Using the Gaussian Elimination Method for Large Banded Matrix Equations

Jerome P.Y. Maa  
*Virginia Institute of Marine Science*

Ming-Hokng Maa  
*Massachusetts Institute of Technology*

Changqing Li  
*Andersen Consulting LLP*

Qing He  
*East China Normal University*

Follow this and additional works at: [https://scholarworks.wm.edu/reports](https://scholarworks.wm.edu/reports)

Part of the [Marine Biology Commons](https://scholarworks.wm.edu/reports)

**Recommended Citation**
Maa, J. P., Maa, M., Li, C., & He, Q. (1997) Using the Gaussian Elimination Method for Large Banded Matrix Equations. VIMS Special Scientific Report No. 135. Virginia Institute of Marine Science, College of William and Mary. [https://doi.org/10.25773/5d0g-av44](https://doi.org/10.25773/5d0g-av44)

This Report is brought to you for free and open access by W&M ScholarWorks. It has been accepted for inclusion in Reports by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).
Using the Gaussian Elimination Method for Large Banded Matrix Equations

Jerome P.-Y. Maa
School of Marine Science
Virginia Institute of Marine Science
College of William and Mary
Gloucester Point, VA 23062

Ming-Hokng Maa
Massachusetts Institute of Technology
Cambridge, MASS

Changqing Li
Andersen Consulting LLP
Minneapolis, MN

Qing He
East China Normal University
ShangHai, China.

Special Scientific Report No. 135

January 1997
Table of content

Table of Content............................................................... I
Abstract ................................................................. II
Introduction ............................................................... 1
Methodology ............................................................... 2
Simultaneous Linear Algebra Equations ................................. 4
General Banded Matrix Equation ........................................... 6
Full Banded-matrix Equation Solver ....................................... 8
A Thrifty Banded Matrix Equation Solver ................................. 9
Discussion and Conclusions ............................................... 13
Acknowledgments .......................................................... 15
References ................................................................. 15
Figures ................................................................. 16
Appendix I, Source Listing of Program P_BANDED.FOR .......... I-1
Appendix II, Source Listing of Program P_THRIFT.FOR .......... II-1
Appendix III, Source Listing of FORTRAN Program ID.FOR .... III-1
Appendix IV, Source Listing of MATLAB Program P_PLOT.M .... IV-1
Appendix V, Source Listing of P_EXACT.M ......................... V-1
Abstract

A special book-keeping method was developed to allow computers with limited random access memory but sufficient hard-disk space to feasibly solve large banded matrix equations by using the Gaussian Elimination method with Partial Pivoting. The computation time for this method is excellent because only a minimum number of disk read/write is required, freeing processor time for number crunching.
INTRODUCTION

Numerical solutions of elliptic partial differential equations with high resolution or large domains are typically approached through iteration methods, e.g., Successive-Over-Relaxation (SOR) (Roache 1972), Multi-Grid (MG) (Brandt 1984), etc. This is primarily because direct approaches such as the Gaussian Elimination method cannot feasibly solve large sets of linear algebra equations (or matrix equations) with limited computer memory. For example, a square two dimensional domain with 200 grids per side will generate a banded coefficient matrix with a dimension of $400 \times 40000$. If four bytes are used to represent real numbers, a computer would require approximately 64 megabytes (MB) of memory to store this matrix alone. If 16 bytes are used to represent complex numbers with improved accuracy, the memory requirement could easily increase to 256 MB. This memory requirement is typically too large for general computers.

Using the Gaussian Elimination method for solving a partial differential equation, however, has two advantages: (1) The numerical scheme is simple and has been available for more than 20 years; (2) The computation time is not affected by the type and complexity of boundary conditions. In contrast, the convergence rate for iteration methods typically degrades with Neumenn type boundary conditions or complex boundary geometries.

In order to practically implement the Gaussian Elimination method, however, the enormous memory requirements must be addressed. With today’s advances in hard-disk storage, computer
hard-disks on the order of gigabytes (GB) or more have become standard and commonplace. It has also become a standard practice for many operating systems to automatically satisfy an application's memory requirements with virtual memory, swapping random access memory with hard-disk storage. This built-in virtual memory, however, has very low efficiency because it was never designed for computations with matrices of very large orders. In this study, a method for solving large banded matrix equations by systematically swapping the contents of a high order matrix between memory and hard-disk is presented. This report will detail the construction of the banded matrix equation, and compare the original Gaussian Elimination method of solution, versus the thrifty banded matrix solver method of solution. Computer source codes are listed in the Appendices and are also available on disk for registered user.

METHODOLOGY

A simple two dimensional elliptic Laplace equation (Eq. 1) is used to demonstrate the construction of a banded matrix equation.

\[
\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0 \quad \text{................................................................. (1)}
\]

For simplicity, we are using Dirichlet boundary conditions in this study. The boundary conditions are specified at the border of a square domain, from 0 to \(\pi\), are as follows: \(p = 2\) at \(x = 0\); \(p = 1\) at \(y = 0\); and \(p = 0\) at both \(x = y = \pi\). For this example,
we can find the analytical solution to compare the results obtained from each approach.

Using an S-transform on the y component yields the analytical solution:

\[ p = \sum_{n=1}^{\infty} \frac{2}{n\pi} \left\{ \left[ 1 - 2(-1)^n \right] \cosh(nx) - \left[ 1 - 2(-1)^n \right] \coth(n\pi) + \cosh(n\pi) \right\} \sinh(nx) + 1 \} \sin(ny) \quad \ldots \ldots \quad (2) \]

The analytical solution (with \( n=4000 \)) is plotted in Fig. 1. The MATLAB program that calculated Eq. 2 can be found in Appendix V. Notice that the analytical solution does not exactly meet the boundary condition \( \phi = 1 \) at \( y = 0 \). This is because the solution is in the form of Fourier series, which converges to the average, i.e., zero, at the boundary \( y = 0 \). In the neighborhood of this boundary, however, \( \phi \) increases sharply and approaches 1 when \( y > 1 \times 10^{-3} \). Two selected profiles of the analytical solution demonstrate this situation in Fig. 2. In Fig. 2a, a linear scale on the y-axis indicates that \( p \rightarrow 1 \) in the neighborhood of \( y = 0 \). In Fig. 2b, a semi-log scale on the y-axis indicates that \( P = 0 \) at \( y = 0 \), but \( p \) increases very fast to approach 1 when \( y > 0.001 \).

The contour plot in Fig. 1 was generated with a spatial resolution of \( \Delta x = \Delta y = 0.03141592 \), which is not high enough to show the significant change of \( p \) for \( 0 < y < 0.001 \). Thus the \( p \) values at \( y = 0.001 \) instead of \( y = 0 \) were used with other \( p \) values to construct the contour plot (Fig. 1).
SIMULTANEOUS LINEAR ALGEBRA EQUATIONS

Using the central finite difference method, Eq. 1 can be rewritten as

\[ p_{i-1,j} + r p_{i,j-1} - 2(1+r) p_{i,j} + r p_{i,j+1} + p_{i+1,j} = 0 \]  \hspace{1cm} (3)

where \( r = (\Delta x/\Delta y)^2 \), and \( \Delta x \) and \( \Delta y \) are the grid sizes selected in the \( x \) and \( y \) directions, respectively. This equation is typically referred to as a five-point-approximation because only five points are involved in one iteration loop of an iteration method (e.g., SOR or MG) or to build a band matrix equation.

To demonstrate the construction of the banded matrix equation points in both the \( x \) and \( y \) directions, is initially chosen. See

\[
\begin{align*}
2 & + 1 - 4 p_1 + p_2 + p_{11} = 0 \\
2 & + p_2 - 4 p_3 + p_4 + p_{12} = 0 \\
2 & + p_3 - 4 p_4 + p_5 + p_{13} = 0 \\
& \vdots & \vdots & \vdots & \vdots & \vdots \\
2 & + p_9 - 4 p_9 + p_{10} + p_{19} = 0 \\
2 & + p_9 - 4 p_{10} + 0 + p_{20} = 0 \\
p_1 & + 1 - 4 p_{11} + p_{12} + p_{21} = 0 \\
p_2 & + p_{11} - 4 p_{12} + p_{13} + p_{22} = 0 \\
p_3 & + p_{12} - 4 p_{13} + p_{14} + p_{23} = 0 \\
& \vdots & \vdots & \vdots & \vdots & \vdots \\
p_{79} & + p_{89} - 4 p_{89} + p_{90} + p_{99} = 0 \\
p_{80} & + p_{89} - 4 p_{90} + 0 + p_{100} = 0 \\
& \vdots & \vdots & \vdots & \vdots & \vdots \\
p_{87} & + p_{97} - 4 p_{98} + p_{99} + 0 = 0 \\
p_{89} & + p_{98} - 4 p_{99} + p_{100} + 0 = 0 \\
p_{90} & + p_{99} - 4 p_{100} + 0 + 0 = 0
\end{align*}
\]  \hspace{1cm} (4)
Fig. 3. The solutions are the p values at each intersection point. In Fig. 3, they are marked as \( p_1, p_2, p_3, \ldots, p_{99}, p_{100} \).

Application of Eq. 3 at these intersection points (from \( p_1 \) to \( p_{100} \)) generates the following simultaneous linear equations.

The constants in the above equations can be moved to the righthand side, and the simultaneous linear equations can be written as a band matrix equation:

\[
\mathbf{A}\mathbf{p} = \mathbf{b} \quad \text{................................. (5)}
\]

where \( \mathbf{A} \) is a banded coefficient matrix, \( \mathbf{p} \) is a column matrix that contains the unknown variable \( p \) in the study domain, and \( \mathbf{b} \) is a column matrix that contains the given boundary conditions.
Matrix $\mathbf{A}$ has a dimension of $M \times N$, ($21 \times 100$), where $M$ is the band width and $N$ is the total number of unknown points within the study domain. The band width is the sum of the upper band width, $MU$, the lower band width, $ML$, and the unit width of diagonal elements, i.e., $M = MU + ML + 1$. The upper band width, $MU$, is the maximum count of the element $P_{i+1,j}$ from the diagonal element, $P_{i,j}$. Here the count represents the number of grid points between $P_{i,j}$ and $P_{i+1,j}$. The measurement begins in the $i$th column and $(j+1)$th row and continues with increasing row number until the maximum row number or the upper boundary is reached, and then continues on the first grid point (or the lower boundary) on the $(i+1)$th column, and ends at the $j$th row. Similarly, $ML$ is the maximum count between element $P_{i-1,j}$ and $P_{i,j}$.

GENERAL BANDED MATRIX EQUATION

In general, the coefficient matrix, $\mathbf{A}$, can be written as follows. Note that in Eq. 2, $r$ need not be 1. Also in Eq. 8,

$$\mathbf{P}^T = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 & \ldots & P_{i-1} & P_i & P_{i+1} & \ldots & P_{98} & P_{99} & P_{100} \end{bmatrix} \quad \ldots \ldots \ldots \quad (7)$$

$$\mathbf{B}^T = \begin{bmatrix} -3 & -2 & -2 & \ldots & -2 & 0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \end{bmatrix} \quad \ldots \quad (8)$$
\[ a_1,u \quad \cdots \quad a_{1,n} \quad a_{1,n-1} \quad a_{1,n} \]

\[ a_{d+1,1} \quad a_{d+1,2} \quad \cdots \quad a_{d+1,n-1} \quad a_{d+1,n} \]

\[ a_{m,1} \quad a_{m,2} \quad \cdots \quad a_{m,n-1} \quad a_{m,n} \]

\[ \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & a_{1,u} & \cdots & a_{1,n-L} & a_{1,n-L-1} & \cdots & a_{1,n-1} & a_{1,n} \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & a_{d+1,2} & a_{d+1,3} & \cdots & a_{d+1,u-1} & a_{d+1,u} & \cdots & a_{d+1,n-L} & a_{d+1,n-L-1} & \cdots & a_{d+1,n-1} & a_{d+1,n} \\
a_{d+1,1} & a_{d+1,2} & a_{d+1,3} & \cdots & a_{d+1,u-1} & a_{d+1,u} & \cdots & a_{d+1,n-L} & a_{d+1,n-L-1} & \cdots & a_{d+1,n-1} & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,u-1} & a_{m,u} & \cdots & a_{m,n-1} & 0 & \cdots & 0 & 0
\end{bmatrix} \quad \begin{equation} \tag{9} \end{equation} \]

\( a_{1,u}, a_{1,u+1}, \ldots, \) and \( a_{1,n} \) are not necessary all in the same row, i.e., row 1. In this example, however, they are all in the same row because of the simple boundary geometry. If the boundary geometry was irregular, then these elements would not be in the same row. Similarly, \( a_{m,1}, a_{m,2}, \ldots, \) and \( a_{m,n} \) are not necessary all in row \( M \).

A problem arises when the above stated approach is used to solve an elliptic partial differential equation when the required resolution is high, or the study domain is large, i.e., the banded matrix equation is large. To illustrate this problem, the previous example will be solved by both the original Gaussian Elimination method with Partial Pivoting and the thrifty banded matrix solver developed for this study.
FULL BANDED MATRIX EQUATION SOLVER

To solve Eq. 1 using the full banded matrix, A, a program P_BANDED.FOR (see Appendix I), which use the FORTRAN subroutines (CGBFA and CGBSL) from Linpack (Dongarra et al. 1979), was established. This program performs adequately on a personal computer (PC) with the coarse grid. Fig. 4 plots the results using a MATLAB program, P_PLOT.M (see Appendix IV), which reads the output generated by P_BANDED.FOR directly. From Fig. 4, it is obviously to see that a higher resolution is required. When the grid size is increased to 102 x 102, however, the program P_BANDED.FOR would require approximately 42 MB of memory to execute. While a Windows 3.1/95 based PC would use virtual memory to execute the program, the computing time would be very long because Windows would be constantly swapping between memory and hard-disk. Because of this reason, we changed to a Unix workstation SUN SPARC II computer with 32 MB of memory to do the job. It is still very slow for this size because of disk and memory swap. But we eventually got the results, which is identical to Fig. 1.

The input data for the program P_BANDED.FOR was generated using another FORTRAN program, ID.FOR, given in Appendix III. The input file generated by ID.FOR was as general as possible so users could easily modify the input file for different equations. Notice, however, that this program was only programmed for a Dirichlet boundary value problem. For Neumann type boundary conditions, the routine that formulates the banded coefficient
matrix would be quite different. The solution procedure for the banded matrix equation, however, would remain essentially the same.

A THRIFTY BANDED MATRIX EQUATION SOLVER

Notice that there are many zero elements in $A$, especially when the band width is large. Eq. 2 indicates that only the diagonal terms ($a_{d,j}$ with $j=1, N$), the two immediate off-diagonal terms ($a_{d-1,j}$ and $a_{d+1,j}$ with $j=1, N$), the upper border terms ($a_{1,j}$ with $j=1, N$), and the lower border terms ($a_{m,j}$ with $j=1, N$) are possibly non-zero terms. All other elements are zero. The entire banded matrix $A$, therefore, can be stored in two much smaller matrices, $V$ and $T$, each with a dimension of $N \times 5$. Matrix $V$ stores the five rows that contain possible non-zero elements as $v_{i,k}$ with $k = 1$ to $5$ and $i = 1$ to $N$. Their locations (all integer values) in the original banded matrix $A$ are stored in matrix $T$. In this way, $A$ can be stored more compactly. For comparison, the band matrix presented in the introduction would only require 1.2 MB as opposed to the original 64 MB of memory to store.

In solving Eq. 2, the two small matrices, $V$ and $T$, are recalled one block at a time to construct a working matrix, $W$, with dimensions of $(M+ML) \times (NK+M)$, where $NK$ is determined by the available computer memory.
Notice that in the working matrix, the original banded matrix was shifted ML rows downward. This is just following the approached used in Linpack. Also notice that some data were excluded after column NK+M when fetch a block (length NK) of data from V and T to construct $W$. These data will be recovered in the construction of next $W$ to maintain data integrity.
In general, the larger the NK, the smaller the number of disk I/O, thereby reducing the processing time, however, this speed reduction is only marginally because the number of disk I/O is already minimized in this process. As a rule of thumb, NK should be at least 4 x M.

After applying the procedure of Gaussian Elimination Method for a banded matrix on \( w \) (Dongarra et al. 1979), the results were saved in the first M rows (row 1 to M). The space from row M+1 to M+ML is used to save the multipliers. This is also the typical approach used in Linpack. In this study, however, only
the first M row, from column 1 to NK, was written into hard disk as a temporary file. The multipliers in row M+1 to M+ML were ignored because there is no use for a single column matrix B. Only for those cases that matrix B is a rectangular matrix, we then need to store these multipliers.

After write a disk file, data in the block from column NK+1 to NK+M is then moved to the beginning of the working matrix, i.e., column 1 to column M. The data ignored when expanding the first block of V and T into W is then retrieved, and the second block (also length of NK) of V and T is then expanded into the working matrix. Some data at the end of this block must again be ignored due to a lack of space in W. This process repeats until the entire V and T have been processed. During this forward elimination process, many temporary files will have been written into the hard-disk. These files are written in binary form to improve I/O times and to conserve disk space.

The backward substitution method implemented in the FORTRAN subroutine CGBSL from Linpack is then used to find the solution, P, also one block at a time. This routine is relatively simple: the last disk file was read first (and deleted), and the backward substitution routine is used to calculate the solutions for this block. This process also repeated for all the blocks to find out P.

Using the finite difference method with dx = dy = 0.031105, i.e., a grid size of 102 x 102. With the thrifty banded matrix solver, we only need a working matrix that has a size of 310 x
400. Thus, the size of executable codes reduces to about 2 MB. Of course, one needs about 40 MB of disk space for writing temporary files, which were deleted at the end. Under this condition, the example problem given previously can be solved using a personal computer with reasonable time. For example, the computer time for solve the same problem using a Pentium 75 MHz PC was about 160 sec. The results are identical to that given in Fig. 1.

Source list of this thrifty banded matrix solver, a FORTRAN program, P_THRIFT.FOR, is given in Appendix II. The input data file is the same as that used for the program P_BANDED.FOR, and the results were also plotted using the MATLAB program P_PLOT.M. Some of the subroutines (e.g., DOMAIN, ICAMAX, CAXPY, CSCAL, IDISTR, and IDISTL) used in this program are identical to those used in the program P_BANDED.FOR. But they are put together to have a complete file.

DISCUSSION AND CONCLUSIONS

The Gaussian Elimination method involves two steps: forward elimination and backward substitution. Mathews (1987) pointed out that the number of computations involved in the forward elimination is proportional to $N^3$, where $N$ is the order of the coefficient matrix. In contrast, the number of computations involved in backward substitution is proportional to $N^2$. Computation time, therefore, is mostly spent in forward elimination. If there are many linear systems (all with the same
coefficient matrix) to be solved, then the forward elimination should only be calculated once, and the results, multipliers, should be saved for different B matrices. While the book-keeping procedure developed in this study could easily be adapted to solve multiple linear systems, the details are out of the scope of this study.

For very large banded matrix equations, i.e., a large N, the round-off error may become significant. To improve accuracy, it may become necessary to use double precision (8 bytes for real numbers). This increase in precision, however, does not significantly increase the memory requirement because the size of the executable code is small to begin with, and sufficient hard-disk space is not difficult to obtain.

While only a simple application was presented in this note, this method can be used in many applications, especially when there are Neumann boundary conditions and an irregular boundary geometry. For those situations, the formulation of the banded matrix equation may differ, but the banded matrix solver remains the same.

The use of the Gaussian Elimination Method with Partial Pivoting, the development of a special book-keeping procedure, and the high availability of large hard-disks allow the practical solution of large banded matrix equations. The feasibility of efficiently calculating the solution of an elliptic partial differential equation with large study domain and/or high resolution on a personal computer was demonstrated. While
developed for computers with limited resources, however, this method could easily compute domains on a global scale when processed on computers with large memory and/or disk resources.

ACKNOWLEDGMENTS


REFERENCES
Brandt, A., 1984, Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, GMD study, number 85, Postfach 1240, Birlinghoven, D-5205 St. Augustin 1, Germany.


Fig. 1. Contours Plot of the Analytical Solution.
Fig. 2. Analytical Solution Profiles at Selected x locations.
Fig. 3. Coarse Grid Points in the Study Domain.
Fig. 4. Numerical Solution Using the Coarse Grid
Appendix I. Source Lists of FORTRAN program P_BANDED.FOR

program p_banded

This program is to calculate the velocity potential \( p \) in the area \( 0.0 \leq x \leq 3.1416, \ 0.0 \leq y \leq 3.1416 \).

a thrify storage method was constructed first, and then, convert to that required by LINPACK, and uses a minor modified LINPAK subroutines to solve it.

Because of the formidable slow pace when running this program with high resolution, we can only limited the parameter \( KQ \) to a small number.

\[
\text{parameter (iq=110,jq=110, IW=310, kq=400, Lq=3, mq=203)}
\]

\[
\text{implicit complex (z)}
\]

\[
\text{character*45 title, label}
\]

\[
\text{character*20 infile, outfile, recfile}
\]

\[
\text{character*1 id, q1}
\]

\[
\text{common /matrx/ za(kq,5), ia(kq,5), zb(kq)}
\]

\[
\text{common /gener/ mp, np, dx, dy, r, p(iq,jq), id(iq,jq)}
\]

\[
\text{common /unknw/ n, imap(kq), jmap(kq), irow(iq)}
\]

\[
\text{common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)}
\]

\[
\text{common /works/ zabd(iw,kq), ipvt(kq)}
\]

boundary conditions

bcx0=2
bcxn=0
bcy0=1
bcyn=0

print*, 'Select 1. p_s.grd; 2. p_m.grd 3. p_L.grd: ', read(*,*) iption
go to (10, 12, 14), iption

10 infile='p_s.grd'
outfile='P_bs.out'
recfile='p_bs.rec'
go to 20

12 infile='p_m.grd'
outfile='P_bm.out'
recfile='p_bm.rec'
go to 20

14 infile='p_L.grd'
outfile='P_bL.out'
recfile='p_bL.rec'
continue

open(9, file=infile, status='old')
open(10, file=outfile, status='unknown')
open(11, file=recfile, status='unknown')
mp, np : Max. grid numbers in x and y direction, respectively.
dx, dy : grid sizes in x and y direction, respectively.

read(9,'(a45)') title
write(11,'(a45)') title
read(9,*) mp, np, dx, dy
if(mp .gt. iq) then
  print*,'IQ is smaller than MP, Change IQ to ',mp
  stop
end if
if(np .gt. jq) then
  print*,'JQ is smaller than nP, Change JQ to ',np
  stop
end if

r = (dx/dy)**2

ID code for each grid point
: 0, interior point
: 1, upper boundary condition
: 2, lower boundary condition
: 3, left boundary condition
: 4, right boundary condition
: 5, left bottom corner B.C.
: 6, left top corner B.C.
: 7, right bottom corner B.C.
: 8, right top corner B.C.
: e, grid point that is not included in the study domain

The following is an example

<table>
<thead>
<tr>
<th></th>
<th>np</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>61111111111118eeeeeeeeeeeeeeeeeeeeee</td>
</tr>
<tr>
<td></td>
<td>3000000000004eeeeecccccccccccccceee</td>
</tr>
<tr>
<td></td>
<td>30000000000011111111111111111111</td>
</tr>
<tr>
<td></td>
<td>3000000000000000000000000000000004</td>
</tr>
<tr>
<td></td>
<td>3000000000000000000000000000000004</td>
</tr>
<tr>
<td></td>
<td>3000000000000000000000000000000004</td>
</tr>
<tr>
<td></td>
<td>3000000000000000000000000000000004</td>
</tr>
<tr>
<td></td>
<td>52200000000000000000000000000000222227</td>
</tr>
<tr>
<td></td>
<td>eee3000000000000000000000000000000004eeee</td>
</tr>
<tr>
<td></td>
<td>eee3000000000000000000000000000000004eeeee</td>
</tr>
<tr>
<td></td>
<td>eee3000000000000000000000000000000004eeee</td>
</tr>
</tbody>
</table>

| i     | mp |
jem| 1 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>522222222222222222222227eeeee</td>
</tr>
</tbody>
</table>

read(9,30) label
write(11,30) label
read(9,30) label
write(11,30) label
30 format(a50)
do j=1,np
   jj=np-j+1
read(9,45) jns, (id(i,jj),i=1,mp)
45 format(i5, (110a1) )
if(jns .ne. jj) then
   write(11,50) jj,jns
50 format(' Sequence is wrong for ID input at ',2i5)
   stop
end if
end do
print*,'Completed reading ID code matrix'
close(9)

construct the unknown COLUMN matrix X, in the matrix eq. AX=B
find each unknown's location: imap(map),jmap(map)
ind(i) : no. of isolated sector in each column, x grid.
for this particular case, ind(i) are all 1.
because no land points in the middle of study domain.
jin(i,index) : begin grid number for an isolated sector in a
column
jout(i,index): end grid number for an isolated sector in a
column
map : the number of total unknown, or the length of X.
later, it is reassigned as N
irow(i) : the total number of unknown vel. Potential in each
column

map=0
do i=1,mp
   icount=0
   in=0
   iout=1
   index=1
   do j=1,np
      if(id(i,j) .eq. '0') then
         icount=icount+1
         map=map+1
         imap(map)=i
         jmap(map)=j
         if(in .eq. 0) then
            jin(i,index)=j
            in=1
            iout=0
         end if
      end if
   end do
   if( id(i,j) .eq. '1' .and. iout .eq. 0) then
      I-3
c
jout(i,index)=j
iout=1
in=0
index=index+1
end if
end do
irow(i)=icount
ind(i)=index-1
c
ind(i) should be >= 1, except for the entire column are all boundary points. if not, something wrong.
c
write(11,55) i, ind(i), irow(i), jin(i,1), jout(i,1)
55 format(' i,ind,irow,jin,jout=',i,i,i,i,i)
if( ind(i) .gt. Lq) then
write(11,60) i, ind(i), Lq, ind(i)
60 format(' At i=',i,' Ind(i)=',i,' > Lq=',i,' Change Lq to',i,' and re-run')
stop
end if
end do
c
n: length of the banded matrix
c
n=map
if(n .lt. 0.8*kq) then
write(*,65) n, kq
65 format('--------------------------------------------------'/
* N=',i,' < KQ=',i,' It is better to reduce KQ, '/
* length of the Banded Matrix, '/ So that KQ is not'
* >> than that required, '/ Instead, one should'
* increase the length of the working matrix, JW, '/
* in order to reduce disk I/O and computing time, '/
* You may continue, or re-run < c/r > : ')
read(*, '(a1)') q1
if( q1 .eq. 'r' .or. q1 .eq. 'R') stop
end if
if(n .gt. kq) then
write(11,70) n, kq, n
70 format('--------------------------------------------------'/
* N=',i,' > KQ=',i,' Increases KQ to ',i,' and re-run')
stop
end if
c
Set up the two small matrices ZA and IA for storing the banded matrix and find out the band width, work size, etc.
c
mu=0  
I-4


```fortran
mL=0
write(11,80)
80 format(' i  j  ia(1), ..., ia(4), ia(5) ',
   * ' za(1) ... za(5) zb')
do map=1,n
i=imap(map)
j=jmap(map)
call domain(i, j, map, ierr)
if( ierr .eq. 0) then
   write(11,90)i,j,ia(map,1),ia(map,2),ia(map,3),ia(map,4),
   * ia(map,5), real(za(map,1)), real(za(map,2)),
   * real(za(map,3)), real(za(map,4)), real(za(map,5)),
   * real(zb(map))
90 format(1x,2i5,2x,5i5,2x,6f6.2)
else
   print*, 'stop, error in DOMAIN at i,j,map=', i,j,map
   print*, 'Check the recording file for details.'
   stop
end if
if( ia(map,5) .ne. 0 ) then
   mu_c=abs(ia(map,5) - ia(map,3) )
else
   mu_c=0
end if
if(mu .lt. mu_c) mu=mu_c
if( ia(map,1) .ne. 0 ) then
   mL_c=abs(ia(map,1) - ia(map,3) )
else
   mL_c=0
end if
if(mL .lt. mL_c) mL=mL_c
end do
Lda=2*mL+mu+1
m=mL+mu+1
write(*,100) m, n, mu, mL, Lda
100 format(' Band width, m = ',i7/
   * ' Data point, N = ',i7/
   * ' Upper B.W., mu = ',i7/
   * ' Lower B.W., mL = ',i7/
   * ' Lda for LINPAK = ',i7)
c if(m .gt. mq) then
   print*, 'M > MQ, please increase MQ to ',m
   stop
end if
c change the thrifty storage to band storage specified by LINPAK,
c LINPAK uses rows ML+1 through 2*ML+MU+1 of ZABD to store the
c original banded matrix. In addition, the first ML rows in ZABD
c are used for elements generated during the triangularization.
c The total number of rows needed in ZABD is 2*ML+MU+1.
```

I-5
C The ML+MU by ML+MU upper left triangle and the ML by ML lower right triangle are not referenced.
C but first clear the matrix

C

-do map=1,n
   do k=1,Lda
      zabd(k,map) = (0.0, 0.0)
   end do
end do
C position the diagonal element

C

kkk==mu+1+ML
-do map=1,n
   if( ia(map,3) .eq. map) then
      zabd(kkk,map) = za(map,3)
   else
      print*,'Check conversion,map,za(i,3)=' ,map,za(map,3)
      stop
   end if
end do
C position the low triangular elements
C
C
-do k=1,2
   if(ia(map,k) .gt. 0) then
      ik=kkk + ia(map,3) - ia(map,k)
jk=map - (ia(map,3) - ia(map,k) )
zabd(ik,jk)==za(map,k)
   end if
-end do
-do k=4,5
   if(ia(map,k) .gt. 0) then
      ik=kkk + ia(map,3) - ia(map,k)
jk=map + ia(map,k) - ia(map,3)
zabd(ik,jk)==za(map,k)
   end if
-end do
-end do
C
print*,'Completed conversion to LINPACK standard'
print*,'The matrix length is from 1 to',n
print*,' Select a checking domain from n1 ,n2 ='
read(*,*) n1, n2
-do map=n1,n2
   write(*,130) imap(map), jmap(map), map, real(zb(map)),
   * (real(zabd(k,map)),k=1,Lda)
   130 format('i,j,map,b=','2i5,i8,f8.1/(10f7.1))
end do
C
call gauss elimination to slove the complex band matrix equ.
print*, 'Calling CGBFA, wait ...

call CGBFA(Lda, n, ml, mu, info)

if (info .eq. 0) then
  print*, 'Successfull called CGBFA '
  print*, 'Want to store results in a file <y/n> : '
  read(*, '(a1)') q1
  if (q1 .eq. 'y' .or. q1 .eq. 'Y') then
    write(11,132)
    format(' i zabd(i,j), j=1,n')
    do i=1,Lda
      write(11,13S) i, (real(zabd(i,j)) ,j=1,n)
    end do
  end if
else
  write(*,140) info
  format(' error code (info) =', i5,' After called CGBFA')
end if

call CGBSL(Lda, n, ml, mu, zB)

c if (info .eq. 0) then
  print*, 'Successfully called CGBRL'
else
  write(*,160) info
  format(' error code (info) =', i5,' after called CGBFA')
end if

do map=1, n
  i=imap(map)
  j=jmap(map)
  p(i,j)=real(zb(map))
  end do

do j=1,np
  p(1,j) = bcx0
  p(mp,j) = bcxn
  end do

do i=1,mp
  p(i,1) = bcy0
  p(i,np) = bcyn
  end do

p(1,1) = 0.5*(bcy0+bcx0)
p(1,np) = 0.5*(bcyn+bcx0)
p(mp,1) = 0.5*(bcy0+bcxn)
p(mp,np) = 0.5*(bcyn+bcxn)

write(10, '(a45)') title
write(10, '(2i5,2f12.8)') mp,np,dx,dy

I-7
do j=1,np
  write(10,200) j, (p(i,j),i=1,mp)
  format(i5/(10f8.4))
end do

close(10)
close(11)

print *, 'program stop'
stop
end

c--------------------------------------------------------

subroutine domain(i, j, map, ier)

parameter (iq=110, jq=110, IW=310, kq=400, Lq=3)
implicit complex (z)
character*1 id
common /matrx/ za(kq,5), ia(kq,5), zb(kq)
common /gener/ mp, np, dx, dy, r, p(iq,jq), id(iq,jq)
common /unknw/ n, imap(kq), jmap(kq), irow(iq)
common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)
common /works/ zabd(iw,kq), ipvt(kq)
common /tempo/ zam(203,kq)

dimension iid(9)

c The finite difference governing equation, the Laplace equation, c with \( r = (\text{dx/dy})^2 \), is

c \[ \begin{align*}
  p(i-1,j) + r*p(i,j-1) - (2+2r)*p(i,j) + r*p(i,j+1) + p(i+1,j) &= 0
\end{align*} \]

c In storage, the coefficient of \( p(i-1,j) \) is stored in (map,1),
the coefficient of \( p(i,j-1) \) is stored in (map,2),
the coefficient of \( p(i,j) \) is stored in (map,3),
and the coefficient of \( p(i,j+1) \) is stored in (map,4),
and the coefficient of \( p(i+1,j) \) is stored in (map,5)

c ier=0
do k=1,9
  iid(k)=0
  icount=0
end do

c For those grid point that are not neighbored to a boundary

c if(id(i-1,j) .eq. '0' .and. id(i+1,j) .eq. '0' .and.
  id(i,j-1) .eq. '0' .and. id(i,j+1) .eq. '0') then
  za(map,1)=1.0
  za(map,2)=r
  za(map,3)=-2.0*(1.0+r)
za(map,4)=r
za(map,5)=1.0
ia(map,1)=map-distl(i,j)+1
ia(map,2)=map-1
ia(map,3)=map
ia(map,4)=map+1
ia(map,5)=map+distr(i,j)-1
zb(map)=(0.0, 0.0)
icount=icount+1
iid(1)=1
end if

for those has a Boundary grind point on the left

if(id(i-1,j) .eq. '3' .and. id(i,j-1) .eq. '0') then
  za(map,1)=0.0
  za(map,2)=r
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=r
  za(map,5)=1.0
  ia(map,1)=0
  ia(map,2)=map-1
  ia(map,3)=map
  ia(map,4)=map+1
  ia(map,5)=map+distr(i,j)-1
  zb(map)=(-2.0, 0.0)
  icount=icount+1
  iid(2)=1
end if

for those has a Boundary grind point on the right

if(id(i+1,j) .eq. '4' .and. id(i,j-1) .eq. '0') then
  za(map,1)=1.0
  za(map,2)=r
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=r
  za(map,5)=0.0
  ia(map,1)=map-distl(i,j)+1
  ia(map,2)=map-1
  ia(map,3)=map
  ia(map,4)=map+1
  ia(map,5)=0
  zb(map)=(0.0, 0.0)
  icount=icount+1
  iid(3)=1
end if

for those has a Boundary grind point on the top
if (id(i,j+1) .eq. '1' .and. id(i-1,j) .eq. '0' .and.
* id(i+1,j) .eq. '0' ) then
    za(map,1)=1.0
    za(map,2)=r
    za(map,3)=-2.0*(1.0+r)
    za(map,4)=0
    za(map,5)=1.0
    ia(map,1)=map-idistl(i,j)+1
    ia(map,2)=map-1
    ia(map,3)=map
    ia(map,4)=0
    ia(map,5)=map+idistr(i,j)-1
    zb(map)=(0.0, 0.0)
    icount=icount+1
    iid(4)=1
end if

for these grid points that has a Boundary at bottom

if (id(i,j-1) .eq. '2' .and. id(i-1,j) .eq. '0' .and.
* id(i+1,j) .eq. '0' ) then
    za(map,1)=1.0
    za(map,2)=0
    za(map,3)=-2.0*(1.0+r)
    za(map,4)=r
    za(map,5)=1.0
    ia(map,1)=map-idistl(i,j)+1
    ia(map,2)=0
    ia(map,3)=map
    ia(map,4)=map+1
    ia(map,5)=map+idistr(i,j)-1
    zb(map)=(-1.0, 0.0)
    icount=icount+1
    iid(5)=1
end if

For those grid points that are left bottom corner

if (id(i-1,j) .eq. '3' .and. id(i,j-1) .eq. '2') then
    za(map,1)=0.0
    za(map,2)=0
    za(map,3)=-2.0*(1.0+r)
    za(map,4)=r
    za(map,5)=1.0
    ia(map,1)=0
    ia(map,2)=0
    ia(map,3)=map
    ia(map,4)=map+1
    ia(map,5)=map+idistr(i,j)-1
    zb(map)=(-3.0, 0.0)
    icount=icount+1
    iid(6)=1
end if
end if

c For those grid points that are left top corner Boundary points

c if(id(i-1,j) .eq. '3' .and. id(i,j+1) .eq. '1') then
  za(map,1)=0.0
t end if

zb(map) = (-2.0, 0.0)
icount=icount+1
iid(7)=1

c For those grid points that are right bottom boundary points

c if(id(i+1,j) .eq. '4' .and. id(i,j-1) .eq. '2' ) then
  za(map,1)=1.0
  za(map,2)=0
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=r
  za(map,5)=0.0
  ia(map,1)=map-idistl(i,j)+1
  ia(map,2)=0
  ia(map,3)=map
  ia(map,4)=map+1
  ia(map,5)=0
  zb(map)=(-1.0, 0.0)
  icount=icount+1
  iid(8)=1
end if

c For those grid points that are right top boundary points

c if(id(i+1,j) .eq. '4' .and. id(i,j+1) .eq. '1') then
  za(map,1)=1.0
  za(map,2)=r
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=0
  za(map,5)=0.0
  ia(map,1)=map-idistl(i,j)+1
  ia(map,2)=map-1
  ia(map,3)=map
  ia(map,4)=0
  ia(map,5)=0
  zb(map)=(0.0, 0.0)
icount=icount+1
iid(9)=1
end if

if( icount .ne. 1) then
  ier=1
  write(11,10) ier, icount, (iid(k),k=1,9)
  format(' ier, icount=',2i5,' id(1)-id(9)=',9i2)
end if

return
end

-----------------------------------------------------------------
c

integer function idistr(i,j)
parameter (iq=110,jq=110, IW=310, kq=400, Lq=3)
implicit complex (z)
character*1 id
common /matrx/ za(kq,5), ia(kq,5), zb(kq)
common /gener/ mp, np, dx, dy, r, p(iq,jq), id(iq,jq)
common /unknw/ n, imap(kq), jmap(kq), irow(iq)
common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)
common /works/ zabd(iw,kq), ipvt(kq)

Calculates the distance between points (i+1,j) and (i,j). It means how many grid points, which are solved for, are in between. It counts vertically from the point(i,j) and up (i,j+1), (i,j+2), ..., (i,np), then (i+1,1), (i+1,2), ..., to (i+1,j). In the matrix equation, this distance represents the up band width at that particular diagonal element (i,j).

do k=1,ind(i)
  if(j .ge. jin(i,k) .and. j .le. jout(i,k) ) mark1=k
  end do

do k=1,ind(i+1)
  if(j .ge. jin(i+1,k) .and. j .le. jout(i+1,k) ) mark2=k
  end do

idd=jout(i,mark1)-j+1 + j-jin(i+1,mark2)

do k=mark1+1, ind(i)
  idd=idd + jout(i,k)-jin(i,k)+1
  end do

do k=mark2-1, ind(i+1)
  idd=idd + jout(i+1,k)-jin(i+1,k)+1
  end do

idistr=idd

return
integer function idistl(i,j)

parameter (iq=110, jq=110, IW=310, kq=400, Lq=3)
implicit complex (z)
character*1 id
common /matrx/ za(kq,5), ia(kq,5), zb(kq)
common /gener/ mp, np, dx, dy, r, p(iq,jq), id(iq,jq)
common /unknw/ n, imap(kq), jmap(kq), irow(iq)
common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)
common /works/ zabd(iw,kq), ipvt(kq)
common /tempo/ zam(203,kq)

Calculate the distance between point (i-1,j) and (i,j). The
distance means how many grid points, which are solved for, are
in between. It counts vertically from (i,j) downward (i,j-1),
(i,j-2), ..., (i,1) and restarted from previous i line
(i-1,np), (i-1,np-1), ..., to (i-1,j).

do k=1,ind(i-1)
  if(j .ge. jin(i-1,k) .and. j .le. jout(i-1,k)) mark1=k
end do

do k=1,ind(i)
  if(j .ge. jin(i,k) .and. j .le. jout(i,k)) mark2=k
end do

idd=jout(i-1,mark1)-j+1 + j-jin(i,mark2)

do k=markl+1,ind(i-1)
  idd=idd+jout(i-1,k)-jin(i-1,k)+1
end do

do k=1,mark2-1
  idd=idd+jout(i,k)-jin(i,k)+1
end do

idistl=idd
return
end

SUBROUTINE CGBFA(LDA, N, ML, MU, INFO)

Original Date of WRITTEN Aug. 14 1978 by MOLEZ, C. B.,
PURPOSE Factors a COMPLEX band matrix by Gaussian elimination
and partial pivoting.

CGBFA is usually called by CGBCO, but it can be called
directly with a saving in time if RCOND is not needed.
On Entry

ABD  COMPLEX(LDA, N), contains the matrix in band storage. The
columns of the matrix are stored in the columns of ABD and
the diagonals of the matrix are stored in rows ML+1
through 2*ML+MU+1 of ABD. See comments below for details.

LDA  INTEGER, the leading dimension of the array ABD .
LDA must be .GE. 2*ML + MU + 1.

N  INTEGER, the order of the original matrix.

ML  INTEGER, number of diagonals below the main diagonal.
   0 .LE. ML .LT. N.

MU  INTEGER, number of diagonals above the main diagonal.
   0 .LE. MU .LT. N. More efficient if ML .LE. MU

On Return

ABD an upper triangular matrix in band storage and the
multipliers which were used to obtain it. The
factorization can be written A = L*U where L is a
product of permutation and unit lower triangular matrices
and U is upper triangular.

IPVT INTEGER(N), an integer vector of pivot indices.

INFO INTEGER
   = 0  normal value.
   = K  if U(K,K) .EQ. 0.0 . This is not an error condition,
      but it does indicate that CGBSL will divide by zero if
      called. Use RCOND in CGBCO for a reliable indication
      of singularity.

This uses rows ML+1 through 2*ML+MU+1 of ABD. In addition, the
first ML rows in ABD are used for elements generated during
the triangularization. The total no. of rows needed in ABD is
2*ML+MU+1. The ML+MU by ML+MU upper left triangle and the ML
c x ML lower right triangle are not referenced.

LINPACK. This version dated 08/14/78 .
Cleve Moler, University of New Mexico, Argonne National Lab.

Subroutines and Functions called from this subroutine

   BLAS CAXPY,CSCAL,ICAMAX
   Fortran ABS,AIMAG,MAX0,MIN0,REAL

REF: DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,

I-14
ROUTINES CALLED CAXPY, CSCAL, ICAMAX

Revised by Jerome P.-Y. Maa on Nov. 96.

1. Move the matrix abd(Lda, 1) and Ipvt(1) to common block. so
   that the limitation of LDA = IW can be relaxed. In other
   words, the parameter IW specified don’t have to be exactly
   equal to LDA.

2. uses implicit statement for complex variables

parameter (IW=310, kq=400)
implicit complex (z)
character*1 q1
common /works/ zabd(iw,kq), ipvt(kq)
REAL
CABS1
CABS1(ZDUM) = ABS(REAL(ZDUM)) + ABS(AIMAG(ZDUM))

FIRST EXECUTABLE STATEMENT CGBFA

M = ML + MU + 1
INFO = 0

ZERO INITIAL FILL-IN COLUMNS

J0 = MU + 2
J1 = MIN0(N,M) - 1
IF (J1 .LT. J0) GO TO 30
DO 20 JZ = J0, J1
   IO = M + 1 - JZ
   DO 10 I = IO, ML
      zABD(I,JZ) = (0.0E0,0.0E0)
   CONTINUE
20 CONTINUE
30 CONTINUE
JZ = J1
JU = 0

GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING

NM1 = N - 1
IF (NM1 .LT. 1) GO TO 130
DO 120 K = 1, NM1
   KP1 = K + 1

ZERO NEXT FILL-IN COLUMN

JZ = JZ + 1
IF (JZ .GT. N) GO TO 50
IF (ML .LT. 1) GO TO 50
DO 40 I = 1, ML
   zABD(I,JZ) = (0.0E0,0.0E0)
40 CONTINUE
50 CONTINUE
C FIND L = PIVOT INDEX

LM = MINO(ML,N-K)
L = ICAMAX(LM+1,zABD(M,K),1) + M - 1
IPVT(K) = L + K - M

ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED

IF (CABS1(zABD(L,K)) .EQ. 0.0E0) GO TO 100

INTERCHANGE IF NECESSARY

IF (L .EQ. M) GO TO 60
  zT = zABD(L,K)
  zABD(L,K) = zABD(M,K)
  zABD(M,K) = zT
60 CONTINUE

COMPUTE MULTIPLIERS

zT = -(1.0E0,0.0E0)/zABD(M,K)
CALL CSCAL(LM, zT, zABD(M+1,K), 1)

ROW ELIMINATION WITH COLUMN INDEXING

JU = MINO(MAXO(JU,MU+IPVT(K)),N)
MM = M
IF (JU .LT. KP1) GO TO 90
DO 80 J = KP1, JU
  L = L - 1
  MM = MM - 1
  zT = zABD(L,J)
  IF (L .EQ. MM) GO TO 70
    zABD(L,J) = zABD(MM,J)
    zABD(MM,J) = zT
70 CONTINUE
  CALL CAXPY(LM, zT, zABD(M+1,K),1, zABD(MM+1,J),1)
80 CONTINUE
90 CONTINUE
GO TO 110
100 CONTINUE
  INFO = K
  print*, 'L,k,abd=',L,k,zabd(L,k)
  read(*,'(a1)') q1
110 CONTINUE
120 CONTINUE
130 CONTINUE
IPVT(N) = N
IF (CABS1(zABD(M,N)) .EQ. 0.0E0) then
  INFO = N
  print*, 'n,zabd(m,n)=', n, zabd(m,n)
I-16
end if

RETURN
END

SUBROUTINE CGBSL(LDA, N, ML, MU, zb)

Original written on Aug. 14, 78, revised on Aug. 1, 82
AUTHOR MOLER, C. B., (U. OF NEW MEXICO)

DESCRIPTION
CGBSL solves the complex band system
A * X = B or CTRANS(A) * X = B
using the factors computed by CGBCO or CGBFA.

On Entry

ABD  COMPLEX(LDA, N), the output from CGBCo or CGBFA,
      passed by common block.
LDA  INTEGER,   the leading dimension of the array ABD .
N    INTEGER,   the order of the original matrix.
ML   INTEGER,   number of diagonals below the main diagonal.
MU   INTEGER,   number of diagonals above the main diagonal.
IPVT INTEGER(N), the pivot vector from CGBCO or CGBFA.
      passed to this subroutine by common block.
zB   COMPLEX(N), the right hand side vector.

On Return

zB   the solution vector X .

Error Condition

A division by zero will occur if the input factor contains a
zero on the diagonal. Technically this indicates singularity
but it is often caused by improper arguments or improper
setting of LDA . It will not occur if the subroutines are
called correctly and if CGBCO has set RCOND .GT. 0.0 or CGBFA
has set INFO .EQ. 0 .

LINPACK. This version dated 08/14/78 .
Cleve Moler, University of New Mexico, Argonne National Lab.

Subroutines and Functions used in this subroutine

BLAS CAXPY, zCDOTC  Fortran CONJG,MIN0

REF:  DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,
      LINPACK USERS GUIDE, SIAM, 1979.

I-17
1. the matrix ZABD, IPVT are transferred by common block
2. delete the Job parameter, so this subroutine only for solving AX=B
3. use implicit for complex variables

parameter (IW=310, kq=400)
implicit complex (z)
common /works/ zabd(iw,kq), ipvt(kq)
dimension zb(1)

\[ M = MU + ML + 1 \]
\[ NM1 = N - 1 \]

SOLVE \( A \times X = B \), but first to do the same factoralization and partial pivoting on the column matrix ZB.

IF (ML .EQ. 0) GO TO 30
IF (NM1 .LT. 1) GO TO 30
DO K = 1, NM1
   LM = MIN0(ML,N-K)
   L = IPVT(K)
   zT = zB(L)
   IF (L .EQ. K) GO TO 10
   zB(L) = zB(K)
   zB(K) = zT
10  CONTINUE
   CALL CAXPY(LM, zT, zABD(M+1, K), 1, zB(K+1), 1)
end do
30  CONTINUE

NOW SOLVE \( A \times X = B \)

DO KB = 1, N
   K = N + 1 - KB
   zB(K) = zB(K)/zABD(M, K)
   LM = MIN0(K, M) - 1
   LA = M - LM
   LB = K - LM
   zT = -zB(K)
   CALL CAXPY(LM, zT, zABD(LA, K), 1, zB(LB), 1)
end do

Recovery sequence of unknown

DO 1000 KB=1,N
   K=N+1-KB
   L=IPVT(K)
   zT=zB(L)
   IF(L . EQ. K) GOTO 1000
   zB(L)=zB(K)
1000 CONTINUE
SUBROUTINE CSCAL(N, CA, CX, INCX)

PURPOSE
Complex vector scale x = a*x

WRITTEN on Oct. 1, 79, REVISION on Aug. 01, 82

CATEGORY NO. D1A6

KEYWORDS BLAS, COMPLEX, LINEAR ALGEBRA, SCALE, VECTOR

AUTHORS
LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)

KINCAID, D. R., (U. OF TEXAS), and KROGH, F. T., (JPL)

DESCRIPTION

BLAS Subprogram

Description of Parameters

-- Input --
N number of elements in input vector(s)
CA complex scale factor
CX complex vector with N elements
INCX storage spacing between elements of CX

-- Output --
CSCAL complex result (unchanged if N .LE. 0)

replace complex CX by complex CA*CX.
For I = 0 to N-1, replace CX(1+I*INCX) with 
   CA * CX(1+I*INCX)

REF: LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
"BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE,"
ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323

ROUTINES CALLED (NONE)

COMPLEX CA, CX(1)

FIRST EXECUTABLE STATEMENT CSCAL

IF(N .LE. 0) RETURN

NS = N*INCX

DO I = 1, NS, INCX
    CX(I) = CA*CX(I)
end do

RETURN
END

SUBROUTINE CAXPY(N, CA, CX, INCX, CY, INCY)
B L A S Subprogram

Description of Parameters

---Input---
N       number of elements in input vector(s)
CA      complex scalar multiplier
CX      complex vector with N elements
INCX    storage spacing between elements of CX
CY      complex vector with N elements
INCY    storage spacing between elements of CY

---Output---
CY      complex result (unchanged if N .LE. 0)

Overwrite complex CY with complex CA*CX + CY.
For I = 0 to N-1, replace
CY(LY+I*INCY) with CA*CX(LX+I*INCX) + CY(LY+I*INCY),
where
LX = 1 if INCX .GE. 0, else LX = (-INCX)*N
and LY is defined in a similar way using INCY.

REFERENCES
LAWSON C. L., HANSON R. J., KINCAID D. R., KROGH F. T.,
*BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323

ROUTINES CALLED (NONE)

COMPLEX CX(1), CY(1), CA

FIRST EXECUTABLE STATEMENT CAXPY
CANORM = ABS(REAL(CA)) + ABS(IMAG(CA))
IF(N.LE.0.OR.CANORM.EQ.0.E0) RETURN
IF(INCX.EQ.INCY.AND.INCX.GT.0) GO TO 20
KX = 1
KY = 1
IF(INCX.LT.0) KX = 1+(1-N)*INCX
IF(INCY.LT.0) KY = 1+(1-N)*INCY
DO 10 I = 1,N
   CY(KY) = CY(KY) + CA*CX(KX)
   KX = KX + INCX
   KY = KY + INCY
10 CONTINUE
RETURN
20 CONTINUE
NS = N*INCX
DO 30 I=1,NS,INCX
CY(I) = CA*CX(I) + CY(I)
30 CONTINUE
RETURN
END

INTEGER FUNCTION ICAMAX(N,CX,INCX)

WRITTEN on Oct. 01, 79, REVISED on Aug. 01, 82
CATEGORY NO. D1A2
KEYWORDS BLAS, COMPLEX, LINEAR ALGEBRA, MAXIMUM COMPONENT, VECTOR
AUTHOR LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
PURPOSE Find the location (or index) of the largest component
of a complex vector
DESCRIPTION

BLAS Subprogram
Description of Parameters

--Input--
N number of elements in input vector(s)
CX complex vector with N elements
INCX storage spacing between elements of CX

--Output--
ICAMAX smallest index (zero if N .LE. 0)

Returns the index of the component of CX having the
largest sum of magnitudes of real and imaginary parts.
ICAMAX = first I, I = 1 to N, to minimize
ABS(REAL(CX(1-INCX+1*INCX))) + ABS(IMAG(CX(1-INCX+1*INCX)))
REF: LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
*BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323
Routines called (none)

COMPLEX CX(1)

***FIRST EXECUTABLE STATEMENT ICAMAX
ICAMAX = 0
IF(N.LE.0) RETURN
ICAMAX = 1
IF(N .LE. 1) RETURN
NS = N*INCX
II = 1
SUMMAX = ABS(REAL(CX(1))) + ABS(IMAG(CX(1)))
DO 20 I=1,NS,INCX
SUMRI = ABS(REAL(CX(I))) + ABS(IMAG(CX(I)))
IF(SUMMAX.GE.SUMRI) GO TO 10
I-21
SUMMAX = SUMRI
ICAMAX = II
10 II = II + 1
20 CONTINUE
RETURN
END

C

COMPLEX FUNCTION zCDOTC(N,CX,INCX,CY,INCY)

C***BEGIN PROLOGUE zCDOTC
C***DATE WRITTEN 791001 (YYMMDD)
C***REVISION DATE 820801 (YYMMDD)
C***CATEGORY NO. D1A4
C***KEYWORDS BLAS,COMPLEX,INNER PRODUCT,LINEAR ALGEBRA,VECTOR
C***AUTHOR LAWSON, C. L., (JPL)
C HANSON, R. J., (SNLA)
C KINCAID, D. R., (U. OF TEXAS)
C KROGH, F. T., (JPL)
C***PURPOSE Dot product of complex vectors, uses complex
C conjugate of first vector
C***DESCRIPTION
C
B L A S Subprogram
Description of Parameters

---Input---
N number of elements in input vector(s)
CX complex vector with N elements
INX storage spacing between elements of CX
CY complex vector with N elements
INCY storage spacing between elements of CY

---Output---
zCDOTC complex result (zero if N .LE. 0)

Returns the dot product for complex CX and CY, uses
CONJUGATE(CX)
zCDOTC=SUM for I=0 to N-1 of CONJ(CX(LX+I*INCX))*CY(LY+I*INCY)
where LX = 1 if INX .GE. 0, else LX = (-INX)*N, and LY is
defined in a similar way using INCY.
C***REFERENCES
C LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
C *BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
C ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
C SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323
C***ROUTINES CALLED (NONE)
C***END PROLOGUE zCDOTC

COMPLEX CX(1),CY(1)

C***FIRST EXECUTABLE STATEMENT zCDOTC
zCDOTC = (0.,0.)
IF(N .LE. 0)RETURN
IF(INCX.EQ.INCY .AND. INCX.GT.0) GO TO 20
KX = 1
KY = 1
IF(INCX.LT.0) KX = 1+(1-N)*INCX
IF(INCY.LT.0) KY = 1+(1-N)*INCY
    DO 10 I = 1,N
    zCDOTC = zCDOTC + CONJG(CX(KX))*CY(KY)
    KX = KX + INCX
    KY = KY + INCY
    CONTINUE
    10 CONTINUE
RETURN
20 CONTINUE
NS = N*INCX
    DO 30 I=1,NS,INCX
    zCDOTC = CONJG(CX(I))*CY(I) + zCDOTC
    30 CONTINUE
RETURN
END
Appendix II. Source List of FORTRAN Program P_THRIFT.FOR

program p_thrify

This program solves the Laplace equation \( p \) in the area
\[ 0.0 \leq x \leq 3.1416, \ 0.0 \leq y \leq 3.1416. \]
using the thrifty band matrix solver
This method was first developed by Li, C. (1995), and later,
 improved and documented by Jerome P.-Y. Maa

The Gover Equation is \[ \frac{d^2 p}{dx^2} + \frac{d^2 p}{dy^2} = 0 \]
The finite difference equation is:
\[ p(i-1,j) + r*p(i,j-1) - 2*(1+r)*p(i,j) + r*p(i,j+1) + p(i+1,j) = 0 \]
The boundary conditions are:
p = 2 at \( x = 0 \)
p = 1 at \( y = 0 \)
p = 0 at \( x = y = \pi \)

INFILE and OUTFILE : to store input and main output data.
RECFILE is used to store routine checking information. If the
program run O.K., it can be deleted.
A MATLAB program PHIPLT.M has been developed to plot the
output results. It uses Matlab version 4.01 for windows, and
read output data from the OUTFILE directly.

Data in the file INFILE are generated by another program
ID.FOR for the examples given in this program.
In this example, there is no need to use complex variable.
But, it was used anyway for general applications.

Jerome P.-Y. Maa Virginia Institute of Marine Science.

parameter (iq=110, jq=110, iw=310, jw=560, kq=10000, Lq=3)

implicit complex (z)
character*45 title, label
character*20 infile, outfile, recfile
character*1 id, q1

common /matrix/ za(kq,5), ia(kq,5), zb(kq)
common /gener/ mp, np, dx, dy, r, mu, mL, m, Lda
common /iando/ p(iq,jq), id(iq,jq)
common /unknw/ n, imap(kq), jmap(kq), irow(iq)
common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)
common /works/ zw(iw,jw), ipvt(kq)

boundary conditions
bcx0=2
bcxn=0
bcy0=1
bcyn=0

print*,'There are three resolutions for this case:'
print*,' coarse, middle, and fine resolution'
print*,'Select 1. p_s.grd; 2. p_m.grd; 3. p_L.grd :
read(*,*) iption
go to (10, 12, 14), iption
10 infile='p_s.grd'
   outfile='p_ts.out'
   recfile='p_ts.rec'
go to 20
12 infile='p_m.grd'
   outfile='p_tm.out'
   recfile='p_tm.rec'
go to 20
14 infile='p_L.grd'
   outfile='p_tL.out'
   recfile='p_tL.rec'
20 continue

open(9, file=infile, status='old')
open(10, file=outfile, status='unknown')
open(11, file=recfile, status='unknown')

mp,np : Max. grid numbers in x and y direction, respectively.
dx,dy : grid sizes in x and y direction, respectively.

read(9,'(a45)') title
write(11,'(a45)') title
read(9,*) mp,np,dx,dy
if(mp .gt. iq) then
   print*,'IQ is smaller than MP, Change IQ to ',mp
   stop
end if
if(np .gt. jq) then
   print*,'JQ is smaller than NP, Change JQ to ',np
   stop
end if

r = (dx/dy)**2

ID code for each grid point
: 0, interior point
: 1, upper boundary condition
: 2, lower boundary condition
: 3, left boundary condition
: 4, right boundary condition
: 5, left bottom corner B.C.
c 6, left top corner B.C.
c 7, right bottom corner B.C.
c 8, right top corner B.C.
c e, grid point that is not included in the study domain

The following is an example

```
read(9,30) label
write(11,30) label
read(9,30) label
write(11,30) label
30 format(a50)
do j=1,np
   jj=np-j+1
   read(9,45) jns, (id(i,jj),i=1,mp)
45 format(i5,(110a1) )
   if(jns .ne. jj) then
      write(11,50) jj,jns
50    format(' Sequence is wrong for ID input at',2i5)
   end if
end do
print*, 'Completed reading ID code matrix'
close(9)
```

construct the unknown COLUMN matrix \( X \), in the full matrix eq. \( AX = B \). Find each unknown’s location: imap(map), jmap(map)

\( \text{ind}(i) \) : no. of isolated sector in each column, x grid.
for this particular case, \( \text{ind}(i) \) are all 1 because no land points in the middle of study domain.

\( \text{jin}(i,index) \): begin grid number for an isolated sector in a column
\( \text{jout}(i,index) \): end grid number for an isolated sector in a column

II-3
map : the number of total unknown, or the length of X.
later, it is reassigned as N
irow(i) : the total number of unknown, P, in each column

map=0
do i=1,mp
   icount=0
   in=0
   iout=1
   index=1
   do j=1,np
      if (id(i,j) .eq. '0') then
         icount=icount+1
         map=map+1
         imap(map)=i
         jmap(map)=j
         if (in .eq. 0) then
            jin(i,index)=j
            in=1
            iout=0
         end if
      end if
   end do
   irow(i)=icount
   ind(i)=index-1
ind(i) should be >= 1, except for the entire column are all
boundary points. if not, something wrong.

write(11,55) i, ind(i), irow(i), jin(i,1), jout(i,1)
55 format(’ i,ind,irow,jin,jout=’,Si5,i5)
if (ind(i) .gt. Lq) then
   write(*,60) i, ind(i), Lq, ind(i)
60 format(’ At i=’,i4,’ Ind(i)=’,i2,’ > Lq=’,i2/
   ’ Change Lq to’,i3,’ and re-run’)
   stop
end if
end do

n: length of the banded matrix

n=map
if(n .lt. 0.8*kq) then
   write(*,65) n, kq
It is better to reduce KQ, length of the Banded Matrix, so that KQ is not >> than that required. Instead, one should increase the length of the working matrix, JW, in order to reduce disk I/O and computing time. You may continue (c), or re-run (r), < c/r > : 

read(*, '(a1)') q1
if( q1 .eq. 'r' .or. q1 .eq. 'R') stop

if(n .gt. kq) then
write(*,70) n, kq, n
format('----------------------------------------------'/
* N',i5,' > KQ=',i5//' Increases KQ to ',i5,' and re-run')
stop

end if

Set up the two small matrices ZA and IA for storing the banded matrix and find out the band width, work size, etc.

mu=0
 mL=0
write(11,80)

format(' i j ia(1) .. .. ia(4),ia(5) '
* ' za(1) .. za(5) zb')
do map=1,n
i=imap(map)
j=jmap(map)
call domain(i, j, map, ierr)
if( ierr .eq. 0) then
write(11,90)i,j,ia(map,1),ia(map,2),ia(map,3),ia(map,4),
* ia(map,5), real(za(map,1)),real(za(map,2)),
* real(za(map,3)), real(za(map,4)), real(za(map,5)),
* real zb(map)
format(1x,2i5,2x,5i5,2x,6f6.2)
else
print*, 'stop, error in DOMAIN at i,j,map='', i,j,map stop
end if

find the band width

if( ia(map,5) .ne. 0 ) then
 mu_c=abs(ia(map,5) - ia(map,3) )
else
 mu_c=0
end if
if(mu .lt. mu_c) mu=mu_c
if( ia(map,1) .ne. 0 ) then
 mL_c=abs(ia(map,1) - ia(map,3) )
else
    mL_c=0
end if
if(mL .lt. mL_c) mL=mL_c
end do

m=mL+mu+1
Lda=m+mL
write(*,100) m, n, mu, mL, Lda
100 format(' Band width, m = ',i7/
    * ' Upper B.W., mu = ',i7/
    * ' Lower B.W., mL = ',i7/
    * ' Lda for Zw = ',i7)
if(Lda .gt. iw) then
    print*,'Please increase IW to ',Lda
    stop
end if

c uses the complex thrifty band matrix solver to solve ZAF*ZX= ZB
c
call bmsolver(ier_code)
c
if(ier_code .eq. 0) then
    do map=1, n
        i=imap(map)
        j=jmap(map)
        p(i,j)=real(zb(map))
    end do
    do j=1,np
        p(1,j) = bcx0
        p(mp,j) = bcxn
    end do
    do i=1,mp
        p(i,1) = bcy0
        p(i,np) = bcyn
    end do
    p(1,1) = 0.5*(bcy0+bcx0)
    p(1,np) = 0.5*(bcyn+bcx0)
    p(mp,1) = 0.5*(bcy0+bcxn)
    p(mp,np) = 0.5*(bcyn+bcxn)
    write(10,'(a45)') title
    write(10,'(2i5,2f12.8)') mp,np,dx,dy
    do j=1,np
        write(10,120) j, (p(i,j),i=1,mp)
120    format(i5/(10f8.4))
    end do
end if

close(10)
close(11)
print*, 'program stop'
print*, 'The solution is in the file ', outfile
print*, 'and other details, if needed, is in ', recfile
stop
end

*****************************************************************

subroutine BMsolver(ierr_code)

It solves a complex banded matrix equation ZAF * ZX = ZB
where ZAF is a complex band matrix with dimension (m * n)
ZX and ZB are two complex column matrices with length (n).

Because of the huge size of ZAF, e.g., (300 x 50000), it is
designed to solve this problem using the following two steps.

First, don't use the full size of ZAF, instead, uses two small
matrices: ZA & IA that each only uses 50000 x 5 to save space.

ZA : a complex matrix (n * 5) to store the coefficient matrix in
a matrix equation ZAF*ZX= ZB. Because of using the finite
difference method to solve an elliptic equation, there are
only 5 elements to be saved for the coefficient matrix.
The band width, however, is much much large than 5 with a
lot of "zero." By doing so, we need another matrix
IA : to save the corresponding locations.

Second, uses a working matrix, ZW(IW,JW), and do a systematical
swap between a hard disk and memory. For this reason, be sure
that you do have enough space in your hard disk. For example, a
complex matrix with size of (300 x 50000) requires 120 MB for
storage, if using 4 byte for a real number. If using 8 bytes,
then, 240 MB is needed.

IW,JW: IW should be >= m+ml, where ml is the lower band width.
The size of JW depends on the available computer memory.
In general, the large the JW, the less the disk IO, and
thus, the faster the computing speed. As a rule of
thumb, you may select JW = 2*IW and tried to see if your
computer has enough memory to run the program.

The procedures follows that given in the subroutine CGBFA &
CGBSL from LINPACK. The major difference is just doing it one
block at a time, stores the results in hard disk sequently.
After forward elimination, reverse the process by reading the
data from hard disk and do back substitute for the solution.

parameter (iq=110, jq=110, iw=310, jw=560, kq=10000)

II-7
Gaussian elimination with partial pivoting

NK: # of columns for the working matrix that will be saved.
NW: The number of column needed for working, nw=m-1+nk
NS: An index to show the number of working matrix.

check working matrix dimension

nk=350
nw=m+nk-1

if(nw .gt. jw) then
write(*,10) iw, jw, Lda, nw
10  format(' The working matrix, ZW(iw,jw), iw,jw=', 2i5/
*     ' IW shoult be >=',i5, ' and JW must be >=',i5/
*     ' Please change program and re-run .'/
*     ' If no memory, you can reduce the NK')
ierr_code=1
return
end if

clear up the working matrix

do i=1,iw
   do j=1,nw
      zw(i,j)=(0.0, 0.0)
   end do
end do

construct the first working matrix, ZW, which has a size of
(Lda x nw). The thrify storaged matrices are expanded, only
to the first (nk x 5) block.

The flag is used to recording only once when the ZA data
starts lost.

ns=0
map1=0
iflag=1
ne=nw
if(ne .gt. n) ne=n
do map=1,ne
  do i=1,5
    j=ia(map,i)
    if(j .ne. 0) then
      if(j .le. nk+m-1) then
        ids=j-map
        zw(m-ids,j)=za(map,i)
      else
        if(iflag .eq. 1) then
          map1=map
          iflag=0
          end if
        end if
      end if
    end do
  end do
c
Gaussian elimination with partial pivoting
c
   j1=min0(n,m)-1
   jz=j1
   ju=0
   nt=0
   index=0
100 continue
c
   if(ju .ne. 0) ju=ju-nk
   if(jz .ne. j1) jz=jz-nk

c index=0, the first, 2nd, ..., block of matrix.
c =1, the last block of matrix
c
   if(index .eq. 0 .and. ne .ne. n) then
     nc=nk
   else
     nc=n-ns*nk-1
   end if
c
   do k=1,nc
     kp1=k+1
     kr=ns*nk + k
   c
   find L = pivot index
   Lm=min0(mL,n-ns*nk-k)
   L=icamax(Lm+1, zw(m,k), 1) + m -1
   ipvt(kr)=L+kr-m
   II-9
zero pivot implies that this column are all zeros, a singular matrix.

if(abs(zw(L,k)) .lt. 0.10e-8) goto 120

interchange if necessary

if(L .ne. m) then
    zt=zw(L,k)
    zw(L,k)=zw(m,k)
    zw(m,k)=zt
end if

compute multipliers

zt=-(1.0e0,0.0e0)/zw(m,k)
call cscal(Lm, zt, zw(m+1,k), 1)

swap ZB array, if necessary

Lp=ipvt(kr)
zt=zb(Lp)
if(Lp .ne. kr) then
    zb(Lp)=zb(kr)
    zb(kr)=zt
end if
call caxpy(Lm, zt, zw(m+1,k), 1, zb(kr+1), 1)

row elimination with column indexing

ju=min0(max0(ju, mu+ipvt(kr)-ns*nk), n-ns*nk)
mm=m
if(ju .ge. kp1) then
    do j=kp1,ju
        L=L-1
        mm=mm-1
        zt=zw(L,j)
        if(L .ne. mm) then
            zw(L,j)=zw(mm,j)
            zw(mm,j)=zt
        end if
        call caxpy(Lm, zt, zw(m+1,k), 1, zw(mm+1,j), 1)
    end do
end if

goto 150
120 continue
print*, 'Zero diagonal element at (L,k)=', L, k
stop
150 continue

c end do
Except the last working matrix, write the upper triangular matrix (from j=1,nk) into hard disk. For the last one, i.e., nc<>nk, go to back substitute directly.

```
if(nc .eq. nk) then
  nt=nt+1
  print*, 'Writing tem. file #', nt
  tmpfile='t'//chrc(nt)//'.tmp'
  open(12,file=tmpfile,form='unformatted')
  write(12) ((zw(i,j), i=1,m), j=1,nk)
  close(12)
```

Moving the rest working matrix forward to the beginning

```
do j=nk+1, nk+m-1
  L=j-nk
  do i=1,Lda
    zw(i,L)=zw(i,j)
  end do
end do
```

Clear the rest working area for reading new working matrix

```
do j=m, nk+m-1
  do i=1,Lda
    zw(i,j)=(0.0,0.0)
  end do
end do
```

Read in a full block of ZA and IA, by two steps

```
ns=ns+1
if( (ns+1)*nk+m-1 .lt. n) then
  ne=nk
  index=0
```

For intermediate blocks, read in ZA and IA by two steps.
First read in the upper triangular matrix that were cut off at the previous time when constructing the working matrix.

```
if(map1 .ne. 0) then
  do map=map1, m-1+ns*nk
    do i=4,5
      j=ia(map,i)
      if(j .ne. 0) then
        if(j .gt. m-1+ns*nk) then
          ids=j-map
          zw(m-ids,j-ns*nk)=za(map,i)
        end if
      end if
    end do
  end do
```

II-11
end do
end if

now read in next block of ZA and IA

iflag=1
map1=0
do k=1,ne
  map=k + ns*nk + m-1
do i=1,5
    j=ia(map,i)
    if(j .ne. 0) then
      if(j .le. ns*nk+ne+m-1) then
        ids=j-map
        zw(m-ids,j-ns*nk)=za(map,i)
      else
        if(iflag .eq. 1) then
          map1=map
          iflag=0
        end if
      end if
    end if
  end do
end do

End of read in a block of ZA and IA.

else

This is to read the last block of ZA and IA

ne=n-(m-1+ns*nk)
index=1
if(map1 .ne. 0) then
  do map=map1,m-1+ns*nk
    do i=4,5
      j=ia(map,i)
      if(j .ne. 0) then
        if(j .gt. m-1+ns*nk) then
          ids=j-map
          zw(m-ids,j-ns*nk)=za(map,i)
        end if
        end if
    end do
  end do
end if

now reads the last block of ZA and IA

iflag=1
map1=0
do k=1,ne

II-12
map=m-1+ns*nk+k
do i=1,5
    j=ia(map,i)
    if(j .ne. 0) then
        if(j .le. m-1+ns*nk+ne) then
            ids=j-map
            zw(m-ids,j-ns*nk)=za(map,i)
        else
            ierr_code=2
            print*, 'If this happen, it is wrong.'
            print*, 'At the last block, no overflow'
            return
        end if
    end do
end do
iflag=1
end if

goto 100
else
   backward substitution from the last submatrix
   if( nt .eq. 0 ) nc=n-1
   do kb=1,nc+1
      kr=n+1-kb
      k=nc+2-kb
      zb(kr)=zb(kr)/zw(m,k)
      Lm=min0(kr,m)-1
      La=m-Lm
      Lb=kr-Lm
      zt=-zb(kr)
      call caxpy(Lm, zt, zw(La,k), 1, zb(Lb), 1)
   end do
   If one loop can include all elements, i.e., for a small banded
   matrix, just stops after this.
   if( nt .eq. 0 ) go to 400
end if
   complete the rest backward substitute
   ns=0
200 continue
   clear the working matrix for reading new submatrix from disk.
   do j=1,nk+m-1
      do i=1,m
         zw(i,j)=(0.0,0.0)
   end do
end do
end do

print*, 'Reading term. file #', nt
open(12, file=tmpfile, form='unformatted')
read(12, ((zw(i,j), i=1,m), j=1,nk)
close(12, status='delete')

do kb=1,nk
   kr=n+1-(nc+1)-ns*nk-kb
   k=nk+1-kb
   zb(kr)=zb(kr)/zw(m,k)
   Lm=min0(kr,m)-1
   La=m-Lm
   Lb=kr-Lm
   zt=-zb(kr)
   call caxpy(Lm, zt, zw(La,k), 1, zb(Lb), 1)
end do
ns=ns+1
nt=nt-1
tmpfile='t'//chrc(nt)//!.tmp'
if(nt .gt. 0) goto 200

400 continue

To restore the original sequence of ZB. Since it starts at the 2nd. we started at 2nd too for restoring.

print*, 'Restore the original sequence'
do 1000 kb=2,n
   k=n+1-kb
   L=ipvt(k)
   zt=zb(L)
   if( L .eq. k ) go to 1000
   print*, 'pivoting at L,k,kb=',L,k,kb
   zb(L)=zb(k)
   zb(k)=zt
1000 continue

return
end

character*3 function chrc(nt)

It changes an input integer number NT to character
CHANGQING LI 06/94

c character*1 c1
c character*2 c2
c character*3 c3

II-14
il=nt
i2=i1/10
if(i2 .gt. 0) then
  i3=i2/10
  if(i3 .gt. 0) then
    i4=i3/10
    if(i4 .gt. 0) then
      print*, "----------------------------------------"
      print*, "The given integer is > 999, not allowed."
      print*, "----------------------------------------"
      chrc='"-1''
    else
      c3=char(48+i3)//char(48+i2-i3*10)//char(48+i1-i2*10)
      chrc=c3
    end if
  else
    c2=char(48+i2)//char(48+i1-i2*10)
    chrc=c2
  end if
else
  c1=char(48+i1)
  chrc=c1
end if

return
end

SUBROUTINE CSCAL(N,CA,CX,INCX)

PURPOSE: Complex vector scale x = a*x

WRITTEN on Oct. 1, 79, REVISION on Aug. 01, 82

CATEGORY NO. D1A6

KEYWORDS BLAS, COMPLEX, LINEAR ALGEBRA, SCALE, VECTOR

AUTHORS LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
  KINCAID, D. R., (U. OF TEXAS), and KROGH, F. T., (JPL)

DESCRIPTION

BLAS Subprogram

Description of Parameters

--Input--
N number of elements in input vector(s)
CA complex scale factor
CX complex vector with N elements
INCX storage spacing between elements of CX

--Output--
CSCAL complex result (unchanged if N .LE. 0)

replace complex CX by complex CA*CX.

II-15
For I = 0 to N-1, replace CX(1+I*INCX) with CA * CX(1+I*INCX)

REF: LAWSON C. L., HANSON R. J., KINCAID D. R., KROGH F. T., "BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE,
ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323

ROUTINES CALLED (NONE)

COMPLEX CA,CX(1)
FIRST EXECUTABLE STATEMENT CSCAL
IF(N .LE. 0) RETURN
NS = N*INCX
DO I = 1,NS,INCX
   CX(I) = CA*CX(I)
end do
RETURN
END

SUBROUTINE CAXPY(N,CA,CX,INCX,CY,INCY)

WRITTEN on Oct. 01, 79, REVISION on April 25, 84
CATEGORY NO. D1A7
KEYWORDS BLAS,COMPLEX,LINEAR ALGEBRA,TRIAD,VECTOR
AUTHOR LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
PURPOSE Complex computation y = a*x + y

DESCRIPTION B L A S Subprogram
Description of Parameters

--Input--
N number of elements in input vector(s)
CA complex scalar multiplier
CX complex vector with N elements
INCX storage spacing between elements of CX
CY complex vector with N elements
INCY storage spacing between elements of CY

--Output--
CY complex result (unchanged if N .LE. 0)

Overwrite complex CY with complex CA*CX + CY.
For I = 0 to N-1, replace
CY(LY+I*INCY) with CA*CX(LX+I*INCX) + CY(LY+I*INCY),
where
LX = 1 if INCX .GE. 0, else LX = (-INCX)*N
and LY is defined in a similar way using INCY.

REFERENCES
LAWSON C. L., HANSON R. J., KINCAID D. R., KROGH F. T.,
*BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
COMPLEX CX(1), CY(1), CA

FIRST EXECUTABLE STATEMENT CAXPY
CANORM = ABS(REAL(CA)) + ABS(AIMAG(CA))
IF(N.LE.0.OR.CANORM.EQ.0.E0) RETURN
IF(INCX.EQ.INCY.AND.INCX.GT.0) GO TO 20
KX = 1
KY = 1
IF(INCX.LT.0) KX = 1+(1-N)*INCX
IF(INCY.LT.0) KY = 1+(1-N)*INCY
DO 10 I = 1, N
CY(KY) = CY(KY) + CA*CX(KX)
KX = KX + INCX
KY = KY + INCY
10 CONTINUE
RETURN
20 CONTINUE
NS = N*INCX
DO 30 I=1,NS,INCX
CY(I) = CA*CX(I) + CY(I)
30 CONTINUE
RETURN
END

INTEGER FUNCTION ICAMAX(N,CX,INCX)

PURPOSE Find the location (or index) of the largest component of a complex vector

DESCRIPTION

B L A S Subprogram

--Input--
  N  number of elements in input vector(s)
  CX  complex vector with N elements
  INCX  storage spacing between elements of CX

--Output--
  ICAMAX  smallest index (zero if N .LE. 0)

  Returns the index of the component of CX having the largest sum of magnitudes of real and imaginary parts.

II-17
ICAMAX = first I, I = 1 to N, to minimize
ABS(REAL(CX(1-INCX+I*INCX))) + ABS(IMAG(CX(1-INCX+I*INCX)))

REFERENCES

LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
* BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323

ROUTINES CALLED (NONE)

COMPLEX CX(1)

C***FIRST EXECUTABLE STATEMENT ICAMAX
ICAMAX = 0
IF(N.LE.0) RETURN
ICAMAX = 1
IF(N .LE. 1) RETURN
NS = N*INCX
II = 1
SUMMAX = ABS(REAL(CX(1))) + ABS(AIMAG(CX(1)))
DO 20 I=1,NS,INCX
SUMRI = ABS(REAL(CX(I))) + ABS(AIMAG(CX(I)))
IF(SUMMAX.GE.SUMRI) GO TO 10
SUMMAX = SUMRI
ICAMAX = II
10 II = II + 1
20 CONTINUE
RETURN
END

-----------------------------------------------

subroutine domain(i, j, map, ier)

parameter (iq=110, jq=110, kq=10000, Lq=3)
implicit complex (z)
character*1 id
common /matrx/ za(kq,5), ia(kq,5), zb(kq)
common /gener/ mp, np, dx, dy, r, mu, mL, m, Lda
common /iando/ p(iq,jq), id(iq,jq)
common /unknw/ n, imap(kq), jmap(kq), irow(iq)
common /labbl/ ind(iq), jin(iq,Lq), jout(iq,Lq)

dimension iid(9)

The finite difference governing equation, the Laplace equation,
with \( r = (dx/dy)^2 \), is

\[ p(i-1,j) + r*p(i,j-1) -(2+2r)*p(i,j) +r*p(i,j+1) +p(i+1,j) = 0 \]

In storage, the coefficient of \( p(i-1,j) \) is stored in \( (map,1) \),
the coefficient of \( p(i,j-1) \) is stored in \( (map,2) \),
the coefficient of \( p(i,j) \) is stored in \( (map,3) \),
the coefficient of \( p(i, j+1) \) is stored in \( (\text{map},4) \),
and the coefficient of \( p(i+1, j) \) is stored in \( (\text{map},5) \)

\[
\text{ien}=0
\]
\[
\text{do } k=1,9
\]
\[
\text{id}(k)=0
\]
\[
\text{end do}
\]
\[
\text{icount}=0
\]

For those grid point that are not neighbored to a boundary

\[
\text{if } (\text{id}(i-1,j) \ \text{.eq.} \ '0' \ \text{.and.} \ \text{id}(i+1,j) \ \text{.eq.} \ '0' \ \text{.and.} \ \text{id}(i,j-1) \ \text{.eq.} \ '0' \ \text{.and.} \ \text{id}(i,j+1) \ \text{.eq.} \ '0') \text{ then}
\]
\[
z(\text{map},1)=1.0
\]
\[
z(\text{map},2)=r
\]
\[
z(\text{map},3)=-2.0*(1.0+r)
\]
\[
z(\text{map},4)=r
\]
\[
z(\text{map},5)=1.0
\]
\[
\text{ia}(\text{map},1)=\text{map}-\text{idistl}(i,j)+1
\]
\[
\text{ia}(\text{map},2)=\text{map}-1
\]
\[
\text{ia}(\text{map},3)=\text{map}
\]
\[
\text{ia}(\text{map},4)=\text{map}+1
\]
\[
\text{ia}(\text{map},5)=\text{map}+\text{idistr}(i,j)-1
\]
\[
\text{zb}(\text{map})=(-2.0, 0.0)
\]
\[
\text{icount}=\text{icount}+1
\]
\[
\text{id}(1)=1
\]
\[
\text{end if}
\]

for those has a Boundary grind point on the left

\[
\text{if } (\text{id}(i-1,j) \ \text{.eq.} \ '3' \ \text{.and.} \ \text{id}(i,j-1) \ \text{.eq.} \ '0' \ \text{.and.} \ \text{id}(i,j+1) \ \text{.eq.} \ '0') \text{ then}
\]
\[
z(\text{map},1)=0.0
\]
\[
z(\text{map},2)=r
\]
\[
z(\text{map},3)=-2.0*(1.0+r)
\]
\[
z(\text{map},4)=r
\]
\[
z(\text{map},5)=1.0
\]
\[
\text{ia}(\text{map},1)=0
\]
\[
\text{ia}(\text{map},2)=\text{map}-1
\]
\[
\text{ia}(\text{map},3)=\text{map}
\]
\[
\text{ia}(\text{map},4)=\text{map}+1
\]
\[
\text{ia}(\text{map},5)=\text{map}+\text{idistr}(i,j)-1
\]
\[
\text{zb}(\text{map})=(-2.0, 0.0)
\]
\[
\text{icount}=\text{icount}+1
\]
\[
\text{id}(2)=1
\]
\[
\text{end if}
\]

for those has a Boundary grind point on the right

\[
\text{if } (\text{id}(i+1,j) \ \text{.eq.} \ '4' \ \text{.and.} \ \text{id}(i,j-1) \ \text{.eq.} \ '0' \ \text{.and.} \ \text{id}(i,j+1) \ \text{.eq.} \ '0') \text{ then}
\]
\[
z(\text{map},1)=1.0
\]

II-19
za(map,2) = r
za(map,3) = -2.0*(1.0+r)
za(map,4) = r
za(map,5) = 0.0
ia(map,1) = map - dist(i,j) + 1
ia(map,2) = map - 1
ia(map,3) = map
ia(map,4) = map + 1
ia(map,5) = 0
zb(map) = (0.0, 0.0)
icount = icount + 1
iid(3) = 1
end if

for those has a Boundary grind point on the top

if (id(i, j+1) .eq. '1' .and. id(i-1, j) .eq. '0' .and. id(i+1, j) .eq. '0') then
za(map,1) = 1.0
za(map,2) = r
za(map,3) = -2.0*(1.0+r)
za(map,4) = 0
za(map,5) = 1.0
ia(map,1) = map - dist(i, j) + 1
ia(map,2) = map - 1
ia(map,3) = map
ia(map,4) = 0
ia(map,5) = map + dist(i, j) - 1
zb(map) = (0.0, 0.0)
icount = icount + 1
iid(4) = 1
end if

for these grid points that has a Boundary at bottom

if (id(i, j-1) .eq. '2' .and. id(i-1, j) .eq. '0' .and. id(i+1, j) .eq. '0') then
za(map,1) = 1.0
za(map,2) = 0
za(map,3) = -2.0*(1.0+r)
za(map,4) = r
za(map,5) = 1.0
ia(map,1) = map - dist(i, j) + 1
ia(map,2) = 0
ia(map,3) = map
ia(map,4) = map + 1
ia(map,5) = map + dist(i, j) - 1
zb(map) = (-1.0, 0.0)
icount = icount + 1
iid(5) = 1
end if
c For those grid points that are left bottom corner
if(id(i-1,j) .eq. '3' .and. id(i,j-1) .eq. '2') then
  za(map,1)=0.0
  za(map,2)=0
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=r
  za(map,5)=1.0
  ia(map,1)=0
  ia(map,2)=0
  ia(map,3)=map
  ia(map,4)=map+1
  ia(map,5)=map+idistr(i,j)-1
  zb(map)=(-3.0, 0.0)
  icount=icount+1
  iid(6)=1
end if

c For those grid points that are left top corner Boundary points
if(id(i-1,j) .eq. '3' .and. id(i,j+1) .eq. '1') then
  za(map,1)=0.0
  za(map,2)=r
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=0.0
  za(map,5)=1.0
  ia(map,1)=0
  ia(map,2)=map-1
  ia(map,3)=map
  ia(map,4)=0
  ia(map,5)=map+idistr(i,j)-1
  zb(map)=(-2.0, 0.0)
  icount=icount+1
  iid(7)=1
end if

c For those grid points that are right bottom boundary points
if(id(i+1,j) .eq. '4' .and. id(i,j-1) .eq. '2') then
  za(map,1)=1.0
  za(map,2)=0
  za(map,3)=-2.0*(1.0+r)
  za(map,4)=r
  za(map,5)=0.0
  ia(map,1)=map-idistl(i,j)+1
  ia(map,2)=0
  ia(map,3)=map
  ia(map,4)=map+1
  ia(map,5)=0
  zb(map)=(-1.0, 0.0)
  icount=icount+1
  iid(8)=1
end if
end if

For those grid points that are right top boundary points

if (id(i+1,j) .eq. '4' .and. id(i,j+1) .eq. '1') then
za(map,1)=1.0
za(map,2)=r
za(map,3)=-2.0*(1.0+r)
za(map,4)=0
za(map,5)=0.0
ia(map,1)=map-idistl(i,j)+1
ia(map,2)=map-1
ia(map,3)=map
ia(map,4)=0
ia(map,5)=0
zb(map)=(0.0, 0.0)
icount=icount+1
iid(9)=1
end if

if( icount .ne. 1) then
ier=1
write(11,10) icount, (iid(k),k=1,9)
10 format(' icount=',i5,' id(1)-id(9)=' ,9i2/
* ' Only one of the ids can be 1, check ID array')
end if
return
end

integer function idistr(i,j)
parameter (iq=110, Lq=3)
common /labbl/ ind(iq),jin(iq,Lq),jout(iq,Lq)

Calculates the distance between points (i+1,j) and (i,j). It means how many grid points, which are looking for, are in between. It counts vertically from the point (i,j) and up (i,j+1), (i,j+2), ..., (i,np), then (i+1,1), (i+1,2), ..., to (i+1,j). In the matrix equation, this distance represents the up band width at that particular diagonal element (i,j)

do k=1,ind(i)
   if(j .ge. jin(i,k) .and. j .le. jout(i,k) ) mark1=k
end do

do k=1,ind(i+1)
   if(j .ge. jin(i+1,k) .and. j .le. jout(i+1,k) ) mark2=k
end do

idd=jout(i,mark1)-j+1 + j-jin(i+1,mark2)

II-22
do k=mark1+1, ind(i)
    idd=idd + jout(i,k)-jin(i,k)+1
end do

d do k=1,mark2-1
    idd=idd + jout(i+1,k)-jin(i+1,k)+1
end do

idistr=idd

return
end

integer function idistl(i,j)

parameter (iq=110, Lq=3)
common /labbl/ ind(iq),jin(iq,Lq),jout(iq,Lq)

Calculate the distance between point (i-1,j) and (i,j). The
distance means how many grid points, which are solved for, are
in between. It counts vertically from (i,j) downward (i,j-1),
(i,j-2), ..., (i,1) and restarted from previous i line
(i-1,np), (i-1,np-1), ..., to (i-1,j).

do k=1,ind(i-1)
    if(j .ge. jin(i-1,k) .and. j .le. jout(i-1,k)) mark1=k
end do

do k=1,ind(i)
    if(j .ge. jin(i,k) .and. j .le. jout(i,k)) mark2=k
end do

idd=jout(i-1,mark1)-j+1 + j-jin(i,mark2)

do k=mark1+1,ind(i-1)
    idd=idd+jout(i-1,k)-jin(i-1,k)+1
end do

do k=1,mark2-1
    idd=idd+jout(i,k)-jin(i,k)+1
end do

idistl=idd
return
end
Appendix III. Source List of FORTRAN Program ID.FOR

program id_matrix

This program generates the input ID matrix, I.D. codes for each cell

parameter (iq=400, jq=200, kq=500, Lq=2000)
character*1 id
character*12 outfile
character*30 header
character*80 title

common/const/ mp, np, dx, dy, r
common/array/ id(iq,jq)
common/bound/ nobc, ibc(Lq), jbc(Lq), zp(Lq)

mp : mesh number in x direction
np : mesh number in y direction
dx : spatial step in x-direction
dy : spatial step in y-direction
r  : square of dx/dy

ID_code for water depth
: 0, interior cell
: 1, upper boundary condition
: 2, lower boundary condition
: 3, left boundary condition
: 4, right boundary condition
: 5, left bottom corner B.C.
: 6, left top corner B.C.
: 7, right bottom corner B.C.
: 8, right top corner B.C.
: e, land point

title=' Laplace equation with specified Dirichlet B.C.'
print*, 'Select 1. large grid; 2. middle grid; '
print*, ' 3. small grid :
read(*,*) iption
go to (10, 12, 14) iption
10 mp=101
   np=101
   outfile='p_L.grd'
go to 20
12 mp=62
   np=62
   outfile='p_m.grd'
go to 20
14 mp=12
   np=12
   outfile='p_s.grd'
20 continue
dx=3.1415926/(mp-1)
dy=3.1415926/(np-1)

velocity potential

if(mp .ge. iq .or. np .ge. jq) write(*,30) mp,np, iq,jq
format(' Given array (mp,np)=',2i5,' is > than allocated',
* (iq,jq)=',2i5)
open(8,file=outfile,form='formatted')
write(8,'(a80)') title
write(8,40) mp, np, dx, dy
format(2i5, 2f12.8)
nobc=0

first establish the ID code for the entire grid point

do j=1, np
  do i=1, mp
    id(i,j)=0'      
  end do
end do

given boundary at the right hand side

do j=2,np-1
  nobc=nobc+1
  id(mp,j)='4'
  ibc(nobc)=mp
  jbc(nobc)=j
  zp(nobc)=0
end do

left side B.C.

do j=2,np-1
  id(1,j)='3'
  nobc=nobc+1
  ibc(nobc)=1
  jbc(nobc)=j
  zp(nobc)=2.0
end do

top B.C.

do i=2,mp-1
  id(i,np)='1'
  nobc=nobc+1
  ibc(nobc)=i
  jbc(nobc)=np

III-2
zp(nobc)=0.0
end do

c bottom B.C.
c
do i=2,mp-1
  id(i,1)=’2’
  nobc=nobc+1
  ibc(nobc)=i
  jbc(nobc)=1
  zp(nobc)=1.0
end do

c check the array assignment
c
if(nobc .gt. Lq) then
  write(*,SO) nobc, Lq, nobc
  format(’ The # of Radiation B.C.=’,i6,’ > Lq=’,i6/
      *   ’ Change the statement to Lq=’,i6,’ and re-run ’)
  stop
end if

c other minor modification
c
id(mp,np)=’e’
id(1,1)=’e’
id(1,np)=’e’
id(mp,1)=’e’

c header=’ I.D. code for each grid point’
write(8,’(a30)’) header
write(8,60) np
format(’ j i = 1, ... ,’,i5)
do j=1,np
  jj=np-j+1
  write(8,70) jj, ( id(i,jj), i=1,mp)
format(i4,1x,130a1)
end do

c header=’Given B.C. at following points’
write(8,’(i5,2x,a30)’) nobc, header
write(8,80)
format(’ n i j B.V.’)
do i=1,nobc
  write(8,90) i,ibc(i),jbc(i),zp(i)
end do
format(3i5,2f12.6)
c
close(8)
stop
end
Appendix IV. Source List of MATLAB program, P_PLOT.M

% Program phi_plt.M
% MathLab M file for plotting the output P matrix generated
% It uses MathLab ver. 4.2C.1 for WINDOWS
%
% Program developed by Jerome P.-Y. Maa at
% Virginia Institute of Marine Science, Feb. 1996
%
% clear the command window first.
%
clc;
disp('Available files for reading are : '); disp(' 1. p bs.out; 2. p_bm.out 3. p_bL.out'); disp(' 4 p_ts.out; 5. p_tm.out 6. p_tL.out'); disp(' 0. exit '); option=input('Select a number : ');
% select an input file. For other users, please change the
% fopen commend to have a proper path specification.
%
while option
    if option == 1
        [fid, message]=fopen('d:\break\p_bs.out');
    end
    if option == 2
        [fid, message]=fopen('d:\break\p_bm.out');
    end
    if option == 3
        [fid, message]=fopen('d:\break\p_bL.out');
    end
    if option == 4
        [fid, message]=fopen('d:\break\p_ts.out');
    end
    if option == 5
        [fid, message]=fopen('d:\break\p_tm.out');
    end
    if option == 6
        [fid, message]=fopen('d:\break\p_tL.out');
    end

% contour values for water depth
% v_dep=[0 0.1 0.2 0.4 0.8 1.0 1.2 1.4 1.6 1.8 2];
% read data
   ftitle=fgetl(fid);
disp(' '); disp(''); disp(ftitle);
mp=fscanf(fid, '%d', 1);
np=fscanf(fid, '%d', 1);

IV-1
disp(['m=',int2str(mp),', n=',int2str(np)]);
dx=fscanf(fid,'%f',1);
dy=fscanf(fid,'%f',1);
disp(['dx=',num2str(dx),', dy=',num2str(dy)]);

read solution

p=[];
for ii=1:np
    nk=fscanf(fid,'%d',1);
    if nk~=ii
        disp(['Seq. error, i=',int2str(nk),'<> ii=',int2str(ii)]);
        disp('paused');
        pause;
    end
    a=fscanf(fid,'%f',mp);
    p(ii,1:mp)=a';
end
clear a;
fclose(fid);

xmin=0;
xmax=3.1415926;
ymin=0;
ymax=xmax;

Plot the contours

clf;
axes('position',[0.1 0.1 0.88 0.88]);
[mmm nnn]=size(p);
dxx=(xmax-xmin)/(nnn-1);
dyy=(ymax-ymin)/(mmm-1);
x=xmin:dxx:xmax;
y=ymin:dyy:ymax;
axis_ratio=(xmax-xmin)/(ymax-ymin);
data_ratio=1;
cl=contour(x, y, p, v_dep);
set(gca,'aspectratio', [axis_ratio, data_ratio]);
set(gca,'xlim', [xmin,xmax], 'ylim', [ymin,ymax]);
hold on;
clabel(cl, 'manual');
xlabel('X'); ylabel('Y');
title('Band Matrix Solution');

disp('Available files for reading are : ');
disp(' 1. p bs.out; 2. p_bm.out; 3. p_bL.out;');
disp(' 4. p_ts.out; 5. p_tm.out; 6. p_tL.out;');
disp(' 0. exit ');
option=input('Select a number : ');
Appendix V. Source listing of P_EXACT.M

% This program calculate the exact solution of a Laplace equation 
% with the following boundary conditions: 
% at x=0,  p = 1; at x=pi, p = 0 
% at y=0,  p = 2; at y=pi, p = 0 
% The analytical solution is provided by Jerome P.-Y. Maa. 
% the results are stored in an ASCII file for later plot with the 
% solution using a banded matrix solver, and a Thrifty banded 
% matrix solver. It also plots the results at end. 
% Because the large value of cosh(x), tanh(x), etc., the solution 
% given in the text was manipulated to the form given in this 
% program.
% One may change the statement of no=xxxx to another number to 
% see the difference.

clear; 

no=4000; 

dx=0.031415926; 

dy=dx; 
m=(pi/dx)+1; 

y=0:dy:pi; x=y; y=y'; 
z=zeros(m,m); 

for n=1:no 

  ff=1-2*(-1)^n; 
  aa=exp(-2*n*pi); 
  bb=exp(-n*x); 
  cc=exp(n*(x-2*pi)); 
  dd=exp(n*(x-pi)); 
  ee=exp(-n*(x+pi)); 
  gg=sin(n*y); 
  
  z=z+2/(n*pi)*gg*((ff*(bb-cc)-dd+ee)/(1-aa)+1); 
end 

fid=fopen('c:\break\P_exact.dat', 'w'); 
format short; 
fprintf(fid,'Analytical solution of a Laplace Equation\n'); 
fprintf(fid,'with b.C.: p=1 @ x=0; p=2 @ y=0; p=0 at x=y=pi\n'); 
for i=1:m 
  fprintf(fid, '%d\n', i); 
  a=z(i,1:m); 

V-1
fprintf(fid, '%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f
 %7.4f %7.4f %7.4f
', a);
fprintf(fid, '\n');
end
fclose(fid);

%mesh(z);
%pause;

v=[0.01 0.1 0.2 0.3 0.4 0.5 0.6 0.8 0.9 1 1.2 1.4 1.6 1.8 1.9 2];
c=contour(x,y,z,v);
axis('square');
clabel(c,'manual');
xlabel('X');
ylabel('Y');

text(2.6, 2.9, ['n=' num2str(no)], 'sc'),
end