and  $x$  is a real vector 9 elements long stored in an array X of dimension 10.

```

CHARACTER*1 UPLO,TRANS,DIAG
INTEGER*8 N,INCX
REAL*8 TP(55),X(10)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 9
INCX = 1
CALL STPMV (UPLO,TRANS,DIAG,N,TP,X,INCX)

```

**Example 2** Form the REAL\*8 matrix-vector product  $T^T x$ , where  $T$  is a 9 by 9 nonunit-diagonal, upper-triangular real matrix stored in packed form in an array TP of dimension 55 and  $x$  is a real vector 9 elements long stored in an array X of dimension 10.

```

INTEGER*8 N,INCX
REAL*8 TP(55),X(10)
N = 6
INCX = 1
CALL STPMV ('UPPER', 'TRANSPOSE', 'NONUNIT', N, TP, X, INCX)

```

**Solve Triangular System****STPSV/CTPSV****Purpose**

Given an  $n$  by  $n$  upper- or lower-triangular matrix  $T$  stored in packed form as described in "Usage," and an  $n$ -vector  $x$ , these subprograms overwrite  $x$  with the solution  $y$  to the system of linear equations  $Ty = x$ . This is the forward elimination or back substitution step of Gaussian elimination. Optionally,  $T$  may be replaced by  $T^T$ , the transpose of  $T$ , or by  $T^*$ , the conjugate transpose of  $T$ . Specifically, these subprograms compute

$$x - T^{-1}x, \quad x - T^{-T}x, \quad \text{and} \quad x - T^{-*}x$$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose of  $T$ .

These subprograms are more primitive than the LINPACK linear equation solvers. As such, they are intended to supplement but not replace them, serving instead as building blocks in constructing optimized linear algebra software. In fact, many of the LINPACK subprograms have been recoded to call these subprograms.

**Storage**

These subprograms allow you to reduce memory usage by supplying the matrix  $T$  in packed form, i.e., stored column by column in a 1-dimensional array. The following examples illustrate the packed storage of a triangular matrix.

**Upper triangular matrix.** If the upper triangular matrix  $T$  is

$$\begin{array}{cccc} 11 & 12 & 13 & 14 \\ 0 & 22 & 23 & 24 \\ 0 & 0 & 33 & 34 \\ 0 & 0 & 0 & 44 \end{array}$$

then the upper triangle of  $T$  is packed column by column into an array **tp** as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| tp(i) | 11 | 12 | 22 | 13 | 23 | 33 | 14 | 24 | 34 | 44 |

Upper-triangular matrix element  $t_{ij}$  is stored in array element **tp**( $j \times (j-1)/2 + i$ ).

**Lower triangular matrix.** If the lower triangular matrix  $T$  is

$$\begin{array}{cccc} 11 & 0 & 0 & 0 \\ 21 & 22 & 0 & 0 \\ 31 & 32 & 33 & 0 \\ 41 & 42 & 43 & 44 \end{array}$$

then the lower triangle of  $T$  is packed column by column into an array **tp** as follows:

| i     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|-------|----|----|----|----|----|----|----|----|----|----|
| tp(i) | 11 | 21 | 31 | 41 | 22 | 32 | 42 | 33 | 43 | 44 |

Lower-triangular matrix element  $t_{ij}$  is stored in array element **tp**( $j \times (2n-j-1)/2 + i$ ).

**Usage**      **SCILIB:**

```

CHARACTER*1 uplo,trans,diag
INTEGER*8 n,incx
REAL*8 tp(lentp),x(lenx)
CALL STPSV (uplo, trans, diag, n, tp, x, incx)

CHARACTER*1 uplo,trans,diag
INTEGER*8 n,incx
COMPLEX*16 tp(lentp),x(lenx)
CALL CTPSV (uplo, trans, diag, n, tp, x, incx)

```

**Input**      **uplo**      Upper/lower triangular option for  $T$ :

                 'L' or 'l'      Solve lower-triangular system (forward elimination)  
                 'U' or 'u'      Solve upper-triangular system (back substitution)

**trans**      Transposition option for  $T$ :

                 'N' or 'n'      Compute  $x = T^{-1}x$   
                 'T' or 't'      Compute  $x = T^{-T}x$   
                 'C' or 'c'      Compute  $x = T^{-*}x$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag**      Specifies whether the matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

                 'N' or 'n'      The diagonal of  $T$  is stored in the array  
                 'U' or 'u'      The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as 'U' or 'u', the diagonal elements are not referenced.

**n**      Number of rows and columns in matrix  $T$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **tp** or **x**.

**tp**      Array of length  $\text{lentp} = n \times (n+1)/2$  containing the  $n$  by  $n$  triangular matrix  $T$ , stored by columns in the packed form described above. Space must be left for the diagonal elements of  $T$  even when **diag** is supplied as 'U' or 'u'.

**x**      Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the right-hand-side  $n$ -vector  $x$ .

**incx**      Increment for the array **x**,  $\text{incx} \neq 0$ :

**incx** > 0       $x$  is stored forward in array **x**, i.e.,  
                  $x_i$  is stored in  $\mathbf{x}((i-1) \times \text{incx} + 1)$ .  
**incx** < 0       $x$  is stored backward in array **x**, i.e.,  
                  $x_i$  is stored in  $\mathbf{x}((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $\mathbf{x}(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**     **x**           The solution vector of the triangular system replaces the input.

**Notes**           These subprograms conform to specifications of the Level 2 BLAS.

The subprograms do not check for singularity of matrix  $T$ .  $T$  is singular if **diag** = 'N' or 'n' and some  $t_{ii} = 0$ . This condition will cause a division by zero to occur. Therefore, the program must detect singularity and take appropriate action to avoid a problem before calling any of these subprograms.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

```

uplo ≠ 'L' or 'l' or 'U' or 'u',
trans ≠ 'N' or 'n' or 'T' or 't' or 'C' or 'c',
diag ≠ 'N' or 'n' or 'U' or 'u',
n < 0, and
incx = 0.

```

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1**     Perform REAL\*8 forward elimination using a 75 by 75 unit-diagonal, lower-triangular real matrix stored in packed form in an array TP of dimension 5500, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

CHARACTER*1 UPLO, TRANS, DIAG
INTEGER*8 N, INCX
REAL*8 TP(5500), X(100)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 75
INCX = 1
CALL STPSV (UPLO, TRANS, DIAG, N, TP, X, INCX)

```

**Example 2**     Perform REAL\*8 back substitution using a 75 by 75 nonunit-diagonal, upper-triangular real matrix stored in packed form in an array TP of dimension 5500, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```

INTEGER*8 N, INCX
REAL*8 TP(5500), X(100)
N = 75
INCX = 1
CALL STPSV ('UPPER', 'NONTRANS', 'NONUNIT', N, TP, X, INCX)

```

**Purpose** Given a scalar  $\alpha$ , an  $m$  by  $n$  matrix  $B$ , and an upper- or lower-triangular matrix  $T$ , these subprograms compute either of the matrix-matrix products  $\alpha TB$  or  $\alpha BT$ . The size of  $T$ , either  $m$  by  $m$  or  $n$  by  $n$ , depends on which matrix product is requested. Optionally,  $T$  may be replaced by  $T^T$ , the transpose of  $T$ , or by  $T^*$ , the conjugate transpose of  $T$ . The resulting matrix product overwrites the input  $B$  matrix. Specifically, these subprograms compute matrix products of the forms

$$\begin{aligned} B - \alpha TB, \quad B - \alpha T^T B, \quad B - \alpha T^* B, \\ B - \alpha BT, \quad B - \alpha BT^T, \quad B - \alpha BT^*. \end{aligned}$$

**Usage** SCILIB:  
 CHARACTER\*1 side,uplo,transt,diag  
 INTEGER\*8 m,n,ldt,ldb  
 REAL\*8 alpha,t(ldt,\*),b(ldb,\*)  
 CALL STRMM (side, uplo, transt, diag, m, n, alpha, t, ldt, b, ldb)

CHARACTER\*1 side,uplo,transt,diag  
 INTEGER\*8 m,n,ldt,ldb  
 COMPLEX\*16 alpha,t(ldt,\*),b(ldb,\*)  
 CALL CTRMM (side, uplo, transt, diag, m, n, alpha, t, ldt, b, ldb)

**Input** **side** Specifies whether triangular matrix  $T$  is the left or right matrix operand:

'L' or 'l'  $T$  is the left matrix operand: for example,  $B - \alpha TB$   
 'R' or 'r'  $T$  is the right matrix operand: for example,  $B - \alpha BT$

**uplo** Upper/lower triangular option for  $T$ :

'L' or 'l'  $T$  is a lower-triangular matrix  
 'U' or 'u'  $T$  is an upper-triangular matrix

**transt** Transposition option for  $T$ :

'N' or 'n' Use matrix  $T$  directly  
 'T' or 't' Use  $T^T$ , the transpose of  $T$   
 'C' or 'c' Use  $T^*$ , the conjugate transpose of  $T$

In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag** Specifies whether the  $T$  matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

'N' or 'n' The diagonal of  $T$  is stored in the array  
 'U' or 'u' The diagonal of  $T$  consists of unstored

When **diag** is supplied as 'U' or 'u', the diagonal elements of  $T$  are not referenced.

**m** Number of rows in matrix  $B$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference **t** or **b**.

**n** Number of columns in matrix  $B$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **t** or **b**.

## Continued

- alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $B - 0$  without referencing **t**.
- t** Array whose upper or lower triangle, as specified by **uplo**, contains the upper- or lower-triangular matrix  $T$ , whose size is indicated by **side**:
- |            |                   |
|------------|-------------------|
| 'L' or 'l' | $T$ is $m$ by $m$ |
| 'R' or 'r' | $T$ is $n$ by $n$ |
- The other triangle of **t** is not referenced. Not used as input if **alpha** = 0.
- ldt** The leading dimension of array **t** as declared in the calling program unit, with  $\text{ldt} \geq \max(\text{the number of rows of } T, 1)$ .
- b** Array containing the  $m$  by  $n$  matrix  $B$ . Not used as input if **alpha** = 0.
- ldb** The leading dimension of array **b** as declared in the calling program unit, with  $\text{ldb} \geq \max(m, 1)$ .

**Output** **b** The indicated matrix product replaces the input.

**Notes** These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**side**  $\neq$  'L' or 'l' or 'R' or 'r',  
**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**transt**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**m**  $< 0$ ,  
**n**  $< 0$ ,  
**ldt** too small, and  
**ldb**  $< \max(m, 1)$ .

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved, for example, by coding the **transt** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix product  $TB$ , where  $T$  is a 6 by 6 nonunit-diagonal, upper-triangular real matrix stored in an array T whose dimensions are 10 by 10 and  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10. The matrix product will overwrite the input  $B$  matrix.

```

CHARACTER*1 SIDE, UPLO, TRANST, DIAG
INTEGER*8 M, N, LDT, LDB
REAL*8 ALPHA, T(10,10), B(10,10)
SIDE = 'L'
UPLO = 'U'
TRANST = 'N'
DIAG = 'N'
M = 6
N = 8
ALPHA = 1.0
LDT = 10
LDB = 10
CALL STRMM (SIDE, UPLO, TRANST, DIAG, M, N, ALPHA, T, LDT, B, LDB)

```

**Example 2** Form the REAL\*8 matrix product  $qBT^T$ , where  $q$  is a real scalar,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10, and  $T$  is a 8 by 8 unit-diagonal lower-triangular real matrix stored in an array T whose dimensions are 10 by 10. The matrix product will overwrite the input  $B$  matrix.

```

INTEGER*8 M, N, LDT, LDB
REAL*8 Q, T(10,10), B(10,10)
M = 6
N = 8
LDT = 10
LDB = 10
CALL STRMM ('RIGHT', 'LOWER', 'TRANS', 'UNIT', M, N, Q, T, LDT, B, LDB)

```

**Matrix-Vector Multiply****STRMV/CTRMV**

**Purpose** Given an  $n$  by  $n$  upper- or lower-triangular matrix  $T$ , and an  $n$ -vector  $x$ , these subprograms compute the matrix-vector products  $Tx$ ,  $T^T x$ , and  $T^*x$ , where  $T^T$  is the transpose of  $T$ , and  $T^*$  is the conjugate transpose of  $T$ . Specifically, these subprograms compute matrix-vector products of the forms

$$x = Tx, \quad x = T^T x, \quad \text{and} \quad x = T^*x.$$

**Usage** **SCILIB:**

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,ldt,incx
REAL*8 t(ldt,n),x(lenx)
CALL STRMV (uplo, trans, diag, n, t, ldt, x, incx)
```

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,ldt,incx
COMPLEX*16 t(ldt,n),x(lenx)
CALL CTRMV (uplo, trans, diag, n, t, ldt, x, incx)
```

**Input** **uplo** Upper/lower triangular option for  $T$ :

'L' or 'l'  $T$  is lower triangular  
'U' or 'u'  $T$  is upper triangular

The other triangle of the array  $t$  is not referenced.

**trans** Transposition option for  $T$ :

'N' or 'n' Compute  $x = Tx$   
'T' or 't' Compute  $x = T^T x$   
'C' or 'c' Compute  $x = T^*x$

where  $T^T$  is the transpose of  $T$  and  $T^*$  is the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag** Specifies whether the matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

'N' or 'n' The diagonal of  $T$  is stored in the array  
'U' or 'u' The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as 'U' or 'u', the diagonal elements are not referenced.

**n** Number of rows and columns in matrix  $T$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $t$  or  $x$ .

**t** Array containing the  $n$  by  $n$  triangular matrix  $T$ .

**ldt** The leading dimension of array  $t$  as declared in the calling program unit, with  $ldt \geq \max(n,1)$ .

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the input vector  $x$ .

**incx** Increment for the array  $x$ ,  $\text{incx} \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,

$x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,

$x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**  $x$  The updated  $x$  vector replaces the input.

**Notes** These subprograms conform to specifications of the Level 2 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**n** < 0,  
**ldt** < max(**n**,1), and  
**incx** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix-vector product  $Tx$ , where  $T$  is a 9 by 9 unit-diagonal lower-triangular real matrix stored in an array T whose dimensions are 10 by 10, and  $x$  is a real vector 9 elements long stored in an array X of dimension 10.

```
CHARACTER*1 UPLO,TRANS,DIAG
INTEGER*8 N,LDT,INCX
REAL*8 T(10,10),X(10)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 9
LDT = 10
INCX = 1
CALL STRMV (UPLO,TRANS,DIAG,N,T,LDT,X,INCX)
```

**Example 2** Form the REAL\*8 matrix-vector product  $T^T x$ , where  $T$  is a 9 by 9 nonunit-diagonal, upper-triangular real matrix stored in an array T whose dimensions are 10 by 10, and  $x$  is a real vector 9 elements long stored in an array X of dimension 10.

```
INTEGER*8 N,LDT
REAL*8 T(10,10),X(10)
N = 6
LDT = 10
CALL STRMV ('UPPER', 'TRANSPOSE', 'NONUNIT', N,T,LDT,X,1)
```

## Solve Triangular Systems

## STRSM/CTRSM

**Purpose** Given a scalar  $\alpha$ , an upper- or lower-triangular matrix  $T$ , and an  $m$  by  $n$  matrix  $B$ , these subprograms compute either of the matrix solutions  $\alpha T^{-1}B$  or  $\alpha BT^{-1}$ . The size of  $T$ , either  $m$  by  $m$  or  $n$  by  $n$ , depends on which matrix solution is requested. Optionally,  $T^{-1}$  may be replaced by  $T^{-T}$ , the inverse of the transpose of  $T$ , or by  $T^{-*}$ , the inverse of the conjugate transpose of  $T$ . The resulting matrix solution overwrites the input  $B$  matrix. Specifically, these subprograms compute matrix solutions of the forms

$$\begin{array}{lll} B - \alpha T^{-1}B, & B - \alpha T^{-T}B, & B - \alpha T^{-*}B, \\ B - \alpha BT^{-1}, & B - \alpha BT^{-T}, & B - \alpha BT^{-*}. \end{array}$$

**Usage** **SCILIB:**

```

CHARACTER*1 side,uplo,transt,diag
INTEGER*8 m,n,ldt,ldb
REAL*8 alpha,t(ldt,*),b(ldb,*)
CALL STRSM (side, uplo, transt, diag, m, n, alpha, t, ldt, b, ldb)

CHARACTER*1 side,uplo,transt,diag
INTEGER*8 m,n,ldt,ldb
COMPLEX*16 alpha,t(ldt,*),b(ldb,*)
CALL CTRSM (side, uplo, transt, diag, m, n, alpha, t, ldt, b, ldb)

```

**Input**

**side** Specifies whether triangular matrix  $T$  is the left or right matrix operand:

‘L’ or ‘l’  $T$  is the left matrix operand: for example,  $B - \alpha T^{-1}B$   
‘R’ or ‘r’  $T$  is the right matrix operand: for example,  $B - \alpha BT^{-1}$

**uplo** Upper/lower triangular option for  $T$ :

‘L’ or ‘l’  $T$  is a lower-triangular matrix  
‘U’ or ‘u’  $T$  is an upper-triangular matrix

**transt** Transposition option for  $T$ :

‘N’ or ‘n’ Use matrix  $T^{-1}$   
‘T’ or ‘t’ Use  $T^{-T}$ , the inverse of the transpose of  $T$   
‘C’ or ‘c’ Use  $T^{-*}$ , the inverse of the conjugate transpose of  $T$

In the real subprograms, ‘C’ and ‘c’ have the same meaning as ‘T’ and ‘t’.

**diag** Specifies whether the  $T$  matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

‘N’ or ‘n’ The diagonal of  $T$  is stored in the array  
‘U’ or ‘u’ The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as ‘U’ or ‘u’, the diagonal elements of  $T$  are not referenced.

**m** Number of rows in matrix  $B$ ,  $m \geq 0$ . If  $m = 0$ , the subprograms do not reference **t** or **b**.

**n** Number of columns in matrix  $B$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference **t** or **b**.

- alpha** The scalar  $\alpha$ . If **alpha** = 0, the subprograms compute  $B - 0$  without referencing **t**.
- t** Array whose upper or lower triangle, as specified by **uplo**, contains the upper- or lower-triangular matrix  $T$ , whose size is indicated by **side**:
- |            |                   |
|------------|-------------------|
| 'L' or 'l' | $T$ is $m$ by $m$ |
| 'R' or 'r' | $T$ is $n$ by $n$ |
- The other triangle of **t** is not referenced. Not used as input if **alpha** = 0.
- ldt** The leading dimension of array **t** as declared in the calling program unit, with  $\text{ldt} \geq \max(\text{the number of rows of } T, 1)$ .
- b** Array containing the  $m$  by  $n$  matrix  $B$ . Not used as input if **alpha** = 0.
- ldb** The leading dimension of array **b** as declared in the calling program unit, with  $\text{ldb} \geq \max(m, 1)$ .

**Output** **b** The indicated matrix solution replaces the input.

**Notes** These subprograms conform to specifications of the Level 3 BLAS.

If an error in the arguments is detected, the subprograms call the error handler XERBLA, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**side**  $\neq$  'L' or 'l' or 'R' or 'r',  
**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**transt**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**m**  $< 0$ ,  
**n**  $< 0$ ,  
**ldt** too small, and  
**ldb**  $< \max(m, 1)$ .

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the **CALL** statement may be improved, for example, by coding the **transt** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Form the REAL\*8 matrix solution  $T^{-1}B$ , where  $T$  is a 6 by 6 nonunit-diagonal, upper-triangular real matrix stored in an array T whose dimensions are 10 by 10 and  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10. The matrix solution will overwrite the input  $B$  matrix.

```

CHARACTER*1 SIDE,UPLO,TRANST,DIAG
INTEGER*8 M,N,LDT,LDB
REAL*8 ALPHA,T(10,10),B(10,10)
SIDE = 'L'
UPLO = 'U'
TRANST = 'N'
DIAG = 'N'
M = 6
N = 8
ALPHA = 1.0
LDT = 10
LDB = 10
CALL STRSM (SIDE,UPLO,TRANST,DIAG,M,N,ALPHA,T,LDT,B,LDB)

```

**Example 2** Form the REAL\*8 matrix solution  $qBT^{-T}$ , where  $q$  is a real scalar,  $B$  is a 6 by 8 real matrix stored in an array B of dimension 10 by 10, and  $T$  is a 8 by 8 unit-diagonal lower-triangular real matrix stored in an array T whose dimensions are 10 by 10. The matrix solution will overwrite the input  $B$  matrix.

```

INTEGER*8 M,N,LDT,LDB
REAL*8 Q,T(10,10),B(10,10)
M = 6
N = 8
LDT = 10
LDB = 10
CALL STRSM ('RIGHT','LOWER','TRANS','UNIT',M,N,Q,T,LDT,B,LDB)

```

**Purpose** Given an  $n$  by  $n$  upper- or lower-triangular matrix  $T$ , and an  $n$ -vector  $x$ , these subprograms overwrite  $x$  with the solution  $y$  to the system of linear equations  $Ty = x$ . This is the forward elimination or back substitution step of Gaussian elimination. Optionally,  $T$  may be replaced by  $T^T$ , the transpose of  $T$ , or by  $T^*$ , the conjugate transpose of  $T$ . Specifically, these subprograms compute

$$x - T^{-1}x, \quad x - T^{-T}x, \quad \text{and} \quad x - T^{-*}x$$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose of  $T$ .

These subprograms are more primitive than the LINPACK linear equation solvers. As such, they are intended to supplement but not replace them, serving instead as building blocks in constructing optimized linear algebra software. In fact, many of the LINPACK subprograms have been recoded to call these subprograms.

**Usage****SCILIB:**

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,ldt,incx
REAL*8 t(ldt,n),x(lenx)
CALL STRSV (uplo, trans, diag, n, t, ldt, x, incx)
```

```
CHARACTER*1 uplo,trans,diag
INTEGER*8 n,ldt,incx
COMPLEX*16 t(ldt,n),x(lenx)
CALL CTRSV (uplo, trans, diag, n, t, ldt, x, incx)
```

**Input**

**uplo** Upper/lower triangular option for  $T$ :

'L' or 'l' Solve lower-triangular system (forward elimination)  
'U' or 'u' Solve upper-triangular system (back substitution)

The other triangle of the array  $t$  is not referenced.

**trans** Transposition option for  $T$ :

'N' or 'n' Compute  $x - T^{-1}x$   
'T' or 't' Compute  $x - T^{-T}x$   
'C' or 'c' Compute  $x - T^{-*}x$

where  $T^{-T}$  is the inverse of the transpose of  $T$ , and  $T^{-*}$  is the inverse of the conjugate transpose. In the real subprograms, 'C' and 'c' have the same meaning as 'T' and 't'.

**diag** Specifies whether the matrix is unit triangular, i.e.,  $t_{ii} = 1$ , or not:

'N' or 'n' The diagonal of  $T$  is stored in the array  
'U' or 'u' The diagonal of  $T$  consists of unstored ones

When **diag** is supplied as 'U' or 'u', the diagonal elements are not referenced.

**n** Number of rows and columns in matrix  $T$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $t$  or  $x$ .

## Continued

- t**        Array containing the  $n$  by  $n$  triangular matrix  $T$ .
- ldt**      The leading dimension of array **t** as declared in the calling program unit, with  $\text{ldt} \geq \max(\mathbf{n}, 1)$ .
- x**        Array of length  $\text{lenx} = (\mathbf{n}-1) \times |\text{incx}| + 1$  containing the right-hand-side  $n$ -vector  $x$ .
- incx**     Increment for the array **x**,  $\text{incx} \neq 0$ :
- incx** > 0     $x$  is stored forward in array **x**, i.e.,  
 $x_i$  is stored in  $\mathbf{x}((i-1) \times \text{incx} + 1)$ .
- incx** < 0     $x$  is stored backward in array **x**, i.e.,  
 $x_i$  is stored in  $\mathbf{x}((i-\mathbf{n}) \times \text{incx} + 1)$ .

Use  $\text{incx} = 1$  if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $\mathbf{x}(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**    **x**        The solution vector of the triangular system replaces the input.

**Notes**     These subprograms conform to specifications of Level 2 BLAS.

The subprograms do not check for singularity of matrix  $T$ .  $T$  is singular if **diag** = 'N' or 'n' and some  $t_{ii} = 0$ . This condition will cause a division by zero to occur. Therefore, the program must detect singularity and take appropriate action to avoid a problem before calling any of these subprograms.

If an error in the arguments is detected, the subprograms call error handler XERBLA, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERBLA (refer to the end of this chapter), may be replaced with a user-supplied version to change the error procedure. Error conditions are

**uplo**  $\neq$  'L' or 'l' or 'U' or 'u',  
**trans**  $\neq$  'N' or 'n' or 'T' or 't' or 'C' or 'c',  
**diag**  $\neq$  'N' or 'n' or 'U' or 'u',  
**n** < 0,  
**ldt** <  $\max(\mathbf{n}, 1)$ , and  
**incx** = 0.

Actual character arguments in a subroutine call may be longer than the corresponding dummy arguments. Therefore, readability of the CALL statement may be improved by coding the **trans** argument as 'NORMAL' or 'NONTRANS' for 'N', 'TRANSPOSE' for 'T', or 'CTRANS' for 'C'. Refer to "Example 2."

**Example 1** Perform REAL\*8 forward elimination using the 75 by 75 unit-diagonal lower-triangular real matrix stored in an array A whose dimensions are 100 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```
CHARACTER*1 UPLO, TRANS, DIAG
INTEGER*8 N, LDA, INCX
REAL*8 A(100,100), X(100)
UPLO = 'L'
TRANS = 'N'
DIAG = 'U'
N = 75
LDA = 100
INCX = 1
CALL STRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)
```

**Example 2** Perform REAL\*8 back substitution using the 75 by 75 nonunit-diagonal, upper-triangular real matrix stored in an array A whose dimensions are 100 by 100, and  $x$  is a real vector 75 elements long stored in an array X of dimension 100.

```
INTEGER*8 N, LDA, INCX
REAL*8 A(100,100), X(100)
N = 75
LDA = 100
INCX = 1
CALL STRSV ('UPPER', 'NONTRANS', 'NONUNIT', N, A, LDA, X, INCX)
```

## Vector-Matrix Multiply and Add

SXMPY

- Purpose** This subprogram computes the matrix-vector product  $xA$ , and adds the result to another vector  $y$ , where  $A$  is an  $m$  by  $n$  matrix,  $x$  is an  $m$ -dimensional row vector, and  $y$  is an  $n$ -dimensional row vector. SCILIB subprogram SGEMV allows more general storage of  $x$  and  $y$  and also admits scaling, subtraction, and transposing  $A$ .
- Usage** SCILIB:  
**INTEGER\*8**  $n,m,lda,incx,incy$   
**REAL\*8**  $a(lda,n),x(lenx),y(leny)$   
**CALL SXMPY** ( $n,incy,y,m,incx,x,lda,a$ )
- Input**
- n** Number of columns in matrix  $A$  and length of row vector  $y$ ,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $x$ , or  $y$ .
- incy** Storage increment between successive elements of vector  $y$  in array  $y$ .  $y_i$  is stored in  $y((i-1) \times incy + 1)$ . Use **incy** = 1 if the vector  $y$  is stored contiguously in array  $y$ , i.e., if  $y_i$  is stored in  $y(i)$ .
- y** Array of length  $leny = (n-1) \times incy + 1$  containing the row vector  $y$ .
- m** Number of rows in matrix  $A$  and length of row vector  $x$ ,  $m \geq 0$ . If  $m = 0$ , the subprogram does not reference  $a$ ,  $x$ , or  $y$ .
- incx** Storage increment between successive elements of vector  $x$  in array  $x$ .  $x_i$  is stored in  $x((i-1) \times incx + 1)$ . Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ .
- x** Array of length  $lenx = (m-1) \times incx + 1$  containing the  $n$ -vector  $x$ .
- lda** The leading dimension of array  $a$  as declared in the calling program unit.
- a** Array containing the  $m$  by  $n$  matrix  $A$ .
- Output**
- y** The updated  $y$  vector replaces the input.
- Notes** Cray research, Inc. has declared this subprogram obsolete in release 6.0 of the UNICOS Math and Scientific Library.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

$m < 0,$   
 $n < 0,$   
 $lda < m,$   
 $incx \leq 0,$  and  
 $incy \leq 0.$

**FORTTRAN Equivalent** Except for the argument error checking, the following FORTRAN subroutine is equivalent to SXMPY.

```

SUBROUTINE SXMPY (N, INCY, Y, M, INCX, X, LDA, A)
INTEGER*8 M, N, LDA, INCX, INCY
REAL*8 A(LDA, N), X(*), Y(*)
DO 120 J = 1, M
 DO 110 I = 1, N
 Y((I-1)*INCY+1) = ! Y(I) =
1 Y((I-1)*INCY+1) + ! Y(I) +
2 X((J-1)*INCX+1) * A(J, I) ! X(J) * A(J, I)
110 CONTINUE
120 CONTINUE
RETURN
END

```

**Example**

Form the REAL\*8 vector-matrix product  $y = y + xA$ , where  $A$  is a 9 by 6 real matrix stored in an array  $A$  whose dimensions are 10 by 10,  $x$  is a real vector 9 elements long stored in row 3 of an array  $X$  of dimension 10 by 10, and  $y$  is a real vector 6 elements long stored in row 7 of an array  $Y$  of dimension 10 by 10.

```

INTEGER*8 M, N, LDA, INCX, INCY
REAL*8 A(10,10), X(10,10), Y(10,10)
M = 9
N = 6
LDA = 10
INCX = 10
INCY = 10
CALL SXMPY (N, INCY, Y(7,1), M, INCX, X(3,1), LDA, A)

```

**Error Handler****XERBLA**

**Purpose** This subprogram is the error handler for many of the subprograms in this chapter, as indicated in the "Notes" section in the applicable subprogram descriptions. As supplied in SCILIB, XERBLA prints the following error message:

```

* SCILIB BLAS routine name called with invalid value of argument number iarg *

```

where *name* is the name of the subprogram in which the error was detected, and *iarg* is the argument number of the offending argument. For example, in SGEMV, *trans* is argument number 1 and *m* is argument number 2. XERBLA then executes a FORTRAN STOP statement.

You may supply a version of XERBLA that alters this action. All SCILIB subprograms that call XERBLA have a RETURN statement after the CALL statement, so your version could perhaps set a flag in a common block and RETURN. The flag could be tested in the program unit that calls the SCILIB subprogram.

**Usage** SCILIB:  
CHARACTER\*6 *name*  
INTEGER\*8 *iarg*  
CALL XERBLA (*name*, *iarg*)

**Input** *name* The name of the subprogram in which the error was detected.  
*iarg* The number of the argument that was found to be in error.

**Notes** This subprogram conforms to specifications of the Level 2 BLAS.



# Linear Equations

## Overview

This chapter describes the state-of-the-art LINPACK software library included with SCILIB. The most important subprograms in this library have been upgraded by incorporating the Level 2 and Level 3 BLAS and other algorithmic changes. Although SCILIB includes all LINPACK subprograms, only those routines optimized for use on CONVEX supercomputers are described in this chapter. Table 4-5 at the end of this chapter lists the LINPACK subprograms that are included in SCILIB but are not documented in the *SCILIB User's Guide*. You may find information for these routines in the *LINPACK User's Guide* included in the SCILIB documentation set. More subprograms will be enhanced in future releases of SCILIB. Thus, it is to your advantage to use LINPACK to realize the enhancements that have already been made as well as those that will be made in the future.

This chapter explains how to use SCILIB subprograms to solve systems of linear equations. The operations covered are:

- solution of a system of linear equations
- calculation of the inverse of a matrix
- evaluation of the determinant of a matrix

These operations are performed for a variety of types of matrices, including:

- real and complex general dense matrices
- real and complex general band matrices
- real and complex positive definite dense matrices
- real and complex positive definite band matrices
- real and complex general tridiagonal matrices
- real and complex positive definite tridiagonal matrices

Refer to Chapter 6 for software to solve sparse symmetric linear equations.

## Chapter Objectives

After you read this chapter you will:

- be familiar with the LINPACK subroutine naming convention
- understand the role of the condition number in solving linear equations
- know how to compute the determinant or inverse of a matrix
- know when not to compute the determinant or inverse of a matrix
- be able to locate documentation for LINPACK subroutines not documented here
- know how to use the described subprograms

## What You Need to Know to Use These Subprograms

### Subroutine Naming Convention

LINPACK uses a subroutine naming convention that encodes the function of each subroutine into its name. LINPACK subprogram names consist of five letters in the form TXXYY.

The first letter in the naming convention indicates one of the four FORTRAN data types, as shown in Table 4-1.

**Table 4-1: LINPACK Naming Convention — Data Type**

| T | Data Type                |
|---|--------------------------|
| S | Single Precision REAL    |
| C | Single Precision COMPLEX |

Table 4-2 shows the next two letters in the naming convention which indicate the form of the matrix or its decomposition.

**Table 4-2: LINPACK Naming Convention — Form or Decomposition**

| XX | Form or Decomposition               |
|----|-------------------------------------|
| GE | General                             |
| GB | General band                        |
| PO | Positive definite                   |
| PB | Positive definite band              |
| PP | Positive definite packed            |
| SI | Symmetric indefinite                |
| SP | Symmetric indefinite packed         |
| HI | Hermitian indefinite                |
| HP | Hermitian indefinite packed         |
| TR | Triangular                          |
| GT | General tridiagonal                 |
| PT | Positive definite tridiagonal       |
| CH | Cholesky decomposition              |
| QR | Orthogonal-triangular decomposition |
| SV | Singular value decomposition        |

Table 4-3 lists the final two letters in the naming convention which indicate the computation of a particular subroutine.

**Table 4-3: LINPACK Naming Convention — Computation**

| YY | Subroutine Computation                    |
|----|-------------------------------------------|
| FA | Factor                                    |
| CO | Factor and estimate condition             |
| SL | Solve                                     |
| DI | Determinant and/or inverse and/or inertia |
| DC | Decompose                                 |
| UD | Update                                    |
| DD | Downdate                                  |
| EX | Exchange                                  |

For example, SGBCO factors a general band (GB) matrix and estimates its condition number (CO) using the single precision REAL data type (S). CGEFA calculates the factorization (FA) of a general dense matrix (GE) using the single precision COMPLEX data type (C).

Table 4-4 shows the valid combinations of T, XX, and YY. Each line indicates the allowable T prefixes and YY suffixes for a particular root name XX.

**Table 4-4: LINPACK Naming Convention — Subprogram Names**

| Valid T | XX | Valid YY |    |    |    |    |    |
|---------|----|----------|----|----|----|----|----|
| S       | C  | GE       | CO | FA | SL | DI |    |
| S       | C  | GB       | CO | FA | SL | DI |    |
| S       | C  | PO       | CO | FA | SL | DI |    |
| S       | C  | PB       | CO | FA | SL | DI |    |
| S       | C  | PP       | CO | FA | SL | DI |    |
| S       | C  | SI       | CO | FA | SL | DI |    |
| S       | C  | SP       | CO | FA | SL | DI |    |
|         | C  | HI       | CO | FA | SL | DI |    |
|         | C  | HP       | CO | FA | SL | DI |    |
| S       | C  | TR       | CO |    | SL | DI |    |
| S       | C  | GT       |    |    | SL |    |    |
| S       | C  | PT       |    |    | SL |    |    |
| S       | C  | CH       | DC |    | UD | DD | EX |
| S       | C  | QR       | DC | SL |    |    |    |
| S       | C  | SV       | DC |    |    |    |    |

LINPACK is organized so that it is usually necessary to call two subprograms to perform the above operations. One subprogram is called to process the matrix and another is called to process a particular right-hand side. This division of labor significantly reduces computer time when there is a sequence of problems involving the same matrix but different right-hand sides. It also allows you the flexibility to choose between subprograms that are fast but use a less reliable, elementary test for singularity and subprograms that are slightly slower but use a significantly more reliable test involving an estimate of the condition number of the coefficient matrix.

## Condition Number

The condition number,  $\kappa(A)$ , of the coefficient matrix  $A$  measures the sensitivity of the solution  $x$  of the system of linear equations  $Ax = b$  to errors in the matrix  $A$  and the right-hand side  $b$ . If  $\delta A$  and  $\delta b$  represent the errors in  $A$  and  $b$ , respectively, and if  $\| \cdot \|$  represents any vector norm and its subordinate matrix norm, the error  $\delta x$  in  $x$  that results from solving  $(A + \delta A)(x + \delta x) = b + \delta b$  instead of  $Ax = b$  is bound by

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \|A^{-1}\| \|\delta A\|} \left( \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

A standard result of numerical analysis shows that the roundoff error introduced by the solution process may be modeled by taking  $\|\delta A\|/\|A\|$  and  $\|\delta b\|/\|b\|$  to be small multiples of the computer's machine epsilon. Computational singularity of  $A$  results in  $\kappa(A) = \infty$ . A more common situation occurs when  $A$  is not numerically singular but is ill-conditioned. When a matrix is ill-conditioned,  $\kappa(A)$  is large, so small errors in the matrix and right-hand side and small roundoff errors introduced during the solution process itself are magnified greatly in the solution.

Since  $1 < \kappa(A) \leq \infty$ , it is more convenient to compute the reciprocal condition number,  $1/\kappa(A)$ , than  $\kappa(A)$  itself. The reciprocal condition number has the interpretation that if  $1/\kappa(A)$  approximately equals  $10^{-d}$ , elements of  $x$  can be expected to have  $d$  fewer significant digits of accuracy than the elements of  $A$  or  $b$ . Consequently, if errors in the coefficient matrix and right-hand side exceed  $1/\kappa(A)$ , or if  $1/\kappa(A)$  is negligible compared to 1.0, then  $x$  may have no significant digits at all.

## Determinant and Inverse

Subprograms for computing the determinant and inverse of a matrix are provided, although it is almost never necessary to compute either one. While papers and reference books extensively use the notation " $\det(A) \neq 0$ " to mean " $A$  is nonsingular," SCILIB includes both more efficient and more reliable subprograms for detecting singularity. Similarly, references frequently use " $A^{-1}b$ " to mean "the solution  $x$  of the system of linear equations  $Ax = b$ ." Again, it is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right-hand-side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors just as efficiently—and more accurately—than by matrix multiplication by the inverse.

## Supplemental Reading

Forsythe, G., and C.B. Moler. 1967. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Inc., New Jersey.

## Subprogram Descriptions

|                                                                                                |      |
|------------------------------------------------------------------------------------------------|------|
| Invert Matrix or Solve Linear Equations<br>MINV .....                                          | 4-6  |
| Solve Symmetric Toeplitz Linear Equations<br>OPFILT .....                                      | 4-8  |
| Factor a General Band Matrix and Estimate its Condition Number<br>SGBCO, CGBCO .....           | 4-9  |
| Determinant of a General Band Matrix<br>SGBDI, CGBDI .....                                     | 4-12 |
| Factor a General Band Matrix<br>SGBFA, CGBFA .....                                             | 4-14 |
| Solve Linear Equations with a General Band Matrix<br>SGBSL, CGBSL .....                        | 4-17 |
| Factor a General Matrix and Estimate its Condition Number<br>SGECO, CGECO .....                | 4-19 |
| Determinant and Inverse of a General Matrix<br>SGEDI, CGEDI .....                              | 4-21 |
| Factor a General Matrix<br>SGEFA, CGEFA .....                                                  | 4-24 |
| Solve Linear Equations with a General Matrix<br>SGESL, CGESL .....                             | 4-26 |
| Solve Linear Equations with a Tridiagonal Matrix<br>SGTSL, CGTSL .....                         | 4-29 |
| Factor a Positive Definite Band Matrix and Estimate its Condition Number<br>SPBCO, CPBCO ..... | 4-31 |
| Determinant of a Positive Definite Band Matrix<br>SPBDI, CPBDI .....                           | 4-34 |
| Cholesky Factorization of a Positive Definite Band Matrix<br>SPBFA, CPBFA .....                | 4-36 |
| Solve Linear Equations with a Positive Definite Band Matrix<br>SPBSL, CPBSL .....              | 4-39 |
| Factor a Positive Definite Matrix and Estimate its Condition Number<br>SPOCO, CPOCO .....      | 4-41 |
| Determinant and Inverse of a Positive Definite Matrix<br>SPODI, CPODI .....                    | 4-43 |
| Cholesky Factorization of a Positive Definite Matrix<br>SPOFA, CPOFA .....                     | 4-46 |
| Solve Linear Equations with a Positive Definite Matrix<br>SPOSL, CPOSL .....                   | 4-48 |
| Solve Linear Equations with a Positive Definite Tridiagonal Matrix<br>SPTSL, CPTSL .....       | 4-50 |

**Purpose** Given a general dense  $n$  by  $n$  matrix  $A$ , this subprogram evaluates the determinant of  $A$ , optionally solves one or more systems of linear equations  $Ax_j=b_j$ , and optionally computes  $A^{-1}$ .

Computational singularity of  $A$  results in  $\det(A)=0$ . The partial product of pivot elements is computed as  $A$  is factored, and  $A$  is declared singular if the absolute value of the partial product ever fails to exceed a user-supplied tolerance. If this condition is detected during factorization, the computation is terminated and the "small" partial determinant is returned to indicate its occurrence. A more common situation, however, is that  $A$  is not numerically singular but happens to be ill-conditioned. When a matrix is ill-conditioned, small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself, are magnified greatly in the solution.

Although singular matrices are characterized by having zero determinants, a small nonzero determinant is unrelated to computational singularity or ill-conditioning. Therefore, the stopping criteria is artificial. Hence, we recommend that you not use this subprogram as it may give a completely unreliable indication of the singularity of  $A$ . The SCILIB subprograms SGECO, SGEDI, and SGESL may be combined to perform the same functions as MINV while providing a more reliable indication of singularity.

**Usage** SCILIB:  
**INTEGER\*8**  $n, ldab, m, job$   
**REAL\*8**  $ab(ldab, n+m), work(2*n), det, tol$   
**CALL** MINV ( $ab, n, ldab, work, det, tol, m, job$ )

**Input**

**ab** Array containing the  $n$  by  $n$  matrix  $A$  in columns 1 through  $n$ , and  $m \geq 0$  right-hand-side vectors  $b_j$  in columns  $n+1$  through  $n+m$ .

**n** The order of matrix  $A$ ,  $n > 0$ .

**ldab** The leading dimension of array **ab** as declared in the calling program unit, with  $ldab \geq n$ .

**tol** Lower limit for the partial product of pivot elements,  $tol > 0$ .  $A$  is considered singular if the magnitude of the partial product of pivot elements ever fails to exceed **tol**.

**m** Number of right-hand-side vectors,  $m \geq 0$ . If  $m = 0$ , no right-hand sides are solved.

**job** Option flag:  
**job** = 0 do not compute  $A^{-1}$   
**job**  $\neq$  0 compute  $A^{-1}$

**Working Storage**

**work** An array of size  $2n$ , used for work space.



**Purpose** Given a real symmetric  $n$  by  $n$  Toeplitz matrix  $A$  and a right-hand-side vector  $b$ , this subprogram solves the system of linear equations  $Ax=b$  by means of the Weiner-Levinson algorithm. A matrix  $A$  is a symmetric Toeplitz matrix if its elements  $a_{ij}$  are given by  $a_{ij} = q_{|i-j|+1}$ . The following illustrates a 3 by 3 symmetric Toeplitz matrix.

$$\begin{matrix} q_1 & q_2 & q_3 \\ q_2 & q_1 & q_2 \\ q_3 & q_2 & q_1 \end{matrix}$$

OPFILT is designed for use only on matrices which do not require pivoting to maintain numerical stability.

**Usage** **SCILIB:**  
**INTEGER\*8** n  
**REAL\*8** x(n),b(n),work(2\*n),q(n)  
**CALL OPFILT** (n, x, b, work, q)

**Input** n The order of matrix  $A$ ,  $n \geq 0$ .  
b The right-hand-side vector  $b$ .  
q The vector that generates the  $A$  matrix via the relationship  $a_{ij} = q_{|i-j|+1}$ .

**Working Storage** work An array of size  $2n$ , used for work space.

**Output** x The solution vector  $x$ .

**Notes** If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$n < 0.$$

An overflow or divide by zero may be produced if the matrix is not suitable for solution with the Weiner-Levinson algorithm.

**Example** Solve the 6 by 6 REAL\*8 symmetric Toeplitz matrix system  $Ax = b$  where  $A$  is represented by array Q of dimension 10, and  $x$  and  $b$  are stored in arrays X and B, also of dimension 10 by 10, respectively.

```

INTEGER*8 N
REAL*8 X(10),B(10),WORK(20),Q(10)
N = 6
CALL OPFILT (N,X,B,WORK,Q)

```

**Purpose**

These subprograms compute the triangular factorization and estimate the condition number of a general nonsymmetric  $n$  by  $n$  band matrix  $A$  stored in a two-dimensional array. A band matrix is a matrix whose nonzero elements all are near the principal diagonal. Specifically,  $a_{ij} = 0$  if  $i-j > ml$  or  $j-i > mu$  for some integers  $ml$  and  $mu$ . The smallest such  $ml$  and  $mu$  for a given matrix are called the lower and upper bandwidths, respectively, and  $m = ml + mu + 1$  is the total bandwidth. The subprograms for band matrices use less storage than the subprograms for full matrices if  $2ml + mu < n$ .

Tridiagonal matrices are the special case  $ml = mu = 1$ . They can be handled more efficiently by the subprograms SGTSL, CGTSL. SCILIB also contains subprograms designed to handle positive definite band matrices. These subprograms are documented elsewhere in this chapter.

Specifically, given  $A$ , these subprograms determine an upper-triangular band matrix  $U$ , and a matrix  $L$  that is the product of elementary lower-triangular band matrices and permutation matrices such that

$$A = LU$$

and compute an estimate of  $\kappa(A)$ , the condition number of  $A$ . Refer to "Condition Number" in the introduction to this chapter for a discussion of  $\kappa(A)$ . When a matrix is ill-conditioned,  $\kappa(A)$  is large, so small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself, are magnified greatly in the solution.

Since  $1 < \kappa(A) \leq \infty$ , these subprograms actually compute the reciprocal condition number,  $1/\kappa(A)$ . The reciprocal condition number has the interpretation that if  $1/\kappa(A)$  approximately equals  $10^{-d}$ , elements of  $x$  can be expected to have  $d$  fewer significant digits of accuracy than the elements of  $A$  or  $b$ . Consequently, if errors in the coefficient matrix and right-hand side exceed  $1/\kappa(A)$ , or if  $1/\kappa(A)$  is negligible compared to 1.0, then  $x$  may have no significant digits at all.

A set of companion subprograms computes the triangular factorization of a general band matrix without estimating its condition number. These companion subprograms are faster but provide a less reliable indication of singularity.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $L(Ux) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(L)\det(U)$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve | Determinant |
|------------|--------------------|--------|-------|-------------|
| REAL*8     | SGBCO              | SGBFA  | SGBSL | SGBDI       |
| COMPLEX*16 | CGBCO              | CGBFA  | CGBSL | CGBDI       |

The inverse of  $A$  will usually be a full  $n$  by  $n$  matrix that cannot be stored in the band storage of  $A$ . Therefore, no direct provision is made for computing  $A^{-1}$ . Calculations formulated in terms of matrix inverses are invariably more efficient when expressed in terms of the solution of sets of linear equations.

**Usage** The following example illustrates the storage of general band matrices. If the original matrix is

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 14 | 0  | 0  | 0  | 0  | 0  |
| 21 | 22 | 23 | 24 | 25 | 0  | 0  | 0  | 0  |
| 31 | 32 | 33 | 34 | 35 | 36 | 0  | 0  | 0  |
| 0  | 42 | 43 | 44 | 45 | 46 | 47 | 0  | 0  |
| 0  | 0  | 53 | 54 | 55 | 56 | 57 | 58 | 0  |
| 0  | 0  | 0  | 64 | 65 | 66 | 67 | 68 | 69 |
| 0  | 0  | 0  | 0  | 75 | 76 | 77 | 78 | 79 |
| 0  | 0  | 0  | 0  | 0  | 86 | 87 | 88 | 89 |
| 0  | 0  | 0  | 0  | 0  | 0  | 97 | 98 | 99 |

then its order is  $n = 9$ , and its lower and upper bandwidths are  $ml = 2$  and  $mu = 3$ .

When Gaussian elimination is performed on a general band matrix, pivoting introduces nonzero elements outside the band.  $L$  can be stored with a lower bandwidth of  $ml$ , but  $U$  requires an upper bandwidth of  $ml + mu$ . You must, therefore, provide storage for the extra  $ml$  diagonals. This is done by presenting the original matrix to the subprogram in an array large enough to satisfy the additional storage requirements. Thus, for the above matrix,  $A$  is given in an array  $\mathbf{a}$  with at least  $2ml + mu + 1 = 8$  rows and  $n = 9$  columns as follows:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| *  | *  | *  | *  | *  | +  | +  | +  | +  |
| *  | *  | *  | *  | +  | +  | +  | +  | +  |
| *  | *  | *  | 14 | 25 | 36 | 47 | 58 | 69 |
| *  | *  | 13 | 24 | 35 | 46 | 57 | 68 | 79 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| 21 | 32 | 43 | 54 | 65 | 76 | 87 | 98 | *  |
| 31 | 42 | 53 | 64 | 75 | 86 | 97 | *  | *  |

The asterisks in the  $ml + mu$  by  $ml + mu$  triangle at the upper-left corner and in the  $mu$  by  $mu$  triangle at the lower-right corner represent elements of  $\mathbf{a}$  that are not referenced, and the plus signs in the first  $ml$  rows indicate elements that may be filled in during the factorization. Thus, if  $a_{ij}$  is an element within the band of  $A$ , then it is stored in  $\mathbf{a}(i - j + ml + mu + 1, j)$ . Therefore, the columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in the rows of  $\mathbf{a}$ , such that the principal diagonal is stored in row  $ml + mu + 1$  of  $\mathbf{a}$ .

**SCILIB:**

```
INTEGER*8 lda,n,ml,mu,ipvt(n)
REAL*8 a(lda,n),rcond,work(n)
CALL SGBCO (a, lda, n, ml, mu, ipvt, rcond, work)
```

```
INTEGER*8 lda,n,ml,mu,ipvt(n)
COMPLEX*16 a(lda,n),work(n)
REAL*8 rcond
CALL CGBCO (a, lda, n, ml, mu, ipvt, rcond, work)
```

|                        |              |                                                                                                                                                                                                                                                                                                                                                      |
|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>           | <b>a</b>     | Array containing the $n$ by $n$ band matrix $A$ in the compressed form described above. If $a_{ij}$ is in the band, it is stored in $a(i-j+ml+mu+1)$ . The columns of $A$ are stored in the columns of $a$ , and the diagonals of $A$ are stored in rows $ml+1$ through $2 \times ml+mu+1$ . The first $ml$ rows are used for work space and output. |
|                        | <b>lda</b>   | The leading dimension of array $a$ as declared in the calling program unit, with $lda \geq 2 \times ml+mu+1$ .                                                                                                                                                                                                                                       |
|                        | <b>n</b>     | The order of matrix $A$ , $n > 0$ .                                                                                                                                                                                                                                                                                                                  |
|                        | <b>ml</b>    | The lower bandwidth of $A$ , i.e., the number of nonzero diagonals below the principal diagonal in the band, $0 \leq ml < n$ .                                                                                                                                                                                                                       |
|                        | <b>mu</b>    | The upper bandwidth of $A$ , i.e., the number of nonzero diagonals above the principal diagonal in the band, $0 \leq mu < n$ . These subprograms are more efficient if $ml \leq mu$ . This usually can be arranged since the factors used by these subprograms can be used to solve either $Ax = b$ or $A^*x = b$ .                                  |
| <b>Working Storage</b> | <b>work</b>  | An array of size $n$ , used for work space.                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>          | <b>a</b>     | The triangular factors replace the input matrix. $a$ must be preserved between the condition number estimation call and any solve or determinant call.                                                                                                                                                                                               |
|                        | <b>ipvt</b>  | The pivot information necessary to construct the permutations in the lower-triangular factor, $L$ . $ipvt$ must be preserved between the condition number estimation call and any solve or determinant call.                                                                                                                                         |
|                        | <b>rcond</b> | An estimate of the reciprocal condition number, $1/\kappa(A)$ . If $rcond$ is small enough so that the logical expression                                                                                                                                                                                                                            |

$$1.0 + rcond .EQ. 1.0$$

is true, then  $A$  can be regarded as singular to working precision.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example** Factor and estimate the reciprocal condition number of the 9 by 9 REAL\*8 general band matrix  $A$  whose lower bandwidth is 2 and whose upper bandwidth is 3.  $A$  is stored as illustrated above in array  $A$  whose dimensions are 8 by 10.

```

INTEGER*8 LDA,N,ML,MU,IPVT(10)
REAL*8 A(8,10),RCOND,WORK(10)
LDA = 8
N = 9
ML = 2
MU = 3
CALL SGBCO (A,LDA,N,ML,MU,IPVT,RCOND,WORK)
IF (1.0 + RCOND .EQ. 1.0) THEN
 handle singular matrix
END IF

```

**Purpose** Given the triangular factorization of a general  $n$  by  $n$  band matrix  $A$ , these subprograms evaluate the determinant of  $A$ . No provision is made to compute  $A^{-1}$  since it will usually be a full  $n$  by  $n$  matrix that cannot be stored in the band storage of  $A$ . Moreover, it is almost never necessary to compute the inverse of a matrix. Mathematical references frequently use " $A^{-1}b$ " to mean "the solution  $x$  of the system of linear equations  $Ax = b$ ." It is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right-hand-side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors just as efficiently, and more accurately, than by matrix multiplication by the inverse.

Specifically, given an  $n$  by  $n$  upper-triangular band matrix  $U$ , and a matrix  $L$  which is the product of elementary lower-triangular band matrices and permutation matrices such that

$$A = LU,$$

the subprograms compute

$$\det(A) = \det(L)\det(U).$$

The triangular factorization of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly faster. The other not only computes the factorization, but also estimates the condition number of the matrix. This takes a little more time, but is considerably more reliable. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Determinant |
|------------|--------------------|--------|-------------|
| REAL*8     | SGBCO              | SGBFA  | SGBDI       |
| COMPLEX*16 | CGBCO              | CGBFA  | CGBDI       |

The companion subprograms are documented elsewhere in this chapter.

**Usage**

**SCILIB:**

```
INTEGER*8 lda,n,ml,mu,ipvt(n)
REAL*8 a(lda,n),det(2)
CALL SGBDI (a, lda, n, ml, mu, ipvt, det)
```

```
INTEGER*8 lda,n,ml,mu,ipvt(n)
COMPLEX*16 a(lda,n),det(2)
CALL CGBDI (a, lda, n, ml, mu, ipvt, det)
```

**Input**

**a** Array containing the triangular factors of the  $n$  by  $n$  general band matrix  $A$  as computed by the companion factorization or condition number estimation subprogram. **a** must have been preserved between the factorization or condition number call and the determinant call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq 2 \times ml + mu + 1$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

## Continued

- ml** The lower bandwidth of  $A$ , i.e., the number of nonzero diagonals below the principal diagonal in the band,  $0 \leq \text{ml} < \mathbf{n}$ .
- mu** The upper bandwidth of  $A$ , i.e., the number of nonzero diagonals above the principal diagonal in the band,  $0 \leq \text{mu} < \mathbf{n}$ .
- ipvt** The pivot information necessary to construct the permutations in the lower-triangular factor,  $L$ . **ipvt** must have been preserved between the factorization or condition number call and the determinant call.
- Output** **det** The determinant of  $A$ , in the form  $\det(A) = \det(1) \times 10^{\det(2)}$ . This expression may underflow or overflow if evaluated; on the CONVEX supercomputer, underflows automatically flush to zero, but overflows normally terminate execution. For REAL\*8 and COMPLEX\*16, overflow cannot occur if  $\det(2) \leq 306$ . If evaluation is safe, an efficient way to do it is with the statement

$$\det(A) = \det(1) * 10.0 ** \text{INT}(\det(2))$$

The value stored in **det(2)** is an integer in REAL or COMPLEX form. **det(1)** is normalized so that either  $\det(1) = 0$  or  $1 \leq |Re(\det(1))| + |Im(\det(1))| < 10$ , where  $Re(z)$  and  $Im(z)$  are the real and imaginary parts of  $z$ ;  $Re(z) = z$  and  $Im(z) = 0$  if  $z$  is real.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**It is almost never necessary to compute the determinant of a matrix.** While it is true that papers and reference books make extensive use of the notation " $\det(A) \neq 0$ " to mean " $A$  is nonsingular," SCILIB includes more efficient and more reliable subprograms for detecting singularity.

**Example** Compute the determinant of a 9 by 9 REAL\*8 general band matrix  $A$  whose lower bandwidth is 2 and whose upper bandwidth is 3.  $A$  is stored in array  $A$  whose dimensions are 8 by 10. The less reliable, but slightly faster, factorization subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,ML,MU,IPVT(10),IER
REAL*8 A(8,10),DET(2),DETA
LDA = 8
N = 9
ML = 2
MU = 3
CALL SGBFA (A,LDA,N,ML,MU,IPVT,IER)
IF (IER .EQ. 0) THEN
 CALL SGBDI (A,LDA,N,ML,MU,IPVT,DET)
 IF (DET(1) .EQ. 0.0) THEN
 DETA = 0.0
 ELSE IF (DET(2) .LE. 306) THEN
 DETA = DET(1) * 10.0 ** INT(DET(2))
 ELSE
 the determinant of A is too large to evaluate
 without overflow
 END IF
ELSE
 DETA = 0.0
END IF

```

**Purpose** These subprograms compute the triangular factorization of a general nonsymmetric  $n$  by  $n$  band matrix  $A$  stored in a two-dimensional array. A band matrix is a matrix whose nonzero elements all are near the principal diagonal. Specifically,  $a_{ij} = 0$  if  $i-j > ml$  or  $j-i > mu$  for some integers  $ml$  and  $mu$ . The smallest such  $ml$  and  $mu$  for a given matrix are called the lower and upper bandwidths, respectively, and  $m = ml + mu + 1$  is the total bandwidth. The subprograms for band matrices use less storage than the subprograms for full matrices if  $2ml + mu < n$ .

Tridiagonal matrices are the special case  $ml = mu = 1$ . They can be handled more efficiently by the subprograms SGTSL or CGTSL. SCILIB also contains subprograms designed to handle positive definite band matrices. These subprograms are documented elsewhere in this chapter.

Specifically, given  $A$ , these subprograms determine an upper-triangular band matrix  $U$ , and a matrix  $L$  that is the product of elementary lower triangular band matrices and permutation matrices such that

$$A = LU.$$

Computational singularity of  $A$  results in one or more zero diagonal elements of  $U$ . This condition is detected during factorization, and a status response is returned to indicate its occurrence. A more common situation, however, is that  $A$  is not numerically singular but happens to be ill-conditioned. When a matrix is ill-conditioned, small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself, are magnified greatly in the solution. A set of companion subprograms computes the triangular factorization of a general band matrix and also estimates its condition number. These companion subprograms provide a more reliable indication of singularity. The small amount of additional time they require is usually worthwhile, especially when developing a program or encountering stability or convergence problems.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $L(Ux) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(L)\det(U)$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Factor | Estimate Condition | Solve | Determinant |
|------------|--------|--------------------|-------|-------------|
| REAL*8     | SGBFA  | SGBCO              | SGBSL | SGBDI       |
| COMPLEX*16 | CGBFA  | CGBCO              | CGBSL | CGBDI       |

The companion subprograms are documented elsewhere in this chapter.

The inverse of  $A$  will usually be a full  $n$  by  $n$  matrix that cannot be stored in the band storage of  $A$ . Therefore, no direct provision is made for computing  $A^{-1}$ . Calculations formulated in terms of matrix inverses are invariably more efficient when expressed in terms of the solution of sets of linear equations.

**Usage** The following example illustrates the storage of general band matrices. If the original matrix is

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 14 | 0  | 0  | 0  | 0  | 0  |
| 21 | 22 | 23 | 24 | 25 | 0  | 0  | 0  | 0  |
| 31 | 32 | 33 | 34 | 35 | 36 | 0  | 0  | 0  |
| 0  | 42 | 43 | 44 | 45 | 46 | 47 | 0  | 0  |
| 0  | 0  | 53 | 54 | 55 | 56 | 57 | 58 | 0  |
| 0  | 0  | 0  | 64 | 65 | 66 | 67 | 68 | 69 |
| 0  | 0  | 0  | 0  | 75 | 76 | 77 | 78 | 79 |
| 0  | 0  | 0  | 0  | 0  | 86 | 87 | 88 | 89 |
| 0  | 0  | 0  | 0  | 0  | 0  | 97 | 98 | 99 |

its order is  $n = 9$  and its lower and upper bandwidths are  $ml = 2$  and  $mu = 3$ .

When Gaussian elimination is performed on a general band matrix, pivoting introduces nonzero elements outside the band.  $L$  can be stored with a lower bandwidth of  $ml$ , but  $U$  requires an upper bandwidth of  $ml + mu$ . You must, therefore, provide storage for the extra  $ml$  diagonals. This is done by presenting the original matrix to the subprogram in an array large enough to satisfy the additional storage requirements. Thus, for the above matrix,  $A$  is given in an array  $\mathbf{a}$  with at least  $2ml + mu + 1 = 8$  rows and  $n = 9$  columns as follows:

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|
| *  | *  | *  | *  | *  | +  | +  | +  | +  |
| *  | *  | *  | *  | +  | +  | +  | +  | +  |
| *  | *  | *  | 14 | 25 | 36 | 47 | 58 | 69 |
| *  | *  | 13 | 24 | 35 | 46 | 57 | 68 | 79 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
| 21 | 32 | 43 | 54 | 65 | 76 | 87 | 98 | *  |
| 31 | 42 | 53 | 64 | 75 | 86 | 97 | *  | *  |

The asterisks in the  $ml + mu$  by  $ml + mu$  triangle at the upper-left corner and in the  $mu$  by  $mu$  triangle at the lower-right corner represent elements of  $\mathbf{a}$  that are not referenced, and the plus signs in the first  $ml$  rows indicate elements that may be filled in during the factorization. Thus, if  $a_{ij}$  is an element within the band of  $A$ , then it is stored in  $\mathbf{a}(i - j + ml + mu + 1, j)$ . Therefore, the columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in the rows of  $\mathbf{a}$ , such that the principal diagonal is stored in row  $ml + mu + 1$  of  $\mathbf{a}$ .

**SCILIB:**

```
INTEGER*8 lda,n,ml,mu,ipvt(n),ier
REAL*8 a(lda,n)
CALL SGBFA (a, lda, n, ml, mu, ipvt, ier)
```

```
INTEGER*8 lda,n,ml,mu,ipvt(n),ier
COMPLEX*16 a(lda,n)
CALL CGBFA (a, lda, n, ml, mu, ipvt, ier)
```

**Input** **a** Array containing the  $n$  by  $n$  band matrix  $A$  in the compressed form described above. If  $a_{ij}$  is in the band, it is stored in  $\mathbf{a}(i - j + ml + mu + 1)$ . Columns of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of  $A$  are stored in rows  $ml + 1$  through  $2 \times ml + mu + 1$ . The first  $ml$  rows are used for work space and output.

|               |             |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | <b>lda</b>  | The leading dimension of array <b>a</b> as declared in the calling program unit, with $lda \geq 2 \times ml + mu + 1$ .                                                                                                                                                                                                                                                                                                            |
|               | <b>n</b>    | The order of matrix $A$ , $n > 0$ .                                                                                                                                                                                                                                                                                                                                                                                                |
|               | <b>ml</b>   | The lower bandwidth of $A$ , i.e., the number of nonzero diagonals below the principal diagonal in the band, $0 \leq ml < n$ .                                                                                                                                                                                                                                                                                                     |
|               | <b>mu</b>   | The upper bandwidth of $A$ , i.e., the number of nonzero diagonals above the principal diagonal in the band, $0 \leq mu < n$ . These subprograms are more efficient if $ml \leq mu$ . This usually can be arranged because factors used by these subprograms can be used to solve either $Ax = b$ or $A^*x = b$ .                                                                                                                  |
| <b>Output</b> | <b>a</b>    | The triangular factors replace the input matrix. <b>a</b> must be preserved between the factorization call and any solve or determinant call.                                                                                                                                                                                                                                                                                      |
|               | <b>ipvt</b> | The pivot information necessary to construct the permutations in the lower triangular factor, $L$ . <b>ipvt</b> must be preserved between the factorization call and any solve or determinant call.                                                                                                                                                                                                                                |
|               | <b>ier</b>  | Status response:<br><br><b>ier</b> = 0      Normal return.<br><b>ier</b> = $k \neq 0$ if $u_{kk} = 0$ . ( $u_{kk}$ is the $k$ th element on the diagonal of upper triangular matrix $U$ ). Technically, this is not an error condition for these subprograms, but it does indicate that $A$ is computationally singular and that a division by zero will occur if the factorization is used to solve a system of linear equations. |

**Notes**      These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example**    Factor the 9 by 9 REAL\*8 general band matrix  $A$  whose lower bandwidth is 2 and whose upper bandwidth is 3.  $A$  is stored, as illustrated above, in array  $A$  whose dimensions are 8 by 10.

```

INTEGER*8 LDA,N,ML,MU,IPVT(10),IER
REAL*8 A(8,10)
LDA = 8
N = 9
ML = 2
MU = 3
CALL SGBFA (A,LDA,N,ML,MU,IPVT,IER)
IF (IER .NE. 0) THEN
 handle singular matrix
END IF

```

**Solve Band Linear Equations****SGBSL/CGBSL**

**Purpose** Given the triangular factorization of a general  $n$  by  $n$  band matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . Optionally, these subprograms will solve the system  $A^*x = b$ , where  $A^*$  is the conjugate transpose of  $A$  (the conjugate transpose of a real matrix is simply the transpose). Specifically, given an  $n$  by  $n$  upper-triangular band matrix  $U$ , and a matrix  $L$  that is the product of elementary lower-triangular band matrices and permutation matrices such that

$$A = LU,$$

and an  $n$ -vector  $b$ , to find  $x$  satisfying  $Ax = b$ , the subprograms successively solve

$$Lw = b$$

and

$$Ux = w,$$

while to solve  $A^*x = b$ , the subprograms successively solve

$$U^*v = b$$

and

$$L^*x = v.$$

Triangular factors of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster. The other not only computes the factorization, but also estimates the condition number of the matrix. This takes a little more time, but is considerably more reliable. Names of companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve |
|------------|--------------------|--------|-------|
| REAL*8     | SGBCO              | SGBFA  | SGBSL |
| COMPLEX*16 | CGBCO              | CGBFA  | CGBSL |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,ml,mu,ipvt(n),job
REAL*8 a(lda,n),b(n)
CALL SGBSL (a, lda, n, ml, mu, ipvt, b, job)
```

```
INTEGER*8 lda,n,ml,mu,ipvt(n),job
COMPLEX*16 a(lda,n),b(n)
CALL CGBSL (a, lda, n, ml, mu, ipvt, b, job)
```

**Input**

**a** Array containing the triangular factors of the  $n$  by  $n$  general band matrix  $A$  as computed by the companion factorization or condition number estimation subprogram. **a** must have been preserved between the factorization or condition number call and the solve call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq 2 \times ml + mu + 1$ .

- n** The order of matrix  $A$ ,  $n \geq 0$ .
- ml** The lower bandwidth of  $A$ , i.e., the number of nonzero diagonals below the principal diagonal in the band,  $0 \leq \mathbf{ml} < \mathbf{n}$ .
- mu** The upper bandwidth of  $A$ , i.e., the number of nonzero diagonals above the principal diagonal in the band,  $0 \leq \mathbf{mu} < \mathbf{n}$ .
- ipvt** Pivot information necessary to construct permutations in the lower-triangular factor,  $L$ . **ipvt** must have been preserved between the factorization or condition number estimation call and the solve call.
- b** The right-hand-side vector  $b$ .
- job** Option flag:
- job** = 0 solve  $Ax = b$   
**job**  $\neq$  0 solve  $A^*x = b$

**Output** **b** The solution vector  $x$  overwrites the right-hand-side vector  $b$ .

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example** Solve a system of linear equations  $Ax = b$ , where  $A$  is a 9 by 9 REAL\*8 general band matrix whose lower bandwidth is 2 and whose upper bandwidth is 3.  $A$  is stored in array  $A$  whose dimensions are 8 by 10.  $b$  is a vector 9 elements long stored in an array  $B$  of dimension 10. The more robust, but slightly slower, condition number estimation subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,ML,MU,IPVT(10),JOB
REAL*8 A(8,10),B(10),RCOND,WORK(10)
LDA = 8
N = 9
ML = 2
MU = 3
JOB = 0
CALL SGBCO (A,LDA,N,ML,MU,IPVT,RCOND,WORK)
IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SGBSL (A,LDA,N,ML,MU,IPVT,B,JOB)
ELSE
 handle singular matrix
END IF

```

**Purpose** These subprograms compute the triangular factorization and estimate the condition number of a general dense  $n$  by  $n$  matrix  $A$ . Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  permutation matrix  $P$ , an  $n$  by  $n$  unit lower-triangular matrix  $L$ , and an  $n$  by  $n$  upper-triangular matrix  $U$ , such that

$$PA = LU$$

and compute an estimate of  $\kappa(A)$ , the condition number of  $A$ . Refer to "Condition Number" in the introduction to this chapter for a discussion of  $\kappa(A)$ . When a matrix is ill-conditioned,  $\kappa(A)$  is large, so small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself, are magnified greatly in the solution.

Since  $1 < \kappa(A) \leq \infty$ , these subprograms actually compute the reciprocal condition number,  $1/\kappa(A)$ . The reciprocal condition number has the interpretation that if  $1/\kappa(A)$  approximately equals  $10^{-d}$ , elements of  $x$  can be expected to have  $d$  fewer significant digits of accuracy than the elements of  $A$  or  $b$ . Consequently, if errors in the coefficient matrix and right-hand side exceed  $1/\kappa(A)$ , or if  $1/\kappa(A)$  is negligible compared to 1.0, then  $x$  may have no significant digits at all.

A set of companion subprograms computes the triangular factorization of a matrix without estimating its condition number. These companion subprograms are faster but provide a less reliable indication of singularity.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $L(Ux) = Pb$ . The determinant of  $A$  can be computed as  $\det(A) = \det(P) \times \det(L) \times \det(U)$ . The inverse of  $A$  may be formed as  $A^{-1} = U^{-1}L^{-1}P$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve | Determinant or inverse |
|------------|--------------------|--------|-------|------------------------|
| REAL*8     | SGECO              | SGEFA  | SGESL | SGEDI                  |
| COMPLEX*16 | CGECO              | CGEFA  | CGESL | CGEDI                  |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,ipvt(n)
REAL*8 a(lda,n),rcond,work(n)
CALL SGECO (a, lda, n, ipvt, rcond, work)
```

```
INTEGER*8 lda,n,ipvt(n)
COMPLEX*16 a(lda,n),work(n)
REAL*8 rcond
CALL CGECO (a, lda, n, ipvt, rcond, work)
```

**Input**

**a** Array containing the  $n$  by  $n$  matrix  $A$ .

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

|                        |              |                                                                                                                                                                                                                                                                                                         |
|------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Working storage</b> | <b>work</b>  | An array of size <b>n</b> , used for work space.                                                                                                                                                                                                                                                        |
| <b>Output</b>          | <b>a</b>     | The triangular factors replace the input matrix: the strict lower triangle of <b>a</b> contains the strict lower triangle of $L$ and the upper triangle of <b>a</b> contains $U$ . <b>a</b> must be preserved between the condition number estimation call and any solve, determinant, or inverse call. |
|                        | <b>ipvt</b>  | The pivot information necessary to construct the permutation matrix $P$ . <b>ipvt</b> must be preserved between the condition number estimation call and any solve, determinant, or inverse call.                                                                                                       |
|                        | <b>rcond</b> | An estimate of the reciprocal condition, $1/\kappa(A)$ . If <b>rcond</b> is small enough so that the logical expression                                                                                                                                                                                 |

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then  $A$  can be regarded as singular to working precision. If **rcond** is exactly zero, then the companion subprograms for solving and computing the inverse may divide by zero.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

The triangular factors are stored in a different format from the format used by the standard LINPACK subprograms, but are compatible with the SCILIB factorization, solve, and determinant and inverse subprograms.

**Example** Factor the 6 by 6 REAL\*8 matrix  $A$  stored in array  $A$  whose dimensions are 10 by 10 and estimate its reciprocal condition number.

```

INTEGER*8 LDA,N,IPVT(10)
REAL*8 A(10,10),RCOND,WORK(10)
LDA = 10
N = 6
CALL SGECO (A,LDA,N,IPVT,RCOND,WORK)
IF (1.0 + RCOND .EQ. 1.0) THEN
 handle singular matrix
END IF

```

**Determinant and Inverse****SGEDI/CGEDI**

**Purpose** Given the triangular factorization of a general dense  $n$  by  $n$  coefficient matrix  $A$ , these subprograms evaluate the determinant of  $A$  and/or compute  $A^{-1}$ . Specifically, given an  $n$  by  $n$  permutation matrix  $P$ , an  $n$  by  $n$  unit lower-triangular matrix  $L$ , and a nonsingular  $n$  by  $n$  upper-triangular matrix  $U$ , such that

$$PA = LU,$$

the subprograms compute

$$\det(A) = \det(P) \times \det(L) \times \det(U)$$

and/or

$$A^{-1} = U^{-1}L^{-1}P.$$

The triangular factors of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes the factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable, especially when  $A^{-1}$  is desired. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Determinant or inverse |
|------------|--------------------|--------|------------------------|
| REAL*8     | SGECO              | SGEFA  | SGEDI                  |
| COMPLEX*16 | CGECO              | CGEFA  | CGEDI                  |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,ipvt(n),job
REAL*8 a(lda,n),det(2),work(n)
CALL SGEDI (a, lda, n, ipvt, det, work, job)
```

```
INTEGER*8 lda,n,ipvt(n),job
COMPLEX*16 a(lda,n),det(2),work(n)
CALL CGEDI (a, lda, n, ipvt, det, work, job)
```

**Input**

**a** Array containing the triangular factors  $L$  and  $U$  of the  $n$  by  $n$  coefficient matrix  $A$  as computed by the companion factorization or condition number estimation subprogram. **a** must have been preserved between the factorization or condition number call and the determinant or inverse call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**ipvt** The pivot information necessary to construct the permutation matrix  $P$  as computed by the companion factorization or condition number estimation subprogram. **ipvt** must have been preserved between the factorization or condition number call and the determinant or inverse call.

**job** Option flag:

```
job = 1 compute only A^{-1}
job = 10 compute only $\det(A)$
job = 11 compute both A^{-1} and $\det(A)$
```

|                        |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Working storage</b> | <b>work</b> | An array of size $n$ , used for work space if $A^{-1}$ is requested.                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Output</b>          | <b>a</b>    | Unchanged if $A^{-1}$ is not requested. Otherwise, $A^{-1}$ overwrites the triangular factors of the coefficient matrix.                                                                                                                                                                                                                                                                                                                                               |
|                        | <b>det</b>  | Not referenced if the determinant is not requested. Otherwise, the determinant of $A$ , in the form $\det(A) = \det(1) \times 10^{\det(2)}$ . This expression may underflow or overflow if evaluated; on the CONVEX supercomputer, underflows automatically flush to zero, but overflows normally terminate execution. For REAL*8 and COMPLEX*16, overflow cannot occur if $\det(2) \leq 306$ . If evaluation is safe, an efficient way to do it is with the statement |

$$\det(A) = \det(1) * 10.0 ** \text{INT}(\det(2))$$

Refer to "Example 2."

The value stored in **det(2)** is an integer in REAL or COMPLEX form. **det(1)** is normalized so that either  $\det(1) = 0$  or  $1 \leq |Re(\det(1))| + |Im(\det(1))| < 10$ , where  $Re(z)$  and  $Im(z)$  are the real and imaginary parts of  $z$ ;  $Re(z) = z$  and  $Im(z) = 0$  if  $z$  is real.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

It is almost never necessary to compute either the determinant or the inverse of a matrix. While papers and reference books extensively use the notation " $\det(A) \neq 0$ " to mean "A is nonsingular," SCILIB includes both more efficient and more reliable subprograms for detecting singularity. Similarly, references frequently use " $A^{-1}b$ " to mean "the solution  $x$  of the system of linear equations  $Ax = b$ ." Again, it is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right-hand-side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors just as efficiently, and more accurately, than by matrix multiplication by the inverse.

**Example 1.** Compute only the inverse of a 6 by 6 REAL\*8 matrix  $A$  stored in array A whose dimensions are 10 by 10. The more robust, but slightly slower condition number estimation subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,IPVT(10),JOB
REAL*8 A(10,10),DET(2),RCOND,WORK(10)
LDA = 10
N = 6
JOB = 1
CALL SGECO (A,LDA,N,IPVT,RCOND,WORK)
IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SGEDI (A,LDA,N,IPVT,DET,WORK,JOB)
ELSE
 handle singular matrix
END IF

```

If the coefficient matrix  $A$  is determined to be nonsingular,  $A^{-1}$  overwrites the coefficient matrix  $A$  in array A.

Continued

**Example 2** Compute only the determinant of a 6 by 6 REAL\*8 matrix *A* stored in array *A* whose dimensions are 10 by 10. The less reliable, but slightly faster factorization subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,IPVT(10),IER, JOB
REAL*8 A(10,10),DET(2),DETA,WORK(10)
LDA = 10
N = 6
JOB = 10
CALL SGEFA (A,LDA,N,IPVT,IER)
IF (IER .EQ. 0) THEN
 CALL SGEDI (A,LDA,N,IPVT,DET,WORK, JOB)
 IF (DET(1) .EQ. 0.0) THEN
 DETA = 0.0
 ELSE IF (DET(2) .LE. 306) THEN
 DETA = DET(1) * 10.0 ** INT(DET(2))
 ELSE
 the determinant of A is too large to evaluate
 without overflow
 END IF
ELSE
 DETA = 0.0
END IF

```

**Purpose** These subprograms compute the triangular factorization of a general dense  $n$  by  $n$  matrix  $A$ . Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  permutation matrix  $P$ , an  $n$  by  $n$  unit lower-triangular matrix  $L$ , and an  $n$  by  $n$  upper-triangular matrix  $U$ , such that

$$PA = LU.$$

Computational singularity of  $A$  results in one or more zero diagonal elements of  $U$ . This condition is detected during factorization, and a status response is returned to indicate its occurrence. A more common situation, however, is that  $A$  is not numerically singular but happens to be ill-conditioned. When a matrix is ill-conditioned, small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself, are magnified greatly in the solution. A set of companion subprograms computes the triangular factorization of a matrix and also estimates its condition number. These companion subprograms provide a more reliable indication of singularity. The small amount of additional time they require is usually worthwhile, especially when developing a program or encountering stability or convergence problems.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $L(Ux) = Pb$ . The determinant of  $A$  can be computed as  $\det(A) = \det(P) \times \det(L) \times \det(U)$ . The inverse of  $A$  may be formed as  $A^{-1} = U^{-1}L^{-1}P$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type | Factor | Estimate Condition | Solve | Determinant or inverse |
|-----------|--------|--------------------|-------|------------------------|
| REAL*4    | SGEFA  | SGECO              | SGESL | SGEDI                  |
| COMPLEX*8 | CGEFA  | CGECO              | CGESL | CGEDI                  |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,ipvt(n),ier
REAL*8 a(lda,n)
CALL SGEFA (a, lda, n, ipvt, ier)
```

```
INTEGER*8 lda,n,ipvt(n),ier
COMPLEX*16 a(lda,n)
CALL CGEFA (a, lda, n, ipvt, ier)
```

**Input**

**a** Array containing the  $n$  by  $n$  matrix  $A$ .

**lda** The leading dimension of array  $a$  as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**Output**

**a** The triangular factors replace the input matrix; the strict lower triangle of  $a$  contains the strict lower triangle of  $L$  and the upper triangle of  $a$  contains  $U$ .  $a$  must be preserved between the factorization call and any solve, determinant, or inverse call.

**ipvt** The pivot information necessary to construct the permutation matrix  $P$ .  $ipvt$  must be preserved between the factorization call and any solve, determinant, or inverse call.

**ier** Status response:

**ier = 0** Normal return.  
**ier = k ≠ 0** if  $u_{kk} = 0$ . ( $u_{kk}$  is the  $k$ th element on the diagonal of upper triangular matrix  $U$ ). Technically, this is not an error condition for these subprograms, but it does indicate that  $A$  is computationally singular and that a division by zero will occur if the factorization is used to solve a system of linear equations or to compute the matrix inverse.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

The triangular factors are stored in a different format from the format used by the standard LINPACK subprograms, but are compatible with the SCILIB condition number estimation, solve, and determinant and inverse subprograms.

**Example** Factor the 6 by 6 REAL\*8 matrix  $A$  stored in array  $A$  whose dimensions are 10 by 10.

```

INTEGER*8 LDA,N,IPVT(10),IER
REAL*8 A(10,10)
LDA = 10
N = 6
CALL SGEFA (A,LDA,N,IPVT,IER)
IF (IER .NE. 0) THEN
 handle singular matrix
END IF

```

**Purpose** Given the triangular factorization of a general dense  $n$  by  $n$  coefficient matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . Optionally, these subprograms will solve the system  $A^*x = b$ , where  $A^*$  is the conjugate transpose of  $A$  (the conjugate transpose of a real matrix is simply the transpose). Specifically, given an  $n$  by  $n$  permutation matrix  $P$ , an  $n$  by  $n$  unit lower-triangular matrix  $L$ , and a nonsingular  $n$  by  $n$  upper-triangular matrix  $U$ , such that

$$PA = LU,$$

and an  $n$ -vector  $b$ , to find  $x$  satisfying  $Ax = b$ , the subprograms compute

$$v = Pb,$$

then successively solve

$$Lw = v$$

and

$$Ux = w,$$

To solve  $A^*x = b$ , the subprograms successively solve

$$U^*v = b$$

and

$$L^*w = v,$$

and then compute

$$x = P^*w.$$

The triangular factors of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes the factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve |
|------------|--------------------|--------|-------|
| REAL*8     | SGECO              | SGEFA  | SGESL |
| COMPLEX*16 | CGECO              | CGEFA  | CGESL |

The companion subprograms are documented elsewhere in this chapter.

#### Usage

##### SCILIB:

```
INTEGER*8 lda,n,ipvt(n),job
REAL*8 a(lda,n),b(n)
CALL SGESL (a, lda, n, ipvt, b, job)
```

```
INTEGER*8 lda,n,ipvt(n),job
COMPLEX*16 a(lda,n),b(n)
CALL CGESL (a, lda, n, ipvt, b, job)
```

## Continued

|               |             |                                                                                                                                                                                                                                                                                             |
|---------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>  | <b>a</b>    | Array containing the triangular factors $L$ and $U$ of the $n$ by $n$ coefficient matrix $A$ as computed by the companion factorization or condition number estimation subprogram. <b>a</b> must have been preserved between the factorization or condition number call and the solve call. |
|               | <b>lda</b>  | The leading dimension of array <b>a</b> as declared in the calling program unit, with $lda \geq \max(n,1)$ .                                                                                                                                                                                |
|               | <b>n</b>    | The order of matrix $A$ , $n \geq 0$ .                                                                                                                                                                                                                                                      |
|               | <b>ipvt</b> | The pivot information necessary to construct the permutation matrix $P$ as computed by the companion factorization or condition number estimation subprogram. <b>ipvt</b> must have been preserved between the factorization or condition number call and the solve call.                   |
|               | <b>b</b>    | The right-hand-side vector $b$ .                                                                                                                                                                                                                                                            |
|               | <b>job</b>  | Option flag:<br><br>$job = 0$ solve $Ax = b$<br>$job \neq 0$ solve $A^*x = b$                                                                                                                                                                                                               |
| <b>Output</b> | <b>b</b>    | The solution vector $x$ overwrites the right-hand-side vector $b$ .                                                                                                                                                                                                                         |

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example 1** Solve a system of linear equations  $Ax = b$ , where  $A$  is a 6 by 6 REAL\*8 matrix stored in array A whose dimensions are 10 by 10, and  $b$  is a vector 6 elements long stored in an array B of dimension 10. The more robust, but slightly slower condition number estimation subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,IPVT(10),JOB
REAL*8 A(10,10),B(10),RCOND,WORK(10)
LDA = 10
N = 6
JOB = 0
CALL SGECO (A,LDA,N,IPVT,RCOND,WORK)
IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SGESL (A,LDA,N,IPVT,B,JOB)
ELSE
 handle singular matrix
END IF

```

If the coefficient matrix  $A$  is determined to be nonsingular, the solution vector  $x$  overwrites the right-hand-side  $b$  in array **b**.

**Example 2** Solve a system of linear equations  $A^T x = b$ , where  $A$  is a 6 by 6 REAL\*8 matrix stored in array A whose dimensions are 10 by 10, and  $b$  is a vector 6 elements long stored in an array B of dimension 10. The less reliable, but slightly faster, factorization subprogram is used to factor the coefficient matrix.

```
INTEGER*8 LDA,N,IPVT(10),IER,JOB
REAL*8 A(10,10),B(10)
LDA = 10
N = 6
JOB = 1
CALL SGEFA (A,LDA,N,IPVT,IER)
IF (IER .EQ. 0) THEN
 CALL SGESL (A,LDA,N,IPVT,B,JOB)
ELSE
 handle singular matrix
END IF
```

If the coefficient matrix  $A$  is determined to be nonsingular, the solution vector  $x$  overwrites the right-hand-side  $b$  in array  $b$ .

**Solve Tridiagonal Linear Equations****SGTSL/CGTSL**

**Purpose** Given an  $n$  by  $n$  tridiagonal matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . A tridiagonal matrix  $A = \{a_{ij}\}$  is a matrix whose nonzero elements lie only on the principal diagonal ( $i = j$ ), the subdiagonal ( $i = j + 1$ ), and the superdiagonal ( $i = j - 1$ ) of the matrix.

These subprograms perform the same function as subprograms SGTSV, DGTSV, CGTSV and ZGTSV, except that the latter are designed for use on tridiagonal matrices that are known to be diagonally dominant. A matrix is diagonally dominant if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \quad i = 1, 2, \dots, n.$$

When a matrix is diagonally dominant, partial pivoting is not required to ensure numerical stability. Eliminating partial pivoting admits vectorized and parallelized algorithms, so SGTSV, DGTSV, CGTSV and ZGTSV are faster than these subprograms. For efficiency, use SGTSV, DGTSV, CGTSV or ZGTSV when it is known that partial pivoting may be safely omitted.

**Usage** The following example illustrates the storage of a tridiagonal matrix. If the original 7 by 7 matrix is

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 11 | 12 | 0  | 0  | 0  | 0  | 0  |
| 21 | 22 | 23 | 0  | 0  | 0  | 0  |
| 0  | 32 | 33 | 34 | 0  | 0  | 0  |
| 0  | 0  | 43 | 44 | 45 | 0  | 0  |
| 0  | 0  | 0  | 54 | 55 | 56 | 0  |
| 0  | 0  | 0  | 0  | 65 | 66 | 67 |
| 0  | 0  | 0  | 0  | 0  | 76 | 77 |

then the subdiagonal is stored in array **c**, the principal diagonal is stored in array **d**, and the superdiagonal is stored in array **e**, as follows:

| i | c(i) | d(i) | e(i) |
|---|------|------|------|
| 1 | *    | 11   | 12   |
| 2 | 21   | 22   | 23   |
| 3 | 32   | 33   | 34   |
| 4 | 43   | 44   | 45   |
| 5 | 54   | 55   | 56   |
| 6 | 65   | 66   | 67   |
| 7 | 76   | 77   | *    |

The asterisks represent elements whose initial contents are not used.

**SCILIB:**

```

INTEGER*8 n,ier
REAL*8 c(n),d(n),e(n),b(n)
CALL SGTSL (n, c, d, e, b, ier)

```

```

INTEGER*4 n,ier
COMPLEX*16 c(n),d(n),e(n),b(n)
CALL CGTSL (n, c, d, e, b, ier)

```

|               |            |                                                                                                                                                                       |
|---------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>  | <b>n</b>   | The order of matrix $A$ , $n > 0$ .                                                                                                                                   |
|               | <b>c</b>   | Array containing the subdiagonal of the tridiagonal matrix, $c(i) = a_{i,i-1}$ , $i = 2, 3, \dots, n$ . On return, <b>c</b> is destroyed, including <b>c(1)</b> .     |
|               | <b>d</b>   | Array containing the principal diagonal of the tridiagonal matrix, $d(i) = a_{ii}$ , $i = 1, 2, \dots, n$ . On return, <b>d</b> is destroyed.                         |
|               | <b>e</b>   | Array containing the superdiagonal of the tridiagonal matrix, $e(i) = a_{i,i+1}$ , $i = 1, 2, \dots, n-1$ . On return, <b>e</b> is destroyed, including <b>e(n)</b> . |
|               | <b>b</b>   | The right-hand-side vector $b$ .                                                                                                                                      |
| <b>Output</b> | <b>b</b>   | The solution vector $x$ overwrites the right-hand-side vector $b$ if <b>ier</b> = 0 is returned.                                                                      |
|               | <b>ier</b> | Status response:<br><br><b>ier</b> = 0      Normal return.<br><b>ier</b> = $k \neq 0$ if the $k$ th element of the diagonal becomes exactly zero.                     |

**Notes**      These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example**      Solve a system of linear equations  $Ax = b$ , where  $A$  is a 7 by 7 REAL\*8 tridiagonal matrix. The subdiagonal of  $A$  is stored in array **C**, the principal diagonal is stored in array **D**, and the superdiagonal is stored in array **E**.  $b$  is a vector 7 elements long stored in an array **B**.

```

INTEGER*8 N, IER
REAL*8 C(10), D(10), E(10), B(10)
N = 7
CALL SGTSL (N, C, D, E, B, IER)
IF (IER .NE. 0) THEN
 handle error condition
END IF

```

**Estimate Condition****SPBCO/CPBCO****Purpose**

These subprograms compute the Cholesky factorization and estimate the condition number of an  $n$  by  $n$  positive definite band matrix  $A$  stored in a two-dimensional array. A matrix  $A$  is positive definite if and only if it is Hermitian; that is,  $A$  is equal to  $A^*$ , its conjugate transpose, and the quadratic form  $x^*Ax$  is positive for all nonzero vectors  $x$ . (The conjugate transpose of a real matrix or vector is simply the transpose.)

A positive definite band matrix is a positive definite matrix whose nonzero elements all are fairly near the principal diagonal. Specifically,  $a_{ij} = 0$  if  $|i-j| > m$  for some integer  $m$ . The smallest such  $m$  for a given matrix is called the half bandwidth, and  $2m+1$  is called the total bandwidth.

Tridiagonal matrices are the special case  $m = 1$ . They can be handled more efficiently by the subprograms SPTSL and CPTSL.

Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  upper-triangular band matrix  $R$ , such that

$$A = R^*R$$

and compute an estimate of  $\kappa(A)$ , the condition number of  $A$ . Refer to "Condition Number" in the introduction to this chapter for a discussion of  $\kappa(A)$ . When a matrix is ill-conditioned,  $\kappa(A)$  is large, so small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself are magnified greatly in the solution.

Since  $1 < \kappa(A) \leq \infty$ , these subprograms actually compute the reciprocal condition number,  $1/\kappa(A)$ . The reciprocal condition number has the interpretation that if  $1/\kappa(A)$  approximately equals  $10^{-d}$ , elements of  $x$  can be expected to have  $d$  fewer significant digits of accuracy than the elements of  $A$  or  $b$ . Consequently, if errors in the coefficient matrix and right-hand side exceed  $1/\kappa(A)$ , or if  $1/\kappa(A)$  is negligible compared to 1.0, then  $x$  may have no significant digits at all.

A set of companion subprograms computes the Cholesky factorization of a matrix without estimating its condition number. These companion subprograms are faster but provide a less reliable indication of singularity.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $R^*(Rx) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(R)^2$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve | Determinant |
|------------|--------------------|--------|-------|-------------|
| REAL*8     | SPBCO              | SPBFA  | SPBSL | SPBDI       |
| COMPLEX*16 | CPBCO              | CPBFA  | CPBSL | CPBDI       |

The inverse of  $A$  will usually be a full  $n$  by  $n$  matrix, which cannot be stored in the band storage of  $A$ . Therefore, no direct provision is made for computing  $A^{-1}$ . Calculations formulated in terms of matrix inverses are invariably more efficient when expressed in terms of the solution of sets of linear equations.

**Usage** The following example illustrates the storage of positive definite band matrices. If the original matrix is

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 0  | 0  | 0  | 0  |
| 12 | 22 | 23 | 24 | 0  | 0  | 0  |
| 13 | 23 | 33 | 34 | 35 | 0  | 0  |
| 0  | 24 | 34 | 44 | 45 | 46 | 0  |
| 0  | 0  | 35 | 45 | 55 | 56 | 57 |
| 0  | 0  | 0  | 46 | 56 | 66 | 67 |
| 0  | 0  | 0  | 0  | 57 | 67 | 77 |

then its order is  $n = 7$  and its half bandwidth is  $m = 2$ . The upper triangle of  $A$  is stored in an array  $\mathbf{a}$  with at least  $m + 1 = 3$  rows and 7 columns as follows:

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| *  | *  | 13 | 24 | 35 | 46 | 57 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 |

The asterisks represent elements in the  $m$  by  $m$  triangle at the upper-left corner of  $\mathbf{a}$  that are not referenced. Thus, if  $a_{ij}$  is an element within the band of the upper triangle of  $A$ , it is stored in  $\mathbf{a}(i-j+m+1,j)$ . In words, the columns of the upper triangle of  $A$  are stored in the columns of  $\mathbf{a}$ , and the diagonals of the upper triangle of  $A$  are stored in the rows of  $\mathbf{a}$ .

**SCILIB:**

```
INTEGER*8 lda,n,m,ier
REAL*8 a(lda,n),rcond,work(n)
CALL SPBCO (a, lda, n, m, rcond, work, ier)
```

```
INTEGER*8 lda,n,m,ier
COMPLEX*16 a(lda,n),work(n)
REAL*8 rcond
CALL CPBCO (a, lda, n, m, rcond, work, ier)
```

**Input**

**a** Array containing the upper triangle of the  $n$  by  $n$  positive definite band matrix  $A$  in the compressed form described above. If  $0 \leq j - i \leq m$ , then  $a_{ij}$  is stored in  $\mathbf{a}(i-j+m+1)$ . Columns of the upper triangle of  $A$  are stored in the columns of  $\mathbf{a}$ , and diagonals of the upper triangle of  $A$  are stored in the rows of  $\mathbf{a}$ .

**lda** The leading dimension of array  $\mathbf{a}$  as declared in the calling program unit, with  $\text{lda} \geq m+1$ .

**n** The order of matrix  $A$ ,  $n > 0$ .

**m** The half bandwidth of  $A$ , i.e., the number of diagonals above the principal diagonal in the band,  $0 \leq m < n$ .

**Working storage**

**work** An array of size  $n$ , used for work space.

**Output**

**a** The Cholesky factor  $R$  replaces the input matrix. The factorization is not complete if  $\text{ier}$  is nonzero.  $\mathbf{a}$  must be preserved between the condition number estimation call and any solve or determinant call.

**rcond** An estimate of the reciprocal condition number,  $1/\kappa(A)$ , if **ier** is zero; unchanged from its input value if **ier** is nonzero. If **ier** is zero and **rcond** is so small that the logical expression

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true,  $A$  can be regarded as singular to working precision.

**ier** Status response:

**ier** = 0 Normal return—factorization complete.

**ier** =  $k \neq 0$  The leading submatrix of order  $k$  is not computationally positive definite, possibly because of roundoff error.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example** Factor the 7 by 7 REAL\*8 positive definite band matrix  $A$  whose half bandwidth is 2 and whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 5 by 10, and estimate its reciprocal condition number.

```

INTEGER*8 LDA,N,M,IER
REAL*8 A(5,10),RCOND,WORK(10)
LDA = 5
N = 7
M = 2
CALL SPBCO (A,LDA,N,M,RCOND,WORK,IER)
IF (IER .NE. 0) THEN
 handle indefinite matrix
ELSE IF (1.0 + RCOND .EQ. 1.0) THEN
 handle singular matrix
END IF

```

**Purpose** Given the Cholesky factorization of an  $n$  by  $n$  positive definite band matrix  $A$ , these subprograms evaluate the determinant of  $A$ . No provision is made to compute  $A^{-1}$  because it will usually be a full  $n$  by  $n$  matrix, which cannot be stored in the band storage of  $A$ . Moreover, it is almost never necessary to compute the inverse of a matrix. Mathematical references frequently use " $A^{-1}b$ " to mean "the solution  $x$  of the system of linear equations  $Ax = b$ ." It is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right-hand-side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors just as efficiently, and more accurately, than by matrix multiplication by the inverse.

Specifically, given an  $n$  by  $n$  upper-triangular band matrix  $R$ , such that

$$A = R^*R,$$

where  $R^*$  is the conjugate transpose of  $R$ , the subprograms compute

$$\det(A) = \det(R)^2.$$

The Cholesky factorization of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes the factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Determinant |
|------------|--------------------|--------|-------------|
| REAL*8     | SPBCO              | SPBFA  | SPBDI       |
| COMPLEX*16 | CPBCO              | CPBFA  | CPBDI       |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,m
REAL*8 a(lda,n),det(2)
CALL SPBDI (a, lda, n, m, det)
```

```
INTEGER*8 lda,n,m
COMPLEX*16 a(lda,n)
REAL*8 det(2)
CALL CPBDI (a, lda, n, m, det)
```

**Input**

- a** Array containing the Cholesky factor  $R$  of the  $n$  by  $n$  positive definite band matrix  $A$  as computed by the companion factorization or condition number estimation subprogram. **a** must have been preserved between the factorization or condition number call and the determinant call.
- lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq n+1$ .
- n** The order of matrix  $A$ ,  $n \geq 0$ .

Continued

- m** The half bandwidth of  $A$ , i.e., the number of diagonals above the principal diagonal in the band,  $0 \leq m < n$ .
- Output** **det** The determinant of  $A$ , in the form  $\det(A) = \det(1) \times 10^{\det(2)}$ . This expression may underflow or overflow if evaluated; on the CONVEX supercomputer, underflows automatically flush to zero, but overflows normally terminate execution. For REAL\*8 and COMPLEX\*16, overflow cannot occur if  $\det(2) \leq 306$ . If evaluation is safe, an efficient way to do it is with the statement

$$\det(A) = \det(1) * 10.0 ** \text{INT}(\det(2))$$

The value stored in **det(2)** is an integer in REAL form. **det(1)** is normalized so that  $\det(1) = 0$  or  $1 \leq \det(1) < 10$ .

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**It is almost never necessary to compute the determinant of a matrix.** While it is true that papers and reference books make extensive use of the notation " $\det(A) \neq 0$ " to mean " $A$  is nonsingular," SCILIB includes both more efficient and more reliable subprograms for detecting singularity.

**Example** Compute the determinant of a 7 by 7 REAL\*8 matrix  $A$  whose half bandwidth is 2 and whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 5 by 10. The less reliable, but slightly faster factorization subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA,N,M,IER
REAL*8 A(5,10),DET(2),DETA
LDA = 5
N = 7
M = 2
CALL SPBFA (A,LDA,N,M,IER)
IF (IER .EQ. 0) THEN
 CALL SPBDI (A,LDA,N,M,DET)
 IF (DET(1) .EQ. 0.0) THEN
 DETA = 0.0
 ELSE IF (DET(2) .LE. 306) THEN
 DETA = DET(1) * 10.0 ** INT(DET(2))
 ELSE
 the determinant of A is too large to evaluate
 without overflow
 END IF
ELSE
 DETA = 0.0
END IF

```

**Purpose** These subprograms compute Cholesky factorization of an  $n$  by  $n$  positive definite band matrix  $A$  stored in a two-dimensional array. A matrix  $A$  is positive definite if and only if it is Hermitian; that is,  $A$  is equal to  $A^*$ , its conjugate transpose, and the quadratic form  $x^*Ax$  is positive for all nonzero vectors  $x$ . (The conjugate transpose of a real matrix or vector is simply the transpose.)

A positive definite band matrix is a positive definite matrix whose nonzero elements all are fairly near the principal diagonal. Specifically,  $a_{ij} = 0$  if  $|i-j| > m$  for some integer  $m$ . The smallest such  $m$  for a given matrix is called the half bandwidth, and  $2m+1$  is called the total bandwidth.

Tridiagonal matrices are the special case  $m = 1$ . They can be handled more efficiently by the subprograms SPTSL and CPTSL.

Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  upper-triangular band matrix  $R$ , such that

$$A = R^*R.$$

Computational singularity of  $A$  results in one or more zero diagonal elements of  $R$ , or, more frequently, in the loss of positive definiteness as evidenced by a negative diagonal element. This condition is detected during factorization, and a status response is returned to indicate its occurrence. A more common situation, however, is that  $A$  is not numerically singular but is ill-conditioned. When a matrix is ill-conditioned, small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself are magnified greatly in the solution. A set of companion subprograms computes Cholesky factorization of a matrix and estimates its condition number. These companion subprograms provide a more reliable indication of singularity. The small amount of additional time they require is usually worthwhile, especially when developing a program or encountering stability or convergence problems.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $R^*(Rx) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(R)^2$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Factor | Estimate Condition | Solve | Determinant |
|------------|--------|--------------------|-------|-------------|
| REAL*8     | SPBFA  | SPBCO              | SPBSL | SPBDI       |
| COMPLEX*16 | CPBFA  | CPBCO              | CPBSL | CPBDI       |

Companion subprograms are documented elsewhere in this chapter.

The inverse of  $A$  will usually be a full  $n$  by  $n$  matrix, which cannot be stored in the band storage of  $A$ . Therefore, no direct provision is made for computing  $A^{-1}$ . Calculations formulated in terms of matrix inverses are invariably more efficient when expressed in terms of the solution of sets of linear equations.

**Usage** The following example illustrates storage of positive definite band matrices. If the original matrix is

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 11 | 12 | 13 | 0  | 0  | 0  | 0  |
| 12 | 22 | 23 | 24 | 0  | 0  | 0  |
| 13 | 23 | 33 | 34 | 35 | 0  | 0  |
| 0  | 24 | 34 | 44 | 45 | 46 | 0  |
| 0  | 0  | 35 | 45 | 55 | 56 | 57 |
| 0  | 0  | 0  | 46 | 56 | 66 | 67 |
| 0  | 0  | 0  | 0  | 57 | 67 | 77 |

then its order is  $n = 7$  and its half bandwidth is  $m = 2$ . The upper triangle of  $A$  is stored in an array  $\mathbf{a}$  with at least  $m + 1 = 3$  rows and 7 columns:

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| *  | *  | 13 | 24 | 35 | 46 | 57 |
| *  | 12 | 23 | 34 | 45 | 56 | 67 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 |

Asterisks represent elements in the  $m$  by  $m$  triangle at the upper-left corner of  $\mathbf{a}$  that are not referenced. Thus, if  $a_{ij}$  is an element within the band of the upper triangle of  $A$ , it is stored in  $\mathbf{a}(i-j+m+1,j)$ . Thus, the columns of the upper triangle of  $A$  are stored in the columns of  $\mathbf{a}$ , and diagonals of the upper triangle of  $A$  are stored in the rows of  $\mathbf{a}$ .

**SCILIB:**

```
INTEGER*8 lda,n,m,ier
REAL*8 a(lda,n)
CALL SPBFA (a, lda, n, m, ier)
```

```
INTEGER*8 lda,n,m,ier
COMPLEX*16 a(lda,n)
CALL CPBFA (a, lda, n, m, ier)
```

**Input** **a** Array containing the upper triangle of the  $n$  by  $n$  positive definite band matrix  $A$  in the compressed form described above. If  $0 \leq j - i \leq m$ , then  $a_{ij}$  is stored in  $\mathbf{a}(i-j+m+1)$ . The columns of the upper triangle of  $A$  are stored in the columns of  $\mathbf{a}$  and the diagonals of the upper triangle of  $A$  are stored in the rows of  $\mathbf{a}$ .

**lda** The leading dimension of array  $\mathbf{a}$  as declared in the calling program unit, with  $lda \geq m+1$ .

**n** The order of matrix  $A$ ,  $n > 0$ .

**m** The half bandwidth of  $A$ , i.e., the number of diagonals above the principal diagonal in the band,  $0 \leq m < n$ .

**Output** **a** The Cholesky factor  $R$  replaces the input matrix. The factorization is not complete if **ier** is nonzero.  $\mathbf{a}$  must be preserved between the condition number estimation call and any solve or determinant call.

**ier** Status response:

- ier** = 0 Normal return—factorization complete.
- ier** =  $k \neq 0$  The leading submatrix of order  $k$  is not computationally positive definite, possibly because of roundoff error.

**Notes**            These subprograms are usage-compatible with the standard LINPACK subprograms with the same names.

**Example**          Factor the 7 by 7 REAL\*8 positive definite band matrix  $A$  whose half bandwidth is 2 and whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 5 by 10, and estimate its reciprocal condition number.

```
INTEGER*8 LDA,N,M,IER
REAL*8 A(5,10)
LDA = 5
N = 7
M = 2
CALL SPBFA (A,LDA,N,M,IER)
IF (IER .NE. 0) THEN
 handle singular or indefinite matrix
END IF
```

## Solve Linear Equations

## SPBSL/CPBSL

**Purpose** Given the Cholesky factorization of an  $n$  by  $n$  positive definite band matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . Specifically, given an  $n$  by  $n$  upper-triangular band matrix  $R$ , such that

$$A = R^*R,$$

where  $R^*$  is the conjugate transpose of  $R$ , and an  $n$ -vector  $b$ , to find  $x$  satisfying  $Ax = b$ , the subprograms successively solve

$$R^*w = b$$

and

$$Rx = w.$$

Cholesky factorization of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve |
|------------|--------------------|--------|-------|
| REAL*8     | SPBCO              | SPBFA  | SPBSL |
| COMPLEX*16 | CPBCO              | CPBFA  | CPBSL |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,m
REAL*8 a(lda,n),b(n)
CALL SPBSL (a, lda, n, m, b)
```

```
INTEGER*8 lda,n,m
COMPLEX*16 a(lda,n),b(n)
CALL CPBSL (a, lda, n, m, b)
```

**Input**

**a** Array containing the Cholesky factor  $R$  of the  $n$  by  $n$  positive definite band matrix  $A$  as computed by the companion factorization or condition number estimation subprogram. **a** must have been preserved between the factorization or condition number call and the solve call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq m+1$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**m** The half bandwidth of  $A$ , i.e., the number of diagonals above the principal diagonal in the band,  $0 \leq m < n$ .

**b** The right-hand-side vector  $b$ .

**Output**

**b** The solution vector  $x$  overwrites the right-hand-side vector  $b$ .

**Notes**

These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example**

Solve a system of linear equations  $Ax = b$ , where  $A$  is a 7 by 7 REAL\*8 positive definite band matrix with half bandwidth 2. The upper triangle of  $A$  is stored in array  $A$  whose dimensions are 5 by 10.  $b$  is a vector 7 elements long stored in an array  $B$  of dimension 10. The more robust, but slightly slower condition number estimation subprogram is used to factor the coefficient matrix.

```
INTEGER*8 LDA,N,IER
REAL*8 A(5,10),B(10),RCOND,WORK(10)
LDA = 5
N = 7
M = 2
CALL SPBCO (A,LDA,N,M,RCOND,WORK,IER)
IF (IER .NE. 0) THEN
 handle indefinite matrix
ELSE IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SPBSL (A,LDA,N,M,B)
ELSE
 handle singular matrix
END IF
```

If the coefficient matrix  $A$  is determined to be positive definite and nonsingular, the solution vector  $x$  overwrites the right-hand-side  $b$  in array  $b$ .

**Purpose** These subprograms compute Cholesky factorization and estimate the condition number of an  $n$  by  $n$  positive definite matrix  $A$  stored in a two-dimensional array and estimate its condition number. A matrix  $A$  is positive definite if and only if it is Hermitian; that is,  $A$  is equal to  $A^*$ , its conjugate transpose, and the quadratic form  $x^*Ax$  is positive for all nonzero vectors  $x$ . (The conjugate transpose of a real matrix or vector is simply the transpose.)

Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  upper-triangular matrix  $R$ , such that

$$A = R^*R$$

and compute an estimate of  $\kappa(A)$ , the condition number of  $A$ . Refer to "Condition Number" in the introduction to this chapter for a discussion of  $\kappa(A)$ . When a matrix is ill-conditioned,  $\kappa(A)$  is large, so small errors in the matrix and right-hand side and small roundoff errors introduced during the solution process itself are magnified greatly in the solution.

Since  $1 < \kappa(A) \leq \infty$ , these subprograms actually compute the reciprocal condition number,  $1/\kappa(A)$ . The reciprocal condition number has the interpretation that if  $1/\kappa(A)$  approximately equals  $10^{-d}$ , elements of  $x$  can be expected to have  $d$  fewer significant digits of accuracy than the elements of  $A$  or  $b$ . Consequently, if errors in the coefficient matrix and right-hand side exceed  $1/\kappa(A)$ , or if  $1/\kappa(A)$  is negligible compared to 1.0, then  $x$  may have no significant digits at all.

A set of companion subprograms computes Cholesky factorization of a matrix without estimating its condition number. These companion subprograms are faster but provide a less reliable indication of singularity.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $R^*(Rx) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(R)^2$ . The inverse of  $A$  may be formed as  $A^{-1} = R^{-1}R^{-*}$ , where  $R^{-*}$  is the conjugate transpose of the inverse of  $R$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Estimate Condition | Factor | Solve | Determinant or inverse |
|------------|--------------------|--------|-------|------------------------|
| REAL*8     | SPOCO              | SPOFA  | SPOSL | SPODI                  |
| COMPLEX*16 | CPOCO              | CPOFA  | CPOSL | CPODI                  |

The companion subprograms are documented elsewhere in this chapter.

### Usage

#### SCILIB:

```
INTEGER*8 lda,n,ier
REAL*8 a(lda,n),rcond,work(n)
CALL SPOCO (a, lda, n, rcond, work, ier)
```

```
INTEGER*8 lda,n,ier
COMPLEX*16 a(lda,n),work(n)
REAL*8 rcond
CALL CPOCO (a, lda, n, rcond, work, ier)
```

|                        |              |                                                                                                                                                                                                                                                                                                                                |
|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>           | <b>a</b>     | Array containing the diagonal and upper triangle of the $n$ by $n$ positive definite matrix $A$ . The elements in the strict lower triangle of <b>a</b> are not referenced.                                                                                                                                                    |
|                        | <b>lda</b>   | The leading dimension of array <b>a</b> as declared in the calling program unit, with $lda \geq \max(n,1)$ .                                                                                                                                                                                                                   |
|                        | <b>n</b>     | The order of matrix $A$ , $n \geq 0$ .                                                                                                                                                                                                                                                                                         |
| <b>Working storage</b> | <b>work</b>  | An array of size $n$ , used for work space.                                                                                                                                                                                                                                                                                    |
| <b>Output</b>          | <b>a</b>     | The Cholesky factor $R$ replaces the input matrix $A$ in the upper triangle of <b>a</b> . The strict lower triangle of <b>a</b> is unchanged. The factorization is not complete if <b>ier</b> is nonzero. <b>a</b> must be preserved between the condition number estimation call and any solve, determinant, or inverse call. |
|                        | <b>rcond</b> | An estimate of the reciprocal condition number, $1/\kappa(A)$ , if <b>ier</b> is zero; unchanged from its input value if <b>ier</b> is nonzero. If <b>ier</b> is zero and <b>rcond</b> is so small that the logical expression                                                                                                 |

$$1.0 + \text{rcond} \text{ .EQ. } 1.0$$

is true, then  $A$  can be regarded as singular to working precision.

**ier** Status response:

**ier** = 0 Normal return—factorization complete.

**ier** =  $k \neq 0$  The leading submatrix of order  $k$  is not computationally positive definite, possibly because of roundoff error.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example** Factor the 6 by 6 REAL\*8 positive definite matrix  $A$  whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 10 by 10, and estimate its reciprocal condition number.

```

INTEGER*8 LDA,N,IER
REAL*8 A(10,10),RCOND,WORK(10)
LDA = 10
N = 6
CALL SPOCO (A,LDA,N,RCOND,WORK,IER)
IF (IER .NE. 0) THEN
 handle indefinite matrix
ELSE IF (1.0 + RCOND .EQ. 1.0) THEN
 handle singular matrix
END IF

```

**Determinant and Inverse****SPODI/CPODI**

**Purpose** Given the Cholesky factorization of an  $n$  by  $n$  positive definite coefficient matrix  $A$ , these subprograms evaluate the determinant of  $A$  and/or compute  $A^{-1}$ . Specifically, given an  $n$  by  $n$  upper-triangular matrix  $R$ , such that

$$A = R^*R,$$

where  $R^*$  is the conjugate transpose of  $R$ , the subprograms compute

$$\det(A) = \det(R)^2$$

and/or

$$A^{-1} = R^{-1}R^{-*}$$

where  $R^{-*}$  is the conjugate transpose of the inverse of  $R$ .

The Cholesky factorization of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable, especially when  $A^{-1}$  is desired. The names of the companion subprograms depend on the data type:

| Data Type  | Estimate Condition | Factor | Determinant or inverse |
|------------|--------------------|--------|------------------------|
| REAL*8     | SPOCO              | SPOFA  | SPODI                  |
| COMPLEX*16 | CPOCO              | CPOFA  | CPODI                  |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```

INTEGER*8 lda,n,job
REAL*8 a(lda,n),det(2)
CALL SPODI (a, lda, n, det, job)
INTEGER*8 lda,n,job
COMPLEX*16 a(lda,n)
REAL*8 det(2)
CALL CPODI (a, lda, n, det, job)

```

**Input**

**a** Array containing the Cholesky factor  $R$  of the  $n$  by  $n$  positive definite coefficient matrix  $A$  in its upper triangle, as computed by the companion factorization or condition number estimation subprogram. The upper triangle of **a** must have been preserved between the factorization or condition number call and the determinant or inverse call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**job** Option flag:

```

job = 1 compute only A^{-1}
job = 10 compute only $\det(A)$
job = 11 compute both A^{-1} and $\det(A)$

```

- Output**
- a** Unchanged if  $A^{-1}$  is not requested. Otherwise, the upper triangle of  $A^{-1}$  overwrites the Cholesky factor of the coefficient matrix. The strict lower triangle of **a** is never changed.
- det** Not referenced if the determinant is not requested. Otherwise, the determinant of  $A$ , in the form  $\det(A) = \det(1) \times 10^{\det(2)}$ . This expression may underflow or overflow if evaluated; on the CONVEX supercomputer, underflows automatically flush to zero, but overflows normally terminate execution. For REAL\*8 and COMPLEX\*16, overflow cannot occur if  $\det(2) \leq 306$ . If evaluation is safe, an efficient way to do it is with the statement

$$\det(A) = \det(1) * 10.0 ** \text{INT}(\det(2))$$

Refer to "Example 2."

The value stored in **det(2)** is an integer in REAL form. **det(1)** is normalized so that  $\det(1) = 0$  or  $1 \leq \det(1) < 10$ .

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**It is almost never necessary to compute either the determinant or the inverse of a matrix.** While papers and reference books extensively use the notation " $\det(A) \neq 0$ " to mean " $A$  is nonsingular," SCILIB includes more efficient and more reliable subprograms for detecting singularity. Similarly, references frequently use " $A^{-1}b$ " to mean "the solution  $x$  of the system of linear equations  $Ax = b$ ." Again, it is more efficient and accurate to compute the solution directly than to invert the coefficient matrix and multiply the inverse times the right-hand-side vector. This is true even if there are many systems of equations, all using the same coefficient matrix; the matrix may be factored once and the systems may be solved from the factors just as efficiently, and more accurately, than by matrix multiplication by the inverse.

Continued

**Example 1** Compute only the inverse of a 6 by 6 REAL\*8 positive definite matrix  $A$  whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 10 by 10. The more robust, but slightly slower condition number estimation subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA, N, IER, JOB
REAL*8 A(10,10), DET(2), RCOND, WORK(10)
LDA = 10
N = 6
JOB = 1
CALL SPOCO (A, LDA, N, RCOND, WORK, IER)
IF (IER .NE. 0) THEN
 handle indefinite matrix
ELSE IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SPODI (A, LDA, N, DET, JOB)
ELSE
 handle singular matrix
END IF

```

If the coefficient matrix  $A$  is determined to be nonsingular,  $A^{-1}$  overwrites the coefficient matrix  $A$  in array  $a$ .

**Example 2** Compute only the determinant of a 6 by 6 REAL\*8 matrix  $A$  stored in array  $A$  whose dimensions are 10 by 10. The less reliable, but slightly faster factorization subprogram is used to factor the coefficient matrix.

```

INTEGER*8 LDA, N, IER, JOB
REAL*8 A(10,10), DET(2), DETA
LDA = 10
N = 6
JOB = 10
CALL SPOFA (A, LDA, N, IER)
IF (IER .EQ. 0) THEN
 CALL SPODI (A, LDA, N, DET, JOB)
 IF (DET(1) .EQ. 0.0) THEN
 DETA = 0.0
 ELSE IF (DET(2) .LE. 306) THEN
 DETA = DET(1) * 10.0 ** INT(DET(2))
 ELSE
 the determinant of A is too large to evaluate
 without overflow
 END IF
ELSE
 DETA = 0.0
END IF

```

**Purpose** These subprograms compute the Cholesky factorization of an  $n$  by  $n$  positive definite matrix  $A$  stored in a two-dimensional array. A matrix  $A$  is positive definite if and only if it is Hermitian; that is,  $A$  is equal to  $A^*$ , its conjugate transpose, and the quadratic form  $x^*Ax$  is positive for all nonzero vectors  $x$ . (The conjugate transpose of a real matrix or vector is simply the transpose.) Specifically, given  $A$ , these subprograms determine an  $n$  by  $n$  upper-triangular matrix  $R$ , such that

$$A = R^*R.$$

Computational singularity of  $A$  results in one or more zero diagonal elements of  $R$ , or, more frequently, in the loss of positive definiteness as evidenced by a negative diagonal element. This condition is detected during the factorization, and a status response is returned to indicate its occurrence. A more common situation, however, is that  $A$  is not numerically singular but happens to be ill-conditioned. When a matrix is ill-conditioned, small errors in the matrix and right-hand side, and small roundoff errors introduced during the solution process itself are magnified greatly in the solution. A set of companion subprograms computes the Cholesky factorization of a matrix and also estimates its condition number. These companion subprograms provide a more reliable indication of singularity. The small amount of additional time they require is usually worthwhile, especially when developing a program or encountering stability or convergence problems.

The triangular factors may be used to solve a system of linear equations,  $Ax = b$ , by successively solving  $R^*(Rx) = b$ . The determinant of  $A$  can be computed as  $\det(A) = \det(R)^2$ . The inverse of  $A$  may be formed as  $A^{-1} = R^{-1}R^{-*}$ , where  $R^{-*}$  is the conjugate transpose of the inverse of  $R$ . These operations are performed by a set of companion SCILIB subprograms whose names depend on the data type:

| Data Type  | Factor | Estimate Condition | Solve | Determinant or inverse |
|------------|--------|--------------------|-------|------------------------|
| REAL*8     | SPOFA  | SPOCO              | SPOSL | SPODI                  |
| COMPLEX*16 | CPOFA  | CPOCO              | CPOSL | CPODI                  |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n,ier
REAL*8 a(lda,n)
CALL SPOFA (a, lda, n, ier)
```

```
INTEGER*8 lda,n,ier
COMPLEX*16 a(lda,n)
CALL CPOFA (a, lda, n, ier)
```

**Input**

- a** Array containing the diagonal and upper triangle of the  $n$  by  $n$  positive definite matrix  $A$ . The elements of the strict lower triangle are not referenced.
- lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .
- n** The order of matrix  $A$ ,  $n \geq 0$ .

**Output**

Continued

---

**a** The Cholesky factor  $R$  replaces the input matrix  $A$  in the upper triangle of **a**. The strict lower triangle of **a** is unchanged. The factorization is not complete if **ier** is nonzero. **a** must be preserved between the factorization call and any solve, determinant, or inverse call.

**ier** Status response:

**ier** = 0 Normal return—factorization complete.

**ier** =  $k \neq 0$  The leading submatrix of order  $k$  is not computationally positive definite, possibly because of roundoff error.

**Notes** These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

**Example** Factor the 6 by 6 REAL\*8 positive definite matrix  $A$  whose upper triangle is stored in the upper triangle of array  $A$  whose dimensions are 10 by 10.

```
INTEGER*8 LDA,N,IER
REAL*8 A(10,10)
LDA = 10
N = 6
CALL SPOFA (A,LDA,N,IER)
IF (IER .NE. 0) THEN
 handle indefinite matrix
END IF
```

**Purpose** Given the Cholesky factorization of an  $n$  by  $n$  positive definite coefficient matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . Specifically, given an  $n$  by  $n$  upper-triangular matrix  $R$ , such that

$$A = R^*R,$$

where  $R^*$  is the conjugate transpose of  $R$ , and an  $n$ -vector  $b$ , to find  $x$  satisfying  $Ax = b$ , the subprograms successively solve

$$R^*w = b$$

and

$$Rx = w.$$

The Cholesky factorization of the coefficient matrix may be computed by either of two companion subprograms. One of them computes only the factorization, using an elementary test for singularity of the coefficient matrix, and is slightly the faster of the two. The other not only computes the factorization, but also estimates the condition number of the matrix. This process takes a little more time, but is considerably more reliable. The names of the companion subprograms depend on the data type:

| Data type  | Estimate condition | Factor | Solve |
|------------|--------------------|--------|-------|
| REAL*8     | SPOCO              | SPOFA  | SPOSL |
| COMPLEX*16 | CPOCO              | CPOFA  | CPOSL |

The companion subprograms are documented elsewhere in this chapter.

**Usage****SCILIB:**

```
INTEGER*8 lda,n
REAL*8 a(lda,n),b(n)
CALL SPOSL (a, lda, n, b)
```

```
INTEGER*8 lda,n
COMPLEX*16 a(lda,n),b(n)
CALL CPOSL (a, lda, n, b)
```

**Input**

**a** Array containing the Cholesky factor  $R$  of the  $n$  by  $n$  positive definite coefficient matrix  $A$  in its upper triangle, as computed by the companion factorization or condition number estimation subprogram. The upper triangle of **a** must have been preserved between the factorization or condition number call and the solve call.

**lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**b** The right-hand-side vector  $b$ .

**Output**

**b** The solution vector  $x$  overwrites the right-hand-side vector  $b$ .

**Notes**

These subprograms are usage compatible with the standard LINPACK subprograms with the same names.

Continued

**Example** Solve a system of linear equations  $Ax = b$ , where  $A$  is a 6 by 6 REAL\*8 positive definite matrix whose upper triangle is stored in array A whose dimensions are 10 by 10, and where  $b$  is a vector 6 elements long stored in an array B of dimension 10. The more robust, but slightly slower condition number estimation subprogram is used to factor the coefficient matrix.

```

 INTEGER*8 LDA,N,IER
 REAL*8 A(10,10),B(10),RCOND,WORK(10)
 LDA = 10
 N = 6
 CALL SPOCO (A,LDA,N,RCOND,WORK,IER)
 IF (IER .NE. 0) THEN
 handle indefinite matrix
 ELSE IF (1.0 + RCOND .NE. 1.0) THEN
 CALL SPOSL (A,LDA,N,B)
 ELSE
 handle singular matrix
 END IF

```

If the coefficient matrix  $A$  is determined to be positive definite and nonsingular, the solution vector  $x$  overwrites the right-hand-side  $b$  in array b.

**Purpose** Given an  $n$  by  $n$  positive definite tridiagonal matrix  $A$ , and a right-hand-side  $n$ -vector  $b$ , these subprograms solve the system of linear equations  $Ax = b$ . A matrix  $A$  is positive definite if and only if it is Hermitian; that is,  $A$  is equal to  $A^*$ , its conjugate transpose, and the quadratic form  $x^*Ax$  is positive for all nonzero vectors  $x$ . (The conjugate transpose of a real matrix or vector is simply the transpose.)

A positive definite tridiagonal matrix is a positive definite matrix  $A = \{a_{ij}\}$  whose nonzero elements lie only on the principal diagonal ( $i = j$ ), the subdiagonal ( $i = j+1$ ), and the superdiagonal ( $i = j-1$ ) of the matrix. Because of conjugate symmetry, the principal diagonal is always real and the subdiagonal and superdiagonal are complex conjugates of each other. Thus, it is not necessary to store both the subdiagonal and the superdiagonal.

**Usage** The following example illustrates the storage of a real positive definite tridiagonal matrix. If the original 7 by 7 matrix is

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 11 | 12 | 0  | 0  | 0  | 0  | 0  |
| 12 | 22 | 23 | 0  | 0  | 0  | 0  |
| 0  | 23 | 33 | 34 | 0  | 0  | 0  |
| 0  | 0  | 34 | 44 | 45 | 0  | 0  |
| 0  | 0  | 0  | 45 | 55 | 56 | 0  |
| 0  | 0  | 0  | 0  | 56 | 66 | 67 |
| 0  | 0  | 0  | 0  | 0  | 67 | 77 |

then the principal diagonal is stored in array  $d$  and the superdiagonal is stored in array  $e$ :

| i | d(i) | e(i) |
|---|------|------|
| 1 | 11   | 12   |
| 2 | 22   | 23   |
| 3 | 33   | 34   |
| 4 | 44   | 45   |
| 5 | 55   | 56   |
| 6 | 66   | 67   |
| 7 | 77   | *    |

The asterisk represents an element that is not referenced.

**SCILIB:**

```
INTEGER*8 n
REAL*8 d(n),e(n-1),b(n)
CALL SPTSL (n, d, e, b)
```

```
INTEGER*8 n
COMPLEX*16 d(n),e(n-1),b(n)
CALL CPTSL (n, d, e, b)
```

**Input**

- n** The order of matrix  $A$ ,  $n > 0$ .
- d** Array containing the principal diagonal of the tridiagonal matrix,  $d(i) = a_{ii}$ ,  $i = 1, 2, \dots, n$ . For CPTSL and ZPTSL, only the real parts of  $d$  are used. On return,  $d$  is destroyed.
- e** Array containing the superdiagonal of the tridiagonal matrix,  $e(i) = a_{i,i+1}$ ,  $i = 1, 2, \dots, n-1$ .

Continued

---

- b**            The right-hand-side vector  $b$ .
- Output**      **b**            The solution vector  $x$  overwrites the right-hand-side vector  $b$ .
- Notes**        These subprograms are usage-compatible with the standard LINPACK subprograms with the same names.
- Caution is necessary since these subprograms do not detect error conditions. An inaccurate solution may be computed or a division by zero may occur if the matrix is indefinite or singular.
- Example**      Solve a system of linear equations  $Ax = b$ , where  $A$  is a 7 by 7 REAL\*8 positive definite tridiagonal matrix. The principal diagonal of  $A$  is stored in array D, and the superdiagonal is stored in array E.  $b$  is a vector 7 elements long stored in an array B.

```
INTEGER*8 N
REAL*8 D(10),E(10),B(10)
N = 7
CALL SPTSL (N,D,E,B)
```

**LINPACK Subprograms not in the *SCILIB User's Guide***

Although SCILIB includes all LINPACK subprograms, the following nonoptimized routines are not documented in the *SCILIB User's Guide*. The *LINPACK User's Guide*, included in the SCILIB documentation set, documents these subprograms.

**Table 4-5: LINPACK Subprograms not in the *SCILIB User's Guide***

| Name  | Function                                                                      |
|-------|-------------------------------------------------------------------------------|
| SCHDC | Cholesky Decomposition of a Symmetric Matrix                                  |
| CCHDC | Cholesky Decomposition of a Hermitian Matrix                                  |
| SCHDD | Recompute the Cholesky Decomposition of a DOWNDATED Symmetric Matrix          |
| CCHDD | Recompute the Cholesky Decomposition of a DOWNDATED Hermitian Matrix          |
| SCHEX | Recompute the Cholesky Decomposition of a PERMUTED Symmetric Matrix           |
| CCHEX | Recompute the Cholesky Decomposition of a PERMUTED Hermitian Matrix           |
| SCHUD | Recompute the Cholesky Decomposition of a UPDATED Symmetric Matrix            |
| CCHUD | Recompute the Cholesky Decomposition of a UPDATED Hermitian Matrix            |
| CHICO | Factor a Hermitian Indefinite Matrix and Estimate its Condition Number        |
| CHIDI | Determinant, Inverse, and Inertia of a Hermitian Indefinite Matrix            |
| CHIFA | Factor a Hermitian Indefinite Matrix                                          |
| CHISL | Solve Linear Equations with a Hermitian Indefinite Matrix                     |
| CHPCO | Factor a Hermitian Indefinite Packed Matrix and Estimate its Condition Number |
| CHPDI | Determinant, Inverse, and Inertia of a Hermitian Indefinite Packed Matrix     |
| CHPFA | Factor a Hermitian Indefinite Packed Matrix                                   |
| CHPSL | Solve Linear Equations with a Hermitian Indefinite Packed Matrix              |
| SPPCO | Factor a Positive Definite Packed Matrix and Estimate its Condition Number    |
| CPPCO | Factor a Positive Definite Packed Matrix and Estimate its Condition Number    |
| SPPDI | Determinant and Inverse of a Positive Definite Packed Matrix                  |
| CPPDI | Determinant and Inverse of a Positive Definite Packed Matrix                  |
| SPPFA | Factor a Positive Definite Packed Matrix                                      |
| CPPFA | Factor a Positive Definite Packed Matrix                                      |
| SPPSL | Solve Linear Equations with a Positive Definite Packed Matrix                 |
| CPPSL | Solve Linear Equations with a Positive Definite Packed Matrix                 |
| SQRDC | <i>QR</i> Decomposition of a General Rectangular Matrix                       |
| CQRDC | <i>QR</i> Decomposition of a General Rectangular Matrix                       |
| SQRSL | Solve Linear Equations using the <i>QR</i> Decomposition                      |
| CQRSL | Solve Linear Equations using the <i>QR</i> Decomposition                      |
| SSICO | Factor a Symmetric Indefinite Matrix and Estimate its Condition Number        |
| CSICO | Factor a Symmetric Indefinite Matrix and Estimate its Condition Number        |
| SSIDI | Determinant, Inverse, and Inertia of a Symmetric Indefinite Matrix            |
| CSIDI | Determinant, Inverse, and Inertia of a Symmetric Indefinite Matrix            |
| SSIFA | Factor a Symmetric Indefinite Matrix                                          |
| CSIFA | Factor a Symmetric Indefinite Matrix                                          |
| SSISL | Solve Linear Equations with a Symmetric Indefinite Matrix                     |
| CSISL | Solve Linear Equations with a Symmetric Indefinite Matrix                     |
| SSPCO | Factor a Symmetric Indefinite Packed Matrix and Estimate its Condition Number |
| CSPCO | Factor a Symmetric Indefinite Packed Matrix and Estimate its Condition Number |

# Eigenvalues and Eigenvectors

## Overview

This chapter describes the state-of-the-art EISPACK library included with SCILIB. Some subprograms in this library have been upgraded by incorporating Level 2 and Level 3 BLAS and other algorithmic improvements. Although all EISPACK subprograms are included in SCILIB, only upgraded ones are described in this chapter. Table 5-1 at the end of this chapter lists the subprograms that are included in SCILIB but not documented in the *SCILIB User's Guide*. You may find information for these subprograms in the *EISPACK Guide* and the *EISPACK Guide Extension* included in the SCILIB documentation set.

Other EISPACK subprograms will be enhanced in subsequent releases of SCILIB. Thus, it is to your advantage to use EISPACK now to realize the enhancements that have already been made as well as those that will be made in the future.

This chapter explains how to use SCILIB subprograms to compute eigenvalues or eigenvalues and eigenvectors of matrices. The operations covered are:

- dense Hermitian eigenproblems,  $Ax = \lambda x$ , with  $A = A^*$
- dense general eigenproblems,  $Ax = \lambda x$ , for arbitrary  $A$
- dense generalized eigenproblems,  $Ax = \lambda Bx$
- banded eigenproblems,  $Ax = \lambda x$

Refer to Chapter 7 for software to compute the eigenvalues or eigenvectors of a real, symmetric, sparse, ordinary or generalized eigenproblem.

## Chapter Objectives

After reading this chapter you will:

- know which version of EISPACK is included in the SCILIB library
- know how to use the described subprograms

## What You Need to Know to Use These Subprograms

EISPACK exists in single- and double-precision versions. Only the single-precision version is included in SCILIB.

## Supplemental Reading

- Garbow, B.S., et al. 1977. "Matrix Eigensystem Routines—EISPACK Guide Extension," *Lecture Notes in Computer Science*, Vol. 51. Springer-Verlag, New York.
- Parlett, B.N. 1980. *The Symmetric Eigenproblem*. Prentice-Hall, New Jersey.
- Smith, B.T., et al. 1976. "Matrix Eigensystem Routines—EISPACK Guide." *Lecture Notes in Computer Science*, Vol. 6, 2nd edition. Springer-Verlag, New York.
- Wilkinson, J.H. 1965. *The Algebraic Eigenproblem*. Oxford University Press, Oxford.

## Subprogram Descriptions

|                                                                     |      |
|---------------------------------------------------------------------|------|
| Eigenvalues and Eigenvectors of a Real Symmetric Matrix             |      |
| RS .....                                                            | 5-3  |
| Eigenvalues and Eigenvectors of a Real Symmetric Tridiagonal Matrix |      |
| TQL2 .....                                                          | 5-5  |
| Eigenvalues of a Real Symmetric Tridiagonal Matrix                  |      |
| TQLRAT .....                                                        | 5-7  |
| Reduce a Real Symmetric Matrix to Real Symmetric Tridiagonal Form   |      |
| TRED1 .....                                                         | 5-9  |
| Reduce a Real Symmetric Matrix to Real Symmetric Tridiagonal Form   |      |
| TRED2 .....                                                         | 5-11 |

**Eigenvalues and Eigenvectors of a Real Symmetric Matrix****RS**

**Purpose** This subprogram computes eigenvalues or eigenvalues and eigenvectors of a full real symmetric  $n$  by  $n$  matrix  $A$ . Specifically, given  $A$ , this subprogram determines  $n$  scalars,  $\lambda_i$ ,  $i = 1, 2, \dots, n$ , for which there exist corresponding nonzero vectors,  $x_i$ , such that

$$Ax_i = \lambda_i x_i.$$

Optionally, the  $x_i$  also may be computed.

**Usage****SCILIB:**

```

INTEGER*8 ldax,n,job,ier
REAL*8 a(ldax,n),w(n),x(ldax,n),work1(n),work2(n)
CALL RS (ldax, n, a, w, job, x, work1, work2, ier)

```

**Input**

**ldax** The leading dimension of arrays **a** and **x** as declared in the calling program unit, with  $\text{ldax} \geq \max(n, 1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**a** Array containing the diagonal and lower triangle of the  $n$  by  $n$  matrix  $A$ . Elements in the strict upper triangle are not referenced.

**job** Option flag:

```

job = 0 compute eigenvalues only
job ≠ 0 compute eigenvalues and eigenvectors

```

**Working storage**

**work1** Array of size  $n$ , used for work space.

**work2** Array of size  $n$ , used for work space.

**Output**

**a** The lower triangle is destroyed if **job** = 0. Not modified if **job** ≠ 0.

**w** The eigenvalues  $\lambda_i$  of  $A$  in ascending order if **ier** = 0 is returned.

**x** Not referenced if eigenvectors are not requested. In this case, **x** can be a dummy variable. Otherwise, eigenvectors of  $A$  if **ier** = 0 is returned. The  $j$ th column of **x** contains the eigenvector  $x_j$  of  $A$  corresponding to the eigenvalue in **w**( $j$ ),  $j = 1, 2, \dots, n$ . Eigenvectors are normalized to have Euclidean length = 1.

|                           |                                                                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ier</b>                | Status response:                                                                                                                                                                                                                                                                                                                            |
| <b>ier = 0</b>            | Normal return.                                                                                                                                                                                                                                                                                                                              |
| <b>ier = k, 1 ≤ k ≤ n</b> | if calculation of the <i>k</i> th eigenvalue failed to converge. <b>w(1), w(2), ..., w(k-1)</b> are eigenvalues, but are not necessarily the smallest and are not necessarily sorted. If eigenvectors are requested, the first <i>k-1</i> columns of <b>x</b> are eigenvectors corresponding to the first <i>k-1</i> elements of <b>w</b> . |
| <b>ier = 10n</b>          | if <b>n &gt; ldax</b> . No eigenvalues or eigenvectors are returned.                                                                                                                                                                                                                                                                        |

**Notes** This subprogram is usage-compatible with the standard single-precision EISPACK subprogram with the same name. It calls EISPACK subprograms TRED1 and TQLRAT or TRED2 and TQL2, which are documented elsewhere in this chapter.

**Example 1** Compute eigenvalues of a 6 by 6 REAL\*8 symmetric matrix *A* whose diagonal and lower triangle are stored in array *A* whose dimensions are 10 by 10. Eigenvalues are stored in array *W* of dimension 10.

```

INTEGER*8 LDA,N,JOB,IER
REAL*8 A(10,10),W(10),X,WORK1(10),WORK2(10)
LDA = 10
N = 6
JOB = 0
CALL RS (LDA,N,A,W,JOB,X,WORK1,WORK2,IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF

```

**Example 2** Compute eigenvalues and eigenvectors of a 6 by 6 REAL\*8 symmetric matrix *A* whose diagonal and lower triangle are stored in array *A* whose dimensions are 10 by 10. Eigenvalues are stored in array *W* of dimension 10; eigenvectors are stored in the first six columns of array *X* of dimension 10 by 10.

```

INTEGER*8 LDAX,N,JOB,IER
REAL*8 A(10,10),W(10),X(10,10),WORK1(10),WORK2(10)
LDAX = 10
N = 6
JOB = 1
CALL RS (LDAX,N,A,W,JOB,X,WORK1,WORK2,IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF

```

**Eigenvalues and Eigenvectors of a Real Symmetric Matrix****TQL2**

**Purpose** This subprogram computes eigenvalues and eigenvectors of a tridiagonal real symmetric  $n$  by  $n$  matrix. Eigenvalues and eigenvectors of a full real symmetric matrix can also be computed by this subprogram if TRED2 has been used to reduce the full matrix to tridiagonal form.

Specifically, given a tridiagonal real symmetric matrix  $A$  or output of TRED2 applied to a full real symmetric matrix  $A$ , this subprogram determines scalars,  $\lambda_i$ ,  $i = 1, 2, \dots, n$ , and nonzero vectors,  $x_i$ ,  $i = 1, 2, \dots, n$ , such that

$$Ax_i = \lambda_i x_i.$$

**Usage** **SCILIB:**

```

INTEGER*8 ldx,n,ier
REAL*8 d(n),e(n),x(ldx,n)
CALL TQL2 (ldx, n, d, e, x, ier)

```

**Input**

**ldx** The leading dimension of array  $x$  as declared in the calling program unit, with  $ldx \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**d** Array containing the diagonal elements of the  $n$  by  $n$  symmetric tridiagonal matrix  $A$ .

**e** Array containing the subdiagonal elements of  $A$  in elements  $e(2)$  through  $e(n)$ .  $e(1)$  is not used as input.

**x** If  $A$  is a tridiagonal matrix,  $x$  must be initialized to the  $n$  by  $n$  identity matrix. If  $A$  is a full matrix,  $x$  contains the transformation matrix produced by TRED2 in reducing the full matrix to tridiagonal form.

**Output**

**d** Eigenvalues,  $\lambda_i$ ,  $i = 1, 2, \dots, n$ , of  $A$  overwrite the input if  $ier = 0$  is returned. Eigenvalues have been sorted into ascending order.

**e** Destroyed.

**x** Eigenvectors of  $A$  if  $ier = 0$  is returned. The  $j$ th column of  $x$  is the eigenvector  $x_j$  of  $A$ , corresponding to the eigenvalue in  $w(j)$ ,  $j = 1, 2, \dots, n$ . Eigenvectors are normalized to have Euclidean length = 1.

**ier** Status response:

**ier = 0** Normal return.

**ier = k,  $1 \leq k \leq n$**  if calculation of the  $k$ th eigenvalue failed to converge.  $w(1), w(2), \dots, w(k-1)$  are eigenvalues, but are not necessarily the smallest and are not necessarily sorted. The first  $k-1$  columns of  $x$  are eigenvectors corresponding to the first  $k-1$  elements of  $w$ .

**Notes** This subprogram is usage compatible with the standard single-precision EISPACK subprogram with the same name.

**Example 1** Compute eigenvalues and eigenvectors of a 6 by 6 tridiagonal REAL\*8 symmetric matrix  $A$  whose diagonal and lower subdiagonal are stored in arrays  $D$  and  $E$  of dimension 10. Eigenvalues are returned in array  $D$ ; eigenvectors are placed in the first six columns of array  $X$  of dimension 10 by 10.

```

 INTEGER*8 LDX,N, IER
 REAL*8 D(10),E(10),X(10,10)
 LDX = 10
 N = 6
 DO J = 1, N
 DO I = 1, N
 X(I, J) = 0.0
 END DO
 X(J, J) = 1.0
 END DO
 CALL TQL2 (LDX,N,D,E,X, IER)
 IF (IER .NE. 0) THEN
 handle convergence failure
 END IF

```

**Example 2** Compute eigenvalues and eigenvectors of a 6 by 6 REAL\*8 symmetric matrix  $A$  whose diagonal and lower triangle are stored in array  $A$  whose dimensions are 10 by 10. Eigenvalues are stored in array  $W$  of dimension 10; eigenvectors are stored in the first six columns of array  $X$  of dimension 10 by 10. (Compare with "Example 2" in the description of RS.)

```

 INTEGER*8 LDAX,N, IER
 REAL*8 A(10,10),W(10),X(10,10),WORK(10)
 LDAX = 10
 N = 6
 CALL TRED2 (LDAX,N,A,W,WORK,X)
 CALL TQL2 (LDAX,N,W,WORK,X, IER)
 IF (IER .NE. 0) THEN
 handle convergence failure
 END IF

```

**Eigenvalues of a Real Symmetric Matrix****TQLRAT**

**Purpose** This subprogram computes the eigenvalues of a tridiagonal real symmetric  $n$  by  $n$  matrix. Eigenvalues of a full real symmetric matrix can also be computed by this subprogram if TRED1 has been used to reduce the full matrix to tridiagonal form.

Specifically, given a tridiagonal real symmetric matrix  $A$  or output of TRED1 applied to a full real symmetric matrix  $A$ , this subprogram determines scalars,  $\lambda_i$ ,  $i = 1, 2, \dots, n$ , for which there exist corresponding nonzero vectors,  $x_i$ ,  $i = 1, 2, \dots, n$ , such that

$$Ax_i = \lambda_i x_i.$$

**Usage** SCILIB:  
**INTEGER\*8** n,ier  
**REAL\*8** d(n),e2(n)  
**CALL TQLRAT (n, d, e2, ier)**

**Input**

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**d** Array containing diagonal elements of the  $n$  by  $n$  symmetric tridiagonal matrix  $A$ .

**e2** Array containing squares of subdiagonal elements of  $A$  in elements **e2(2)** through **e2(n)**. **e2(1)** is not used as input.

**Output**

**d** Eigenvalues,  $\lambda_i$ ,  $i = 1, 2, \dots, n$ , of  $A$  overwrite the input if **ier** = 0 is returned. Eigenvalues have been sorted into ascending order.

**e2** Destroyed.

**ier** Status response:

**ier** = 0 Normal return.  
**ier** =  $k \neq 0$  If calculation of the  $k$ th eigenvalue failed to converge. **w(1)**, **w(2)**, ..., **w(k-1)** are eigenvalues, but are not necessarily the smallest ones.

**Notes** This subprogram is usage-compatible with the standard single-precision EISPACK subprogram with the same name.

**Example 1** Compute eigenvalues of a 6 by 6 tridiagonal REAL\*8 symmetric matrix  $A$  whose diagonal is stored in array D of dimension 10. Squares of the lower subdiagonal elements of  $A$  are stored in array E2, also of dimension 10. The eigenvalues are returned in array D.

```

INTEGER*8 N, IER
REAL*8 D(10), E2(10)
N = 6
CALL TQLRAT (N, D, E2, IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF

```

**Example 2** Compute eigenvalues of a 6 by 6 REAL\*8 symmetric matrix  $A$  whose diagonal and lower triangle are stored in array  $A$  whose dimensions are 10 by 10. Eigenvalues are stored in array  $W$  of dimension 10. (Compare with "Example 1" in the description of RS.)

```
INTEGER*8 LDA,N,IER
REAL*8 A(10,10),W(10),WORK1(10),WORK2(10)
LDA = 10
N = 6
CALL TRED1 (LDA,N,A,W,WORK1,WORK2)
CALL TQLRAT (N,W,WORK2,IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF
```

**Reduce Real Symmetric Matrix to Tridiagonal Form****TRED1**

**Purpose** This subprogram uses orthogonal-similarity transformations to reduce a full real symmetric  $n$  by  $n$  matrix  $A$  to symmetric tridiagonal form without accumulating reduction transformations. The reduced form may be passed to subprogram TQLRAT, documented elsewhere in this chapter, to find the eigenvalues of  $A$ .

Specifically, given  $A$ , this subprogram determines an  $n$  by  $n$  tridiagonal matrix  $T$  that is orthogonally similar to  $A$ , i.e., such that there exists an  $n$  by  $n$  orthogonal matrix  $Q$  for which

$$Q^T A Q = T.$$

**Usage** SCILIB:

```
INTEGER*8 lda,n
REAL*8 a(lda,n),d(n),e(n),e2(n)
CALL TRED1 (lda, n, a, d, e, e2)
```

**Input** **lda** The leading dimension of array **a** as declared in the calling program unit, with  $lda \geq \max(n,1)$ .

**n** The order of matrix  $A$ ,  $n \geq 0$ .

**a** Array containing the diagonal and lower triangle of the  $n$  by  $n$  matrix  $A$ . Elements in the strict upper triangle are not referenced.

**Output** **a** The diagonal and lower triangle are destroyed.

**d** Array containing diagonal elements of the tridiagonal matrix  $T$ .

**e** Array containing the subdiagonal elements of  $T$  in elements **e(2)** through **e(n)**. **e(1) = 0**.

**e2** Array containing squares of subdiagonal elements of  $T$  in elements **e(2)** through **e(n)**. **e2(1) = 0**.

**Notes** This subprogram is usage-compatible with the standard single-precision EISPACK subprogram with the same name.

Output arrays **e** and **e2** are redundant. Some EISPACK subprograms that can be used following TRED1 require **e** as input and some require **e2**.

**Example 1** Reduce the 6 by 6 REAL\*8 symmetric matrix  $A$  whose diagonal and lower triangle are stored in array  $A$  whose dimensions are 10 by 10 to tridiagonal form.

```
INTEGER*8 LDA,N
REAL*8 A(10,10),D(10),E(10),E2(10)
LDA = 10
N = 6
CALL TRED1 (LDA,N,A,D,E,E2)
```

**Example 2** Compute eigenvalues of a 6 by 6 REAL\*8 symmetric matrix *A* whose diagonal and lower triangle are stored in array *A* whose dimensions are 10 by 10. Eigenvalues will be stored in array *W* of dimension 10. (Compare with "Example 1" in the description of RS.)

```
INTEGER*8 LDA,N,IER
REAL*8 A(10,10),W(10),WORK1(10),WORK2(10)
LDA = 10
N = 6
CALL TRED1 (LDA,N,A,W,WORK1,WORK2)
CALL TQLRAT (N,W,WORK2,IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF
```

**Reduce Real Symmetric Matrix to Tridiagonal Form****TRED2**

**Purpose** This subprogram uses orthogonal similarity transformations to reduce a full real symmetric  $n$  by  $n$  matrix  $A$  to symmetric tridiagonal form and accumulates reduction transformations. This reduced form and the transformation matrix may be passed to subprogram TQL2, documented elsewhere in this chapter, to find eigenvalues and eigenvectors of  $A$ .

Specifically, given  $A$ , this subprogram determines an  $n$  by  $n$  orthogonal matrix  $X$  and an  $n$  by  $n$  symmetric tridiagonal matrix  $T$  such that

$$X^T A X = T.$$

**Usage****SCILIB:**

```
INTEGER*8 ldax,n
REAL*8 a(ldax,n),d(n),e(n),x(ldax,n)
CALL TRED2 (ldax, n, a, d, e, x)
```

**Input**

**ldax** The leading dimension of arrays **a** and **x** as declared in the calling program unit, with  $\text{ldax} \geq \max(\text{n}, 1)$ .

**n** The order of matrix  $A$ ,  $\text{n} \geq 0$ .

**a** Array containing the diagonal and lower triangle of the  $\text{n}$  by  $\text{n}$  matrix  $A$ . Elements in the strict upper triangle are not referenced.

**Output**

**d** Array containing diagonal elements of the tridiagonal matrix  $T$ .

**e** Array containing subdiagonal elements of  $T$  in elements  $e(2)$  through  $e(\text{n})$ .  $e(1) = 0$ .

**x** The transformation matrix  $X$  that reduces  $A$  to tridiagonal form.

**Notes**

This subprogram is usage-compatible with the standard single-precision EISPACK subprogram with the same name.

**Example 1**

Reduce the 6 by 6 REAL\*8 symmetric matrix  $A$  whose diagonal and lower triangle are stored in array **A** whose dimensions are 10 by 10 to tridiagonal form and accumulate the transformation matrix.

```
INTEGER*8 LDAX, N
REAL*8 A(10,10),D(10),E(10),X(10,10)
LDAX = 10
N = 6
CALL TRED2 (LDAX, N, A, D, E, X)
```

**Example 2** Compute eigenvalues and eigenvectors of a 6 by 6 REAL\*8 symmetric matrix *A* whose diagonal and lower triangle are stored in array *A* whose dimensions are 10 by 10. Eigenvalues will be stored in array *W* of dimension 10; eigenvectors will be stored in the first six columns of array *X* of dimension 10 by 10. (Compare with "Example 2" in the description of RS.)

```
INTEGER*8 LDAX,N,IER
REAL*8 A(10,10),W(10),X(10,10),WORK(10)
LDAX = 10
N = 6
CALL TRED2 (LDAX,N,A,W,WORK,X)
CALL TQL2 (LDAX,N,W,WORK,X,IER)
IF (IER .NE. 0) THEN
 handle convergence failure
END IF
```

**EISPACK Subprograms not in the *SCILIB User's Guide***

Although SCILIB includes all EISPACK subprograms, the following nonoptimized routines are not documented in the *SCILIB User's Guide*. The *EISPACK Guide* and the *EISPACK Guide Extension*, included in the SCILIB documentation set, document these subprograms.

**Table 5-1: EISPACK Subprograms not in the *SCILIB User's Guide***

| Name   | Function                                                                 |
|--------|--------------------------------------------------------------------------|
| BAKVEC | Back Transform Eigenvectors following FIGI                               |
| BALANC | Balance a Real General Matrix                                            |
| BALBAK | Back Transform Eigenvectors following BALANC                             |
| BANDR  | Reduce a Real Symmetric Band Matrix to Real Symmetric Tridiagonal Form   |
| BANDV  | Determine Some Eigenvectors of a Real Symmetric Band Matrix              |
| BISECT | Determine Some Eigenvectors of a Real Symmetric Tridiagonal Matrix       |
| BQR    | Determine Some Eigenvalues of a Real Symmetric Band Matrix               |
| CBABK2 | Back Transform Eigenvectors following CBAL                               |
| CBAL   | Balance a Complex General Matrix                                         |
| CG     | Determine Eigenvalues/vectors of a Complex General Matrix                |
| CH     | Determine Eigenvalues/vectors of a Complex Hermitian Matrix              |
| CINVT  | Determine Some Eigenvectors of a Complex Upper Hessenberg Matrix         |
| COMBAK | Back Transform Eigenvectors following COMHES                             |
| COMHES | Reduce a Complex General Matrix to Complex Upper Hessenberg Form         |
| COMLR  | Determine the Eigenvalues of a Complex Upper Hessenberg Matrix           |
| COMLR2 | Determine the Eigenvalues/vectors of a Complex Hessenberg Matrix         |
| COMQR  | Determine the Eigenvalues of a Complex Upper Hessenberg Matrix           |
| COMQR2 | Determine the Eigenvalues/vectors of a Complex Upper Hessenberg Matrix   |
| CORTB  | Back Transform Eigenvectors following CORTH                              |
| CORTH  | Reduce a Complex General Matrix to Complex Upper Hessenberg Form         |
| ELMBAK | Back Transform Eigenvectors following ELMHES                             |
| ELMHES | Reduce a Real General Matrix to Real Upper Hessenberg Form               |
| ELTRAN | Accumulate the Transformations in the Reduction by ELMHES                |
| FIGI   | Transform a Real Non-symmetric Tridiagonal Matrix to Real Symmetric Form |
| FIGI2  | Transform a Real Non-symmetric Tridiagonal Matrix to Real Symmetric Form |
| HQR    | Determine the Eigenvalues of a Real Upper Hessenberg Matrix              |
| HQR2   | Determine the Eigenvalues/vectors of a Real Upper Hessenberg Matrix      |
| HTRIB3 | Back Transform Eigenvectors following HTRID3                             |
| HTRIBK | Back Transform Eigenvectors following HTRIDI                             |
| HTRID3 | Reduce a Complex Hermitian Matrix to Real Symmetric Tridiagonal Form     |
| HTRIDI | Reduce a Complex Hermitian Matrix to Real Symmetric Tridiagonal Form     |
| IMTQL1 | Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix         |
| IMTQL2 | Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix |
| IMTQLV | Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix         |
| INVIT  | Determine Some Eigenvectors of a Real Upper Hessenberg Matrix            |
| MINFIT | Solve a Least Squares Problem with a Real Rectangular Coefficient Matrix |
| ORTBAK | Back Transform Eigenvectors following ORTHES                             |
| ORTHES | Reduce a Real General Matrix to Real Upper Hessenberg Form               |

| Name   | Function                                                                       |
|--------|--------------------------------------------------------------------------------|
| ORTRAN | Accumulate the Transformations in the Reduction by ORTHES                      |
| QZHES  | Partially Reduce a Real General Generalized Eigenproblem                       |
| QZIT   | Complete the Reduction of a Real General Generalized Eigenproblem              |
| QZVAL  | Determine the Eigenvalues of a Reduced Real General Generalized Eigenproblem   |
| QZVEC  | Determine the Eigenvectors of a Reduced Real General Generalized Eigenproblem  |
| RATQR  | Determine Some Extreme Eigenvalues of a Real Symmetric Tridiagonal Matrix      |
| REBAK  | Back Transform Eigenvectors following REDUC or REDUC2                          |
| REBAKB | Back Transform Eigenvectors following REDUC2                                   |
| REDUC  | Reduce a Real Symmetric Generalized Eigenproblem to Standard Form              |
| REDUC2 | Reduce a Real Symmetric Generalized Eigenproblem to Standard Form              |
| RG     | Determine the Eigenvalues/vectors of a Real General Matrix                     |
| RGG    | Determine the Eigenvalues/vectors of a Real General Generalized Eigenproblem   |
| RSB    | Determine the Eigenvalues/vectors of a Real Symmetric Band Matrix              |
| RSG    | Determine the Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem |
| RSGAB  | Determine the Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem |
| RSGBA  | Determine the Eigenvalues/vectors of a Real Symmetric Generalized Eigenproblem |
| RSM    | Determine All Eigenvalues and Some Eigenvectors of a Real Symmetric Matrix     |
| RSP    | Determine the Eigenvalues/vectors of a Real Symmetric Packed Matrix            |
| RST    | Determine the Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix       |
| RT     | Determine the Eigenvalues/vectors of a Real Tridiagonal Matrix                 |
| SVD    | Compute the Singular Value Decomposition of a Real Rectangular Matrix          |
| TINVIT | Determine Some Eigenvectors of a Real Symmetric Tridiagonal Matrix             |
| TQL1   | Determine the Eigenvalues of a Real Symmetric Tridiagonal Matrix               |
| TRBAK1 | Back Transform Eigenvectors following TRED1                                    |
| TRBAK3 | Back Transform Eigenvectors following TRED3                                    |
| TRED3  | Reduce a Real Symmetric Matrix to Real Symmetric Tridiagonal Form              |
| TRIDIB | Determine Some Eigenvalues of a Real Symmetric Tridiagonal Matrix              |
| TSTURM | Determine Some Eigenvalues/vectors of a Real Symmetric Tridiagonal Matrix      |

# Fast Fourier Transforms

## Overview

This chapter explains how to use the SCILIB Fast Fourier Transform (FFT) subprograms. The operations covered are

- one-dimensional complex-to-complex FFT subprograms
- one-dimensional real-to-complex and complex-to-real FFT subprograms
- one-dimensional simultaneous complex-to-complex FFT subprograms
- one-dimensional simultaneous real-to-complex and complex-to-real FFT subprograms

## Chapter Objectives

After reading this chapter you will

- understand the SCILIB FFT subprogram restrictions
- know how to augment subprograms with zero-value data points
- know how to use the described subprograms

## What You Need to Know to Use These Subprograms

Strictly speaking, an FFT is not a type of transform but a class of algorithms for efficiently computing the discrete Fourier transform (DFT). Although the DFT is defined for any number of data points, the SCILIB FFT subprograms restrict the number of points to certain forms. For single one-dimensional transforms, the number of points must be a power of two:

$$l = 2^p, \quad p \geq 0.$$

For simultaneous transforms, the number of points in each direction must be a product of powers of two, three, and five:

$$l = 2^p 3^q 5^r, \quad p, q, r \geq 0.$$

While these restrictions limit the utility of the subprograms, the gain in speed is enormous. You can frequently adapt your data set to the SCILIB FFT subprograms by augmenting it with enough zero-value data points to reach the next acceptable number of points. Doing so slightly changes the problem, which may or may not be important, depending on the problem. For example, adding zero-value points to a time series changes the implied sampling frequency, but adding zero-value points to data sets before using FFT subprograms to compute convolutions does not change the result.

## Supplemental Reading

Brigham, E.O. 1974. *The Fast Fourier Transform*. Prentice-Hall, New Jersey.

Rabiner, L.R., and B. Gold. 1975. *Theory and Application of Digital Signal Processing*. Prentice-Hall, New Jersey.

## Subprogram Descriptions

|                                                           |      |
|-----------------------------------------------------------|------|
| One-Dimensional Complex-to-Complex FFT<br>CFFT2 .....     | 6-3  |
| Simultaneous One-Dimensional FFT<br>CFTFAX, CFFTMLT ..... | 6-5  |
| Complex-to-Real One-Dimensional FFT<br>CRFFT2 .....       | 6-9  |
| Real-to-Complex One-Dimensional FFT<br>RCFFT2 .....       | 6-11 |
| Simultaneous One-Dimensional FFT<br>FFTFAX, RFFTMLT ..... | 6-13 |

## One-Dimensional Complex-to-Complex FFT

CFFT2

**Purpose** Given an array of complex data, this subprogram computes the one-dimensional unscaled forward or inverse discrete Fourier transform using a radix 2 fast Fourier transform (FFT) algorithm.

The one-dimensional unscaled forward discrete Fourier transform of  $z(n)$ , for  $n = 1, 2, \dots, l$ , is defined by

$$Z(m) = \sum_{n=1}^l z(n) e^{-2\pi i(m-1)(n-1)/l}$$

for  $m = 1, 2, \dots, l$  and  $i = \sqrt{-1}$ .

Alternatively, the one-dimensional unscaled inverse discrete Fourier transform of  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is defined by

$$z(n) = \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

for  $n = 1, 2, \dots, l$ .

CFFT2 requires that  $l$  be a power of 2, i.e., of the form  $l = 2^p$ , where  $p \geq 0$ .

**Usage** Because it is common to use one data set length repetitively, this subprogram has a separate initialization call so that the setup can be performed only once for each different transform size. You will, therefore, always have at least two **CALL** statements to the FFT subprogram, using the same working storage array. Refer to "Example."

**SCILIB:**

```

INTEGER*8 init,isgn,l
COMPLEX*16 zin(1),zout(1)
REAL*8 work(5*1)
CALL CFFT2 (init, isgn, l, zin, work, zout)

```

**Input**

**init** Initialization flag:

**init**  $\neq 0$  initialize **work** for subsequent transforms of length **l**.  
**init**  $= 0$  compute transform.

**isgn** Operation flag: if **init**  $= 0$ ,

**isgn**  $< 0$  compute unscaled forward transform.  
**isgn**  $> 0$  compute unscaled inverse transform.

The sign of the exponential is the same as the sign of **isgn**.

**l** Number of data points, of the form  $l = 2^p$ , with  $p \geq 0$ .

**zin** Array of data to be transformed. Not used if **init**  $\neq 0$ . **zin** may be equivalenced to **work**, in which case the input values may be overwritten.

**Working Storage**

**work** A work array of size  $5 * l$ . If **init**  $\neq 0$ , **work** is initialized for computing transforms of length **l**. If **init**  $= 0$ , **work** must have been initialized by a previous call with this value of **l** in which **init**  $\neq 0$ .

**Output**     **zout**     Array of transformed data if **init** = 0. Not used if **init** ≠ 0.

**Notes**       It is usual to scale the inverse transform by multiplying the summation by 1/l so that the inverse transform of the forward transform of a data set returns the original data. This subprogram omits this scaling, meaning that the inverse transform of the forward transform of a data set is the original data multiplied by l.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**init** = 0 and **isgn** = 0;  
l not a power of 2.

**Example**       Compute the forward discrete Fourier transform of two COMPLEX\*16 data sets of length 1024. The length of working storage is  $5 \times 1024 = 5120$ .

```

 INTEGER*8 INIT,ISGN,L
 COMPLEX*16 ZIN1(1024),ZOUT1(1024),ZIN2(1024),ZOUT2(1024)
 REAL*8 WORK(5120)
 L = 1024
 INIT = 1
 CALL CFFT2 (INIT,ISGN,L,ZIN1,WORK,ZOUT1) ! INITIALIZE
 INIT = 0
 ISGN = -1
 CALL CFFT2 (INIT,ISGN,L,ZIN1,WORK,ZOUT1) ! FIRST TRANSFORM
 CALL CFFT2 (INIT,ISGN,L,ZIN2,WORK,ZOUT2) ! SECOND TRANSFORM

```

**Purpose** Given a number of sets of one-dimensional complex data with real and imaginary parts in separate real arrays, subroutine CFFTMLT computes all of their one-dimensional forward or inverse discrete Fourier transforms using a radix 2-3-5 fast Fourier transform (FFT) algorithm.

The one-dimensional forward discrete Fourier transform of a complex set of data  $z(n)$ , for  $n = 1, 2, \dots, l$ , is defined by

$$Z(m) = \sum_{n=1}^l z(n) e^{-2\pi i(m-1)(n-1)/l}$$

for  $m = 1, 2, \dots, l$  and  $i = \sqrt{-1}$ .

Alternatively, the one-dimensional scaled inverse discrete Fourier transform of  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is defined by

$$z(n) = \frac{1}{l} \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

for  $n = 1, 2, \dots, l$ .

This subprogram performs forward or scaled inverse transform operations simultaneously on a number of data sets. It requires that the length  $l$  of the data sets be a product of powers of 2, 3, and 5, i.e., of the form

$$l = 2^p 3^q 5^r,$$

where  $p, q, r \geq 0$ . Refer to "Notes" for a partial list of permissible values of  $l$ .

The complex data,  $z$  or  $Z$ , are stored with real and imaginary parts in separate real arrays,  $x$  and  $y$ , respectively.

**Usage** Because it is common to use one data set length repetitively, subroutine CFFTMLT has a separate initialization subprogram, CFTFAX, so that the setup can be performed only once for each different transform size. You will, therefore, always have at least one CALL CFTFAX statement and at least one CALL CFFTMLT statement, using the same working storage arrays. Refer to "Example."

**SCILIB:**

```
INTEGER*8 work3(19),incl,incn,l,n,isgn
REAL*8 x(lenxy),y(lenxy),work1(4*1*n),work2(2*1)
```

Initialization call:

```
CALL CFTFAX (l, work3, work2)
```

Transform call:

```
CALL CFFTMLT (x, y, work1, work2, work3, incl, incn, l, n, isgn)
```

**Input**      **x** and **y** Arrays containing **n** data sets, each consisting of **l** data points, to be transformed. Typically, **x** and **y** will be two- or three-dimensional arrays with each data set being a one-dimensional array section. Refer to "Notes" for suggested usages.

Treating **x** and **y** as one-dimensional arrays results in

$$\text{lenxy} = (l-1) \times \text{incl} + (n-1) \times \text{incn} + 1.$$

The real and imaginary parts of the *i*th data point of the *j*th data set,  $1 \leq i \leq l$ ,  $1 \leq j \leq n$ , are stored in

$$x((i-1) \times \text{incl} + (j-1) \times \text{incn} + 1)$$

and

$$y((i-1) \times \text{incl} + (j-1) \times \text{incn} + 1),$$

respectively.

**incl**      Storage increment between successive elements of the same data set, **incl** > 0. Use **incl** = 1 if each data set is stored contiguously in **x** and **y**.

**incn**      Storage increment between corresponding data points of successive data sets, **incn** > 0.

**l**          Number of data points in each data set, of the form  $l = 2^p 3^q 5^r$ , with  $p, q, r \geq 0$ .

**n**          The number of data sets, **n** > 0.

**isgn**      Operation flag:

**isgn** = -1    compute forward transform.

**isgn** = +1    compute inverse transform.

The sign of the exponential is the same as the sign of **isgn**.

**Working Storage**      **work1**    A work array of size  $4 * l * n$ .

**work2**    A work array of size  $2 * l$ , initialized by CFTFAX for use in CFFTMLT.

**work3**    A work array of size 19, initialized by CFTFAX for use in CFFTMLT. CFTFAX returns **work3**(1) = -99 if **l** is not factorable as specified above.

**Output**      **x** and **y**    The transformed data replaces the input.

## Notes

Typically, **x** and **y** will be two- or three-dimensional arrays with each data set being a one-dimensional section of the arrays, i.e., all but one subscript will be constant within a data set.

If **x** and **y** are two-dimensional arrays of dimension **ldxy** by **mdxy**, and if the data sets are stored in the columns of **x** and **y**, then  $1 \leq \text{ldxy}$ ,  $n \leq \text{mdxy}$ ,  $\text{incl} = 1$ , and  $\text{incn} = \text{ldxy}$ . For example:

```
CALL CFFTMLT (x, y, work1, work2, work3, 1, ldxy, 1, n, isgn)
```

If **x** and **y** are two-dimensional arrays as above and data sets are stored in rows of **x** and **y**,  $1 \leq \text{mdxy}$ ,  $n \leq \text{ldxy}$ ,  $\text{incl} = \text{ldxy}$ , and  $\text{incn} = 1$ . For example:

```
CALL CFFTMLT (x, y, work1, work2, work3, ldxy, 1, 1, n, isgn)
```

The subprograms generally are faster if the data sets are the rows of the arrays, so that  $\text{incn} = 1$ , rather than the columns. Otherwise, it is generally better to have an odd value of  $\text{incn}$ .

If **x** and **y** are three-dimensional arrays of dimension **ldxy** by **mdxy** by **ndxy**, then  $\text{incl}$  and  $\text{incn}$  will usually be 1, **ldxy**, or **ldxy** × **mdxy**, depending on which of the subscripts of the three-dimensional array varies within a data set, which subscript varies between data sets, and which remains constant. Specifically, if the subscript that varies within a data set is the

- 1st subscript, use  $\text{incl} = 1$ .
- 2nd subscript, use  $\text{incl} = \text{ldxy}$ .
- 3rd subscript, use  $\text{incl} = \text{ldxy} \times \text{mdxy}$ .

Similarly, if the subscript that varies between data sets is the

- 1st subscript, use  $\text{incn} = 1$ .
- 2nd subscript, use  $\text{incn} = \text{ldxy}$ .
- 3rd subscript, use  $\text{incn} = \text{ldxy} \times \text{mdxy}$ .

If an error in the arguments is detected, CFFTMLT calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

```
work3(1) = -99,
incl ≤ 0,
incn ≤ 0,
l not of the form 2p 3q 5r for p,q,r ≥ 0,
n < 0, and
isgn ≠ ± 1.
```

The following list indicates some of the permissible values of **l**. Although **l** can be any value of the form  $2^p 3^q 5^r$  where  $p, q, r \geq 0$ , this list only shows values not exceeding 1000:

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, 32, 36, 40, 45, 48, 50, 54, 60, 64, 72, 75, 80, 81, 90, 96, 100, 108, 120, 125, 128, 135, 144, 150, 160, 162, 180, 192, 200, 216, 225, 240, 243, 250, 256, 270, 288, 300, 320, 324, 360, 375, 384, 400, 405, 432, 450, 480, 486, 500, 512, 540, 576, 600, 625, 640, 648, 675, 720, 729, 750, 768, 800, 810, 864, 900, 960, 972, and 1000.

**Example 1** Compute the forward discrete Fourier transform of 256 complex data sets of length 1024. Real and imaginary parts of data sets are stored as columns of arrays X and Y whose dimensions are 1025 by 256.

```

INTEGER*8 WORK3(19), INCL, INCN, L, N, ISGN
REAL*8 X(1025,256), Y(1025,256), WORK1(1048576), WORK2(512)
L = 1024
INCL = 1
N = 256
INCN = 1025
ISGN = -1
CALL CFTFAX (L, WORK3, WORK2)
IF (WORK3(1) .EQ. -99) THEN
 handle error condition
END IF
CALL CFFTMLT (X, Y, WORK1, WORK2, WORK3, INCL, INCN, L, N, ISGN)

```

**Example 2** Compute the inverse discrete Fourier transform of 1024 complex data sets of length 256. Real and imaginary parts of data sets are stored as rows of arrays X and Y whose dimensions are 1025 by 256.

```

INTEGER*8 WORK3(19), INCL, INCN, L, N, ISGN
REAL*8 X(1025,256), Y(1025,256), WORK1(1048576), WORK2(512)
L = 256
INCL = 1025
N = 1024
INCN = 1
ISGN = 1
CALL CFTFAX (L, WORK3, WORK2)
IF (WORK3(1) .EQ. -99) THEN
 handle error condition
END IF
CALL CFFTMLT (X, Y, WORK1, WORK2, WORK3, INCL, INCN, L, N, ISGN)

```

## Complex-to-Real One-Dimensional FFT

## CRFFT2

**Purpose** A complex sequence  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is conjugate-symmetric about  $Z(l/2+1)$  if

$$\text{Im}(Z(1)) = \text{Im}(Z(l/2 + 1)) = 0$$

and

$$Z(l/2+1 + m) = \bar{Z}(l/2+1 - m), \quad m = 1, 2, \dots, l/2-1,$$

where  $\bar{Z}$  is the complex conjugate of  $Z$ .

Given an array of conjugate-symmetric complex data, this subprogram computes the one-dimensional, complex-to-real, unscaled forward or unscaled inverse, discrete Fourier transform using a radix 2 fast Fourier transform (FFT) algorithm optimized for real output.

The one-dimensional unscaled forward discrete Fourier transform of a data set,  $z(n)$ , for  $n = 1, 2, \dots, l$ , is defined by

$$Z(m) = \sum_{n=1}^l z(n) e^{-2\pi i(m-1)(n-1)/l}$$

for  $m = 1, 2, \dots, l$  and  $i = \sqrt{-1}$ .

Alternatively, the one-dimensional unscaled inverse discrete Fourier transform of the data set  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is defined by

$$z(n) = \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

for  $n = 1, 2, \dots, l$ .

This subprogram requires that  $l$  be a power of 2, i.e., of the form  $l = 2^p$ , where  $p \geq 0$ .

**Usage** Because it is common to use one data set length repetitively, this subprogram has a separate initialization call so that the setup can be performed only once for each different transform size. Therefore, you will always have at least two **CALL** statements to the FFT subprogram, using the same working storage array.

**SCILIB:**

```

INTEGER*8 init,isgn,l
COMPLEX*16 zin(l/2+1)
REAL*8 work(3*l+4),xout(l)
CALL CRFFT2 (init, isgn, l, zin, work, xout)

```

**Input**

**init** Initialization flag:

**init**  $\neq 0$  initialize **work** for subsequent transforms of length **l**.  
**init**  $= 0$  compute transform.

**isgn** Operation flag: if **init**  $= 0$ ,

**isgn**  $< 0$  compute forward transform.  
**isgn**  $> 0$  compute unscaled inverse transform.

The sign of the exponential is the same as the sign of **isgn**.

|                        |             |                                                                                                                                                                                                                                                                                 |
|------------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        | <b>l</b>    | Number of data points, of the form $l = 2^p$ , with $p \geq 0$ .                                                                                                                                                                                                                |
|                        | <b>zin</b>  | Array containing the first $l/2+1$ elements of the data set to be transformed. Not used if <b>init</b> $\neq 0$ .                                                                                                                                                               |
| <b>Working Storage</b> | <b>work</b> | A work array of size $3 * l + 4$ .<br><br>If <b>init</b> $\neq 0$ , <b>work</b> is initialized for computing transforms of length $l$ . If <b>init</b> $= 0$ , <b>work</b> must have been initialized by a previous call with this value of $l$ in which <b>init</b> $\neq 0$ . |
| <b>Output</b>          | <b>xout</b> | Array of transformed data if <b>init</b> $= 0$ . Not used if <b>init</b> $\neq 0$ .                                                                                                                                                                                             |

**Notes** It is usual to scale the inverse transform by multiplying the summation by  $1/l$  so that the inverse transform of the forward transform of a data set returns the original data. This subprogram omits this scaling, meaning that the inverse transform of the forward transform of a data set is the original data multiplied by  $l$ .

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**init**  $= 0$  and **isgn**  $= 0$ ;  
**l** not a power of 2.

**Example** Compute the forward discrete Fourier transform of two conjugate-symmetric COMPLEX\*16 data sets of length 1024. Only the first 513 elements of the input data sets are stored. The length of working storage is  $3*1024+4 = 3076$ .

```

INTEGER*8 INIT,ISGN,L
COMPLEX*16 ZIN1(513),ZIN2(513)
REAL*8 WORK(3076),XOUT1(1024),XOUT2(1024)
L = 1024
INIT = 1
CALL CRFFT2 (INIT,ISGN,L,ZIN1,WORK,XOUT1) ! INITIALIZE
INIT = 0
ISGN = -1
CALL CRFFT2 (INIT,ISGN,L,ZIN1,WORK,XOUT1) ! FIRST TRANSFORM
CALL CRFFT2 (INIT,ISGN,L,ZIN2,WORK,XOUT2) ! SECOND TRANSFORM

```

## Real-to-Complex One-Dimensional FFT

RCFFT2

**Purpose** Given an array of real data, this subprogram computes the one-dimensional, real-to-complex, unscaled forward or inverse, discrete Fourier transform using a radix 2 fast Fourier transform (FFT) algorithm optimized for real input.

The one-dimensional unscaled forward discrete Fourier transform of a data set,  $z(n)$ , for  $n = 1, 2, \dots, l$ , is defined by

$$Z(m) = \sum_{n=1}^l z(n) e^{-2\pi i(m-1)(n-1)/l}$$

for  $m = 1, 2, \dots, l$  and  $i = \sqrt{-1}$ .

When the sequence  $z(n)$  is real, the sequence  $Z(m)$  is conjugate-symmetric about  $Z(l/2+1)$ , i.e.,

$$\text{Im}(Z(1)) = \text{Im}(Z(l/2 + 1)) = 0$$

and

$$Z(l/2+1 + m) = \bar{Z}(l/2+1 - m), \quad m = 1, 2, \dots, l/2-1,$$

where  $\bar{Z}$  is the complex conjugate of  $Z$ .

This subprogram actually computes twice the above quantity:

$$Z(m) = 2 \sum_{n=1}^l z(n) e^{-2\pi i(m-1)(n-1)/l}$$

Alternatively, the one-dimensional unscaled inverse discrete Fourier transform of the data set  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is defined by

$$z(n) = \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

for  $n = 1, 2, \dots, l$ .

Again, twice the above quantity is what is actually computed:

$$z(n) = 2 \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

This subprogram requires that  $l$  be a power of 2, i.e., of the form  $l = 2^p$ , where  $p \geq 0$ .

**Usage**

Because it is common to use one data set length repetitively, this subprogram has a separate initialization call so that the setup can be performed only once for each different transform size. Therefore, you will always have at least two **CALL** statements to the FFT subprogram, using the same working storage array.

**SCILIB:**

```

INTEGER*8 init, isgn, l
REAL*8 xin(1), work(3*1+4)
COMPLEX*16 zout(1/2+1)
CALL RCFFT2 (init, isgn, l, xin, work, zout)

```

|                        |             |                                                                                                                                                                                                                                                                               |
|------------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input</b>           | <b>init</b> | Initialization flag:<br><b>init</b> $\neq$ 0 initialize <b>work</b> for subsequent transforms of length <b>l</b> .<br><b>init</b> = 0 compute transform.                                                                                                                      |
|                        | <b>isgn</b> | Operation flag: if <b>init</b> = 0,<br><b>isgn</b> < 0 compute forward transform.<br><b>isgn</b> > 0 compute unscaled inverse transform.<br><br>The sign of the exponential is the same as the sign of <b>isgn</b> .                                                          |
|                        | <b>l</b>    | Number of data points, of the form $l = 2^p$ , with $p \geq 0$ .                                                                                                                                                                                                              |
|                        | <b>xin</b>  | Array containing the data set to be transformed. Not used if <b>init</b> $\neq$ 0.                                                                                                                                                                                            |
| <b>Working Storage</b> | <b>work</b> | A work array of size $3 * l + 4$ . If <b>init</b> $\neq$ 0, <b>work</b> is initialized for computing transforms of length <b>l</b> . If <b>init</b> = 0, <b>work</b> must have been initialized by a previous call with this value of <b>l</b> in which <b>init</b> $\neq$ 0. |
| <b>Output</b>          | <b>zout</b> | Array containing the first $l/2+1$ elements of transformed data if <b>init</b> = 0. Not used if <b>init</b> $\neq$ 0.                                                                                                                                                         |

**Notes** It is usual to scale the inverse transform by multiplying the summation by  $1/l$  so that the inverse transform of the forward transform of a data set returns the original data. This subprogram replaces this scaling with scaling both forward and inverse transforms by a factor of 2, meaning that the inverse transform of the forward transform of a data set does not return the original data.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are

**isgn** = 0, and  
**l** not a power of 2.

**Example** Compute the forward discrete Fourier transform of two REAL\*8 data sets of length 1024. Only the first 513 elements of the transformed data are computed. The length of working storage is  $3*1024+4 = 3076$ .

```

INTEGER*8 INIT, ISGN, L
REAL*8 XIN1(1024), XIN2(1024), WORK(3076)
COMPLEX*16 ZOUT1(513), ZOUT2(513)
L = 1024
INIT = 1
CALL RCFFT2 (INIT, ISGN, L, XIN1, WORK, ZOUT1) ! INITIALIZE
INIT = 0
ISGN = -1
CALL RCFFT2 (INIT, ISGN, L, XIN1, WORK, ZOUT1) ! FIRST TRANSFORM
CALL RCFFT2 (INIT, ISGN, L, XIN2, WORK, ZOUT2) ! SECOND TRANSFORM

```

**Purpose**

Given a number of one-dimensional real data sets, subroutine RFFTMLT computes the nonredundant portion of all of their one-dimensional forward real-to-complex discrete Fourier transforms using a radix 2-3-5 fast Fourier transform (FFT) algorithm optimized for real input. Alternatively, given the nonredundant parts of a number of conjugate-symmetric one-dimensional complex data sets, subroutine RFFTMLT computes the inverse complex-to-real discrete Fourier transform using a radix 2-3-5 FFT algorithm optimized for real output.

The multiple forward, scaled, real-to-complex Fourier transform of a real set of data  $z(n)$ , for  $n = 1, 2, \dots, l$ , is defined by:

$$Z(m) = \sum_{n=1}^l \frac{1}{l} z(n) e^{-2\pi i(m-1)(n-1)/l}$$

for  $m = 1, 2, \dots, l$ , where  $i = \sqrt{-1}$  and

$Z(1, 2, \dots, l)$  is one of the real input data sets.

The sequence  $Z(m)$  is conjugate-symmetric about  $Z(l/2+1)$ , i.e.,

$$\text{Im}(Z(1)) = \text{Im}(Z(l/2+1)) = 0$$

and

$$Z(l/2+1+m) = \bar{Z}(l/2+1-m), \quad m = 1, 2, \dots, l/2-1,$$

where  $\bar{Z}$  is the complex conjugate of  $Z$ . Therefore, the nonredundant part consists of the first  $l/2+1$  elements of  $Z$ , which is all of  $Z$  that is computed or stored.

Alternatively, if  $Z(m)$ , for  $m = 1, 2, \dots, l$ , is a conjugate-symmetric complex data set, the multiple backward, unscaled complex-to-real Fourier transform of  $Z(m)$  is defined by:

$$z(n) = \sum_{m=1}^l Z(m) e^{+2\pi i(m-1)(n-1)/l}$$

for  $n = 1, 2, \dots, l$ . Only the nonredundant part of  $Z$  is used.

This subprogram performs forward or inverse transform operations simultaneously on a number of data sets. It requires that the length  $l$  of the data sets be a product of powers of 2, 3, and 5, i.e., of the form:

$$l = 2^p 3^q 5^r,$$

where  $p, q, r \geq 0$ , and where either  $l = 1$  or  $l$  is even. Refer to "Notes" for a partial list of permissible values of  $l$ .

**Usage** Because it is common to use one data set length repetitively, subroutine RFFTMLT has a separate initialization subprogram, FFTFAX, so that the setup can be performed only once for each different transform size. You will, therefore, always have at least one **CALL FFTFAX** statement and at least one **CALL RFFTMLT** statement, using the same working storage arrays. Refer to "Example."

**SCILIB:**

```
INTEGER*8 work3(19),incl,incn,l,n,isgn
REAL*8 x(lenx),work1(2*1*n),work2(2*1)
```

Initialization call:

```
CALL FFTFAX (l, work3, work2)
```

Transform call:

```
CALL RFFTMLT (x, work1, work2, work3, incl, incn, l, n, isgn)
```

**Input** **x** Array containing **n** one-dimensional data sets, each consisting of **l** real data points or the first  $l/2+1$  complex data points of a conjugate-symmetric complex data set of length **l**, to be transformed. Typically, **x** is a two- or three-dimensional array with each set of data being a one-dimensional array section. Refer to "Notes" for suggested usages.

Treating **x** as a one-dimensional array results in:

$$\text{lenx} = (l+1) \times \text{incl} + (n-1) \times \text{incn} + 1.$$

For a forward real-to-complex transform, the *i*th real data point of the *j*th data set,  $1 \leq i \leq l$ ,  $1 \leq j \leq n$ , is stored in:

$$x((i-1) \times \text{incl} + (j-1) \times \text{incn} + 1).$$

For an inverse complex-to-real transform, the real part of the *i*th data point of the *j*th data set,  $1 \leq i \leq l/2+1$ ,  $1 \leq j \leq n$ , is stored in:

$$x((2 \times i - 2) \times \text{incl} + (j-1) \times \text{incn} + 1)$$

and the imaginary part is stored in:

$$x((2 \times i - 1) \times \text{incl} + (j-1) \times \text{incn} + 1),$$

respectively.

**incl** Storage increment between successive elements of the same data set, **incl** > 0. Use **incl** = 1 if each data set is stored contiguously in **x**.

**incn** Storage increment between corresponding data points of successive data sets, **incn** > 0.

**l** Number of data points in each complete data set, of the form  $l = 2^p 3^q 5^r$ , with  $q, r \geq 0$  and either  $l = 1$  or  $p \geq 1$ .

**n** The number of data sets, **n** > 0.

**isgn** Option flag:  
**isgn** = -1 compute real-to-complex forward transform.  
**isgn** = +1 compute complex-to-real inverse transform.

**Working Storage**

**work1** A work array of size  $2 * l * n$ .

**work2** A work array of size  $2 * l$ , initialized by FFTFAX for use in RFFTMLT.

Continued

**work3** A work array of size 19, initialized by FFTFAX for use in RFFTMLT. FFTFAX returns **work3**(1) = -99 if *l* is not factorable as specified above.

**Output**     **x**     The transformed data replaces the input.

For a forward real-to-complex transform, the real part of the *i*th output point of the *j*th data set,  $1 \leq i \leq l/2+1$ ,  $1 \leq j \leq n$ , is stored in:

$$x((2 \times i - 2) \times \text{incl} + (j - 1) \times \text{incn} + 1)$$

and the imaginary part is stored in

$$x((2 \times i - 1) \times \text{incl} + (j - 1) \times \text{incn} + 1),$$

respectively. If needed, the remaining  $(l/2 - 1) \times n$  complex output values may be formed by using the conjugate-symmetry condition.

For an inverse complex-to-real transform, the *i*th real output point of the *j*th data set,  $1 \leq i \leq l$ ,  $1 \leq j \leq n$ , is stored in:

$$x((i - 1) \times \text{incl} + (j - 1) \times \text{incn} + 1).$$

**Notes**     Typically, **x** will be a two- or three-dimensional array with each set of data being a one-dimensional section of the array, i.e., all but one subscript will be constant within a data set.

If **x** is a two-dimensional array of dimension **ldx** by **mdx**, and if the data sets are stored in the columns of **x**, then  $1 \leq \text{ldx}$ ,  $n \leq \text{mdx}$ , **incl** = 1, and **incn** = **ldx**. For example:

**CALL RFFTMLT (x, work1, work2, work3, 1, ldx, l, n, isgn)**

If **x** is a two-dimensional array as above and data sets are stored in rows of **x**, then  $1 \leq \text{mdx}$ ,  $n \leq \text{ldx}$ , **incl** = **ldx**, and **incn** = 1. For example:

**CALL RFFTMLT (x, work1, work2, work3, ldx, 1, l, n, isgn)**

The subprograms generally are faster if the data sets are the rows of the arrays, so that **incn** = 1, rather than the columns. Otherwise, it is generally better to have an odd value of **incn**.

If **x** is a three-dimensional array of dimension **ldx** by **mdx** by **ndx**, then **incl** and **incn** will usually be 1, **ldx**, or **ldx** × **mdx**, depending on which of the subscripts of the three-dimensional array varies within a data set, which subscript varies between data sets, and which remains constant. Specifically, if the subscript that varies within a data set is the

- 1st subscript, use **incl** = 1.
- 2nd subscript, use **incl** = **ldx**.
- 3rd subscript, use **incl** = **ldx** × **mdx**.

Similarly, if the subscript that varies between data sets is the

- 1st subscript, use **incn** = 1.
- 2nd subscript, use **incn** = **ldx**.
- 3rd subscript, use **incn** = **ldx** × **mdx**.

If an error in the arguments is detected, RFFTMLT calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

```

work3(1) = -99,
incl ≤ 0,
incn ≤ 0,
l not of the required form 2p 3q 5r, with l = 1 or l even,
n ≤ 0, and
isgn ≠ ± 1.

```

The following list indicates some of the permissible values of l. Although l can be any even value of the form 2<sup>p</sup> 3<sup>q</sup> 5<sup>r</sup> where q, r ≥ 0 and either l = 1 or p ≥ 1, this list only shows the values not exceeding 1000:

1, 2, 4, 6, 8, 10, 12, 16, 18, 20, 24, 30, 32, 36, 40, 48, 50, 54, 60, 64, 72, 80, 90, 96, 100, 108, 120, 128, 144, 150, 160, 162, 180, 192, 200, 216, 240, 250, 256, 270, 288, 300, 320, 324, 360, 384, 400, 432, 450, 480, 486, 500, 512, 540, 576, 600, 640, 648, 720, 750, 768, 800, 810, 864, 900, 960, 972, and 1000.

**Example 1** Compute the forward discrete Fourier transform of 256 real data sets of length 1024, stored as columns of arrays X whose dimensions are 1025 by 256.

```

INTEGER*8 WORK3(19), INCL, INCN, L, N, ISGN
REAL*8 X(1025, 256), WORK1(1048576), WORK2(512)
L = 1024
INCL = 1
N = 256
INCN = 1025
ISGN = -1
CALL FFTFAX (L, WORK3, WORK2)
IF (WORK3(1) .EQ. -99) THEN
 handle error condition
END IF
CALL RFFTMLT (X, WORK1, WORK2, WORK3, INCL, INCN, L, N, ISGN)

```

**Example 2** Compute the inverse discrete Fourier transform of 1024 sets of conjugate-symmetric complex data of length 256, stored as rows of arrays X whose dimensions are 1025 by 256.

```

INTEGER*8 WORK3(19), INCL, INCN, L, N, ISGN
REAL*8 X(1025, 256), WORK1(1048576), WORK2(512)
L = 256
INCL = 1025
N = 1024
INCN = 1
ISGN = 1
CALL FFTFAX (L, WORK3, WORK2)
IF (WORK3(1) .EQ. -99) THEN
 handle error condition
END IF
CALL RFFTMLT (X, WORK1, WORK2, WORK3, INCL, INCN, L, N, ISGN)

```

# Correlation and Convolution Subprograms

## Overview

This chapter explains how to use the SCILIB subprograms available for correlations, convolutions, and related operations such as filtering by means of convolutions.

## Chapter Objectives

After reading this chapter you will

- know how use the described subprograms to compute correlation and convolution

## What You Need to Know to Use These Subprograms

The subprograms presented here can be used to compute both discrete correlations and discrete convolutions. See the specific subprogram descriptions for details.

## Supplemental Reading

Rabiner, L.R., and B. Gold. 1975. *Theory and Application of Digital Signal Processing*. Prentice-Hall, New Jersey.

## Subprogram Descriptions

|                                       |     |
|---------------------------------------|-----|
| Discrete Correlation<br>FILTERG ..... | 7-2 |
| Discrete Correlation<br>FILTERS ..... | 7-4 |

**Purpose** This subprogram computes the fully engaged portion of the discrete correlation of a data vector and a filter vector. It can be used to compute the complete discrete correlation (the fully engaged portion plus the *tails*) by appending zeros to the ends of the data vector. Refer to "Example 2."

If  $f_i, i = 1, 2, \dots, m$ , and  $x_i, i = 1, 2, \dots, n$ , are a filter vector and a data vector, respectively, their discrete correlation  $\tilde{y}_i$  is defined by

$$\tilde{y}_i = \sum_j f_j x_{i+j-1},$$

for  $i = -m+2, -m+3, \dots, n$ , where the sum is taken over all indices  $j$  for which both  $f_j$  and  $x_{i+j-1}$  are defined.

This subprogram computes only the fully engaged portion of the correlation, i.e., the part where the sums have exactly  $\min(m, n)$  terms. Hence, if  $m \leq n$ , it computes

$$y_i = \sum_{j=1}^m f_j x_{i+j-1},$$

for  $i = 1, 2, \dots, n-m+1$ .

The discrete convolution  $\tilde{z}_i$  of the vectors  $f$  and  $x$  is defined by

$$\tilde{z}_i = \sum_j f_{i-j+1} x_j,$$

for  $i = 1, 2, \dots, m+n-1$ , where the sum is taken over all indices  $j$  for which both  $f_{i-j+1}$  and  $x_j$  are defined.

This subprogram computes only the fully engaged portion of the convolution, i.e., the part where the sums have exactly  $\min(m, n)$  terms. Hence, if  $m \leq n$ , it computes

$$z_i = \sum_{j=1}^m f_{m+1-j} x_{i+j-1}$$

for  $i = 1, 2, \dots, n-m+1$ . A comparison of the definitions of  $y_i$  and  $z_i$  shows that the convolution may be computed by storing the  $f$  vectors in reverse order before calling FILTERG.

**Usage****SCILIB:**

```

INTEGER*8 m,n
REAL*8 f(m),x(n),y(n-m+1)
CALL FILTERG (f, m, x, n, y)

```

**Input**

**f** Array containing the filter vector  $f$  of length  $m$ .

**m** The length of the  $f$  vector,  $m > 0$ .

**x** Array containing the data vector  $x$  of length  $n$ .

**n** The length of the  $x$  vector,  $n \geq m$ .

**Output**     **y**           The fully engaged correlation vector  $y$  of length  $n-m+1$ .

**Notes**           To compute the complete correlation vector, including both tails as well as the fully engaged portion, append  $m-1$  zeros to each end of the  $x$  vector. The fully engaged portion of the correlation of the resulting vector is the complete correlation corresponding to the original  $x$  vector. Refer to "Example 2."

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$\begin{aligned} m &\leq 0, \text{ and} \\ n &\leq m. \end{aligned}$$

**Example 1**       Compute the fully engaged portion of the discrete correlation of the REAL\*8 vectors  $f = (2, 1)$  and  $x = (4, 1, 3, 5, 2)$  stored in arrays F and X, respectively. In this instance,  $m = 2$  and  $n = 5$ .

```
INTEGER*8 M,N
REAL*8 F(2),X(5),Y(4)
DATA F / 2.0 , 1.0 /
DATA X / 4.0 , 1.0 , 3.0 , 5.0 , 2.0 /
M = 2
N = 4
CALL FILTERG (F,M,X,N,Y)
```

The result is  $y = (9, 5, 11, 12)$ .

**Example 2**       Compute the complete discrete correlation of the REAL\*8 vectors  $f = (1, 2)$  and  $x = (1, 3, 5)$  stored in arrays F and X, respectively. Thus  $m = 2$ , and to get the complete correlation, we append  $m-1=1$  zero to each end of  $x$ , getting  $\bar{x} = (0, 1, 3, 5, 0)$  and  $n = 5$ .

```
INTEGER*8 M,N
REAL*8 F(2),X(5),Y(4)
DATA F / 1.0 , 2.0 /
DATA X / 0.0 , 1.0 , 3.0 , 5.0 , 0.0 /
M = 2
N = 4
CALL FILTERG (F,M,X,N,Y)
```

The result is  $y = (2, 7, 13, 5)$ .

**Purpose**

This subprogram computes the fully engaged portion of the discrete correlation of a data vector and a symmetric filter vector. It can be used to compute the complete discrete correlation (the fully engaged portion plus the *tails*) by appending zeros to the ends of the data vector. Refer to "Example 2."

If  $f_i, i = 1, 2, \dots, m$ , and  $x_i, i = 1, 2, \dots, n$ , are a filter vector and a data vector, respectively, their discrete correlation  $\tilde{y}_i$  is defined by

$$\tilde{y}_i = \sum_j f_j x_{i+j-1}$$

for  $i = -m+2, -m+3, \dots, n$ , where the sum is taken over all indices  $j$  for which both  $f_j$  and  $x_{i+j-1}$  are defined.

The filter vector is assumed to be symmetric, i.e.,  $f_i = f_{m+1-i}, i = 1, 2, \dots, [m/2]$ , so only the first  $[m/2]$  elements of  $f$  must be stored.

This subprogram computes only the fully engaged portion of the correlation, i.e., the part where the sums have exactly  $\min(m, n)$  terms. Hence, if  $m \leq n$ , it computes

$$y_i = \begin{cases} f_{(m+1)/2} x_{i+(m+1)/2} + \sum_{j=1}^{(m-1)/2} f_j (x_{i+j-1} + x_{i+m-j}), & \text{if } m \text{ odd} \\ \sum_{j=1}^{m/2} f_j (x_{i+j-1} + x_{i+m-j}), & \text{if } m \text{ even} \end{cases}$$

for  $i = 1, 2, \dots, n-m+1$ .

The fully engaged portion of the discrete convolution  $z_i$  of the vectors  $f$  and  $x$  is defined by

$$z_i = \sum_{j=1}^m f_{m+1-j} x_{i+j-1}$$

for  $i = 1, 2, \dots, n-m+1$ . Because  $f$  is a symmetric vector,  $y_i = z_i$  for  $i = 1, 2, \dots, n-m+1$ ; i.e., the fully engaged discrete correlation is identical to the fully engaged discrete convolution.

**Usage****SCILIB:**

```
INTEGER*8 m,n
REAL*8 f((m+1)/2),x(n),y(n-m+1)
CALL FILTERS (f, m, x, n, y)
```

**Input**

- f** Array containing the first  $(m+1)/2$  elements of the filter vector  $f$  of length  $m$ .
- m** The length of the  $f$  vector,  $m > 0$ .

- x**        Array containing the data vector  $x$  of length  $n$ .
- n**        The length of the  $x$  vector,  $n \geq m$ .
- Output**    **y**        The fully engaged correlation vector  $y$  of length  $n-m+1$ .
- Notes**     To compute the complete correlation vector, including both tails as well as the fully engaged portion, append  $m-1$  zeros to each end of the  $x$  vector. The fully engaged portion of the correlation of the resulting vector is the complete correlation corresponding to the original  $x$  vector.

If an error in the arguments is detected, the subprogram calls the error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$m \leq 0, \text{ and} \\ n \leq m.$$

Refer to "Example 2."

- Example 1**    Compute the fully engaged portion of the discrete correlation of the REAL\*8 vectors  $f = (-1, 2, -1)$  and  $x = (4, 1, 3, 5, 2)$  stored in arrays F and X, respectively. In this instance,  $m = 3$  and  $n = 5$ .

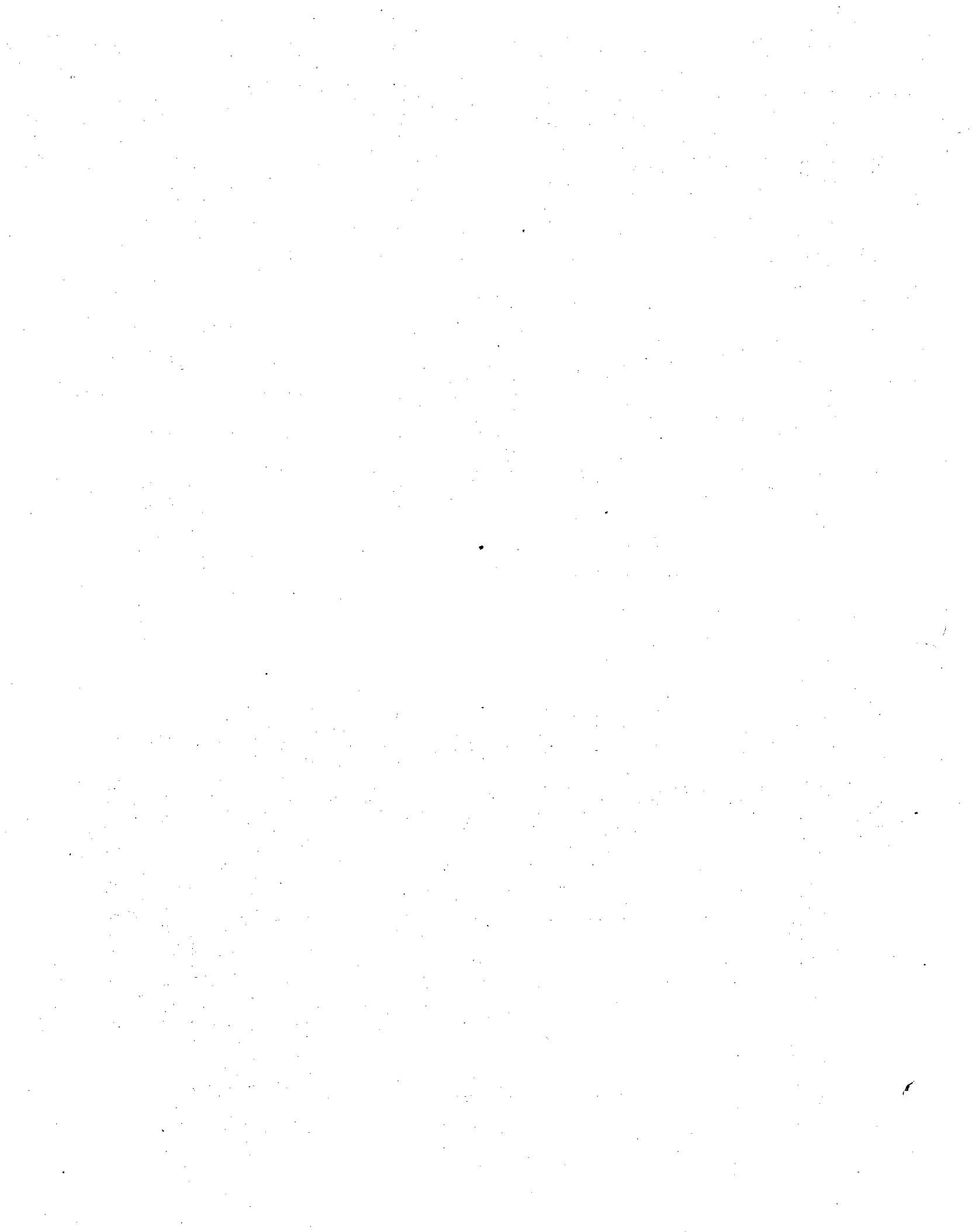
```
INTEGER*8 M,N
REAL*8 F(2),X(5),Y(3)
DATA F / -1.0 , 4.0 /
DATA X / 4.0 , 1.0 , 3.0 , 5.0 , 2.0 /
M = 3
N = 4
CALL FILTERS (F,M,X,N,Y)
```

The result is  $y = (-3, 6, 15)$ .

- Example 2**    Compute the complete discrete correlation of the REAL\*8 vectors  $f = (1, 2, 1)$  and  $x = (1, 3, 5)$  stored in arrays F and X, respectively. Thus  $m = 3$ , and to get the complete correlation, we append  $m-1 = 2$  zeros to each end of  $x$ , getting  $\bar{x} = (0, 0, 1, 3, 5, 0, 0)$  and  $n = 7$ .

```
INTEGER*8 M,N
REAL*8 F(2),X(7),Y(5)
DATA F / 1.0 , 2.0 /
DATA X / 0.0 , 0.0 , 1.0 , 3.0 , 5.0 , 0.0 , 0.0 /
M = 2
N = 4
CALL FILTERS (F,M,X,N,Y)
```

The result is  $y = (1, 5, 12, 13, 5)$ .



# Linear Recurrences

## Overview

This chapter explains how to use SCILIB subprograms for a variety of linear recurrence operations.

## Chapter Objectives

After reading this chapter you will

- be able to recognize a recurrence
- know how to use the described subprograms

## What You Need to Know to Use These Subprograms

A recurrence is a loop-carried data dependency between a value calculated during one iteration of a loop and used during a subsequent iteration. When the FORTRAN compiler detects a recurrence, it cannot vectorize the loop. Therefore, the following subprograms, which use special, vectorizable algorithms, can be used to optimize FORTRAN loops containing certain linear recurrences.

## Subprogram Descriptions

|                                                                                   |      |
|-----------------------------------------------------------------------------------|------|
| Solve a First Order Linear Recurrence<br>FOLR, FOLRP .....                        | 8-2  |
| Solve a First Order Linear Recurrence<br>FOLR2, FOLR2P .....                      | 8-4  |
| Solve a First Order Linear Recurrence with Constant Coefficient<br>FOLRC .....    | 8-7  |
| Solve for the Last Term of a First Order Linear Recurrence<br>FOLRN, FOLRNP ..... | 8-9  |
| Compute the Vector of Partial Products of a Vector<br>RECPP .....                 | 8-12 |
| Compute the Vector of Partial Sums of a Vector<br>RECPS .....                     | 8-14 |
| Solve a Second Order Linear Recurrence<br>SOLR .....                              | 8-16 |
| Solve a Second Order Linear Recurrence<br>SOLR3 .....                             | 8-19 |
| Solve for the Last Term of a Second Order Linear Recurrence<br>SOLRN .....        | 8-22 |

**Purpose** Given real vectors  $a$  and  $x$  of length  $n$ , these subprograms solve the first order linear recurrence

$$\begin{aligned} y_1 &= x_1 \\ y_i &= x_i \pm a_i y_{i-1}, \quad i = 2, 3, \dots, n, \end{aligned}$$

overwriting the input  $x$  vector with the resulting  $y$  vector.

The operation indicated by “ $\pm$ ” above is specified by the subprogram name used:

FOLR “ $\pm$ ” represents “ $-$ ”:  $y_i = x_i - a_i y_{i-1}$   
 FOLRP “ $\pm$ ” represents “ $+$ ”:  $y_i = x_i + a_i y_{i-1}$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,inca,incx
REAL*8 a(lena),x(lenx)
CALL FOLR (n, a, inca, x, incx)
```

```
INTEGER*8 n,inca,incx
REAL*8 a(lena),x(lenx)
CALL FOLRP (n, a, inca, x, incx)
```

**Input**

**n** Number of elements of vectors  $a$  and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$  or  $x$ .

**a** Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca**  $> 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .

**inca**  $< 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ , **incx**  $\neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**Output**

**x** If  $n = 0$ , then  $x$  is unchanged. Otherwise, the recurrence’s solution vector overwrites the input.

**Notes**

The result is unspecified if  $a$  and  $x$  overlap such that any element of  $a$  shares a memory location with any element of  $x$ .

If an error in the arguments is detected, the subprograms call error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$\begin{aligned} n &< 0, \\ inca &= 0, \text{ and} \\ incx &= 0. \end{aligned}$$

**FORTRAN  
Equivalent**

```

SUBROUTINE FOLR (N, A, INCA, X, INCX)
INTEGER*8 N, INCA, INCX
REAL*8 A(*), X(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
DO 10 I = 2, N
 X(IX) = X(IX) - A(IA) * X(IX-INCX)
 IA = IA + INCA
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example**

Solve the REAL\*8 first order linear recurrence

$$\begin{aligned} y_1 &= x_1 \\ y_i &= x_i - a_i y_{i-1}, \quad i = 2, 3, \dots, n, \end{aligned}$$

where  $a$  and  $x$  are vectors 10 elements long stored in one-dimensional arrays A and X of dimension 20, and  $y$  overwrites  $x$ .

```

INTEGER*8 N, INCA, INCX
REAL*8 A(20), X(20)
N = 10
INCA = 1
INCX = 1
CALL FOLR (N, A, INCA, X, INCX)

```

**Purpose** Given real vectors  $a$  and  $x$  of length  $n$ , these subprograms solve the first order linear recurrence

$$\begin{aligned} y_1 &= x_1 \\ y_i &= x_i \pm a_i y_{i-1}, \quad i = 2, 3, \dots, n, \end{aligned}$$

for the  $y$  vector.

The operation indicated by “ $\pm$ ” above is specified by the subprogram name used:

FOLR2    “ $\pm$ ” represents “ $-$ ”:     $y_i = x_i - a_i y_{i-1}$   
 FOLR2P    “ $\pm$ ” represents “ $+$ ”:     $y_i = x_i + a_i y_{i-1}$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,inca,incx
REAL*8 a(lena),x(lenx),y(leny)
CALL FOLR2 (n, a, inca, x, incx, y, incy)
```

```
INTEGER*8 n,inca,incx
REAL*8 a(lena),x(lenx),y(leny)
CALL FOLR2P (n, a, inca, x, incx, y, incy)
```

**Input**

**n**    Number of elements of vectors  $a$ ,  $x$ , and  $y$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$ ,  $x$ , or  $y$ .

**a**    Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca**    Increment for the array  $a$ :

**inca**  $> 0$      $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .

**inca**  $< 0$      $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**x**    Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx**    Increment for the array  $x$ , **incx**  $\neq 0$ :

**incx**  $> 0$      $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$      $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**incy** Increment for the array **y**, **incy**  $\neq$  0:

**incy**  $>$  0 **y** is stored forward in array **y**, i.e.,  
 $y_i$  is stored in  $y((i-1) \times \text{incy} + 1)$ .

**incy**  $<$  0 **y** is stored backward in array **y**, i.e.,  
 $y_i$  is stored in  $y((i-n) \times \text{incy} + 1)$ .

Use **incy** = 1 if the vector **y** is stored contiguously in array **y**, i.e., if  $y_i$  is stored in  $y(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output** **y** Array of length  $\text{leny} = (n-1) \times |\text{incy}| + 1$  containing the  $n$ -vector **y**. If  $n = 0$ , then **y** is unchanged. Otherwise, **y** is the recurrence's solution vector.

**Notes** The result is unspecified if **a**, **x**, or **y** overlap such that any element of **a**, **x**, or **y** shares a memory location with any other element of **a**, **x**, or **y**.

If an error in the arguments is detected, the subprograms call error handler XERSCI, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**n**  $<$  0,  
**inca** = 0,  
**incx** = 0, and  
**incy** = 0.

**FORTTRAN**  
Equivalent

```

SUBROUTINE FOLR2 (N, A, INCA, X, INCX, Y, INCY)
INTEGER*8 N, INCA, INCX, INCY
REAL*8 A(*), X(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCY .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IY = 1 + INCY
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
IF (INCY .LT. 0) IY = 1 - (N-2) * INCY
Y(IY-INCX) = X(IX-INCX)
DO 10 I = 2, N
 Y(IY) = X(IX) - A(IA) * Y(IY-INCX)
 IA = IA + INCA
 IX = IX + INCX
 IY = IY + INCY
10 CONTINUE
RETURN
END

```

**Example** Solve the REAL\*8 first order linear recurrence

$$\begin{aligned}
 y_1 &= x_1 \\
 y_i &= x_i - a_i y_{i-1}, \quad i = 2, 3, \dots, n,
 \end{aligned}$$

where  $a$ ,  $x$  and  $y$  are vectors 10 elements long stored in one-dimensional arrays A, X, and Y of dimension 20.

```

INTEGER*8 N, INCA, INCX, INCY
REAL*8 A(20), X(20), Y(20)
N = 10
INCA = 1
INCX = 1
INCY = 1
CALL FOLR2 (N, A, INCA, X, INCX, Y, INCY)

```

## First Order Linear Recurrence

## FOLRC

**Purpose** Given a real coefficient  $\alpha$  and a real vector  $a$  of length  $n$ , this subprogram solves the first order linear recurrence

$$\begin{aligned} x_1 &= a_1 \\ x_i &= a_i + \alpha x_{i-1}, \quad i = 2, 3, \dots, n. \end{aligned}$$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx,inca
REAL*8 x(lenx),a(lena),alpha
CALL FOLRC (n, x, incx, a, inca, alpha)
```

**Input**

**n** Number of elements of vectors  $a$  and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$  or  $x$ .

**incx** Increment for the array  $x$ ,  $incx \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**a** Array of length  $lena = (n-1) \times |inca| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca** > 0  $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times inca + 1)$ .

**inca** < 0  $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times inca + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**alpha** The scalar  $\alpha$ .

**Output**

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ . If  $n = 0$ , then  $x$  is unchanged. Otherwise, the recurrence's solution vector is returned.

**Notes** The result is unspecified if  $a$  and  $x$  overlap such that any element of  $a$  shares a memory location with any element of  $x$ .

If an error in the arguments is detected, the subprogram calls error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$\begin{aligned} n &< 0, \\ incx &= 0, \text{ and} \\ inca &= 0. \end{aligned}$$

**FORTRAN  
Equivalent**

```

SUBROUTINE FOLRC (N, X, INCX, A, INCA, ALPHA)
INTEGER*8 N, INCX, INCA
REAL*8 X(*), A(*), ALPHA
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
X(IX-INCX) = A(IA-INCA)
DO 10 I = 2, N
 X(IX) = A(IA) + ALPHA * X(IX-INCX)
 IA = IA + INCA
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example** Solve the REAL\*8 first order linear recurrence

$$\begin{aligned} x_1 &= a_1 \\ x_i &= a_i + 4x_{i-1}, \quad i = 2, 3, \dots, n, \end{aligned}$$

where  $a$  and  $x$  are vectors 10 elements long stored in one-dimensional arrays A and X of dimension 20.

```

INTEGER*8 N, INCX, INCA
REAL*8 X(20), A(20), ALPHA
N = 10
INCX = 1
INCA = 1
ALPHA = 4.0
CALL FOLRC (N, X, INCX, A, INCA, ALPHA)

```

Last Term of First Order Linear RecurrenceFOLRN/FOLRNP

**Purpose** Given real vectors  $a$  and  $x$  of length  $n$ , these subprograms solve for the last term of the first order linear recurrence

$$y_1 = x_1$$

$$y_i = x_i \pm a_i y_{i-1}, \quad i = 2, 3, \dots, n,$$

i.e., returning  $y_n$ .

The operation indicated by “ $\pm$ ” above is specified by the subprogram name used:

FOLRN “ $\pm$ ” represents “ $-$ ”:  $y_i = x_i - a_i y_{i-1}$   
 FOLRNP “ $\pm$ ” represents “ $+$ ”:  $y_i = x_i + a_i y_{i-1}$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,inca,incx
REAL*8 yn,FOLRN,a(lena),x(lenx)
yn = FOLRN (n, a, inca, x, incx)

INTEGER*8 n,inca,incx
REAL*8 yn,FOLRNP,a(lena),x(lenx)
yn = FOLRNP (n, a, inca, x, incx)
```

**Input**

**n** Number of elements of vectors  $a$  and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$  or  $x$ .

**a** Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca**  $> 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .  
**inca**  $< 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ :

**incx**  $\geq 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .  
**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

If **incx** = 0, then  $x_i = x(1)$  for all  $i$ . Refer to “Notes” to see how to use FOLRNP with **incx** = 0 to evaluate a polynomial. Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to “BLAS Indexing Conventions” in the introduction to Chapter 2.

**Output**      **yn**      If  $n = 0$ , then **yn** = 0. Otherwise, the last term of the recurrence's solution is returned.

**Notes**            The result is unspecified if **a** and **x** overlap such that any element of **a** shares a memory location with any element of **x**.

FOLRNP may be used to evaluate a polynomial  $p(x) = \sum_{i=0}^n a_i x^i$  by recognizing that  $p(x)$  is the last term of the recurrence

$$\begin{aligned} y_0 &= a_n \\ y_i &= a_{n-i} + y_{i-1}x, \quad i = 1, 2, \dots, n. \end{aligned}$$

Refer to "Example 2."

If an error in the arguments is detected, the subprograms call error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

$$\begin{aligned} n &< 0, \text{ and} \\ \text{inca} &= 0. \end{aligned}$$

**FORTRAN  
Equivalent**

```

REAL*8 FUNCTION FOLRN (N, A, INCA, X, INCX)
INTEGER*8 N, INCA, INCX
REAL*8 A(*), X(*)
FOLRN = 0.0
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
FOLRN = X(IX-INCX)
DO 10 I = 2, N
 FOLRN = X(IX) - A(IA) * FOLRN
 IA = IA + INCA
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

Continued

**Example 1** Solve for the last term of the REAL\*8 first order linear recurrence

$$\begin{aligned} y_1 &= x_1 \\ y_i &= x_i - a_i y_{i-1}, \quad i = 2, 3, \dots, n, \end{aligned}$$

where  $a$  and  $x$  are vectors 10 elements long stored in one-dimensional arrays A and X of dimension 20.

```
INTEGER*8 N, INCA, INCX
REAL*8 YN, FOLRN, A(20), X(20)
N = 10
INCA = 1
INCX = 1
YN = FOLRN(N, A, INCA, X, INCX)
```

**Example 2**

Evaluate the REAL\*8 polynomial  $p(x) = \sum_{i=0}^{10} a_i x^i$ , where  $a$  is a vector 11 elements long stored in one-dimensional array A of dimension 0:20.

```
INTEGER*8 N, INCA, INCX
REAL*8 P, FOLRNP, A(0:20), X
N = 11
INCA = -1
INCX = 0
P = FOLRNP(N, A, INCA, X, INCX)
```

**Purpose** Given real vector  $a$  of length  $n$ , this subprogram computes the  $n$ -vector  $x$  of partial products

$$x_i = \prod_{j=1}^i a_j, \quad i = 1, 2, \dots, n.$$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

**SCILIB:**

```
INTEGER*8 n,incx,inca
REAL*8 x(lenx),a(lena)
CALL RECPP (n, x, incx, a, inca)
```

**Input**

**n** Number of elements of vectors  $a$  and  $x$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$  or  $x$ .

**incx** Increment for the array  $x$ ,  $incx \neq 0$ :

**incx** > 0  $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .

**incx** < 0  $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**a** Array of length  $lena = (n-1) \times |inca| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca** > 0  $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times inca + 1)$ .

**inca** < 0  $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times inca + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ . If  $n = 0$ , then  $x$  is unchanged. Otherwise, the vector of partial products replaces the input.

**Notes**

The result is unspecified if  $a$  and  $x$  overlap such that any element of  $a$  shares a memory location with any element of  $x$ .

If an error in the arguments is detected, the subprograms call error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

```
n < 0,
incx = 0, and
inca = 0.
```

**FORTRAN**  
Equivalent

```

SUBROUTINE RECPP (N, X, INCX, A, INCA)
INTEGER*8 N, INCX, INCA
REAL*8 X(*), A(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
X(IX-INCX) = A(IA-INCA)
DO 10 I = 2, N
 X(IX) = X(IX-INCX) * A(IA)
 IA = IA + INCA
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example**

Compute the REAL\*8 vector of partial products of the vector  $a$ , where  $a$  is a vector 10 elements long stored in a one-dimensional array A of dimension 20. The result is stored in array X, also of dimension 20.

```

INTEGER*8 N, INCX, INCA
REAL*8 X(20), A(20)
N = 10
INCA = 1
INCX = 1
CALL RECPP (N, X, INCX, A, INCA)

```

**Purpose** Given real vector  $a$  of length  $n$ , this subprogram computes the  $n$ -vector  $x$  of partial sums

$$x_i = \sum_{j=1}^i a_j, \quad i = 1, 2, \dots, n.$$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,incx,inca
REAL*8 x(lenx),a(lena)
CALL RECPS (n, x, incx, a, inca)
```

**Input**

**n** Number of elements of vectors  $a$  and  $x$ ,  $n \geq 0$ . If  $n = 0$ , the subprograms do not reference  $a$  or  $x$ .

**incx** Increment for the array  $x$ ,  $incx \neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times incx + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times incx + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**a** Array of length  $lena = (n-1) \times |inca| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca**  $> 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times inca + 1)$ .

**inca**  $< 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times inca + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**

**x** Array of length  $lenx = (n-1) \times |incx| + 1$  containing the  $n$ -vector  $x$ . If  $n = 0$ , then  $x$  is unchanged. Otherwise, the vector of partial sums replaces the input.

**Notes**

The result is unspecified if  $a$  and  $x$  overlap such that any element of  $a$  shares a memory location with any element of  $x$ .

If an error in the arguments is detected, the subprograms call error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

```
n < 0,
incx = 0, and
inca = 0.
```

**FORTTRAN**  
**Equivalent**

```

SUBROUTINE RECPS (N, X, INCX, A, INCA)
INTEGER*8 N, INCX, INCA
REAL*8 X(*), A(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1 + INCA
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-2) * INCA
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
X(IX-INCX) = A(IA-INCA)
DO 10 I = 2, N
 X(IX) = X(IX-INCX) + A(IA)
 IA = IA + INCA
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example**

Compute the REAL\*8 vector of partial sums of the vector  $a$ , where  $a$  is a vector 10 elements long stored in a one-dimensional array A of dimension 20. The result is stored in array X, also of dimension 20.

```

INTEGER*8 N, INCX, INCA
REAL*8 X(20), A(20)
N = 10
INCA = 1
INCX = 1
CALL RECPS (N, X, INCX, A, INCA)

```

**Purpose** Given real vectors  $a$  and  $b$  of length  $n$  and the first two elements of  $n$ -vector  $x$ , this subprogram solves the second order linear recurrence

$$x_i = a_{i-2}x_{i-1} + b_{i-2}x_{i-2}, \quad i = 3, 4, \dots, n.$$

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

SCILIB:

```
INTEGER*8 n,inca,incb,incx
REAL*8 a(lena),b(lenb),x(lenx)
CALL SOLR (n, a, inca, b, incb, x, incx)
```

**Input**

**n** Number of elements of vectors  $a$ ,  $b$ , and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $b$ , or  $x$ .

**a** Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca**  $> 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .

**inca**  $< 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**b** Array of length  $\text{lenb} = (n-1) \times |\text{incb}| + 1$  containing the  $n$ -vector  $b$ .

**incb** Increment for the array  $b$ :

**incb**  $> 0$   $b$  is stored forward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-1) \times \text{incb} + 1)$ .

**incb**  $< 0$   $b$  is stored backward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-n) \times \text{incb} + 1)$ .

Use **incb** = 1 if the vector  $b$  is stored contiguously in array  $b$ , i.e., if  $b_i$  is stored in  $b(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the first two elements,  $x_1$  and  $x_2$  of the  $n$ -vector  $x$ .

**incx** Increment for the array  $x$ , **incx**  $\neq 0$ :

**incx**  $> 0$   $x$  is stored forward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $< 0$   $x$  is stored backward in array  $x$ , i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array  $x$ , i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**     **x**     If  $n = 0$ , then **x** is unchanged. Otherwise, the recurrence's solution vector replaces the input.

**Notes**        The result is unspecified if **a**, **b**, or **x** overlap such that any element of *a*, *b*, or *x* shares a memory location with any other element of *a*, *b*, or *y*.

If an error in the arguments is detected, the subprogram calls error handler XERSCI, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**n** < 0,  
**inca** = 0,  
**incb** = 0, and  
**incx** = 0.

**FORTRAN  
Equivalent**

```

SUBROUTINE SOLR (N, A, INCA, B, INCB, X, INCX)
INTEGER*8 N, INCA, INCB, INCX
REAL*8 A(*), B(*), X(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCB .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1
IB = 1
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-3) * INCA
IF (INCB .LT. 0) IB = 1 - (N-3) * INCB
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
DO 10 I = 3, N
 X(IX+INCX) = A(IA) * X(IX) + B(IB) * X(IX-INCX)
 IA = IA + INCA
 IB = IB + INCB
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example** Solve the REAL\*8 second order linear recurrence

$$x_i = a_{i-2}x_{i-1} + b_{i-2}x_{i-2}, \quad i = 3, 4, \dots, n,$$

where  $a$ ,  $b$ , and  $x$  are vectors 10 elements long stored in one-dimensional arrays A, B, and X of dimension 20.

```
INTEGER*8 N, INCA, INCB, INCX
REAL*8 A(20), B(20), X(20)
N = 10
INCA = 1
INCB = 1
INCX = 1
X(1) = ...
X(2) = ...
CALL SOLR (N, A, INCA, B, INCB, X, INCX)
```

## Second Order Linear Recurrence

SOLR3

**Purpose** Given real vectors  $a$ ,  $b$ , and  $x$  of length  $n$ , this subprogram solves the second order linear recurrence

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 \\ y_i &= x_i + a_{i-2}y_{i-1} + b_{i-2}y_{i-2}, \quad i = 3, 4, \dots, n. \end{aligned}$$

overwriting the input  $x$  vector with the resulting  $y$  vector.

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage**

**SCILIB:**

```
INTEGER*8 n,inca,incb,incx
REAL*8 a(lena),b(lenb),x(lenx)
CALL SOLR3 (n, a, inca, b, incb, x, incx)
```

**Input**

**n** Number of elements of vectors  $a$ ,  $b$ , and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $b$ , or  $x$ .

**a** Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

$\text{inca} > 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .  
 $\text{inca} < 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use  $\text{inca} = 1$  if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**b** Array of length  $\text{lenb} = (n-1) \times |\text{incb}| + 1$  containing the  $n$ -vector  $b$ .

**incb** Increment for the array  $b$ :

$\text{incb} > 0$   $b$  is stored forward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-1) \times \text{incb} + 1)$ .  
 $\text{incb} < 0$   $b$  is stored backward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-n) \times \text{incb} + 1)$ .

Use  $\text{incb} = 1$  if the vector  $b$  is stored contiguously in array  $b$ , i.e., if  $b_i$  is stored in  $b(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the  $n$ -vector  $x$ .

**incx**    Increment for the array **x**, **incx**  $\neq$  0:

**incx**  $>$  0    **x** is stored forward in array **x**, i.e.,

$x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $<$  0    **x** is stored backward in array **x**, i.e.,

$x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector **x** is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**    **x**    If  $n = 0$ , then **x** is unchanged. Otherwise, the recurrence's solution vector replaces the input.

**Notes**    The result is unspecified if **a**, **b**, or **x** overlap such that any element of **a**, **b**, or **x** shares a memory location with any other element of **a**, **b**, or **x**.

If an error in the arguments is detected, the subprogram calls error handler XERSCI, which prints an error message and executes a FORTRAN STOP statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**n**  $<$  0,  
**inca** = 0,  
**incb** = 0, and  
**incx** = 0.

FORTTRAN  
Equivalent

```

SUBROUTINE SOLR3 (N, A, INCA, B, INCB, X, INCX)
INTEGER*8 N, INCA, INCB, INCX
REAL*8 A(*), B(*), X(*)
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCB .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1
IB = 1
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-3) * INCA
IF (INCB .LT. 0) IB = 1 - (N-3) * INCB
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
DO 10 I = 3, N
 X(IX+INCX) = X(IX+INCX) + A(IA) * X(IX) + B(IB) * X(IX-INCX)
 IA = IA + INCA
 IB = IB + INCB
 IX = IX + INCX
10 CONTINUE
RETURN
END

```

**Example** Solve the REAL\*8 second order linear recurrence

$$\begin{aligned}
 y_1 &= x_1 \\
 y_2 &= x_2 \\
 y_i &= x_i + a_{i-2}y_{i-1} + b_{i-2}y_{i-2}, \quad i = 3, 4, \dots, n.
 \end{aligned}$$

where  $a$ ,  $b$ , and  $x$  are vectors 10 elements long stored in one-dimensional arrays A, B, and X of dimension 20, and  $y$  overwrites  $x$ .

```

INTEGER*8 N, INCA, INCB, INCX
REAL*8 A(20), B(20), X(20)
N = 10
INCA = 1
INCB = 1
INCX = 1
CALL SOLR3 (N, A, INCA, B, INCB, X, INCX)

```

**Purpose** Given real vectors  $a$  and  $b$  of length  $n$  and the first two elements of  $n$ -vector  $x$ , this subprogram solves for the last term of the second order linear recurrence

$$x_i = a_{i-2}x_{i-1} + b_{i-2}x_{i-2}, \quad i = 3, 4, \dots, n,$$

i.e., returning  $x_n$ .

The vectors may be stored in one-dimensional arrays or either in rows or columns of two-dimensional arrays, and the indexing through the arrays may be either forward or backward.

**Usage****SCILIB:**

```
INTEGER*8 n,inca,incb,incx
REAL*8 xn,SOLRN,a(lena),b(lenb),x(lenx)
xn = SOLRN (n, a, inca, b, incb, x, incx)
```

**Input**

**n** Number of elements of vectors  $a$ ,  $b$ , and  $x$  to be used in the recurrence,  $n \geq 0$ . If  $n = 0$ , the subprogram does not reference  $a$ ,  $b$ , or  $x$ .

**a** Array of length  $\text{lena} = (n-1) \times |\text{inca}| + 1$  containing the  $n$ -vector  $a$ .

**inca** Increment for the array  $a$ :

**inca**  $> 0$   $a$  is stored forward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-1) \times \text{inca} + 1)$ .

**inca**  $< 0$   $a$  is stored backward in array  $a$ , i.e.,  
 $a_i$  is stored in  $a((i-n) \times \text{inca} + 1)$ .

Use **inca** = 1 if the vector  $a$  is stored contiguously in array  $a$ , i.e., if  $a_i$  is stored in  $a(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**b** Array of length  $\text{lenb} = (n-1) \times |\text{incb}| + 1$  containing the  $n$ -vector  $b$ .

**incb** Increment for the array  $b$ :

**incb**  $> 0$   $b$  is stored forward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-1) \times \text{incb} + 1)$ .

**incb**  $< 0$   $b$  is stored backward in array  $b$ , i.e.,  
 $b_i$  is stored in  $b((i-n) \times \text{incb} + 1)$ .

Use **incb** = 1 if the vector  $b$  is stored contiguously in array  $b$ , i.e., if  $b_i$  is stored in  $b(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**x** Array of length  $\text{lenx} = (n-1) \times |\text{incx}| + 1$  containing the first two elements,  $x_1$  and  $x_2$  of the  $n$ -vector  $x$ .

**incx** Increment for the array **x**, **incx**  $\neq$  0:

**incx**  $>$  0  $x$  is stored forward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-1) \times \text{incx} + 1)$ .

**incx**  $<$  0  $x$  is stored backward in array **x**, i.e.,  
 $x_i$  is stored in  $x((i-n) \times \text{incx} + 1)$ .

Use **incx** = 1 if the vector  $x$  is stored contiguously in array **x**, i.e., if  $x_i$  is stored in  $x(i)$ . Refer to "BLAS Indexing Conventions" in the introduction to Chapter 2.

**Output**    **xn**    If  $n = 0$ , then **xn** = 0. Otherwise, the last term of the recurrence's solution is returned.

**x**    If  $n = 0$ , then **x** is unchanged. Otherwise, **x** is overwritten with intermediate results. These intermediate results are not necessarily what is shown in "FORTRAN Equivalent".

**Notes**    The result is unspecified if **a**, **b**, or **x** overlap such that any element of  $a$ ,  $b$ , or  $x$  shares a memory location with any other element of  $a$ ,  $b$ , or  $y$ .

If an error in the arguments is detected, the subprogram calls error handler XERSCI, which prints an error message and executes a FORTRAN **STOP** statement. The standard version of XERSCI (refer to Chapter 9, Miscellaneous Routines) may be replaced with a user-supplied version to change the error procedure. Error conditions are:

**n**  $<$  0,  
**inca** = 0,  
**incb** = 0, and  
**incx** = 0.

**FORTTRAN**  
**Equivalent**

```

REAL*8 FUNCTION SOLRN (N, A, INCA, B, INCB, X, INCX)
INTEGER*8 N, INCA, INCB, INCX
REAL*8 A(*), B(*), X(*)
SOLRN = 0.0
IF (N .LT. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCA .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCB .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IF (INCX .EQ. 0) THEN
 CALL XERSCI (...)
 RETURN
END IF
IA = 1
IB = 1
IX = 1 + INCX
IF (INCA .LT. 0) IA = 1 - (N-3) * INCA
IF (INCB .LT. 0) IB = 1 - (N-3) * INCB
IF (INCX .LT. 0) IX = 1 - (N-2) * INCX
DO 10 I = 3, N ! CAUTION: X NOT NECESSARILY RETURNED AS SHOWN
 X(IX+INCX) = A(IA) * X(IX) + B(IB) * X(IX-INCX)
 IA = IA + INCA
 IB = IB + INCB
 IX = IX + INCX
10 CONTINUE
IF (N .EQ. 1) THEN
 SOLRN = X(IX-INCX)
ELSE
 SOLRN = X(IX)
END IF
RETURN
END

```

**Example**

Solve for the last term of the REAL\*8 second order linear recurrence

$$x_i = a_{i-2}x_{i-1} + b_{i-2}x_{i-2}, \quad i = 3, 4, \dots, n,$$

where  $a$ ,  $b$ , and  $x$  are vectors 10 elements long stored in one-dimensional arrays A, B, and X of dimension 20.

```

INTEGER*8 N, INCA, INCB, INCX
REAL*8 XN, SOLRN, A(20), B(20), X(20)
N = 10
INCA = 1
INCB = 1
INCX = 1
X(1) = ...
X(2) = ...
XN = SOLRN (N, A, INCA, B, INCB, X, INCX)

```

# Miscellaneous Routines

## Overview

This chapter explains how to use SCILIB subprograms for operations that do not fit in the categories covered by other chapters. It includes a description of subprograms that

- sort the elements of a vector into ascending or descending order
- report errors detected in the usage of SCILIB subprograms

## Chapter Objectives

After reading this chapter you will

- know how to use the described subprograms
- know how to change the method of error reporting in SCILIB subprograms

## What You Need to Know to Use These Subprograms

Knuth, D.E. 1973. *The Art of Computer Programming*, Vol. 3: Sorting and Searching. Addison-Wesley, Menlo Park, California.

## Subprogram Descriptions

|                                    |     |
|------------------------------------|-----|
| Sort an Array into Ascending Order |     |
| ORDERS .....                       | 9-2 |
| SCILIB Error Handler               |     |
| XERSCI .....                       | 9-4 |

**ORDERS****Sort Array into Ascending Order****Purpose**

This subprogram sorts data items into ascending order based on a sort key that may be all or part of each data record. The key may be a real number, a signed or unsigned integer, or a character string. Instead of rearranging the elements of the array, ORDERS returns an index vector containing the indices of the data elements in sorted order. The method has a linear computational complexity and is stable, that is, the original order of data with equal keys is preserved.

This subprogram uses a bucket sort algorithm to sort the data records. The length of the sort key determines the number of passes through the data. ORDERS has the option of processing either one or two bytes of the key per pass through the data. Using two bytes halves the number of passes at the expense of increased working storage and overhead per pass.

**Usage**

You may omit the last 1, 2, or 3 arguments from the CALL statement if the indicated default value is acceptable. Thus, the argument list may contain 5, 6, 7, or 8 arguments.

**SCILIB:**

```
INTEGER*8 mode,work(lwork),indx(n),n,ldx,keylen,iradix
REAL*8 x(ldx,n)
CALL ORDERS (mode, work, x, indx, n, ldx, keylen, iradix)
```

```
INTEGER*8 mode,work(lwork),x(ldx,n),indx(n),n,ldx,keylen,iradix
CALL ORDERS (mode, work, x, indx, n, ldx, keylen, iradix)
```

**Input**

**mode** Variable indicating data type and initial order option:

- mode** = 0 The key is an unsigned binary number of length  $8 \times \text{keylen}$  bits, and **indx** does not contain an initial order on input.
- mode** = 1 The key is a signed integer (INTEGER\*8), and **indx** does not contain an initial order on input.
- mode** = 2 The key is a real number (REAL\*8), and **indx** does not contain an initial order on input.
- mode** = 10 The key is an unsigned binary number of length **keylen** bytes, and **indx** contains an initial ordering on input.
- mode** = 11 The key is a signed integer (INTEGER\*8), and **indx** contains an initial ordering on input.
- mode** = 12 The key is a real number (REAL\*8), and **indx** contains an initial ordering on input.

**x** Array containing the data to be sorted.

**indx** If **mode** = 10, 11, or 12, an array of indices containing the initial data ordering, i.e.,  $x(1, \text{indx}(i))$  is the  $i$ th smallest data item. Not used as input if **mode** = 0, 1, or 2.

**n** Number of data items to be sorted,  $n \geq 1$ .

**ldx** The leading dimension of array **x** as declared in the calling program unit. Default = 1.

**keylen** Length of sort key as number of 8-bit bytes. **keylen** must be specified as 8 unless **mode** = 0 or 10. Default = 8.

**iradix** The number of bytes of the key that are processed per pass over the data, **iradix** = 1 or 2. Default = 1.

**Working Storage**     **work**     An array whose length, **lwork**, depends on **iradix**:

**iradix** = 1     **lwork** = 257  
**iradix** = 2     **lwork** = 65537

**Output**     **mode**     Status response:

**mode** ≥ 0     Normal return (unchanged from input).  
**mode** = -1     ORDERS called with fewer than 5 arguments.  
**mode** = -2     ORDERS called with more than 8 arguments.  
**mode** = -3      $ldx \leq 0$  or  $ldx > 2^{24}$ .  
**mode** = -4      $keylen > 8 \times ldx$ .  
**mode** = -5     **iradix** ≠ 1 or 2.  
**mode** = -6      $keylen \leq 0$ .  
**mode** = -7      $n \leq 0$  or  $n > 2^{24}$ .  
**mode** = -8     **mode** ≠ 0, 1, 2, 10, 11, or 12.  
**mode** = -9      $keylen \neq 8$  for real or integer sort.

**indx**     Array of indices containing the sorted data ordering, i.e., **x(1,indx(i))** is the *i*th smallest data item.

**Notes**     You can combine several CALL ORDERS statements to use either a larger field or more than one field as the sort key. Make the first call to sort on the least significant part of the key with **mode** = 0, 1, or 2; then use **mode** = 10, 11, or 12 to sort on additional fields in increasing order of significance. Refer to "Example 2."

**Example 1**     Sort the elements of a REAL\*8 vector *x* into ascending order, where *x* is a vector 100 elements long stored in a one-dimensional array **X** of dimension 200. The index array is returned in a one-dimensional array **INDX** of dimension 200.

```

INTEGER*8 MODE, WORK(257), INDX(200), N, LDX, KEYLEN, IRADIX
REAL*8 X(200)
MODE = 2
N = 100
LDX = 1
KEYLEN = 8
IRADIX = 1
CALL ORDERS (MODE, WORK, X, INDX, N, LDX, KEYLEN, IRADIX)

```

**Example 2**     A sparse matrix is represented by a set of NZ triples,  $\{i, j, A_{ij}\}$ . Sort the triples into order with primary key *i* and secondary key *j*. The data *i*, *j*, and  $A_{ij}$  are stored in arrays **IROW**, **JCOL**, and **AIJ**, all of dimension 500 and the resulting index array is returned in a one-dimensional array **INDX**, also of dimension 500.

```

INTEGER*8 IROW(500), JCOL(500), MODE, WORK(257), INDX(500), NZ
REAL*8 AIJ(500)
MODE = 1
CALL ORDERS (MODE, WORK, JCOL, INDX, NZ, 1, 8, 1)
MODE = 11
CALL ORDERS (MODE, WORK, IROW, INDX, NZ, 1, 8, 1)

```

**Purpose** XERSCI is the error handler for many of the subprograms in the SCILIB library, as indicated in the "Notes" section in the subprogram descriptions. As supplied in SCILIB, XERSCI prints one of the following error messages:

```

* SCILIB subprogram name called with invalid value of argument number iarg *

```

or

```

* Error detected by SCILIB subprogram name: text of error message *

```

or

```

* Error iarg detected by SCILIB subprogram name: text of error message *

```

where *name* is the name of the subprogram in which the error was detected, *iarg* is the argument number of the offending argument, and *text of error message* is a character string. XERSCI then executes a FORTRAN STOP statement.

You may supply a version of XERSCI that alters this action. All SCILIB subprograms that call XERSCI have a RETURN statement after the CALL statement, so your version could perhaps set a flag in a common block and RETURN. The flag could be tested in the program unit that calls the SCILIB subprogram.

**Usage****SCILIB:**

```
CHARACTER*1 name,messag
INTEGER*8 iarg
CALL XERSCI (name, iarg, messag)
```

**Input**

**name** The name of the subprogram in which the error was detected.

**iarg** If *iarg* > 0, the error message is printed in the first form given above and *iarg* is the number of the argument that was found to be in error. If *iarg* = 0, the error message is printed in the second form given above and *iarg* is not part of the error message. If *iarg* < 0, the error message is printed in the third form given above and *iarg* is the error number.

**messag** The text of the error message to be printed if *iarg* ≤ 0. Not used as input if *iarg* > 0.

# Reporting Problems

This appendix introduces the CONVEX Technical Assistance Center (TAC) and *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

## Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

## The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke *contact*, it prompts you for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

## Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- full path name of the program or utility in question
- version number of the program or utility in question

## UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

## Finding the Program Path Name

Use the *which* command to find the full path name of a program or utility. The next screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility.

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which(1)* man page. You can also use the *info* online information system. Enter **info which** at the system prompt.

If you use the C shell (*cs*h), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, but faster.

## Finding the Program Version Number

Use the *vers* command to find the version number of a program or utility. The next screen illustrates using the *vers* command to find the version number of the loader (*ld*) utility. Enter **vers**, then the path name of the program or utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the *vers* command, refer to the *vers(1)* man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

## Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

### Using a *.contact* File

When you invoke *contact*, it prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.

2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

## Aborting the Report

To abort a contact report, either press the interrupt key (usually **CTRL-C**) or choose the abort option when prompted by the *contact* utility. Using **CTRL-C** to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

## Submitting the *dead.report* File

After you abort a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

```
contact -r
```

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

## Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press **CTRL-Z**. To return to the contact session, enter **fg**. Using **CTRL-Z** and the *fg* (foreground) command lets you toggle back and forth between the *contact* utility and the shell. You cannot, however, use **CTRL-Z** and *fg* to toggle back and forth if you are using the Bourne shell (*sh*).

## Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press **RETURN**. Other prompts require more than a one-line response; to move to the next prompt, press **CTRL-D**.

## Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

## Reporting Problems

- `~e` start the text editor (defined in the EDITOR environment variable)
- `~h` display a list of available tilde-escape sequences
- `~p` print the contact report to the terminal screen
- `~r filename` read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.
- `~` insert a single tilde as the first character in the line

## Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- name and version of the product
- one-line summary of the problem
- detailed description of the problem
- priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation related to the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts.

**Step 1a** To invoke the *contact* utility, enter `contact` at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software or documentation question. The next screen illustrates the *contact* command and the system response.

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

- Step 1b If there is a *.contact* file in your home directory, *contact* skips the first prompt. The next screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory.

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

- Step 2 The *contact* utility prompts for the version number of the product. If you do not know the version number, use **CTRL-Z** to suspend the session. Use the *which* (or *whence* if you use *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.
- Step 3 The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.
- Step 4 The *contact* utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the *adb(1)* or *csd(1)* man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.
- Step 5 The *contact* utility prompts for the priority of the problem. The next screen illustrates this prompt and priority levels from which to choose; you must enter a priority

```
Enter a problem priority, based on the following:
1) Critical - work cannot proceed until the problem is resolved.
2) Serious - work can proceed around the problem, with difficulty.
3) Necessary - problem has to be fixed.
4) Annoying - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

number.

- Step 6 The *contact* utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.
- Step 7 The *contact* utility prompts for any other pertinent comments. Please include all relevant information.
- Step 8 The *contact* utility prompts for suggestions regarding documentation supporting the product. Indicate whether documentation could be revised to address the problem.

†

## Reporting Problems

Step 9 The *contact* utility asks for names of files necessary to reproduce the problem. The next screen illustrates the *contact* prompt and sample user response.

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

### Note

Tilde-escape sequences are not recognized in responses to this prompt. In *contact*, a tilde in this section means your home directory. This convention is based on use of the tilde for expanding file names in *csk*.

If files specified are small text files, they are automatically included in the *contact* report. If the files are too large to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the *tar* command (refer to the *tar(1)* man page for further information) or enter each file name in the directory on a single line in the *contact* report.

Step 10 The *contact* utility prompts you to review, edit, submit, or abort the *contact* report. The next screen illustrates this prompt.

```
Please select one of the following options:
```

- 1) Review the problem report.
- 2) Edit the problem report.
- 3) Submit the problem report.
- 4) Abort the problem report.

```
>
```

Choose the number of the option you want to select. These options let you do the following:

**Review** review the text of the *contact* report. You are then prompted again to select an option.

**Edit** edit the text of the *contact* report. If you choose to edit the report, *contact* puts you in your default text editor.

**Submit** sends the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell.

**Abort** saves the text of the report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell.

# Index

## A

accessing SCILIB 1-3  
Ada, calling SCILIB from 1-1  
application compiler 1-4  
arithmetic mode 1-3  
ASAP, automatic self allocating processors  
1-5  
automatic self allocating processors (ASAP)  
1-5

## B

backward storage 2-3  
BAKVEC 5-13  
BALANC 5-13  
BALBAK 5-13  
band matrix 3-16, 3-34, 3-64, 3-68, 4-9, 4-12,  
4-14, 4-17, 4-29, 4-31, 4-34, 4-36, 4-39,  
4-50  
BANDR 5-13  
BANDV 5-13  
Basic Linear Algebra Subprograms 1-2, 2-1  
BEGIN\_TASKS compiler directive 1-6  
bibliography xxiii  
BISECT 5-13  
BLAS 1-2, 2-1  
BLAS, Extended 3-1, 4-1, 5-1  
BLAS indexing conventions 2-3  
BLAS, Level 2 3-1, 4-1, 5-1  
BLAS, Level 3 3-1, 4-1, 5-1  
BQR 5-13

## C

C, calling SCILIB from 1-1  
calling SCILIB from Ada 1-1  
calling SCILIB from C 1-1  
CAXPY 2-36  
CBABK2 5-13  
CBAL 5-13  
CCHDC 4-52  
CCHDD 4-52  
CCHEX 4-52  
CCHUD 4-52  
CCOPY 2-40  
CDOTC 2-42  
CDOTU 2-42  
CFFT2 6-3  
CFFTMLT 6-5  
CFTFAX 6-5  
CG 5-13  
CGBCO 4-9  
CGBDI 4-12  
CGBFA 4-14  
CGBMV 3-16  
CGBSL 4-17  
CGECO 4-19  
CGEDI 4-21  
CGEFA 4-24  
CGEMM 3-20  
CGEMMS 3-23  
CGEMV 3-27  
CGERC 3-30

CGERU 3-30  
CGESL 4-26  
CGTSL 4-29  
CH 5-13  
CHBMV 3-34  
CHEMM 3-47  
CHEMV 3-50  
CHER 3-53  
CHER2 3-55  
CHER2K 3-58  
CHERK 3-61  
CHICO 4-52  
CHIDI 4-52  
CHIFA 4-52  
CHISL 4-52  
Cholesky factorization 4-31, 4-36, 4-41, 4-46  
CHPCO 4-52  
CHPDI 4-52  
CHPFA 4-52  
CHPMV 3-38  
CHPR 3-41  
CHPR2 3-44  
CHPSL 4-52  
CINVT 5-13  
CLUSEQ 2-6  
CLUSFGE 2-6  
CLUSFGT 2-6  
CLUSFLE 2-6  
CLUSFLT 2-6  
CLUSIGE 2-6  
CLUSIGT 2-6  
CLUSILE 2-6  
CLUSILT 2-6  
CLUSNE 2-6  
cluster 2-6  
COMBAK 5-13  
COMHES 5-13  
COMLR 5-13  
COMLR2 5-13  
compiler directives 1-5  
COMQR 5-13  
COMQR2 5-13  
condition number 4-4, 4-9, 4-19, 4-31, 4-41  
*contact*, aborting the report A-3, A-6  
*contact*, editing the report A-6  
*contact*, ending a response A-3  
*contact*, ending the report A-6  
*contact* file, skipping first prompt by using  
A-2  
*contact*, including files in the report A-6  
*contact*, invoking A-1, A-4  
*contact*, prerequisites A-1  
*contact*, prompts A-4  
*contact*, reporting problems A-1  
*contact*, restrictions on tilde-escape sequences  
A-6  
*contact*, reviewing the report A-6  
*contact*, skipping first prompt by using *con-*  
*contact* file A-2  
*contact*, step-by-step discussion of prompts  
A-4  
*contact*, submitting *dead.report* file A-3

*contact*, submitting the report A-6  
*contact*, suspending the report A-3  
*contact*, tilde-escape sequences A-3  
*contact*, tips on using A-2  
 CONVEX VECLIB Documentation Set 1-1  
 CONVEX VECLIB User's Guide 1-1  
 convolution 7-2, 7-4  
 correlation 7-2, 7-4  
 correlation and convolution subprograms 7-1  
 CORTB 5-13  
 CORTH 5-13  
 count vector elements 2-10, 2-11, 2-12  
 CPBCO 4-31  
 CPBDI 4-34  
 CPBFA 4-36  
 CPBSL 4-39  
 CPOCO 4-41  
 CPODI 4-43  
 CPOFA 4-46  
 CPOSL 4-48  
 CPPCO 4-52  
 CPPDI 4-52  
 CPPFA 4-52  
 CPPSL 4-52  
 CPTSL 4-50  
 CQRDC 4-52  
 CQRSL 4-52  
 Cray SCILIB 1-1  
 CRFFT2 6-9  
 CROT 2-51  
 CROTG 2-53  
 CSCAL 2-59  
 CSICO 4-52  
 CSIDI 4-52  
 CSIFA 4-52  
 CSISL 4-52  
 CSPCO 4-52  
 CSSCAL 2-59  
 CSUM 2-61  
 CSWAP 2-62  
 CSYMM 3-47  
 CSYR2K 3-58  
 CSYRK 3-61  
 CTBMV 3-64  
 CTBSV 3-68  
 CTPMV 3-72  
 CTPSV 3-75  
 CTRMM 3-78  
 CTRMV 3-81  
 CTRSM 3-83  
 CTRSV 3-86  
 CXpa 1-3

**D**

data types, default 1-2  
*dead.report* file, submitting A-3  
*dead.report* file, using *-r* option to submit A-3  
 default data types 1-2  
 determinant 4-4, 4-12, 4-21, 4-34, 4-43  
 DFT 6-1

discrete Fourier transform 6-1  
 documentation, on-line 1-8  
 documentation, ordering xxiv  
 dot product 2-42, 2-49

**E**

eigenvalues 5-1, 5-3, 5-5, 5-7, 5-9, 5-11  
 eigenvectors 5-1, 5-3, 5-5, 5-11  
 EISPACK 1-2, 1-9, 5-1  
*EISPACK Guide* 1-10  
*EISPACK Guide Extension* 1-10  
 ELMLBAK 5-13  
 ELMHES 5-13  
 ELTRAN 5-13  
 END\_TASKS compiler directive 1-6  
 error handling, SCILIB 1-8  
 error reporting A-1  
 Euclidean norm 2-45  
 Extended BLAS 1-2, 3-1, 4-1, 5-1

**F**

fast Fourier transforms 6-1  
*fc* command line 1-3  
 FFT, complex-to-real 6-9  
 FFT, one-dimensional 6-3, 6-9, 6-11  
 FFT, one-dimensional, simultaneous 6-5, 6-13  
 FFT, real-to-complex 6-11, 6-13  
 FFTFAX 6-13  
 FIGI 5-13  
 FIGI2 5-13  
 FILTERG 7-2  
 filtering 7-2, 7-4  
 FILTERS 7-4  
 find 2-6, 2-25, 2-27, 2-29, 2-31, 2-64, 2-67  
 first order recurrence 8-2, 8-4, 8-7, 8-9  
 floating point formats 1-3  
 FOLR 8-2  
 FOLR2 8-4  
 FOLR2P 8-4  
 FOLRC 8-7  
 FOLRN 8-9  
 FOLRNP 8-9  
 FOLRP 8-2  
 FORCE\_PARALLEL compiler directive 1-5  
 FORTRAN array argument association 2-2  
 FORTRAN storage of arrays 2-2  
 forward storage 2-3  
 further reference xxiii

**G**

GATHER 2-9  
 gather 2-9  
 Givens rotation 2-51, 2-53  
 Givens rotation, modified 2-55, 2-57

**H**

HQR 5-13  
 HQR2 5-13  
 HTRIB3 5-13

HTRIBK 5-13  
 HTRID3 5-13  
 HTRIDI 5-13

## I

ICAMAX 2-17  
 IEEE arithmetic mode 1-3  
 IEEE compatibility 1-3  
 ILZ 2-10  
 ILLZ 2-11  
 ILSUM 2-12  
 IMTQL1 5-13  
 IMTQL2 5-13  
 IMTQLV 5-13  
 increment 2-3  
 increment arguments 2-3  
 INCX 2-3  
 index 2-25, 2-27, 2-29, 2-31, 2-64, 2-67  
 index of maximum 2-13, 2-21  
 index of maximum absolute value 2-17  
 index of minimum 2-15, 2-23  
 index of minimum absolute value 2-19  
 INFLMAX 2-13  
 INFLMIN 2-15  
 inner product 2-42, 2-49  
 INTMAX 2-21  
 INTMIN 2-23  
 inverse 4-4, 4-6, 4-21, 4-43  
 INVIT 5-13  
 ISAMAX 2-17  
 ISAMIN 2-19  
 ISEARCH 2-25  
 ISMAX 2-21  
 ISMIN 2-23  
 ISRCHEQ 2-25  
 ISRCHFG 2-25  
 ISRCHFGT 2-25  
 ISRCHFLE 2-25  
 ISRCHFLT 2-25  
 ISRCHIGE 2-25  
 ISRCHIGT 2-25  
 ISRCHILE 2-25  
 ISRCHILT 2-25  
 ISRCHMEQ 2-27  
 ISRCHMGE 2-27  
 ISRCHMGT 2-27  
 ISRCHMLE 2-27  
 ISRCHMLT 2-27  
 ISRCHMNE 2-27  
 ISRCHNE 2-25

## L

-l option 1-3  
 Level 2 BLAS 3-1, 4-1, 5-1  
 Level 3 BLAS 3-1, 4-1, 5-1  
 linear equations 4-1, 4-6, 4-8  
 linking SCILIB 1-3  
 LINPACK 1-2, 1-9, 4-1  
 LINPACK Users' Guide 1-10  
 locate 2-6, 2-10, 2-11, 2-25, 2-27, 2-29, 2-31,  
 2-64, 2-67

-lscilib 1-3  
 LU factorization 4-9, 4-14, 4-19, 4-24  
 -lveclib8 1-3

## M

man pages 1-8  
 matrix inverse 4-4, 4-6, 4-21, 4-43  
 matrix-matrix multiply 3-5, 3-7, 3-20, 3-23,  
 3-47  
 matrix-matrix multiply, triangular 3-78  
 matrix-vector multiply 3-11, 3-13, 3-16, 3-27,  
 3-34, 3-38, 3-50, 3-64, 3-72, 3-81  
 matrix-vector multiply and add 3-32  
 maximum 2-13, 2-17, 2-21  
 MINFIT 5-13  
 minimum 2-15, 2-19, 2-23  
 MINV 4-6  
 modified Givens rotation 2-55, 2-57  
 MXM 3-5  
 MXMA 3-7  
 MXV 3-11  
 MXVA 3-13

## N

native arithmetic mode 1-3  
 negative 2-11  
 NEXT\_TASK compiler directive 1-6  
 nonzeros 2-10  
 norm 2-34, 2-45

## O

on-line documentation 1-8  
 OPFLT 4-8  
 ordered vector 2-29, 2-31  
 ordering documentation xxiv  
 ORDERS 9-2  
 ORTBK 5-13  
 ORTHES 5-13  
 ORTRAN 5-14  
 OSRCHF 2-29  
 OSRCHI 2-29  
 OSRCHM 2-31

## P

packed matrix 3-38, 3-41, 3-44, 3-72, 3-75  
 parallel processing 1-5  
 partial products 8-12  
 partial sums 8-14  
 performance analysis 1-3  
 pipelined arithmetic unit 1-5  
 polynomial, evaluate 8-9  
 positive 2-11  
 positive definite matrix 4-31, 4-34, 4-36,  
 4-39, 4-41, 4-43, 4-46, 4-48, 4-50  
 products, partial 8-12  
 profiling 1-3  
 programmer's reference 1-8

## Q

QZHES 5-14  
 QZIT 5-14  
 QZVAL 5-14  
 QZVEC 5-14

## R

rank-2k update 3-58  
 rank-k update 3-61  
 rank-one update 3-30, 3-41, 3-53  
 rank-two update 3-44, 3-55  
 RATQR 5-14  
 RCFFT2 6-11  
 REBAK 5-14  
 REBAKB 5-14  
 RECPP 8-12  
 RECPS 8-14  
 recurrence, first order 8-2, 8-4, 8-7, 8-9  
 recurrence, second order 8-16, 8-19, 8-22  
 REDUC 5-14  
 REDUC2 5-14  
 re-entrance 1-5  
 RFFTMLT 6-13  
 RG 5-14  
 RGG 5-14  
 RS 5-3  
 RSB 5-14  
 RSG 5-14  
 RSGAB 5-14  
 RSGBA 5-14  
 RSM 5-14  
 RSP 5-14  
 RST 5-14  
 RT 5-14

## S

SASUM 2-34  
 SAXPY 2-36  
 SCASUM 2-34  
 SCATTER 2-38  
 scatter 2-38  
 SCHDC 4-52  
 SCHDD 4-52  
 SCHEX 4-52  
 SCHUD 4-52  
 SCILIB 1-1, 1-3  
 SCILIB error handling 1-8  
 SCILIB, linking 1-3  
 SCILIB Programmer's Reference 1-8  
 SCNRM2 2-45  
 SCOPY 2-40  
 SDOT 2-42  
 search vector 2-6, 2-25, 2-27, 2-29, 2-31,  
 2-64, 2-67  
 second order recurrence 8-16, 8-19, 8-22  
 SGBCO 4-9  
 SGBDI 4-12  
 SGBFA 4-14  
 SGBMV 3-16  
 SGBSL 4-17  
 SGECO 4-19

SGEDI 4-21  
 SGEFA 4-24  
 SGEMM 3-20  
 SGEMMS 3-23  
 SGEMV 3-27  
 SGER 3-30  
 SGESL 4-26  
 SGTSL 4-29  
 SMXPY 3-32  
 SNRM2 2-45  
 SOLR 8-16  
 SOLR3 8-19  
 SOLRN 8-22  
 sort array 9-2  
 sparse 2-6, 2-9, 2-38, 2-47, 2-49, 2-64, 2-67  
 SPAXPY 2-47  
 SPBCO 4-31  
 SPBDI 4-34  
 SPBFA 4-36  
 SPBSL 4-39  
 SPDOT 2-49  
 SPOCO 4-41  
 SPODI 4-43  
 SPOFA 4-46  
 SPOSL 4-48  
 SPPCO 4-52  
 SPPDI 4-52  
 SPPFA 4-52  
 SPPSL 4-52  
 SPTSL 4-50  
 SQRDC 4-52  
 SQRSL 4-52  
 SROT 2-51  
 SROTG 2-53  
 SROTM 2-55  
 SROTMG 2-57  
 SSBMV 3-34  
 SSCAL 2-59  
 SSICO 4-52  
 SSIDI 4-52  
 SSIFA 4-52  
 SSISL 4-52  
 SSPCO 4-52  
 SSPMV 3-38  
 SSPR 3-41  
 SSPR2 3-44  
 SSUM 2-61  
 SSWAP 2-62  
 SSYMM 3-47  
 SSYMV 3-50  
 SSYR 3-53  
 SSYR2 3-55  
 SSYR2K 3-58  
 SSYRK 3-61  
 STBMV 3-64  
 STBSV 3-68  
 storage, backward 2-3  
 storage, forward 2-3  
 STPMV 3-72  
 STPSV 3-75  
 stride 2-3

stride arguments 2-3  
 STRMM 3-78  
 STRMV 3-81  
 STRSM 3-83  
 STRSV 3-86  
 subprogram prefixes 1-7  
 sums, partial 8-14  
 supplemental reading xxiii, 1-10, 2-4, 3-3,  
 4-5, 5-2, 6-2, 7-1  
 SVD 5-14  
 SXMPY 3-89

## T

TAC, technical assistance center xxiv  
 TAC, Technical Assistance Center A-1  
 Technical Assistance Center 1-8  
 technical assistance center, TAC xxiv  
 Technical Assistance Center (TAC) A-1  
 thread, definition 1-5  
 tilde-escape sequences A-3  
 tilde-escape sequences, restrictions on use A-6  
 TINVIT 5-14  
 Toeplitz matrix 4-8  
 TQL1 5-14  
 TQL2 5-5  
 TQLRAT 5-7  
 TRBAK1 5-14  
 TRBAK3 5-14  
 TRED1 5-9  
 TRED2 5-11  
 TRED3 5-14  
 triangular factorization 4-9, 4-14, 4-19, 4-24  
 triangular matrix-matrix multiply 3-78  
 triangular solve 3-68, 3-75, 3-83, 3-86  
 tridiagonal linear equations 4-29, 4-50  
 TRIDIB 5-14  
 trouble reports A-1  
 TSTURM 5-14

## U

UNICOS Math and Scientific Library A-1  
 UNIX-to-UNIX Communication Protocol A-1  
 UNIX-to-UNIX copy command, *ucp* A-1  
 UUCP, connection to TAC A-1  
*ucp*, UNIX-to-UNIX copy command A-1

## V

VECLIB 1-2  
 VECLIB8 1-2  
 vector 1-4  
 vector-matrix multiply and add 3-89  
*vers* command, using to find program version  
 number A-2

## W

Weiner-Levinson algorithm 4-8  
*whence* command, using to find program path  
 name A-2  
 WHENEQ 2-64  
 WHENFGE 2-64

WHENFGT 2-64  
 WHENFLE 2-64  
 WHENFLT 2-64  
 WHENIGE 2-64  
 WHENIGT 2-64  
 WHENILE 2-64  
 WHENILT 2-64  
 WHENMEQ 2-67  
 WHENMGE 2-67  
 WHENMGT 2-67  
 WHENMLE 2-67  
 WHENMLT 2-67  
 WHENMNE 2-67  
 WHENNE 2-64

*which* command, using to find program path  
 name A-2

## X

XERBLA 3-91  
 XERSCI 1-8, 9-4

## Z

zeros 2-10

abine 11-11-60  
11-11-60

11-11-60

... to ... and ... of the ...  
... in ... in ... efforts ...  
... in ...

... in ... in ...  
... in ... in ...  
... in ... in ...

Series of horizontal lines for data entry.

FOR INFORMATION OF DOCUMENTATION

| Number | Page | Location |
|--------|------|----------|
|        |      |          |
|        |      |          |
|        |      |          |

...  
...  
...  
...  
...

**CONVEX SCILIB User's Guide**  
710-013630-001, First Edition

**Reader's Forum**

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

---

---

---

---

---

---

---

---

---

---

**From:**

Name \_\_\_\_\_ Title \_\_\_\_\_

Company \_\_\_\_\_ Date \_\_\_\_\_

Address and Phone No. \_\_\_\_\_

**FOR ADDITIONAL INFORMATION OR DOCUMENTATION:**

| Location                                   | Phone Number                  |
|--------------------------------------------|-------------------------------|
| From all locations in continental U.S.     | 1(800)952-0379                |
| From locations in Alaska, Hawaii, & Canada | 1(214)497-4379                |
| From all other locations                   | Contact nearest CONVEX office |

Direct mail orders to: CONVEX Computer Corporation  
Customer Service  
PO Box 833851  
Richardson TX 75083-3851 USA

(Fold Here First)

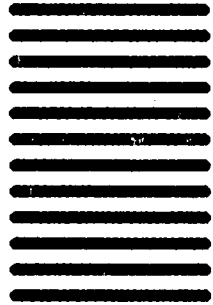


NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation  
Customer Service  
PO Box 833851  
Richardson TX 75083-3851  
USA



(Fold Here Second)

(Tape or Staple)