1-1-1997

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

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

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 x 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 π, 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:

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

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, ϕ increases sharply and approaches 1 when $y > 1\text{x}10^{-3}$. 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

$$p_{i-1,j} + r\,p_{i,j-1} - 2(1+r)\,p_{i,j} + r\,p_{i,j+1} + p_{i+1,j} = 0 \quad \ldots\ldots\ldots\ldots \quad (3)$$

where $r = (\Delta x/\Delta y)^2$, and Δx and Δy 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 e points in both the x and y directions, is initially chosen. See

$$\begin{array}{rrrrrrrrrrr}
2 & + & 1 & - & 4\,p_1 & + & p_2 & + & p_{11} & = & 0 \\
2 & + & p_2 & - & 4\,p_3 & + & p_4 & + & p_{12} & = & 0 \\
2 & + & p_3 & - & 4\,p_4 & + & p_5 & + & p_{13} & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
2 & + & p_8 & - & 4\,p_9 & + & p_{10} & + & p_{19} & = & 0 \\
2 & + & p_9 & - & 4\,p_{10} & + & 0 & + & p_{20} & = & 0 \\
p_1 & + & 1 & - & 4\,p_{11} & + & p_{12} & + & p_{21} & = & 0 \\
p_2 & + & p_{11} & - & 4\,p_{12} & + & p_{13} & + & p_{22} & = & 0 \\
p_3 & + & p_{12} & - & 4\,p_{13} & + & p_{14} & + & p_{23} & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
p_{79} & + & p_{88} & - & 4\,p_{89} & + & p_{90} & + & p_{99} & = & 0 \\
p_{80} & + & p_{89} & - & 4\,p_{90} & + & 0 & + & p_{100} & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
p_{88} & + & p_{97} & - & 4\,p_{98} & + & p_{99} & + & 0 & = & 0 \\
p_{89} & + & p_{98} & - & 4\,p_{99} & + & p_{100} & + & 0 & = & 0 \\
p_{90} & + & p_{99} & - & 4\,p_{100} & + & 0 & + & 0 & = & 0
\end{array} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

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, ..., p_{99}, p_{100}.

Application of Eq. 3 at these intersection points (from p_1 to quation, a course grid, $\Delta x = \Delta y = 0.28559933$, i.e., 12 grid 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:

$$\mathbf{A}\,\mathbf{P} = \mathbf{B} \qquad \text{..... (5)}$$

where **A** is a banded coefficient matrix, **P** is a column matrix that contains the unknown variable p in the study domain, and **B** is a column matrix that contains the given boundary conditions.

$$\mathbf{A} = \begin{bmatrix}
0 & 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 1 \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\
. & . & . & \dots & . & . & . & \dots & . & . & . & \dots & . & . \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\
0 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 1 \\
-4 & -4 & -4 & \dots & -4 & -4 & -4 & \dots & -4 & -4 & -4 & \dots & -4 & -4 \\
1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 0 \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\
. & . & . & \dots & . & . & . & \dots & . & . & . & \dots & . & . \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\
1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 1 & 0 & \dots & 0 & 0
\end{bmatrix} \qquad \text{..... (6)}$$

$$P^T = \left[P_1 \;\; P_2 \;\; P_3 \;\; P_4 \;\; \ldots\ldots \;\; P_{i-1} \;\; P_i \;\; P_{i+1} \;\; \ldots \;\; P_{98} \;\; P_{99} \;\; P_{100}\right] \quad \ldots\ldots\ldots\ldots \quad (7)$$

$$B^T = \left[-3 \;\; -2 \;\; -2 \;\; .. \;\; -2 \;\; -1 \;\; 0 \;\; 0 \;\; \ldots \;\; 0 \;\; 1 \;\; 0 \;\; \ldots \;\; 0 \;\; 1 \;\; 0 \;\; \ldots \;\; 0 \;\; 0\right] \quad .. \quad (8)$$

Matrix **A** has a dimension of M x N, (21 x 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 $P_{i,i}$.

GENERAL BANDED MATRIX EQUATION

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

$$
\mathbf{A} = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & a_{1,u} & \cdots & a_{1,n-L} & a_{1,n-L-1} & \cdots & a_{1,n-1} & a_{1,n} \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot & \cdot \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & a_{d-1,2} & a_{d-1,3} & \cdots & a_{d-1,u-1} & a_{d-1,u} & \cdots & a_{d-1,n-L} & a_{d-1,n-L-1} & \cdots & a_{d-1,n-1} & a_{d-1,n} \\
a_{d,1} & a_{d,2} & a_{d,3} & \cdots & a_{d,u-1} & a_{d,u} & \cdots & a_{d,n-L} & a_{d,n-L-1} & \cdots & a_{d,n-1} & a_{d,n} \\
a_{d+1,1} & a_{d+1,2} & a_{d+1,3} & \cdots & a_{d+1,u-1} & a_{d+1,u} & \cdots & a_{d+1,n-L} & a_{d+1,n-L-1} & \cdots & a_{d+1,n-1} & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot & \cdot \\
0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,u-1} & a_{m,u} & \cdots & a_{m,n-L} & 0 & \cdots & 0 & 0
\end{bmatrix} \quad \cdots \quad (9)
$$

$a_{1,u}$, $a_{1,u+1}$, ... , 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, $a_{m,1}$, $a_{m,2}$,..., and $a_{m,L}$ 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 x 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) x (NK+M), where NK is determined by the available computer memory.

$$
W = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & a_{1,u} & a_{1,u+1} & \cdots & a_{1,nk+m-1} & a_{1,nk+m} \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & a_{d-1,2} & a_{d-1,3} & \cdots & a_{d-1,u-1} & a_{d-1,u} & a_{d-1,u+1} & \cdots & a_{d-1,nk+m-1} & a_{d-1,nk+m} \\
a_{d,1} & a_{d,2} & a_{d,3} & \cdots & a_{d,u-1} & a_{d,u} & a_{d,u+1} & \cdots & a_{d,nk+m-1} & a_{d,nk+m} \\
a_{d+1,1} & a_{d+1,2} & a_{d+1,3} & \cdots & a_{d+1,u-1} & a_{d+1,u} & a_{d+1,u+1} & \cdots & a_{d+1,nk+m-1} & a_{d+1,nk+m} \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,u-1} & a_{m,u} & a_{m,u+1} & \cdots & a_{m,n+m-1} & a_{m,nk+m}
\end{bmatrix} \qquad (10)
$$

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.

$$
W = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & 0 & u_{1,M} & \cdots & u_{1,NK+M} \\
0 & 0 & 0 & \cdots & 0 & u_{2,M-1} & u_{2,M} & \cdots & u_{2,NK+M} \\
0 & 0 & 0 & \cdots & u_{3,M-2} & u_{3,M-1} & u_{3,M} & \cdots & u_{3,NK+M} \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & 0 & 0 & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & 0 & 0 & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & 0 & 0 & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & 0 & u_{M-2,3} & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
0 & u_{M-1,2} & u_{M-1,3} & \cdots & u_{M-1,M-2} & u_{M-1,M-1} & u_{M-1,M} & \cdots & u_{M-1,NK+M} \\
d_{M,1} & d_{M,2} & d_{M,3} & \cdots & d_{M,M-2} & d_{M,M-1} & d_{M,M} & \cdots & d_{M,NK+M} \\
m_{M+1,1} & m_{M+1,2} & m_{M+1,3} & \cdots & m_{M+1,M-2} & m_{M+1,M-1} & m_{M+1,M} & \cdots & m_{M+1,NK+M} \\
m_{M+2,1} & m_{M+2,2} & m_{M+2,3} & \cdots & m_{M+2,M-2} & m_{M+2,M-1} & m_{M-2,M} & \cdots & m_{M+2,NK+M} \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
\cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\
m_{M+ML-2,1} & m_{M+ML-2,2} & m_{M+ML-2,3} & \cdots & \cdot & \cdot & \cdot & \cdots & m_{M+ML-2,NK+M} \\
m_{M+ML-1,1} & m_{M+ML-1,2} & m_{M+ML-1,3} & \cdots & \cdot & \cdot & \cdot & \cdots & m_{M+ML-1,NK+M} \\
m_{M+ML,1} & M_{M+ML,2} & m_{M+ML,3} & \cdots & \cdot & \cdot & \cdot & \cdots & m_{M+ML,NK+M}
\end{bmatrix} . \quad (11)
$$

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 thery 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 expending the first block of **V** and **T** into **W** is then retrieved, and the second block (also lenth 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 Neumenn 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

Financial support from the U.S. Department of the Interior, Minerals Management Service, Office of International and Marine Minerals, under contract number 14-35-0001-30740 is sincerely acknowledged.

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.

Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewwart, 1979, Linpack Users Guide, SIAM.

Mathews, J. H., 1987, Numerical Methods for Computer Science, Engineering, and Mathematics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 507pp.

Roache, P. J., 1972, Computational Fluid Dynamics, Hermosa Publishers, Albuquerque, New Mexico, 434 pp.

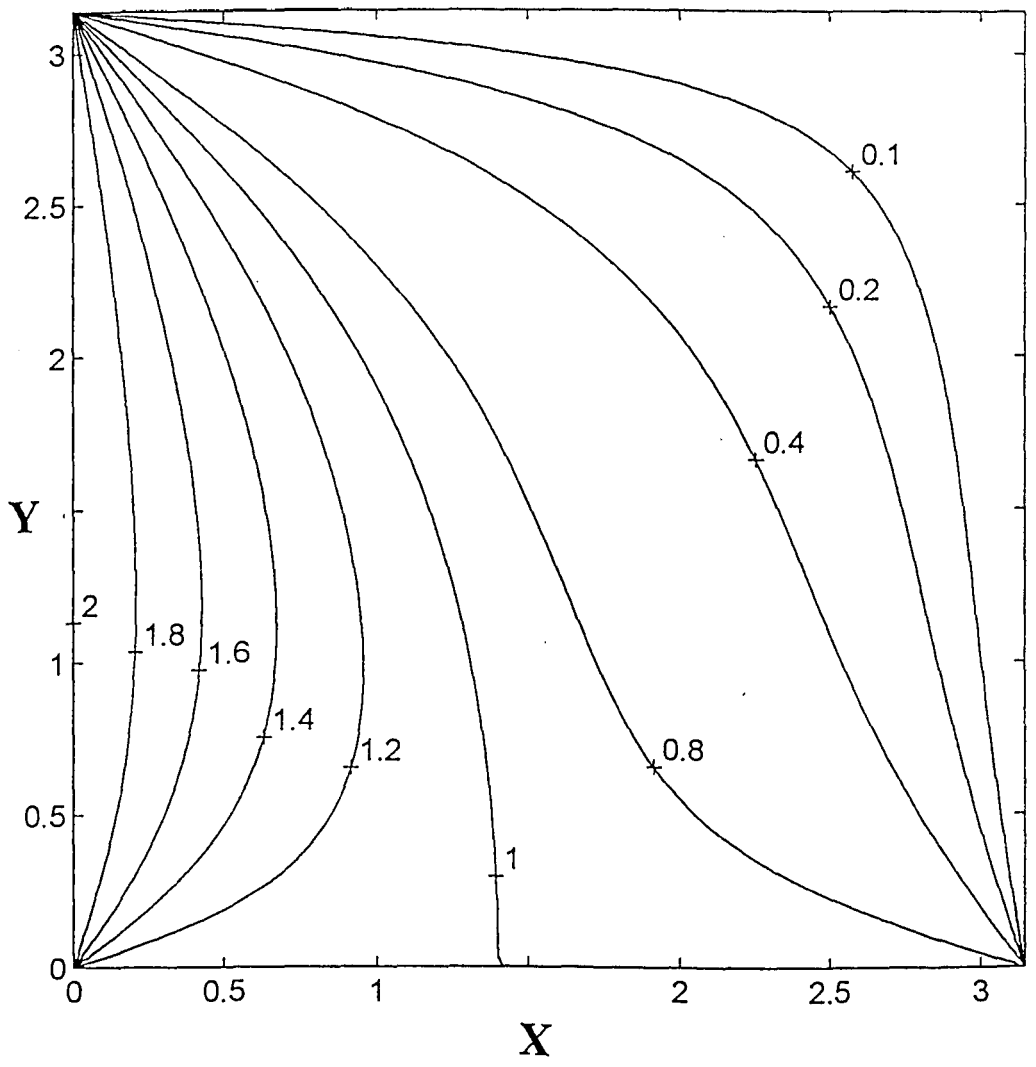

Fig. 1. Contours Plot of the Analytical Solution.

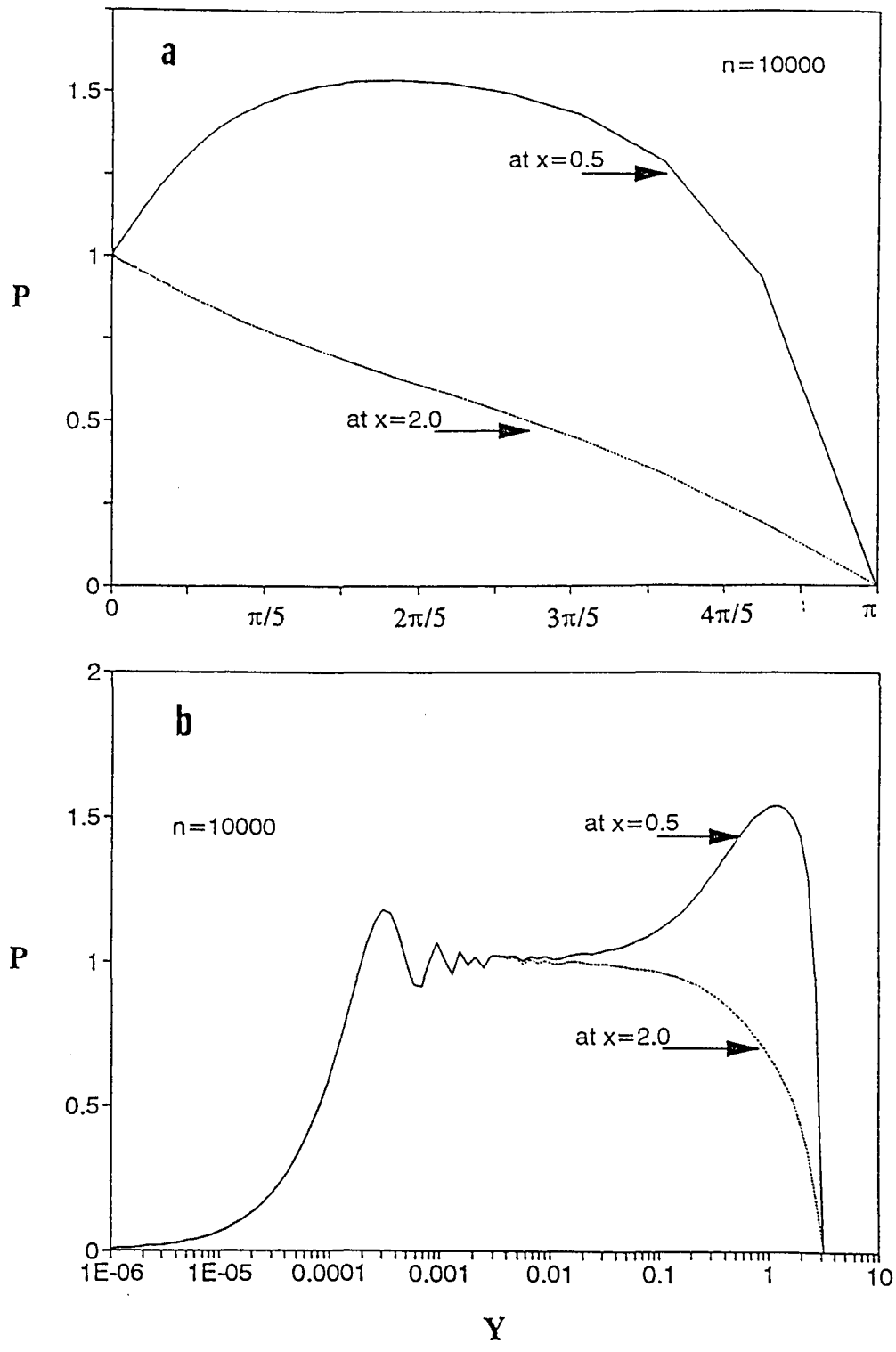

Fig. 2. Analytical Solution Profiles at Selected x locations.

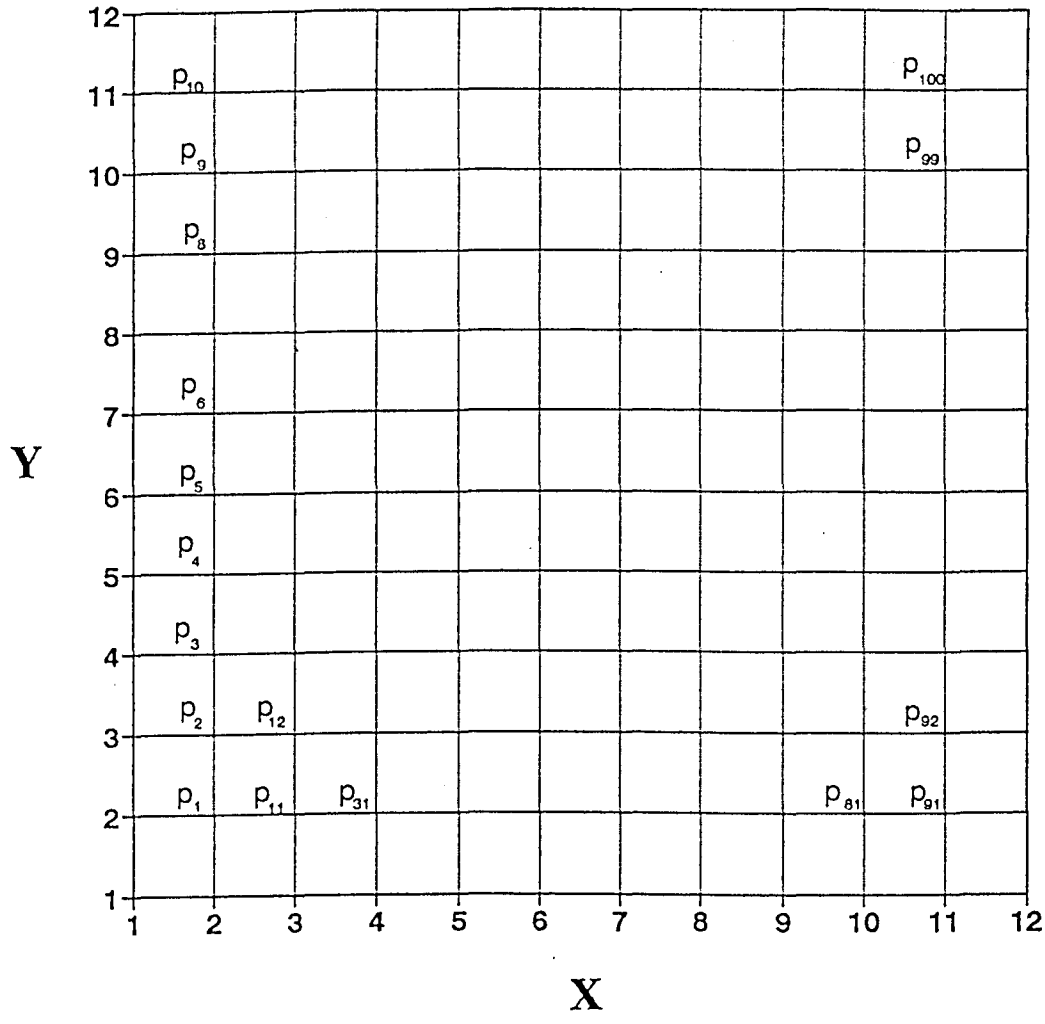

Fig. 3. Coarse Grid Points in the Study Domain.

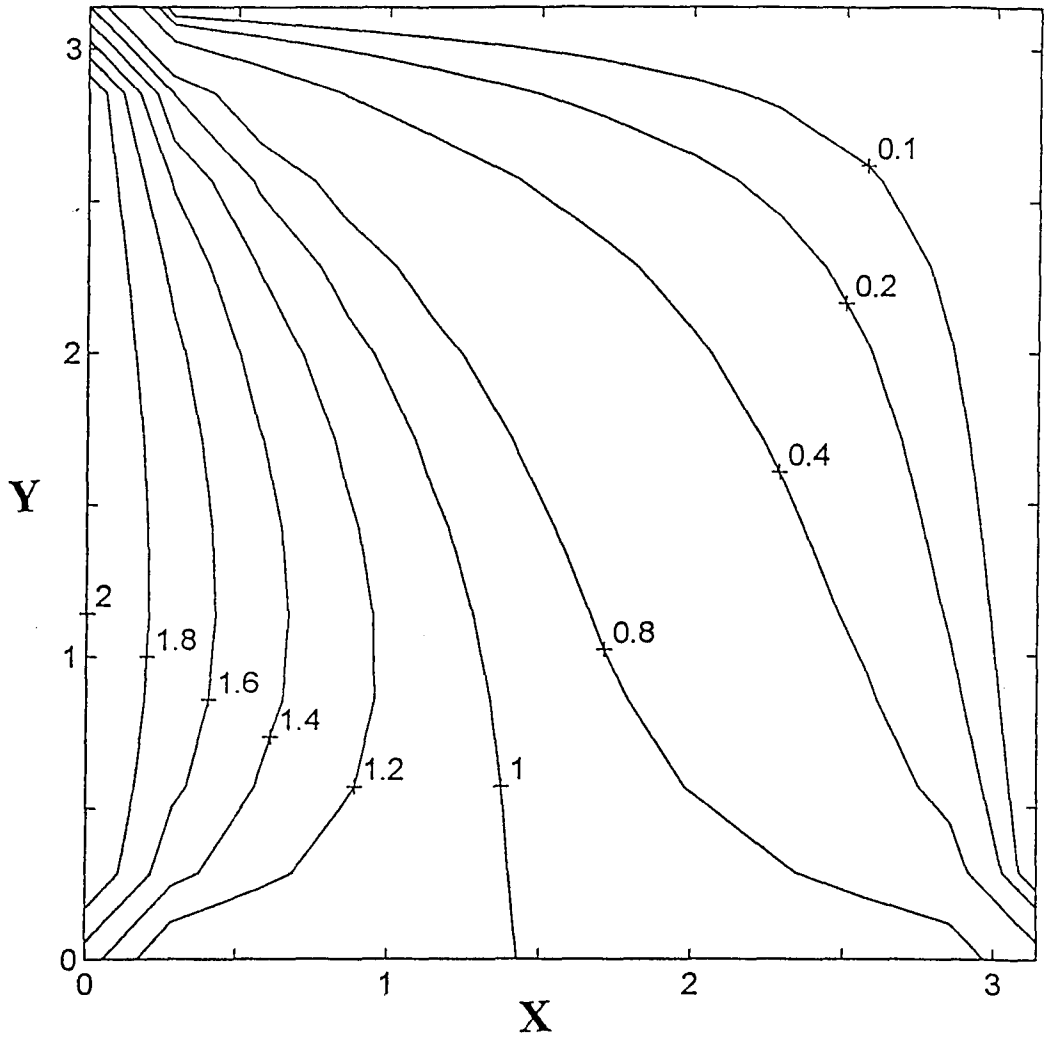

Fig. 4. Numerical Solution Using the Coarse Grid

Appendix I. Source Lists of FORTRAN program P_BANDED.FOR

```
      program p_banded
c
c This program is to calculate the velocity potential (p) in the
c area 0.0 <= x <= 3.1416,  0.0 <= y <= 3.1416.
c
c a thrify storage method was constructed first, and then,
c convert to that required by LINPACK, and uses a minor modified
c LINPAK subroutines to solve it.
c
c Because of the formidible slow pace when runing this program
c with high resolution, we can only limited the parameter KQ to a
c small number.
c
      parameter (iq=110,jq=110, IW=310, kq=400, Lq=3, mq=203)
      implicit complex (z)
      character*45 title, label
      character*20 infile, outfile, recfile
      character*1 id, q1
      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)
c
c     boundary conditions
c
      bcx0=2
      bcxn=0
      bcy0=1
      bcyn=0
c
      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'
 20   continue
c
      open(9, file=infile, status='old')
      open(10, file=outfile, status='unknown')
      open(11, file=recfile, status='unknown')
```

```
c
c   mp,np : Max. grid numbers in x and y direction, respectively.
c   dx,dy : grid sizes in x and y direction, respectively.
c
      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
c
      r = (dx/dy)**2
c
c     ID code for each grid point
c     : 0, interior point
c
c     : 1, upper boundary condition
c     : 2, lower boundary condition
c     : 3, left boundary condition
c     : 4, right boundary condition
c
c     : 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
c     : e, grid point that is not included in the study domain
c
c     The following is an example
c
c     j
c     ^
c     |       np  6111111111118eeeeeeeeeeeeeeeeeee
c     |           3000000000004eeeeeeeeeeeeeeeeeee
c     |           3000000000001111111111111111118
c     |           3000000000000000000000000000004
c     |           3000000000000000000000000000004
c     |           3000000000000000000000000000004
c     |           3000000000000000000000000000004
c     |           5220000000000000000000000222227
c     |           eee3000000000000000000004eeeeee
c     |           eee3000000000000000000004eeeeee
c     |           eee3000000000000000000004eeeeee
c     |         1 eee5222222222222222222227eeeeee
c     |           1                              mp
c     |-------------------------------------------> i
c
      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)
c
c  construct the unknow COLUMN matrix X, in the matrix eq. AX=B
c  find each unknown's location: imap(map),jmap(map)
c
c  ind(i) : no. of isolated sector in each culomn, x grid.
c  for this particular case, ind(i) are all  1.
c  because no land points in the middle of study domain.
c  jin(i,index) : begin grid number for an isolated sector in a
c                 column
c  jout(i,index): end grid number for an isolated sector in a
c                 culomn
c  map      : the number of total unknow, or the length of X.
c             later, it is reassigned as N
c  irow(i)  : the total number of unknow vel. Potential in each
c             column
c
      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
c
            if( id(i,j) .eq. '1' .and. iout .eq. 0) then
```

```
                  jout(i,index)=j
                  iout=1
                  in=0
                  index=index+1
               end if
            end do
         irow(i)=icount
         ind(i)=index-1
c
c  ind(i) should be >= 1, except for the entire column are all
c  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=',5i5)

         if( ind(i) .gt. Lq) then
            write(11,60) 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
c
c n: length of the banded matrix
c
         n=map
         if(n .lt. 0.8*kq) then
            write(*,65) n, kq
 65         format('
-------------------------------------------------'/
     *         ' N=',i5,' << KQ=',i5//' 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=',i5,' > KQ=',i5//
     *         ' Increases KQ to ',i5, ' and re-run')
            stop
         end if
c
c Set up the two small matrices ZA and IA for storing the banded
c matrix and find out the band width, work size, etc.
c
         mu=0
```

```
      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)
c
      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
c
      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
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 .
```

```
C The ML+MU by ML+MU upper left triangle and the ML by ML lower
c right triangle are not referenced.
c
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
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
c
c  position the low triangular elements
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))
         read(*,'(a1)') q1
         end do
c
c  call gauss elimination to slove the complex band matrix equ.
```

```
c
      print*,'Calling CGBFA, wait ...'
      call CGBFA(Lda, n, ml, mu, info)
c
      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)
 132        format('   i          zabd(i,j), j=1,n')
            do i=1,Lda
               write(11,135) i, (real(zabd(i,j)),j=1,n)
 135           format(i5/(10f8.3))
               end do
            end if
        else
         write(*,140) info
 140     format(' error code (info) =', i5,' After called CGBFA')
        end if
c
      call CGBSL(lda, n, ml, mu, zB)
c
      if (info .eq. 0) then
         print*,'Successfully called CGBRL'
        else
         write(*,160) info
 160     format(' error code (info) =', i5,' after called CGBFA')
        end if
c
      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(10,200) j, (p(i,j),i=1,mp)
 200     format(i5/(10f8.4))
         end do

      close(10)
      close(11)

      print *,'program stop'
      stop
      end
c
c-------------------------------------------------------
c
      subroutine domain(i, j, map, ier)
c
      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)
c
      dimension iid(9)
c
c The finite difference governing equation, the Laplace equation,
c with   r = (dx/dy)**2, is
c
c  p(i-1,j) +r*p(i,j-1) -(2+2r)*p(i,j) +r*p(i,j+1) +p(i+1,j) = 0
c
c In storage, the coefficient of p(i-1,j) is stored in (map,1),
c             the coefficient of p(i,j-1) is stored in (map,2),
c             the coefficient of p(i,j)   is stored in (map,3),
c             the coefficient of p(i,j+1) is stored in (map,4),
c         and the coefficient of p(i+1,j) is stored in (map,5)
c
      ier=0
      do k=1,9
         iid(k)=0
         end do
      icount=0
c
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-idistl(i,j)+1
            ia(map,2)=map-1
            ia(map,3)=map
            ia(map,4)=map+1
            ia(map,5)=map+idistr(i,j)-1
            zb(map)=(0.0, 0.0)
            icount=icount+1
            iid(1)=1
            end if
c
c   for those has a Boundary grind point on the left
c
         if(id(i-1,j) .eq. '3' .and. id(i,j-1) .eq. '0' .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+idistr(i,j)-1
            zb(map)=(-2.0, 0.0)
            icount=icount+1
            iid(2)=1
            end if
c
c   for those has a Boundary grind point on the right
c
         if(id(i+1,j) .eq. '4' .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)=0.0
            ia(map,1)=map-idistl(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
c
c   for those has a Boundary grind point on the top
c
```

```
      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
c
c for these grid points that has a Boundary at bottom
c
      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
c
c For those grid points that are left bottom corner
c
      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
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
         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
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
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
c
      if( icount .ne. 1) then
          ier=1
          write(11,10) ier, icount, (iid(k),k=1,9)
 10       format(' ier, icount=',2i5,' id(1)-id(9)=',9i2)
          end if
      return
      end
c
c-----------------------------------------------------------------
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)
c
c  Calculates the distance between points (i+1,j) and (i,j).  It
c  means how many grid points, which are solved for, are in
c  between. It counts vertically from the point(i,j) and up
c  (i,j+1),(i,j+2), ..., (i,np), then (i+1,1), (i+1,2), ..., to
c  (i+1,j).  In the matrix equation, this distance represents the
c  up band width at that particular diagonal element (i,j).
c
      do k=1,ind(i)
         if(j .ge. jin(i,k) .and. j .le. jout(i,k) ) mark1=k
         end do
c
      do k=1,ind(i+1)
         if(j .ge. jin(i+1,k) .and. j .le. jout(i+1,k) ) mark2=k
         end do
c
      idd=jout(i,mark1)-j+1 + j-jin(i+1,mark2)
c
      do k=mark1+1, ind(i)
         idd=idd + jout(i,k)-jin(i,k)+1
         end do
c
      do k=1,mark2-1
         idd=idd + jout(i+1,k)-jin(i+1,k)+1
         end do
c
      idistr=idd
c
      return
```

```
      end
c
c
      integer function idistl(i,j)
c
      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)
c
c  Calculate the distance between point (i-1,j) and (i,j).  The
c  distance means how many grid points, which are solved for, are
c  in between. It counts vertically from (i,j) downward (i,j-1),
c  (i,j-2), ..., (i,1) and restarted from previous i line
c  (i-1,np), (i-1,np-1), ..., to (i-1,j).
c
      do k=1,ind(i-1)
         if(j .ge. jin(i-1,k) .and. j .le. jout(i-1,k)) mark1=k
         end do
c
      do k=1,ind(i)
         if(j .ge. jin(i,k) .and. j .le. jout(i,k)) mark2=k
         end do
c
      idd=jout(i-1,mark1)-j+1 + j-jin(i,mark2)
c
      do k=mark1+1,ind(i-1)
         idd=idd+jout(i-1,k)-jin(i-1,k)+1
         end do
c
      do k=1,mark2-1
         idd=idd+jout(i,k)-jin(i,k)+1
         end do
c
      idistl=idd
      return
      end
c
c
      SUBROUTINE CGBFA(LDA, N, ML, MU, INFO)
C
C  Original Date of WRITTEN  Aug. 14 1978 by MOLER, C. B.,
C  PURPOSE  Factors a COMPLEX band matrix by Gaussian elimination
C           and partial pivoting.
C
C  CGBFA is usually called by CGBCO, but it can be called
C        directly with a saving in time if  RCOND  is not needed.
```

```
C
C  On Entry
C
C ABD  COMPLEX(LDA, N), contains the matrix in band storage.  The
C      columns of the matrix are stored in the columns of ABD and
C      the diagonals of the matrix are stored in rows ML+1
C      through 2*ML+MU+1 of  ABD. See comments below for details.
C
C LDA  INTEGER,    the leading dimension of the array  ABD .
C                    LDA must be .GE. 2*ML + MU + 1 .
C
C N    INTEGER,    the order of the original matrix.
C
C ML   INTEGER,    number of diagonals below the main diagonal.
C                  0 .LE. ML .LT. N .
C
C MU   INTEGER,    number of diagonals above the main diagonal.
C                 0 .LE. MU .LT. N. More efficient if  ML .LE. MU
C
C  On Return
C
C ABD  an upper triangular matrix in band storage and the
C      multipliers which were used to obtain it.  The
C      factorization can be written  A = L*U where  L  is a
C      product of permutation and unit lower triangular matrices
C      and  U  is upper triangular.
C
C   IPVT INTEGER(N), an integer vector of pivot indices.
C
C   INFO INTEGER
C    = 0  normal value.
C    = K  if  U(K,K) .EQ. 0.0 .  This is not an error condition,
C         but it does indicate that CGBSL will divide by zero if
C         called.  Use RCOND  in CGBCO for a reliable indication
C         of singularity.
C
C  This uses rows ML+1 through 2*ML+MU+1 of ABD. In addition, the
C  first ML rows in ABD are used for elements generated during
C  the triangularization.  The total no. of rows needed in ABD is
C  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.
C
C  LINPACK.  This version dated 08/14/78 .
C  Cleve Moler, University of New Mexico, Argonne National Lab.
C
C  Subroutines and Functions called from this subroutine
C
C     BLAS CAXPY,CSCAL,ICAMAX
C     Fortran ABS,AIMAG,MAX0,MIN0,REAL
C
C  REF: DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,
C               *LINPACK USERS  GUIDE*, SIAM, 1979.
```

```
C  ROUTINES CALLED  CAXPY,CSCAL,ICAMAX
C
c  Revised by Jerome P.-Y. Maa on Nov. 96.
c 1. Move the matrix abd(Lda, 1) and Ipvt(1) to common block.  so
c    that the limitiation of LDA = IW can be relaxed.  In other
c    words, the parameter IW specified don't have to be exactly
c    equal to LDA.
c 2. uses implicit statement for complex variables
c
      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))
C
C  FIRST EXECUTABLE STATEMENT  CGBFA
C
      M = ML + MU + 1
      INFO = 0
C
C ZERO INITIAL FILL-IN COLUMNS
C
      J0 = MU + 2
      J1 = MIN0(N,M) - 1
      IF (J1 .LT. J0) GO TO 30
      DO 20 JZ = J0, J1
         I0 = M + 1 - JZ
         DO 10 I = I0, ML
            zABD(I,JZ) = (0.0E0,0.0E0)
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
      JZ = J1
      JU = 0
C
C GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 130
      DO 120 K = 1, NM1
         KP1 = K + 1
C
C ZERO NEXT FILL-IN COLUMN
C
         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
C FIND L = PIVOT INDEX
C
         LM = MIN0(ML,N-K)
         L = ICAMAX(LM+1,zABD(M,K),1) + M - 1
         IPVT(K) = L + K - M
C
C ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (CABS1(zABD(L,K)) .EQ. 0.0E0) GO TO 100
C
C INTERCHANGE IF NECESSARY
C
            IF (L .EQ. M) GO TO 60
               zT = zABD(L,K)
               zABD(L,K) = zABD(M,K)
               zABD(M,K) = zT
   60       CONTINUE
C
C COMPUTE MULTIPLIERS
C
            zT = -(1.0E0,0.0E0)/zABD(M,K)
            CALL CSCAL(LM, zT, zABD(M+1,K), 1)
C
C ROW ELIMINATION WITH COLUMN INDEXING
C
            JU = MIN0(MAX0(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)
```

```
         end if
c
      RETURN
      END
c
c
c
      SUBROUTINE CGBSL(LDA, N, ML, MU, zb)
C
C Original written on Aug. 14, 78, revised on  Aug. 1, 82
C   AUTHOR  MOLER, C. B., (U. OF NEW MEXICO)
C
C DESCRIPTION
C
C     CGBSL solves the complex band system
C     A * X = B  or  CTRANS(A) * X = B
C     using the factors computed by CGBCO or CGBFA.
C
C On Entry
C
C  ABD   COMPLEX(LDA, N), the output from CGBCo or CGBFA,
C                         passed by common block.
C  LDA   INTEGER,     the leading dimension of the array  ABD .
C  N     INTEGER,     the order of the original matrix.
C  ML    INTEGER,     number of diagonals below the main diagonal.
C  MU    INTEGER,     number of diagonals above the main diagonal.
C  IPVT  INTEGER(N), the pivot vector from CGBCO or CGBFA.
C                     passed to this subroutine by common block.
C  zB    COMPLEX(N), the right hand side vector.
C
C On Return
C
C  zB      the solution vector  X .
C
C Error Condition
C
C  A division by zero will occur if the input factor contains a
C  zero on the diagonal.  Technically this indicates singularity
C  but it is often caused by improper arguments or improper
C  setting of LDA .  It will not occur if the subroutines are
C  called correctly and if CGBCO has set RCOND .GT. 0.0  or CGBFA
C  has set INFO .EQ. 0 .
C
C  LINPACK.  This version dated 08/14/78 .
C  Cleve Moler, University of New Mexico, Argonne National Lab.
C
C  Subroutines and Functions used in this subroutine
C
C     BLAS CAXPY, zCDOTC  Fortran CONJG,MIN0
C
C  REF:  DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,
C              LINPACK USERS  GUIDE, SIAM, 1979.
```

```
C
c  Modified by Jerome Maa on Nov. 1996
c  1. the matrix ZABD, IPVT are transferred by common block
c  2. delete the Job parameter, so this subroutine only for
c     solving   AX=B
c  3. use implicit for complex variables
c
      parameter (IW=310, kq=400)
      implicit complex (z)
      common /works/ zabd(iw,kq), ipvt(kq)
      dimension zb(1)
c
      M = MU + ML + 1
      NM1 = N - 1
C
C  SOLVE  A * X = B, but first to do the same factoralization and
C                    partial pivoting on the column matrix ZB.
C
      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
C
C  NOW SOLVE  A*X = B
C
      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
C
C  Recovery sequence of unknown
C
      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)
```

```
          zB(K)=zT
 1000     continue
C
      RETURN
      END
c
c
      SUBROUTINE CSCAL(N,CA,CX,INCX)
C
C  PURPOSE  Complex vector scale x = a*x
C  WRITTEN on Oct. 1, 79, REVISION on Aug. 01, 82
C  CATEGORY NO.  D1A6
C  KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,SCALE,VECTOR
C  AUTHORS  LAWSON, C. L., (JPL),   HANSON, R. J., (SNLA)
C       KINCAID, D. R., (U. OF TEXAS), and KROGH, F. T., (JPL)
C
C  DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       CA  complex scale factor
C       CX  complex vector with N elements
C     INCX  storage spacing between elements of CX
C
C     --Output--
C    CSCAL  complex result (unchanged if N .LE. 0)
C
C     replace complex CX by complex CA*CX.
C     For I = 0 to N-1, replace CX(1+I*INCX) with
C                               CA * CX(1+I*INCX)
C  REF: 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
      COMPLEX CA,CX(1)
C  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
c
c
      SUBROUTINE CAXPY(N,CA,CX,INCX,CY,INCY)
C
```

```
C  WRITTEN on Oct. 01, 79, REVISION on April 25, 84
C  CATEGORY NO.  D1A7
C  KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,TRIAD,VECTOR
C  AUTHOR  LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
C          KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
C  PURPOSE  Complex computation y = a*x + y
C  DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       CA  complex scalar multiplier
C       CX  complex vector with N elements
C     INCX  storage spacing between elements of CX
C       CY  complex vector with N elements
C     INCY  storage spacing between elements of CY
C
C     --Output--
C       CY  complex result (unchanged if N .LE. 0)
C
C     Overwrite complex CY with complex  CA*CX + CY.
C     For I = 0 to N-1, replace
C       CY(LY+I*INCY) with CA*CX(LX+I*INCX) + CY(LY+I*INCY),
C       where
C       LX = 1 if INCX .GE. 0, else LX = (-INCX)*N
C       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
      COMPLEX CX(1),CY(1),CA
C  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
C
C
      INTEGER FUNCTION ICAMAX(N,CX,INCX)
C
C  WRITTEN on Oct. 01, 79, REVISed on Aug. 01, 82
C  CATEGORY NO.  D1A2
C  KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,MAXIMUM COMPONENT,VECTOR
C  AUTHOR  LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
C          KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
C  PURPOSE  Find the location (or index) of the largest component
C  of a complex vector
C  DESCRIPTION
C
C                B L A S  Subprogram
C  Description of Parameters
C
C  --Input--
C       N  number of elements in input vector(s)
C      CX  complex vector with N elements
C    INCX  storage spacing between elements of CX
C
C  --Output--
C   ICAMAX  smallest index (zero if N .LE. 0)
C
C   Returns the index of the component of CX having the
C   largest sum of magnitudes of real and imaginary parts.
C   ICAMAX = first I, I = 1 to N, to minimize
C   ABS(REAL(CX(1-INCX+I*INCX))) + ABS(IMAG(CX(1-INCX+I*INCX)))
C  REF:  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
C
      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
C
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 complx
C            conjugate of first vector
C***DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       CX  complex vector with N elements
C     INCX  storage spacing between elements of CX
C       CY  complex vector with N elements
C     INCY  storage spacing between elements of CY
C
C     --Output--
C    zCDOTC  complex result (zero if N .LE. 0)
C
C  Returns the dot product for complex CX and CY, uses
C  CONJUGATE(CX)
C  zCDOTC=SUM for I=0 to N-1 of CONJ(CX(LX+I*INCX))*CY(LY+I*INCY)
C  where LX = 1 if INCX .GE. 0, else LX = (-INCX)*N, and LY is
C  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
C
      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
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 Porgram P_THRIFT.FOR

```
      program p_thrify
c
c  This program solves the Laplace equation (p) in the area
c  0.0 <= x <= 3.1416,  0.0 <= y <= 3.1416.
c  using the thrifty band matrix solver
c  This method was first developed by Li, C. (1995), and later,
c  improved and documented by  Jerome P.-Y. Maa
c
c  The Govern Equation is    d^2(p)/dx^2 + d^2(p)/dy^2 = 0
c  The finite difference equation is:
c  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
c
c  The boundary conditions are:
c  p = 2 at x = 0
c  p = 1 at y = 0
c  p = 0 at x = y = pi
c
c  INFILE and OUTFILE : to store input and main output data.
c  RECFILE is used to store routine checking information.  If the
c          program run O.K., it can be deleted.
c  A MATLAB program PHIPLT.M has been developed to plot the
c  output results.  It uses Matlab version 4.01 for windows, and
c  read output data from the OUTFILE directly.
c
c  Data in the file INFILE are generated by another program
c  ID.FOR  for the examples given in this program.
c
c  In this example, there is no need to use complex variable.
c  But, it was used anyway for general applications.
c
c  Jerome P.-Y. Maa    Virginia Institute of Marine Science.
c
      parameter (iq=110, jq=110, iw=310, jw=560, kq=10000, Lq=3)
c
      implicit complex (z)
      character*45 title, label
      character*20 infile, outfile, recfile
      character*1 id, q1
c
      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)
      common /works/ zw(iw,jw), ipvt(kq)
c
c     boundary conditions
c
      bcx0=2
```

```
      bcxn=0
      bcy0=1
      bcyn=0
c
      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
c
      open(9, file=infile, status='old')
      open(10, file=outfile, status='unknown')
      open(11, file=recfile, status='unknown')
c
c  mp,np : Max. grid numbers in x and y direction, respectively.
c  dx,dy : grid sizes in x and y direction, respectively.
c
      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
c
      r = (dx/dy)**2
c
c     ID code for each grid point
c     : 0, interior point
c
c     : 1, upper boundary condition
c     : 2, lower boundary condition
c     : 3, left boundary condition
c     : 4, right boundary condition
c
c     : 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
c      : e, grid point that is not included in the study domain
c
c      The following is an example
c
c      j
c      ^        np  6111111111118eeeeeeeeeeeeeeeeeee
c      |            3000000000004eeeeeeeeeeeeeeeeeee
c      |            3000000000001111111111111111118
c      |            3000000000000000000000000000004
c      |            3000000000000000000000000000004
c      |            3000000000000000000000000000004
c      |            3000000000000000000000000000004
c      |            5220000000000000000000000222227
c      |            eee3000000000000000000004eeeeee
c      |            eee3000000000000000000004eeeeee
c      |            eee3000000000000000000004eeeeee
c      |             1  eee5222222222222222222227eeeeee
c      |                1                                  mp
c      |-------------------------------------------------> i
c
       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)
c
c  construct the unknow COLUMN matrix X, in the full matrix eq.
c  AX=B.  Find each unknown's location: imap(map),jmap(map)
c
c  ind(i) : no. of isolated sector in each culomn, x grid.
c           for this particular case, ind(i) are all 1 because
c           no land points in the middle of study domain.
c  jin(i,index): begin grid number for an isolated sector in a
c                column
c  jout(i,index): end grid number for an isolated sector in a
c                 culomn
```

```
c  map      : the number of total unknow, or the length of X.
c             later, it is reassigned as N
c  irow(i)  : the total number of unknow, P, in each column
c
      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
c
            if( id(i,j) .eq. '1' .and. iout .eq. 0) then
               jout(i,index)=j
               iout=1
               in=0
               index=index+1
            end if
         end do
      irow(i)=icount
      ind(i)=index-1
c
c  ind(i) should be >= 1, except for the entire column are all
c  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=',5i5)

      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
c
c n: length of the banded matrix
c
      n=map
      if(n .lt. 0.8*kq) then
         write(*,65) n, kq
```

```
 65      format('-----------------------------------------------'/
     *      ' N=',i5,' << KQ=',i5//' 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
         end if
       if(n .gt. kq) then
         write(*,70) n, kq, n
 70      format('-----------------------------------------------'/
     *       ' N=',i5,' > KQ=',i5//
     *       ' Increases KQ to ',i5, ' and re-run')
         stop
       end if
c
c Set up the two small matrices ZA and IA for storing the banded
c matrix and find out the band width, work size, etc.
c
       mu=0
       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)
c
       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
            stop
           end if
c
c  find the band width
c
           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
c
       m=mL+mu+1
       Lda=m+mL
       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 Zw     = ',i7)
       if(Lda .gt. iw) then
          print*,'Please increase IW to ',Lda
          stop
          end if
c
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 oher details, if needed, is in  ', recfile
      stop
      end
c
c
*****************************************************************
*
c
      subroutine BMsolver(ierr_code)
c
c It solves a complex banded matrix equation ZAF * ZX = ZB
c where ZAF is a complex band matrix with dimension (m * n)
c    ZX and ZB are two complex column matrices with length (n).
c
c Because of the hudge size of ZAF, e.g., (300 x 50000), it is
c designed to solve this problem using the following two steps.
c
c First, don't use the full size of ZAF, instead, uses two small
c matrices: ZA & IA that each only uses 50000 x 5 to save space.
c
c ZA : a complex matrix (n * 5) to store the coefficent matrix in
c      a matrix equation ZAF*ZX= ZB.  Because of using the finite
c      difference method to solve an elliptic equation, there are
c      only 5 elements to be saved for the coefficient matrix.
c      The band width, however, is much much large than 5 with a
c      lot of "zero."  By doing so, we need another matrix
c IA : to save the corresponding locations.
c
c Second, uses a working matrix, ZW(IW,JW), and do a systematical
c
c swap between a hard disk and memory.  For this reason, be sure
c that you do have enough space in your hard disk. For example, a
c complex matrix with size of (300 x 50000) requires 120 MB for
c storage, if using 4 byte for a real number.  If using 8 bytes,
c then, 240 MB is needed.
c
c IW,JW: IW should be >=  m+ml, where ml is the lower band width.
c        The size of JW depends on the available computer memory.
c        In general, the large the JW, the less the disk IO, and
c        thus, the faster the computing speed.  As a rule of
c        thumb, you may select JW = 2*IW and tried to see if your
c        computer has enough memory to run the program.
c
c The proceedures follows that given in the subroutine CGBFA &
c CGBSL from LINPACK.  The major difference is just doing it one
c block at a time, stores the results in hard disk sequently.
c After forward elimination, reverse the process by reading the
c data from hard disk and do back substitute for the solution.
c
      parameter (iq=110, jq=110, iw=310, jw=560, kq=10000)
```

```
c
      implicit complex (z)
      character*8 tmpfile
      character*3 chrc
      character*1 id, q1
c
      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 /works/ zw(iw,jw), ipvt(kq)
c
      cabs1(zdum)=abs(real(zdum)) + abs(aimag(zdum))
c
c  Gaussian elimination with partial pivoting
c
c  NK: # of columns for the working matrix that will be saved.
c  NW: The number of column needed for working, nw=m-1+nk
c  NS: An index to show the number of working matrix.
c
c  check working matrix dimension
c
      nk=350
      nw=m+nk-1
c
      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
c
c  clear up the working matrix
c
      do i=1,iw
         do j=1,nw
            zw(i,j)=(0.0, 0.0)
            end do
         end do
c
c construct the first working matrix, ZW, which has a size of
c (Lda x nw).  The thrify storaged matrices are expanded, only
c to the first (nk x 5) block.
c
c The flag is used to recording only once when the ZA data
c starts lost.
c
      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
c Gaussian elimination with partial pivoting
c
      j1=min0(n,m)-1
      jz=j1
      ju=0
      nt=0
      index=0
c
  100 continue
c
      if(ju .ne. 0) ju=ju-nk
      if(jz .ne. j1) jz=jz-nk
c
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
c   find L = pivot index
c
      Lm=min0(mL,n-ns*nk-k)
      L=icamax(Lm+1, zw(m,k), 1) + m -1
      ipvt(kr)=L+kr-m
c
```

```
c    zero pivot implies that this column are all zeros, a singular
c    matrix.
c
       if(cabs1(zw(L,k)) .lt. 0.10e-8) goto 120
c
c  interchange if necessary
c
       if(L .ne. m) then
          zt=zw(L,k)
          zw(L,k)=zw(m,k)
          zw(m,k)=zt
          end if
c
c  compute multipliers
c
       zt=-(1.0e0,0.0e0)/zw(m,k)
       call cscal(Lm, zt, zw(m+1,k), 1)
c
c  swap ZB array, if necessary
c
       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)
c
c  row elimenation with column indexing
c
       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
c
          goto 150
  120     continue
          print*,'Zero diagonal element at (L,k)=', L, k
          stop
  150     continue
c
          end do
```

```
c
c  Eexcept the last working matrix, write the upper triangular
c  matrix (from j=1,nk) into hard disk.  For the last one, i.e.,
c  nc<>nk, go to back substitute directly.
c
      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)
c
c  moving the rest working matrix forward to the beginning
c
      do j=nk+1, nk+m-1
         L=j-nk
         do i=1,Lda
            zw(i,L)=zw(i,j)
            end do
         end do
c
c  clear the rest working area for reading new working matrix
c
      do j=m, nk+m-1
         do i=1,Lda
            zw(i,j)=(0.0,0.0)
            end do
         end do
c
c read in a full block of ZA and IA, by two steps
c
      ns=ns+1
      if( (ns+1)*nk+m-1 .lt. n) then
         ne=nk
         index=0
c
c  For intermediate blocks, read in ZA and IA by two steps.
c  First read in the upper triangular matrix that were cut off
c  at the previous time when constructing the working matrix.
c
         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
c
c  now read in next block of ZA and IA
c
         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
c
c  End of read in a block of ZA and IA.
c
       else
c
c  This is to read the last block of ZA and IA
c
         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
c
c now reads the last block of ZA and IA
c
            iflag=1
         map1=0
         do k=1,ne
```

```
            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
c
          goto 100
c
       else
c
c    backward substitution from the last submatrix
c
        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
c
c   If one loop can include all elements, i.e., for a small banded
c   matrix, just stops after this.
c
        if( nt .eq. 0 ) go to 400
       end if
c
c   complete the rest backward substitute
c
       ns=0
 200   continue
c
c   clear the working matrix for reading new submatrix from disk.
c
       do j=1,nk+m-1
          do i=1,m
             zw(i,j)=(0.0,0.0)
```

```
            end do
         end do
c
      print*,'Reading tem. file #', nt
      open(12,file=tmpfile,form='unformatted')
      read(12) ((zw(i,j),i=1,m),j=1,nk)
      close(12,status='delete')
c
      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
c
c  To restore the original sequence of ZB.  Since it starts
c  at the 2nd.  we started at 2nd too for restoring.
c
      print*,'Restore the original sequency'
      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
c
      return
      end
c
c
      character*3 function chrc(nt)
c
c It changes an input integer number NT to character
c CHANGQING LI 06/94
c
      character*1 c1
      character*2 c2
      character*3 c3
c
```

```
      i1=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
C
      return
      end
C
C
      SUBROUTINE CSCAL(N,CA,CX,INCX)
C
C  PURPOSE  Complex vector scale x = a*x
C  WRITTEN on Oct. 1, 79, REVISION on Aug. 01, 82
C  CATEGORY NO.  D1A6
C  KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,SCALE,VECTOR
C  AUTHORS  LAWSON, C. L., (JPL),   HANSON, R. J., (SNLA)
C        KINCAID, D. R., (U. OF TEXAS), and KROGH, F. T., (JPL)
C
C  DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       CA  complex scale factor
C       CX  complex vector with N elements
C     INCX  storage spacing between elements of CX
C
C     --Output--
C    CSCAL  complex result (unchanged if N .LE. 0)
C
C     replace complex CX by complex CA*CX.
```

```
C      For I = 0 to N-1, replace CX(1+I*INCX) with
C                                CA * CX(1+I*INCX)
C   REF: 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
      COMPLEX CA,CX(1)
C   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
c
c
      SUBROUTINE CAXPY(N,CA,CX,INCX,CY,INCY)
C
C   WRITTEN on Oct. 01, 79, REVISION on April 25, 84
C   CATEGORY NO.  D1A7
C   KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,TRIAD,VECTOR
C   AUTHOR  LAWSON, C. L., (JPL),  HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
C   PURPOSE  Complex computation y = a*x + y
C   DESCRIPTION
C
C                 B L A S  Subprogram
C     Description of Parameters
C
C      --Input--
C         N  number of elements in input vector(s)
C        CA  complex scalar multiplier
C        CX  complex vector with N elements
C      INCX  storage spacing between elements of CX
C        CY  complex vector with N elements
C      INCY  storage spacing between elements of CY
C
C      --Output--
C        CY  complex result (unchanged if N .LE. 0)
C
C      Overwrite complex CY with complex  CA*CX + CY.
C      For I = 0 to N-1, replace
c         CY(LY+I*INCY) with CA*CX(LX+I*INCX) + CY(LY+I*INCY),
c         where
c         LX = 1 if INCX .GE. 0, else LX = (-INCX)*N
C         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
      COMPLEX CX(1),CY(1),CA
C   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
C
C
      INTEGER FUNCTION ICAMAX(N,CX,INCX)
C
C   WRITTEN on Oct. 01, 79, REVISed on Aug. 01, 82
C   CATEGORY NO.  D1A2
C   KEYWORDS  BLAS,COMPLEX,LINEAR ALGEBRA,MAXIMUM COMPONENT,VECTOR
C   AUTHOR  LAWSON, C. L., (JPL), HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS), KROGH, F. T., (JPL)
C   PURPOSE  Find the location (or index) of the largest component
C            of a complex vector
C   DESCRIPTION
C
C                  B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       CX  complex vector with N elements
C     INCX  storage spacing between elements of CX
C
C     --Output--
C    ICAMAX  smallest index (zero if N .LE. 0)
C
C      Returns the index of the component of CX having the
C      largest sum of magnitudes of real and imaginary parts.
```

```
C      ICAMAX = first I, I = 1 to N, to minimize
C      ABS(REAL(CX(1-INCX+I*INCX))) + ABS(IMAG(CX(1-INCX+I*INCX)))
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
C
      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
c
c-------------------------------------------------------------
c
      subroutine domain(i, j, map, ier)
c
      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)
c
      dimension iid(9)
c
c The finite difference governing equation, the Laplace equation,
c with  r = (dx/dy)**2, is
c
c p(i-1,j) +r*p(i,j-1) -(2+2r)*p(i,j) +r*p(i,j+1) +p(i+1,j) = 0
c
c In storage, the coefficient of p(i-1,j) is stored in (map,1),
c             the coefficient of p(i,j-1) is stored in (map,2),
c             the coefficient of p(i,j)   is stored in (map,3),
```

```
c             the coefficient of p(i,j+1) is stored in (map,4),
c          and the coefficient of p(i+1,j) is stored in (map,5)
c
      ier=0
      do k=1,9
         iid(k)=0
         end do
      icount=0
c
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-idistl(i,j)+1
         ia(map,2)=map-1
         ia(map,3)=map
         ia(map,4)=map+1
         ia(map,5)=map+idistr(i,j)-1
         zb(map)=(0.0, 0.0)
         icount=icount+1
         iid(1)=1
         end if
c
c  for those has a Boundary grind point on the left
c
       if(id(i-1,j) .eq. '3' .and. id(i,j-1) .eq. '0' .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+idistr(i,j)-1
         zb(map)=(-2.0, 0.0)
         icount=icount+1
         iid(2)=1
         end if
c
c  for those has a Boundary grind point on the right
c
       if(id(i+1,j) .eq. '4' .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)=0.0
            ia(map,1)=map-idistl(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
c
c  for those has a Boundary grind point on the top
c
         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
c
c for these grid points that has a Boundary at bottom
c
         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
c
```

```
c For those grid points that are left bottom corner
c
      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
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
         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
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
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
c
      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
c
c-----------------------------------------------------------------
c
      integer function idistr(i,j)
      parameter (iq=110, Lq=3)
      common /labbl/ ind(iq),jin(iq,Lq),jout(iq,Lq)
c
c  Calculates the distance between points (i+1,j) and (i,j).  It
c  means how many grid points, which are looking for, are in
c  between. It counts vertically from the point(i,j) and up
c  (i,j+1),(i,j+2), ..., (i,np), then (i+1,1), (i+1,2), ..., to
c  (i+1,j).  In the matrix equation, this distance represents the
c  up band width at that particular diagonal element (i,j)
c
      do k=1,ind(i)
         if(j .ge. jin(i,k) .and. j .le. jout(i,k) ) mark1=k
         end do
c
      do k=1,ind(i+1)
         if(j .ge. jin(i+1,k) .and. j .le. jout(i+1,k) ) mark2=k
         end do
c
      idd=jout(i,mark1)-j+1 + j-jin(i+1,mark2)
c
```

```
      do k=mark1+1, ind(i)
         idd=idd + jout(i,k)-jin(i,k)+1
         end do
c
      do k=1,mark2-1
         idd=idd + jout(i+1,k)-jin(i+1,k)+1
         end do
c
      idistr=idd
c
      return
      end
c
c
      integer function idistl(i,j)
c
      parameter (iq=110, Lq=3)
      common /labbl/ ind(iq),jin(iq,Lq),jout(iq,Lq)
c
c  Calculate the distance between point (i-1,j) and (i,j).  The
c  distance means how many grid points, which are solved for, are
c  in between. It counts vertically from (i,j) downward (i,j-1),
c  (i,j-2), ..., (i,1) and restarted from previous i line
c  (i-1,np), (i-1,np-1), ..., to (i-1,j).
c
      do k=1,ind(i-1)
         if(j .ge. jin(i-1,k) .and. j .le. jout(i-1,k)) mark1=k
         end do
c
      do k=1,ind(i)
         if(j .ge. jin(i,k) .and. j .le. jout(i,k)) mark2=k
         end do
c
      idd=jout(i-1,mark1)-j+1 + j-jin(i,mark2)
c
      do k=mark1+1,ind(i-1)
         idd=idd+jout(i-1,k)-jin(i-1,k)+1
         end do
c
      do k=1,mark2-1
         idd=idd+jout(i,k)-jin(i,k)+1
         end do
c
      idistl=idd
      return
      end
```

Appendix III. Source List of FORTRAN Program ID.FOR

```
      program id_matrix
c
c  This program generates the input ID matrix, I.D. codes for
c  each cell
c
      parameter (iq=400,jq=200, kq=500, Lq=2000)
      character*1 id
      character*12 outfile
      character*30 header
      character*80 title
c
      common/const/ mp, np, dx, dy, r
      common/array/ id(iq,jq)
      common/bound/ nobc, ibc(Lq), jbc(Lq), zp(Lq)
c
c  mp     : mesh number in x direction
c  np     : mesh number in y direction
c  dx     : spatial step in x-direction
c  dy     : spatial step in y-direction
c   r     : square of dx/dy
c
c  ID_code for water depth
c       : 0, interior cell
c       : 1, upper boundary condition
c       : 2, lower boundary condition
c       : 3, left boundary condition
c       : 4, right boundary condition
c       : 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, land point
c
      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
```

```
c
      dx=3.1415926/(mp-1)
      dy=3.1415926/(np-1)
c
c    velocity potential
c
      if(mp .ge. iq .or. np .ge. jq) write(*,30) mp,np, iq,jq
 30   format(' Given array (mp,np)=',2i5,' is > than allocated',
     *       ' (iq,jq)=',2i5)
c
      open(8,file=outfile,form='formatted')
      write(8,'(a80)') title
      write(8,40) mp, np, dx, dy
  40  format(2i5, 2f12.8)
c
      nobc=0
c
c first establish the ID code for the entire grid point
c
      do j=1, np
         do i=1, mp
            id(i,j)='0'
            end do
         end do
c
c  given boundary at the right hand side
c
      do j=2,np-1
         nobc=nobc+1
         id(mp,j)='4'
         ibc(nobc)=mp
         jbc(nobc)=j
         zp(nobc)=0
         end do
c
c  left side B.C.
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
c
c top B.C.
c
      do i=2,mp-1
         id(i,np)='1'
         nobc=nobc+1
         ibc(nobc)=i
         jbc(nobc)=np
```

```
            zp(nobc)=0.0
            end do
c
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
c check the array assignment
c
       if(nobc .gt. Lq) then
          write(*,50) nobc, Lq, nobc
 50       format(' The # of Radiation B.C.=',i6,' > Lq=',i6/'
     *      ' Change the statement to Lq=',i6,' and re-run ')
          stop
          end if
c
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
 60    format('  j          i = 1, ..., ',i5)
       do j=1,np
          jj=np-j+1
          write(8,70) jj,(id(i,jj), i=1,mp)
 70       format(i4,1x,130a1)
          end do
c
       header='Given B.C. at following points'
       write(8,'(i5,2x,a30)') nobc, header
       write(8,80)
 80    format('    n     i     j       B.V.')
       do i=1,nobc
          write(8,90) i,ibc(i),jbc(i),zp(i)
          end do
 90    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(ftitle);
   mp=fscanf(fid, '%d', 1);
   np=fscanf(fid, '%d', 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 : ');
   end
```

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 a 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);
```

```
    fprintf(fid, '%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f
                  %7.4f %7.4f %7.4f\n', 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
```


VIMS SH 1 V48 no.135
Maa, P.-Y.
Using the Gaussian
elimination method for
large banded matrix