6. Solving the Poisson Equation

So far we have dealt only with either the advection or *maiertous* equations or the vorticity transport equation. We also need methods to solve the elliptic Poisson equation for the stream function ψ

$$(1)\ \nabla^2\psi = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \zeta$$

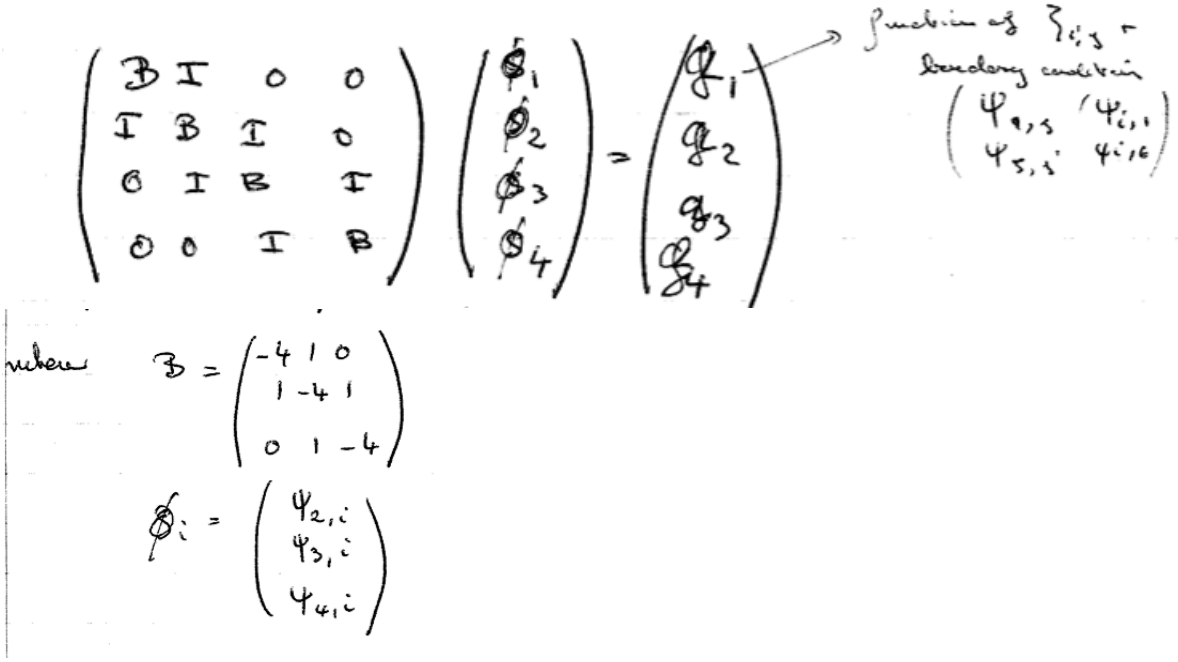
Using standard second-order finite differences, (1) can be approximated by:

$$\frac{\psi_{i+1,k} + \psi_{j-1,f} + \psi_{i,f+1} + \psi_{i,k-1} - 4\psi_{i,f}}{\Delta x^2} = \zeta_{i,j}$$

We know that the $\zeta_{i,j}$ we want the $\psi_{i,j}$. In practice, it is a very large set of equations that we are trying to solve.

Example:

The set of equations is for a 5 by 6 domain.



Block tridigonal set of equations can be used to our advantage in many schemes to solve the set.

6.1 Iteration Methods

These methods require the solution of a linear system of equations. The order of this system may be very large. However, the properties of these systems allows us to construct effective iteration methods for their solution.

We want to solve the matrix equation $Ax = b$ where x and b are n-dimensional vectors and A is a matrix of order n. We are trying to then to find the zeros of the vector function $f(x) = Ax - b$. We can then convert this to a fixed point problem by defining the functions $g(x) = x - f(x) = (I - A)x + b$. We are then now looking for vectors x such that $x = g(x)$. The easiest iteration method is:

$\rightarrow$Choose an initial guess x_o

$\rightarrow$ Define $x^{m+1} = g(x^m)$

$x^{m+1} = (I - A)x^m + b = Mx^m + b$

We denote the error ε^m by $\varepsilon^m = x - x^m$.

then

$\varepsilon^{m+1} = M\varepsilon^m = M^m\varepsilon(0)$

The convergence will therefore depend on the conditions under which M^m will approach zero. If M is convergent or $F *_R (M) < 1$, then the error will eventually reach zero.

In practice, we express any matrix A as the same** $A = D - E - F$ where D is diagonal, E and F are strictly lower and upper triangular u^{th} order matrix.

$Ax = b \rightarrow Dx = (E + F)x + b$

a) Method 1 (insert alt. names here)

We define the iteration scheme as:

$$\boxed{x^{m+1} = D^{-1}(E + F)x^m + D^{-1}b}$$

or in component form

$$x_i^{m+1} = -\sum_{f=1, f\neq i}^{n} (\frac{a_{if}}{a_{ii}})x_f^m + \frac{b_i}{a_{ii}}$$

The method will converge if

$S_R(M) < 1$ when $M = D - 1(E + F)$

(b) Method 2

The iteration scheme is defined as

$\boxed{O - E)x^{m+1} = Fx^m + b}$

In component form

$\| \; a_{ii}x_i^{m+1} = -\sum_{f<i} a_{if}x_f^{m+1} - \sum_{f>i} a_{if}x_f^m + b_i$

The iteration matrix M is $M = (D - E)^{-}1F$ and converges if $S_R(M) < 1$

These methods require the storage* of only one vector x since we replace the components of x^m by components of x^{m+1} as seen as they are amplified faster.

(c) Method 3 or (insert names here)

$A = D - E - F$

$D^{-1}A = I - L - U$

$L, U \rightarrow$ strictly lower and triangular matrices.

We define R, a residual vector which is the error vector at any stage of the calculation.

$R^m = D^{-1}b - x^m + Lx^{m+1} + Ux^m$

if we consider the iteration scheme

$(I - L)x^{m+1} = Ux^m + D^{-1}b$

*We now define the iteration scheme to be used

(4) $\boxed{x^{m+1} = x^m + \alpha R^m}$

where α is the called a relaxation parameter.

In general, we can show that $0 < \alpha < 2$ for the method to work.

This is an error correction* method. If $\alpha > 1$ we are overcorrecting the scheme. If $\alpha < 1$ we are under-correcting the scheme. If $\alpha = 1$ similar* to method 2. the iterative scheme can be rewritten

$x^{m+1} = x^m - \alpha x^m + \alpha Lx^{m+1} + \alpha Ux^m + \alpha D^{-1}b$

Solving for x^{m+1} gives:

$(1 - \alpha L)x^{m+1} = (I - \alpha I + \alpha U)x^m + \alpha D^{-1}b$

$x^{m+1} = (I - \alpha L)^{-1}(L - \alpha I + \alpha U)x^m) + (I - \alpha L)^{-1}\alpha D^{-1}b$

OR

$\boxed{x^{m+1} = Mx^m + (I - \alpha L)^{-1}\alpha D^{-1}b}$

with $M = (I - \alpha L)^{-1}(I - \alpha I + aU)$

Method 3 will then converge if $S_R(M) < 1$, we then want to know the optimal α for which $\parallel M_3 \parallel$ is a minimum.

*If we define $M(\alpha) = (I - \alpha L)^{-1}(\alpha U + (1-\alpha)I)$

then $S_R(M) \geq \mid \alpha - 1 \mid$

$\phi(\lambda) = deb(\lambda I - M) = deb(\lambda I - (I - \alpha L)^{-1}(\alpha U + (1 - \alpha)$

$= deb[\lambda(I - \alpha L) - (\alpha U + (1-\alpha)I)]$

since: (1) deb [AB] = deb A deb B (2) deb $(I - \alpha L) = 1$

If $\lambda_i(\alpha)$ are the roots of $\phi(\lambda)$ (and aigeuadles of M) then from polynomials theory:

$(-1)^n \Pi^u \lambda_i(\alpha) = \phi(0)$ but also,

$\phi(0) = (\alpha - 1)^n$ from (5).

This then $\rightarrow$ that Max $\mid \lambda_i \mid \geq \mid \alpha - 1 \mid$

$\longrightarrow$ This method MAY converge ONLY if $0 < \alpha < 2$ but not for other values.

vspace10mm

d. The Steady Solution

As discussed above, the solution of a steady elliptic equation by iteration is analogous to solving a time-dependant problem to an asymptotic* steady state (error = 0 or $\dfrac{\partial \psi}{\partial t} = 0$. Suppose we consider the time dependent for ψ with a diffusion equation source* foem, ζ, and a diffusion term*

$$\boxed{\frac{\partial \psi}{\partial t} = \nabla^2 \psi - \zeta}$$

We are not interested in the significance of the traunests*, but as the solution approaches a steady state, it also approaches the desired solution for the Poisson equation.

In order to solve the equation over a region, we need to know an* the boundary curve C enclosing the region either

(1) The values of ψ

(2) It's normal derivations*

(3)A combination of the two

Applyinf FTCS, (6) can be rewritten as

$$(7)\psi_{i,j}^{n+1} = \psi_{i,j}^n + \frac{\Delta t}{\Delta x^2}(\psi_{i+1,f}^n + \psi_{i-1,f}^n + \psi_{i,f-1}^n + \psi_{i,f+1}^n - 4\psi_{i,j}^n) - \Delta t \zeta_{i,j}$$

Assuming $\Delta x = \Delta y$ for the time being.

The errors for the iteration values $\psi_{i,j}^n$ are then
$\varepsilon_{i,f}^n = \psi_{i,f} - \psi_{i,j,}n$
Exact value or "∞" iterations

(7) can be rewritten as:
$\varepsilon_{i,j}^{n+1} = E_{i,j}^n + \dfrac{\Delta t}{\Delta x^2}[\varepsilon_{i+1,f} + \varepsilon_{i-1,f} + \varepsilon_{i,j+1} + \varepsilon_{i,j-1} - 4\varepsilon_{i,j}]$
which is independent of ζ and therefore the stability properties of (6) are not affected by ζ.

The stability criteria for the diffusion equation in kuro* diversion is $\alpha\dfrac{\Delta t}{\Delta x^2} \leq \dfrac{1}{4}$ (instead of $\dfrac{1}{2}$ for diverted*)

For $\alpha = 1$, the criteria is $\Delta t \leq \dfrac{\Delta x^2}{4}$ since we wish to approach the asymptote solution as fast as possible, we consider the largest $\Delta t = \dfrac{\Delta x^2}{4}$ which gives for (7):

$$\boxed{\psi_{i,f}^{n+1} = \frac{1}{4}[\psi_{i+1,f}^n + \psi_{i-1,f}^n + \psi_{i,f+1}^n + \psi_{i,f-1}^n - \Delta x^2 \zeta_{i,j}]}$$

This is the solution by Method 1 for $\Delta x = \Delta y$. Each ψ^{n+1} is calculated independent of the sequence in i, j) and therefore in a seus* simultaneously.

If we define a mesh ratio of $\delta = \dfrac{\Delta x}{\Delta y}$, the same method gives

(10) $$\boxed{\psi_{i,f}^{n+1} = \frac{1}{2(1+\delta^2)}[\psi_{i+1,f}^n + \psi_{i-1,f}^n + \delta^2\psi_{i,f+1}^n + \delta^2\psi_{i,f-1}^n - \Delta x^2 \zeta_{i,j}]}$$

The of the convergence rate can proceed from the analysis of the error equation.
$E^{k+1} = GE^k$

Fraunkel (1950):
The highest and loudest* wavelength error components* damp most slowly. This* regardless of the wtrial* error distribution, * there components will dominate for k large.

*Now we can improve method 1. Equation (10) is a two-time level equation which requires storage of $\psi_{i.j}^{n+1}$** and ψ^n. If we swap

the $\psi_{i,j}^n$ by new values when ever it is possible in equation (10), then we obtain:

$$(11)\quad \boxed{\psi_{i,f}^{n+1} = \frac{1}{2(1+\delta^2)}[\psi_{i+1,f}^n + \psi_{i-1,j}^{n+1} + \delta^2\psi_{i,j-1}^{n+1} + \delta^2\psi_{i,f+1}^n - \Delta x^2\zeta_{i,j}]}$$

This is the solution by Method 2 and only one storage level is needed. Frankel (1950) showed that asymptotically, k method 1 iterations are worth $2k$ method 1 iterations and only require half the storage.

Now, it was also found that optimum convergence could be achieved by "over-relaxing" or "under-relaxing" depending on whether weightening* seridudals* were* of the same or opposite sign.

Frankel (1950) developed a method of applying over relaxation to method 2. It is called Successive Over-Relaxation or SOR or Method 3.

(11) can be rewritten as the following with the bracketed term manipulated by a relaxation coefficient α

$$(12)\quad \boxed{\psi_{i,f}^{n+1} = \psi_{i,f}^n + \frac{\alpha}{2(1+\delta^2)}[\psi_{i+1,j}^n + \psi_{i-1,j}^{n+1} + \delta^2\psi_{i,f+1}^n + \delta^2\psi_{i,j-1}^{n+1} - \Delta x^2\zeta_{i,f} - 2(1+\delta^2)\psi_{i,f}^n]}$$

We saw in section c) that $1 \leq \alpha \leq 2$ for convergence (over-relaxation). The optimum value α_o depends on the mesh, the shape of the domain, and the tyoe of boundary conditions.

For a *Dirichtel* problem in a rectangular domain of size $(I-1)\Delta x$ by $(\sigma-1)\Delta y$, Frankel (1950) showed that

$\alpha_0 = 2(\dfrac{1-\sqrt{1-\beta}}{\beta})$

with $\beta = \dfrac{\cos\frac{\pi}{I-1} + \delta^2\cos\frac{\pi}{J-1}}{1+\delta^2})^2$

Because of its simplicity and effectiveness, the sole this sole method has been the most popular of the iterative methods for solving the Poissen* equation in complicated fluid dynamics problems. However, this method takes considerable computer time, and has been now replaced by faster, accurate direct methods in most problems.

The SOR method is very flexible and can be used under a wide range of conditions, including irregular boundaries, inr*** points....

6.2 Direct methods

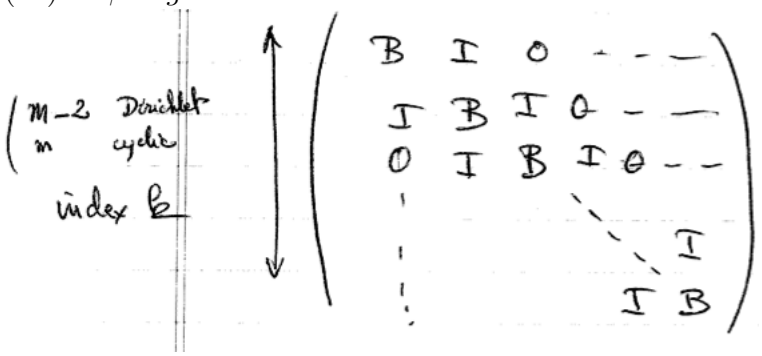
There is another class of solvers, the direct methods, that treat the finite difference equations as a large linear system and employ same tools from linear algebra to operate on the matrix of the finite difference coefficients. The methods take advantage of the sparseness and regular structure of the coefficient matrices to minimize storage requirements and operation counts*. Although they require more storage than the iterative methods, they require fewer operations.

Let's now consider the Posson** equation on a rectangle

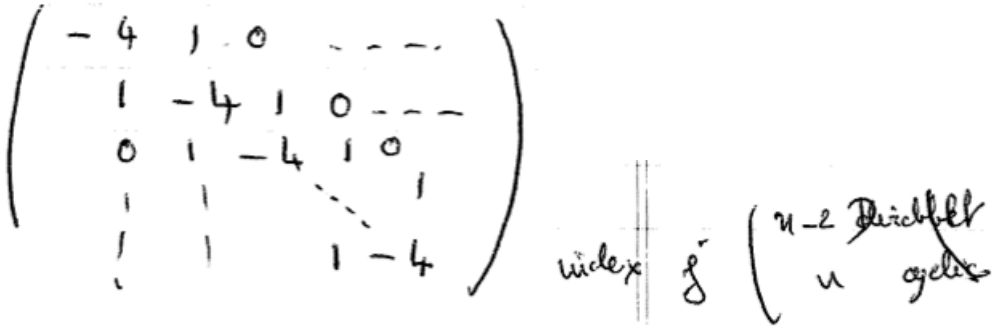
(13) $\nabla^2\psi = \zeta$ $\boxed{n \times m \text{ domain}}$

As described at the beginning of this chapter, the FD form can be expressed as

(14) $M\psi = g$ in matrix form where M is of the form



and B of the form



→ The vectors ψ and g are vectors of subvectors

$$\begin{matrix} \phi & k \\ g & k \end{matrix}$$

of solution values along the k^{th} vector of the mesh.

(correspond to the Si and Li of the beginning of the chapter)

The matrix M is very sparse did cheers* a very regular black structure. All the direct methods take advantage of this structure. Irregularly shaped boundaries destroy this structure and thus prohibit the use of direct method.

Generalized form of Mockey's method or Fouanier Transform method***

We define the Matrix Q such that $Q^{-1}BQ = \Lambda$

where $\Lambda = \begin{matrix} \lambda_i & o \\ o & \lambda_n \end{matrix}$. Thus Q is the matrix ehose colius are the eigeuvectos of B and theλ_i, the corresponding eigeuvalues.

We can also define the transform

$\Phi_k = Q^{-1}\phi k$; $G_k = Q^{-1}g_k$

For the k^{th} now of (14), we have

$I\phi_{k-1} + B\phi_k + I\phi_{k+1} = gk$

Multiply by Q^{-1} and asig (15)

$I\Phi_{k-1} + Q^{-1}BQ\Phi_k + I\Phi_{k+1} = G_k$

(Where $Q^{-1}BQ\Phi_k$ is the eiguvalues)

Then for each value of λ_ν, eigalue, we get a triagonal system

$\Phi_{\nu,k-1} + \lambda_\nu\Phi_{\nu,k} + \Phi_{\nu,k+1} = G_\nu, k$

Each of the triagonal systems can be easily solved by Gaussion eliminators. The solution is then given by the uieux* transformation:

(17) $\phi_k = Q\Phi_k$

*In particular, if the Fourier transform of (15) are defined a:

$$\phi_{j,k} = \sum_\nu \Phi_{\nu,k} e^{i\frac{2\pi\nu f}{n}}$$

$$g_{j,k} = \sum_\nu G_{\nu,k} e^{i\frac{2\pi\nu f}{n}}$$

FFT Transform

and are substituted in the FD form of the Poisson's equation, then for the part j, k we get:

(18) $$\sum_\nu \Phi_{\nu,k-1} e^{i\frac{2\pi\nu j}{n}} + \sum_\nu \Phi_{\nu,k} e^{i\frac{2\pi\nu(f-1)}{n}} + e^{i\frac{2\nu(f+1)}{n}} + \sum_\nu \Phi_{\nu,k+1} e^{i\frac{2\pi\nu f}{n}} - 4\sum_\nu \Phi_{\nu,k} e^{i\frac{n\pi\nu f}{n}} = \sum_\nu G_{\nu,k} e^{i\frac{2\pi\nu j}{n}}$$

Due to the orthigonality of the $e^{i\frac{n\pi\nu j}{n}}$ terms, we have n independent tridiagonal systems for the $\Phi_{\nu,k}$

(19) $\boxed{\Phi_{\nu,k-1} + (-4 + e^{\frac{i2\pi\nu}{n}} + e^{\frac{-in\pi\nu}{n}})\Phi_{\nu,k} + \Phi_{\nu,k+1} = G_{\nu,k}}$

so that $\lambda_\nu = -4 + 2\cos(\frac{2\pi\nu}{n})$.

Once the triidiogonal* system are solved the solution is recovered *usicky* a back-transform or inverse FFT.

b) Cydic reduction

Mockney* rec....ded* for efficiency to reduce the order of the drichigonal systems by applying a** or more passes of cydic reduction (or odd-even reduction) before performing the Fourier Transform.

(20):
$I\phi_{k-2} + B\phi_{k-1} + I\phi_k = g_{k-1}$
$I\phi_{k-1} + B\phi_k + I\phi_{k+1} = g_k$
$I\phi_k + B\phi_{k+1} + I\phi_{k+2} = g_{k+1}$
Multiply the second by $-B$ and *add the threer* gives

(21) $\boxed{I\phi_{k-2} + (2I - B^2)\phi_k + I\phi_{k+2} = g_{k-1} + g_{k+1} - B_{gk}}$

The system is now reduced to only even-numbered rows. At this point, we can apply the Fourier Transformation method as at* the even rows and we use (20) to get the odd rows. This is referred to as Hockney's method.

One does not have to perform the Fouirner Transformation. If m is a power of 2, $m = 2^{\rho+1}$, the reduction process can be performed until we are left with only one row of numbers to solve for.

We can define the recursions
$B^{p+1} = 2I - (B^{(p)})^2$
$g_k^{(p+1)} = g_{k-2^p}^{(p)} + g_{k+2^p}^{(p)} - B^{(p)}g_k$
for : $k = 2^p, m - 2^p$, every 2^p, and $p = 1, ..., \varphi$
After φ reductions, we are left with:

(22) $\boxed{I\phi_o + B^{(\rho)}\phi_{2\rho} + I\phi_m = g_{2\rho}^{(\rho)}}$

ϕ_o and ϕ_m are known from the boundary conditions $\rightarrow$ $\phi_{2\rho}$ can be found easily.

Advantages:

• Every $g_k^{(p+1)}$ can overwrite the previous $g_k^{(p)} \rightarrow$ Little storage is needed

• Each $B^{(p)}$ is a polynomial of B of order 2^p and can be re-expressed* as a sequence of tridiagonal matrices $B^p = A_1 A_2 A_3 ... A_{2p}$

Disadvantages:

• The computation of $g^{(p)}$ is subject to severe round off errors. (Buzbe* et. al., 1970)* $\rightarrow$ unstable for many values of p*.

there are various ways to stabilize the method ("Bumuam variants an eydic reduction" or "the Bunenman algorithms"). The algorithms are mathematically identical to the previous derivations, but are not prone to round off errors.

c) Block Method

Used mostly for the general forms of an elliptic equation which can not be solved by the two previous methods and is too time consuming if done by iterative methods.

General form of an elliptic equation

$(23) a(x,y,)\phi_{xx} + b(x,y)\phi_{xy} + c(x,y,)\phi_{yy} + d(x,y,)\phi_x + e(x,y,)\phi_y + f(x,y)\phi = g(x,y)$

In finite-difference form, the matrices are:

$$M\phi = \begin{bmatrix} A_1 & C_1 & & & \\ B_2 & A_2 & C_2 & & \\ & B_3 & A_3 & C_3 & \\ & & \ddots & \ddots & \\ & & & B_m & A_m \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_m \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_m \end{bmatrix}$$

with

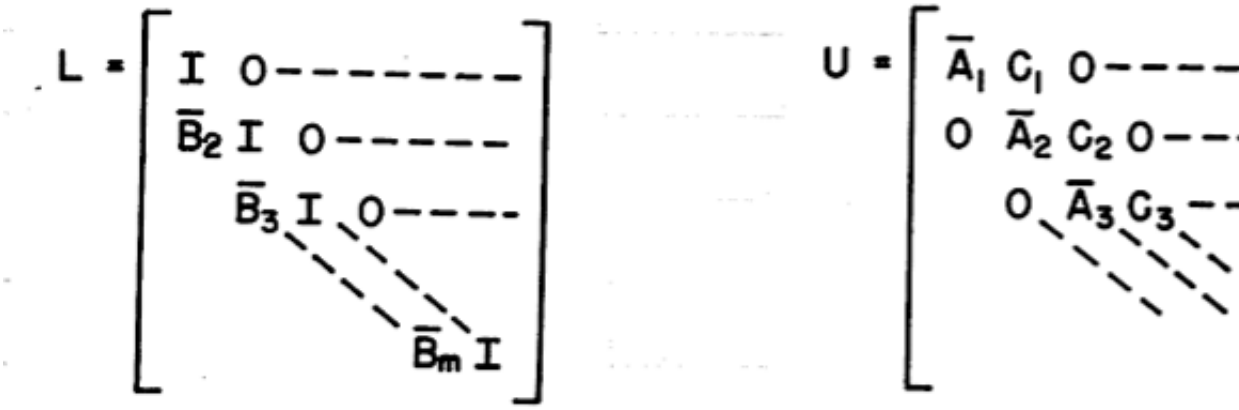
$$A_K = \begin{bmatrix} f_{1,K} - 2\left(\frac{a_{1,K}}{\Delta X^2} + \frac{C_{1,K}}{2\Delta X}\right) & \frac{a_{1,K}}{\Delta X^2} + \frac{d_{1,K}}{2\Delta X} & & \\ \frac{a_{2,K}}{\Delta X^2} - \frac{d_{2,K}}{2\Delta X} & f_{2,K} - 2\left(\frac{a_{2,K}}{\Delta X^2} + \frac{C_{2,K}}{\Delta Y^2}\right) & & \\ & & & \\ & & & \end{bmatrix}_{n \times n}$$

$$B_K = \begin{bmatrix} \frac{C_{1,K}}{\Delta Y^2} - \frac{e_{1,K}}{2\Delta Y} & \frac{-b_{1,K}}{4\Delta X \Delta Y} & & \\ \frac{b_{2,K}}{4\Delta X \Delta Y} & & & \\ & & & \\ & & & \end{bmatrix}_{n \times n}$$

$$C_K = \begin{bmatrix} \frac{C_{1,K}}{\Delta Y^2} + \frac{e_{1,K}}{2\Delta Y} & \frac{b_{1,K}}{4\Delta X \Delta Y} & & \\ \frac{-b_{2,K}}{4\Delta X \Delta Y} & & & \\ & & & \\ & & & \end{bmatrix}_{n \times n}$$

$$\phi_K = \begin{bmatrix} \phi_{1,K} \\ \phi_{2,K} \\ \vdots \\ \phi_{n,K} \end{bmatrix} \qquad g_K = \begin{bmatrix} g_{1,K} \\ g_{2,K} \\ \vdots \\ g_{n,K} \end{bmatrix}$$

M can be factored. $M = LU$
L = Lower Triangular
U = Upper ———



$\vec{A_k}$ and $\vec{B_k}$ are given by:
(24)
$\vec{A_{k-1}}\vec{B_k} - *B_k$
$\vec{A_k} = A_k - \vec{B_k}C_{k-1}$
where $\vec{A_1} = A_1$

On a vector computer*, these recursions* can be preformed very efficently since A_k, B_k, and C_k are tridiagonal. To obtain ϕ_k, we first solve $L\phi = g$ (forward sweep) by

(25) $\phi_k^\alpha = g_k - \vec{B_k}\phi_{k-1} \qquad (\phi_1 = g_1) \qquad k = 2m$

Then $U\phi = \phi$ is solved (backward sweep)

(26) $A_m\phi_m = \phi_m^\alpha$
$A_k\phi_k = \phi_k^\alpha - C_k\phi_{k+1}^\alpha \qquad k = m-1, 1$

This method can handle very general boundary conditions, but are as before restricted to rectangular domain. Very fast as a vector machine, but is machine specific. *Camed salroutunes* are *available.