農業工程學報 第 44 卷第 4 期
中華民國 87 年 12 月出版

Journal of Chinese Agricultural Engineering
Vol. 44, No. 4, December 1998

曲線座標系下通用指標法於有限差分法之應用

Application of a General Index Method to Finite Differences in the Curvilinear Coordinates

農業工程研究中心副研究員兼組長

蔡存孝

Tswn-Syau Tsay

國立成功大學土木系副教授

朱聖浩

Shen-Haw Ju

摘 要

本研究中，爲儲存對稱型及非對稱型超大型矩陣通用指標已被開發，且該法被應用於有限差分法在曲線座標系下，利用共軛梯度類似法求解偏微分方程。本通用指標法特別強調有限差分法中整體矩陣的型態。本法之優點計有(1)可應用於解各種控制方程式及邊界條件，(2)可適用於高階之展開估算，(3)與問題之次元無關，(4)可有效率的解複雜問題及非對稱型矩陣，(5)可適用於解一般型之超大型矩陣，(6)由於僅儲存非零之矩陣元素，因此，在最耗時的矩陣相乘步驟顯得非常有效率，(7)適用於現階段已開發之解矩陣方法。本研究將通用指標法結合共軛梯度類似法應用於二維及三維之地下水丘數值模擬，結果發現該方法比傳統方法有效率。

關鍵詞：地下水丘，共軛梯度類似法，指標法。

ABSTRACT

A general index method for storing un-symmetric and symmetric, sparse matrices was developed and applied to finite differences in the generalized curvilinear coordinates for solving differential equations using conjugate-gradient-like methods. In this general index method, forming the global matrix for finite differences is emphasized. The advantages of this general index method are (1) adaptable to various governing and boundary conditions, (2) flexible for higher order approximation, (3) independent of problem dimension, (4) efficient for complex problems when global matrix is not symmetric, (5) convenient for general sparse matrices, (6) computationally efficient in the most time consuming procedure of matrix multiplication (since only non-zero terms are stored), and (7) flexible and applicable to any developed matrix solver. In this paper, numerical simulation of two- and three-dimensional ground water mounding were used to

illustrate and results were compared, which indicated that the general index method is much more efficient than the traditional numerical procedure of the finite difference method.

Keywords: Ground water mounding, Conjugate gradient like method, Index method.

1. INTRODUCTION

In numerical methods, the governing partial differential equation, subject to boundary and initial conditions is solved by dividing the domain into a collection of points or elemental cells, discretizing the equation to a set of algebraic equations on this collection, and solving the resulting linear system of equations. Traditional element-by-element methods (e.g. SOR) are preferred since it is simple and memory efficient. Because the element-by-element methods in finite differences do not require storing the global coefficient, they are rather simple for computations in the Cartesian coordinates.

In recent years, conjugate-gradient-like methods were reported to be computationally efficient (Bercobier and Rosenthal, 1986; Suetomi and Sekimoto, 1989). Among them, using incomplete Choleski decomposition as preconditioner was reported to be the best (Obeysekare, 1987). Nevertheless, this method is not valid if the global coefficient matrix is not stored.

Finite differences in the generalized curvi-linear-coordinates is useful for modeling moving and free boundary problems (Crank, 1984). Required memory of element-by-element methods can be very small as the global coefficient matrix is not formed. Hence numerous calculation due to the algorithm of simulating moving boundary problems has to be performed. This means that each coefficient of the global matrix must be re-calculated in each iteration loop of locating the free surface; thus, the required computation time is intense. For example, for SOR, usually it will be about 10 to 20 times slower than the method if the global coefficient matrix was stored.

Based upon the two reasons stated above, a general index method for storing the global coefficient matrix effectively is proposed. This general index method coupled with the conjugate-gradient-like methods preconditioned by incomplete Choleski method can save much computation time for simulations of ground water mounding with an moving and free surface boundary algorithm.

2. Description of the Index Method

To store an un-symmetric matrix **A**, seven arrays are required: four global index arrays **LL**, **LU**, **LLL**, **LLU**; one global coefficient array **D** to store the diagonal terms; and two global coefficient arrays **TSU**, **TSL** to store non-zero terms in the upper triangle and lower triangle, respectively.

2.1 Numbering nodes and the non-zero term locations in global matrix

For sky-line method, numbering nodes is a very important step since the CPU time and computer memory are highly dependent on the arrangement of the nodal number. A renumbering scheme is usually used to find a minimum, banded global matrix (George and Liu, 1981). However, the nodal arrangement is not as important for conjugate-gradient-like methods as for sky-line method since the number of non-zero terms in this global matrix will not be changed when the nodal number is changed. Hence, the simplest way of numbering nodes in the general index method is by

natural order. Suppose the total number of nodes in the x-direction and the y-direction is NX and NY, respectively; then, the node numbers for a two-dimensional problem (numbered by natural order) are

$$II = I + (J-1)\times NX \quad \text{.....(1)}$$

Likewise, the node numbers of a three-dimensional problem are

$$II = I + (J-1)\times NX + (K-1)\times NX\times NY \quad \text{...(2)}$$

where I, J, and K are the indices in the X, Y, and Z coordinates. For example, a two-dimensional domain with 20 grid points can be numbered as is indicated in Fig. 1:

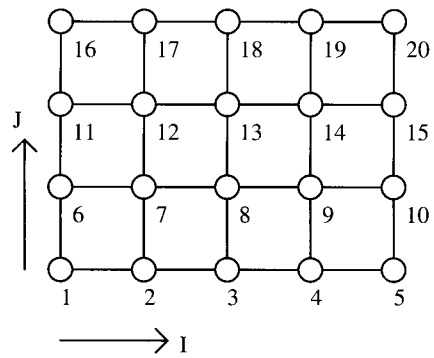

Fig. 1 A uniform 5 by 4 grid domain

A domain of NX by NY nodes implies a system of NX by NY linear equations, constructed by each node's governing equation or boundary condition. For example, the node (I, J), governed by the Laplace's equation for a two-dimensional problem and discretized by the standard, second order accurate, five-point approximation in the finite difference method (Fig. 2), has the algebraic relation

$$4\phi_{I,J} - \phi_{I+1,J} - \phi_{I-1,J} - \phi_{I,J+1} - \phi_{I,J-1} = 0, \quad \text{.....(3)}$$

assuming the grid spacing is uniform.

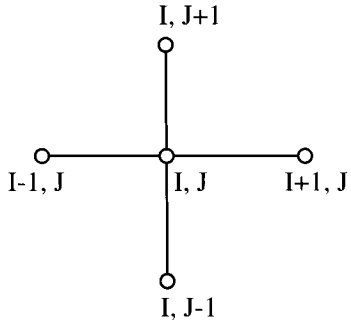

Fig. 2 Standard second order accuracy five-point Laplacian

Defining a nodal index array **MPQR** for the locations of the five grid points in (3) gives

$$MPQR(1) \equiv NODE(I,J-1) = I + (J-2)\times NX \text{ ..(4a)}$$

$$MPQR(2) \equiv NODE(I-1,J) = I - 1 + (J-1)\times NX \text{ (4b)}$$

$$MPQR(3) \equiv NODE(I,J) = I + (J-1)\times NX \text{ .....(4c)}$$

$$MPQR(4) \equiv NODE(I+1,J) = I + 1 + (J-1)\times NX \text{ (4d)}$$

$$MPQR(5) \equiv NODE(I,J+1) = I + (J)\times NX \text{ .....(4e)}$$

Hence for any node, the nodal array **MPQR**, Eqs. (4a) - (4e), determines the nodal points involved in the governing equation for that node. For NODE (2, 3), node 12 in Fig. 1, the nodal array **MPQR** shows that node 12 is related to nodes 7, 11, 13 and 17. It also indicates the 12^{th} equation implicates a non-zero diagonal term and other non-zero terms of this equation are on row 12 in the global matrix **A** and they are located in columns 7, 11, 13, 17. This process of identifying nodal points coefficients ends when MPQR(3) is equal to II (Eq. (1)), a diagonal term in the global matrix. The **MPQR** value is < or > II the non-zero term indicates the non-zero term locates in the lower or upper triangular global matrix. This index array is accompanied with a nodal coefficient array **XE** to indicate the coefficient magnitude. For the example in Fig. 1,

XE(1) = -1.0

XE(2) = -1.0

XE(3) = 4.0

XE(4) = -1.0

XE(5) = -1.0

In programming, the dimension size of **MPQR** is determined by the maximum number of related nodes for any node. For a two-dimensional rectangular mesh problem with NX nodes in X direction and NY in Y direction, the boundary conditions involve 3 points and the governing equation involves 5 points (Fig. 2); hence, the dimension size of **MPQR** is 5 and the dimension size of **XE** is 5 as well. Finally, the location of **XE** in the global matrix **A** will be determined by applying the global index arrays as will be described in the following section.

2.2 Global index arrays and global matrices for conjugate-gradient-like methods (CGL)

Four global index arrays **LL, LU, LLL** and **LLU** are used to store locations of non-zero nodes in an un-symmetric matrix. For a symmetric matrix, **LL=LU** and **LLL=LLU**. The non-zero terms of the upper and lower triangular matrices are numbered by natural order. Array **LU** stores the last non-zero term number of each column of the upper triangular matrix, and array **LL** stores the last non-zero term of each row of the lower triangular matrix. The row number of the non-zero terms in the upper triangular matrix is stored in array **LLU**, and the column number of the non-zero terms in the lower triangular matrix is stored in array **LLL**. The magnitude of the non-zero terms of the global matrix are stored in the global coefficient arrays: **TSU**; **TSL**; and **D**. Consider the following matrix in Fig. 3, the global index arrays and the global coefficient arrays are shown in Table 1a - 1c. The size of arrays **LU**, **LL** and **D** are determined by the problem unknowns. However, the size of arrays **LLU** and **TSU** and of **LLL** and **TSL** are determined by the largest number in LU(I) and LL(I) respectively. For the example shown in Table 1a - 1c, the size of arrays **LLU** and **TSU** is 9, and the size of arrays **LLL** and **TSL** is 8. These four global index arrays can be evaluated by the nodal index arrays **MPQR**. **LU** and **LL** can be determined by the procedure in Table 2, and **LLU** and **LLL** are determined by Table 3.

$$\begin{bmatrix} d1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & d2 & a1 & a2 & 0 & a5 & 0 \\ b1 & 0 & d3 & 0 & a3 & 0 & a7 \\ 0 & b2 & b3 & d4 & a4 & a6 & a8 \\ 0 & 0 & b4 & 0 & d5 & 0 & a9 \\ 0 & b5 & 0 & b6 & b7 & d6 & 0 \\ 0 & b8 & 0 & 0 & 0 & 0 & d7 \end{bmatrix}$$

Fig. 3 Matrix example

Table 1a Global index arrays LU and LL and diagonal global coefficient array D for Fig. 3

I	1	2	3	4	5	6	7
LU(I)	0	0	1	2	4	6	9
LL(I)	0	0	1	3	4	7	8
D(I)	d1	d2	d3	d4	d5	d6	d7

Table 1b Global index arrays LLU and global coefficient array TSU for Fig. 3

I'=LU(I)	1	2	3	4	5	6	7	8	9
LLU((I'))	2	2	3	4	2	4	3	4	5
TSU((I'))	a1	a2	a3	a4	a5	a6	a7	a8	a9

Table 1c Global index arrays LLL and global coefficient array TSL for Fig. 3

I" = LL(I)	1	2	3	4	5	6	7	8
LLL((I"))	1	2	3	3	2	4	5	2
TSL((I"))	b1	b2	b3	b4	b5	b6	b7	b8

Table 2 Procedure to determine LU and LL by MPQR

```
for I=1 to NX
    for J=1 to NY {sweep all nodes}
        II=(J-1) × NX + I {Eq. (1)}
        call subroutine for nodal array MPQR at node (I, J) {Eq. (4a) - (4e)}
        for k=1 to mdf {mdf is the number of involved nodes at node (I, J)}
            JJ = MPQR(k) {a temporary number}
            if JJ > II then LU(JJ) = LU(JJ) + 1 {the upper triangular global matrix}
            if JJ < II then LL(JJ) = LL(JJ) + 1 {the lower triangular global matrix}
        next k
    next J
next I
for IJ=2 to NX × NY {summing up number of global index arrays}
    LL(IJ) = LL(IJ) + LL(IJ-1)
    LU(IJ) = LU(IJ) + LU(IJ-1)
next IJ
```

Table 3 Procedure to determine LLU and LLL by MPQR

```
for I=1 to NX
    for J=1 to NY
        II=(J-1) × NX + I
        call subroutine - nodal index array MPQR at node (I, J)
        for k=1 to mdf {mdf is the number of involved nodes at node (I, J)}
            JJ = MPQR(k)
            if JJ > II then
                LU(JJ) = LU(JJ) + 1
                LLU(LU(JJ)) = II {row number is II for column LU(JJ)}
            if JJ < II then
                LL(JJ) = LL(JJ) + 1
                LLL(LL(II)) = JJ {column number is JJ for row LL(II)}
            endif
        next k
    next J
next I
sort global index arrays LLU and LLL
```

The global matrix is found by using the nodal index arrays and the global index arrays. Using these index arrays, one can efficiently direct magnitude of the nodal coefficient array **XE** to the global matrix **A** (Table 4), evaluate the incomplete Choleski decomposition (Appendix I and II), and conduct matrix multiplication in conjugate-gradient-like methods (Table 5). The procedure of evaluating incomplete Choleski decomposition requires an extra array (**id**). This array stores the non-zero term row (or column) number from **LLU** (or **LLL**). This procedure is very similar to a forward-backward substitution.

Table 4 Procedure of assigning nodal coefficient array into global matrix

```
II=(J-1) × NY + I {Eq. (1)}
for IJ=1 to NX × NY {sweep all nodes}
  jj = MPQR(IJ) {nodal index}
  if (jj = II) then
   TS(II)=XE(IJ) {the diagonal terms}
  elseif (II > IJ) then {binary search process}
    j1 = LL(II-1)+1
    j2 = LL(II)
    kk=(j1+j2)/2
    for k=1 to 20
     if (LLL(kk) = jj) then
      TSL(kk)=XE(IJ) {the lower triangular terms}
       elseif (LLL(kk) > jj) then
          j2 = kk - 1
       else
          j1 = kk + 1
       endif
        kk = (j1 + j2)/2
    next k
  else
    j1 = LU(II-1)+1
    j2 = LU(II)
    kk=(j1+j2)/2
    for k=1 to 20
     if (LLU(kk) = jj) then
      TSU(kk)=XE(IJ) {the upper triangular terms}
       elseif (LLU(kk) > jj) then
          j2 = kk - 1
       else
          j1 = kk + 1
       endif
       kk = (j1 + j2)/2
    next k
  endif
next IJ
```

Table 5 Procedure of matrix multiplication using general index method

```
c**    {C}=[A]{B} {[A] is composed of {TSU},
       {TSL} and {TS}}
for i = 1 to NX × NY {the diagonal term}
   C(i)=TS(i) × B(i)
next IJ
for i = 1 to NX × NY {the upper triangular matrix}
   ii = LU(i-1)+1
   jj = LU(i)
   aa = B(i)
   for j = ii to jj
      C(LLU(j)) = C(LLU(j)) + TSU(j) × aa
   next j
next i
for i = 1 to NX × NY {the lower triangular matrix}
   ii = LL(i-1)+1
   jj = LL(i)
   aa = C(i)
   for j = ii to jj
      aa = aa + TSL(j) × B(LLL(j))
   next j
   C(i) = aa
next i
```

2.3 Global index array and global matrix for sky-line method (SK)

For sky-line method, only the elements below the first non-zero term at each column are stored. A transformation array, **LS**, is used to store the number of elements along the diagonal. An example is shown in Fig.4. Usually, we can use a one-dimensional array, **F**, to store the global matrix. The relationship between the full two-dimensional symmetric array, **G**, and the sky-line one-dimensional symmetric array, **F**, is

$$G(i,j) = F(LS(j)+i-j) \quad i <= j \qquad (5)$$

J	1	2	3	4	5	6	7	8	9	10	
LS(j)	1	3	5	9	12	15	21	23	29	32	i

$$\begin{bmatrix} 1 & 2 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 3 & 4 & 7 & 0 & 0 & 16 & 0 & 0 & 0 \\ & & 5 & 8 & 10 & 0 & 17 & 0 & 0 & 0 \\ & & & 9 & 11 & 13 & 18 & 0 & 24 & 0 \\ & & & & 12 & 14 & 19 & 0 & 25 & 0 \\ & & & & & 15 & 20 & 0 & 26 & 0 \\ & & & & & & 21 & 22 & 27 & 0 \\ & & & & & & & 23 & 28 & 30 \\ & & & & & & & & 29 & 31 \\ & & & & & & & & & 32 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{matrix}$$

Fig. 4 Sky-line method.

3. Required Memory for index method

With double precision a real number takes 8 bytes of memory and a integer number takes 4 bytes of memory. The required memory of the general index method for NX by NY nodes can be approximated by

LU and **LL**	$==> 4\times NX\times NY\times 2$	bytes
LLU and LLL	$==> 4\times NX\times NY\times 4\times 2$	bytes
TSU and TSL	$==> 8\times NX\times NY\times 4\times 2$	bytes
D	$==> 8\times NX\times NY$	bytes

The general index method requires $112\times(NX\times NY)$ bytes of memory to store the non-zero terms of the global matrix. For sky-line method, the required memory for NX by NY nodes can be approximated by a banded matrix with a bandwidth of NX, so the total required memory of the global matrix and the global index vector **LS** is

$$8\times NX\times NX\times NY + NX\times NY\times 4 = (8\times NX + 4)(NX\times NY) \text{ bytes.}$$

The ratio of the required memory of general index method for SK and the general index method for CGL increases with increasing NX as is shown in Fig. 5. The required memory of a problem with NX=100 is approximated reduced by more than 7 times when index method for CGL is used.

4. Numerical experiments

The general index method for conjugate-gradient-like methods and sky-line method were applied to two- and three-dimensional ground water mounding problems and their results were compared. Ground water mounding is the rise of the water table above the regional level in an area of an aquifer in order to provide sufficient head to distribute the water supplied by a localized source (e.g., recharge from a spreading basin, waste water discharge system, irrigation, flooding, leakage from lagoon or land fill) to that area. The shape and height of the mound depends on many factors including the source geometry, rate of supply of recharge, the geologic structure, hydraulic conductivity and its variations, flow/head controls

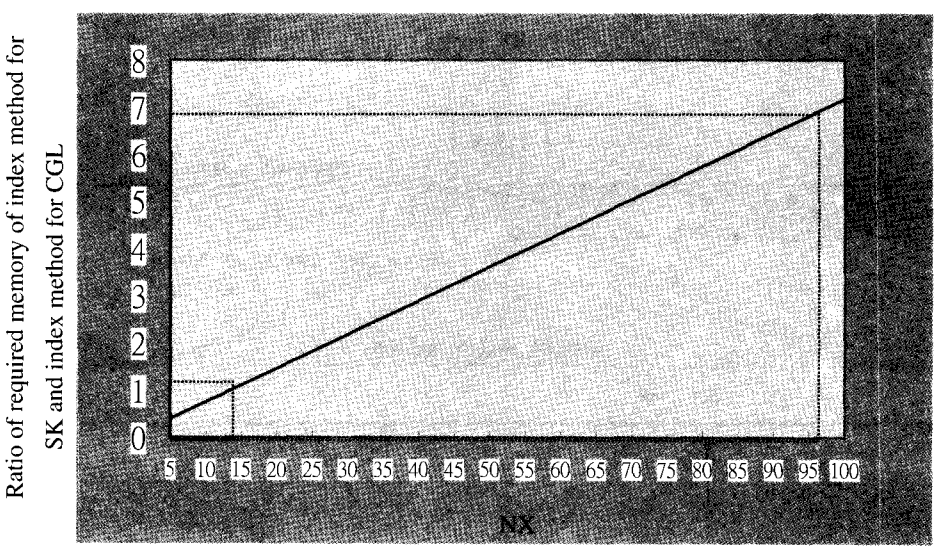

Fig. 5 Ratio of required memory of index method for SK and index method for CGL

and flow in the aquifer. The flow pattern due to mounding transports and mixes substances, introduced by the recharge, with the ground water.uFor this moving boundary problem, an iteration scheme (Appendix III) is used to locate the free surface due to recharge. In computations, the locations of the non-zero terms for the system of linear equations are the same for each iteration, which means that process of constructing the global index arrays **LL**, **LU**, **LLL**, and **LLU** is executed only once at the beginning of this iteration procedure. Non-zero terms of the global matrix have to be computed in each iteration since the free surface is changed during this iteration scheme.

Fig. 6 is a sketch of the two-dimensional flow field and boundary conditions for steady recharge and flow through a two-layered aquifer with a horizontal, impervious bottom and finite, constant head lateral boundaries.

The governing equation, derived by substituting Darcy's law ($\underline{q} = -\underline{\underline{K}} \cdot \nabla\phi$) into the continuity equation ($\nabla \cdot \underline{q} = 0$ for an incompressible fluid and rigid soil skeleton), is

$\nabla \cdot (\underline{\underline{K}} \cdot \nabla\phi) = 0$, where ϕ = piezometric head and $\underline{\underline{K}}$ = hydraulic conductivity tensor.

The boundary conditions are:

(1) Dirichlet boundary condition on x = 0 and x = L (constant head);

(2) Neumann boundary condition on the impervious bottom (no flow);

(3) the free surface boundary includes

(a) the pressure is zero, and

$$\text{(b)}\ \frac{\partial F}{\partial t} + \frac{\underline{q} \cdot \nabla F}{\theta} = \begin{cases} \dfrac{\underline{N} \cdot \nabla F}{\theta} & \text{for area receiving recharge} \\ 0 & \text{for area not receiving recharge} \end{cases}$$

where F the difference between potential head and elevation head, $\underline{N}$ is recharge rate and θ is porosity. Assuming the thickness of the lower layer is H_L, the internal boundary conditions on the vertical location of H_L satisfies equivalent pressure head on either aquifer and equivalent flux through this "internal" boundary between the lower and the upper aquifer.

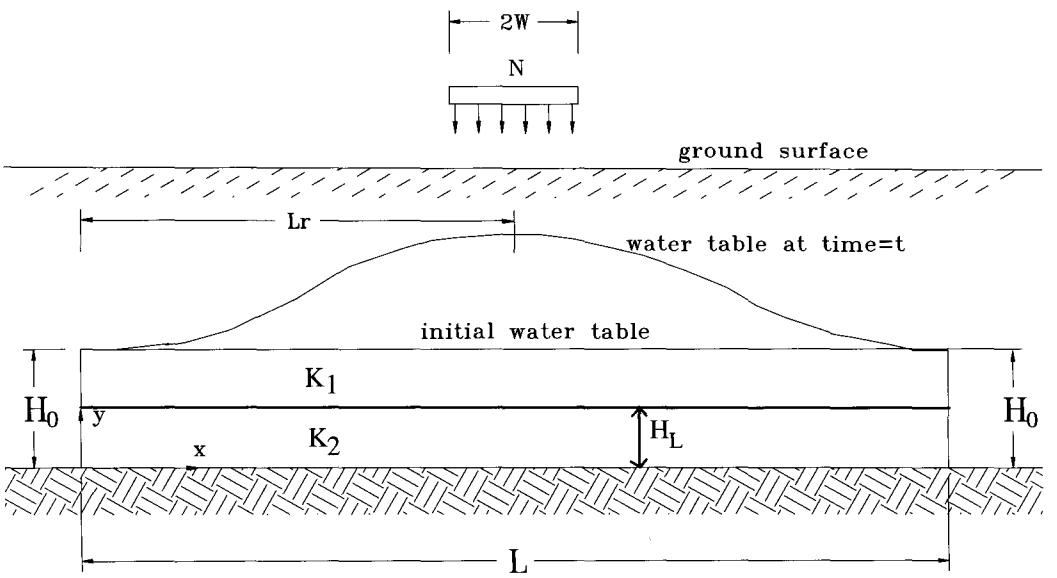

Fig. 6 Configuration of ground water mounding model

In order to estimate the ground water mound height, computations in the generalized curvilinear coordinates was applied. The governing equation ($\nabla \cdot (\underline{\underline{K}} \cdot \nabla \phi) = 0$) in the generalized curvilinear coordinates is expressed (Tsay, 1995)

$$
\begin{aligned}
&-\sum_k \sum_k \sum_m \sum_n \sum_l \sum_j \sum_i K_{ij} \frac{\partial x_j}{\partial \xi^l} \frac{\partial x_i}{\partial \xi^n} g^{lm} g^{kk} \Gamma^n_{kk} \phi_{\xi^m} \\
&+\sum_k \sum_m \sum_n \sum_l \sum_j \sum_i K_{ij} g^{lm} g^{kn} \left(\frac{\partial x_j}{\partial \xi^l} \frac{\partial x_i}{\partial \xi^n} \right)_{\xi^k} \phi_{\xi^m} \\
&+\sum_k \sum_m \sum_n \sum_l \sum_j \sum_i K_{ij} \frac{\partial x_j}{\partial \xi^l} \frac{\partial x_i}{\partial \xi^n} g^{kn} \left(g^{lm} \right)_{\xi^k} \phi_{\xi^m} \\
&+\sum_k \sum_m \sum_n \sum_l \sum_j \sum_i K_{ij} \frac{\partial x_j}{\partial \xi^l} \frac{\partial x_i}{\partial \xi^n} g^{kn} g^{lm} \phi_{\xi^m \xi^k} = 0.
\end{aligned}
$$

where ξ, η, and ζ are axes of the generalized curvilinear coordinates, Γ^k_{ij} is the Christoffel tensor, and g^{ij} is contravariant metric tensor.

Computations of the steady state ground water mound location by different methods were compared by the number of iterations for various schemes to solve the system of linear equations to achieve an error < 10^{-6} and by CPU time using different numbers of grid points. The procedure requires an initial guess for the water table elevation, computation of ϕ in the flow domain, and an iteration scheme.

4.1 Two-dimensional example

This example had a recharge width of 10 m with its center located 100 m from the left boundary and steadily supplying water at a rate (N) of 0.001 cm/sec to an aquifer with 40 m initial saturated thickness (H_0) and 200 m long (L). The hydraulic conductivity of the upper aquifer (K_1) was 0.002 cm/sec and lower aquifer (K_2) is 0.003 cm/sec, respectively. Three grids for the problem were used: 22 by 22, 57 by 57, and 85 by 85. All computations using double precision were carried out on a HPRISC 9000/735 machine, with double precision and a convergence criteria of 10^{-6}.

Four iterative schemes and the sky-line method were compared for the above example:

SL: sky-line

ICG: preconditioned conjugate gradient by incomplete Choleski decomposition

IGCR(n): preconditioned generalized conjugate residual by incomplete Choleski decomposition with inner loop = n

DGCR(n): preconditioned generalized conjugate residual by diagonal scaling with inner loop = n

SOR: point successive-over-relaxation

Fig. 7 shows number of iterations vs. their error by various iterative schemes for a 57 by 57 grid problem. It is indicated that IGCR(30) is the best method from error convergence. The figure also shows that IGCR(10) and IGCR(20) were able to solve this non-symmetric matrix while other methods were not suitable (i.e., SOR, GCR(30), DGCR(10), DGCR(20) and DRCR(30)) or were divergent (i.e., ICG and CG) as CG and ICG only works for solving systems of linear equations with symmetric coefficient matrices. It is evident that the application of preconditioner reduces the iterations (comparing GCR(n), DGCR(n) and IGCR(n)). In this case, using incomplete Choleski decomposition showed a great improvement.

Comparing IGCR(n) and its extensions, Fig. 8 indicates a larger number of inner loop, n, usually reduces the iterations for error convergence. However, a larger inner loop takes more memory. The memory increases proportionally to n by a constant equivalent to the number of unknowns. Hence, even increasing n improves the rate of convergence, an inner loop number larger than the maximum efficient number does not improve computational efficiency but a waste of memory.

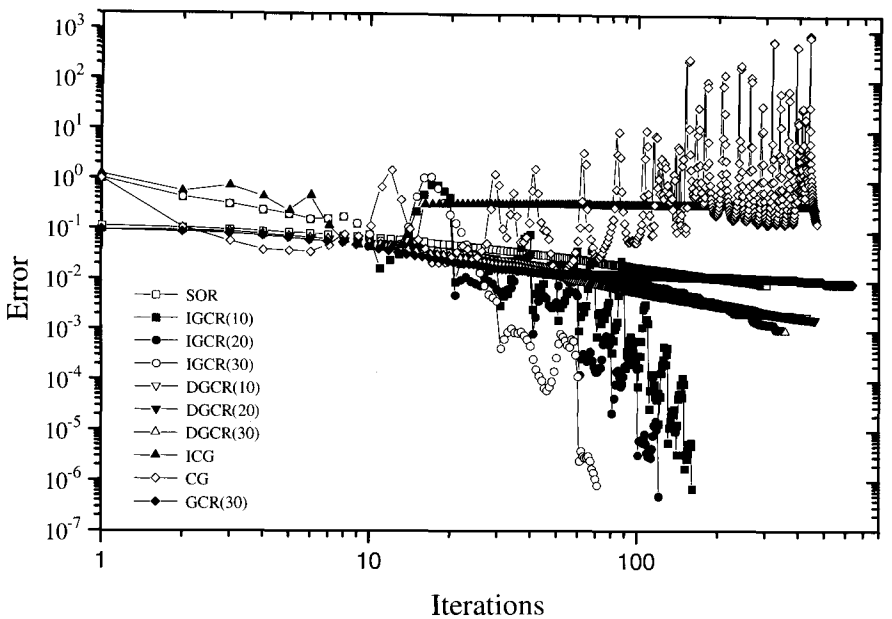

Fig. 7 Error vs. iterations for various iterative schemes applied on a 57 by 57 grid ground water mounding problem by finite differences

Fig. 8 shows that the maximum efficient inner loop number of IGCR(n) for the case tested is around 40. If computation memory was considered as well, a inner loop from 5 to 10 would be sufficient for this problem.

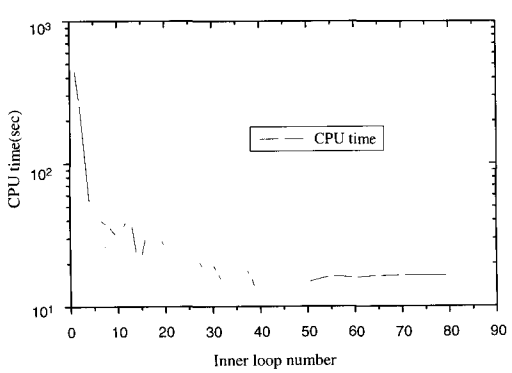

Fig. 8 CPU time vs. inner loop number for IGCR applied on a 57 by 57 grid ground water mounding problem by finite differences

Comparing CPU time, SOR and DGCR(n) are not as efficient as IGCR(n) and SL (Table 6) even though SOR uses the least memory (Table 7). For this two-dimensional problem, SL is the most efficient scheme as it uses LU decomposition instead of iteration schemes. The computation efficiency of SL contributes to the band width of the global matrix is narrow for a two-dimensional problem; hence, zero terms stored by SL are about the same as the non-zero terms stored by IGCR(n).

4.2 Three-dimensional example

This example described the free surface received a rate of 0.005 cm/sec recharge on an area 1m^2 for a homogeneous aquifer (K=0.01 cm/sec) with initial thickness 15 m. The aquifer has horizontal dimensions of 28 m by 28 m (i.e., 784 m^2 area).

Table 6 CPU time (sec) for various methods

Method grid	IGCR(10)	IGCR(20)	IGCR(30)	DGCR(30)	SL	SOR
22x22	0.66	0.45	0.53	7.12	0.32	54.44
57x57	29.92	32.70	23.24	---	9.56	---
85x85	178.13	148.14	151.15	---	37.18	---

Table 7 Approximate necessary memory (mega-byte) for conjugate-gradient-like methods, Sky-Line method, and SOR

Method grid	IGCR(10)	IGCR(20)	IGCR(30)	DGCR(30)	SL	SOR
22x22	0.183	0.259	0.334	0.305	0.216	0.019
57x57	1.268	1.775	2.283	2.066	3.247	0.128
85x85	2.526	3.967	4.885	4.605	10.392	0.284

Three methods are tested by comparing their iterations vs. error. The results indicates that ICG is not valid (Fig. 9). Regarding error convergence and CPU time (Table 8), IGCR(n) was found to be the best (most efficient) method. SL did not advantage over IGCR(n) for three-dimensional problem because its global matrix has a very wide band width which means a great number of zero terms were stored by SL (Table 9). LU decomposition of SL is no longer efficient because there were many more operations for SL than for IGCR(n).

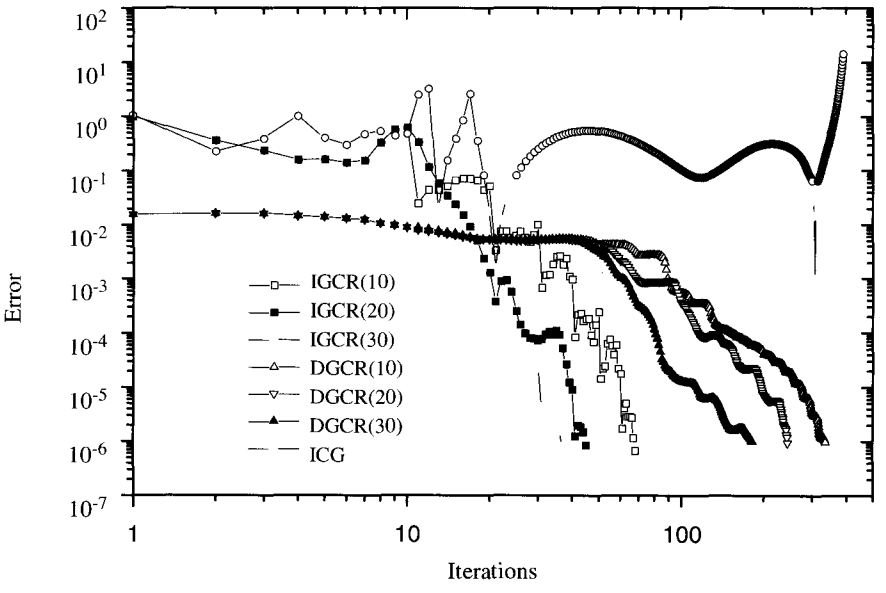

Fig. 8 Error vs. iterations for various iterative schemes applied on a 22 by 22 by 22 grid ground water mounding problem by finite differences

Table 8 CPU time (sec) for various methods

Method grid	IGCR(10)	IGCR(20)	IGCR(30)	DGCR(30)	SL
15x15x15	11.20	9.16	8.86	33.14	171.79
22x22x22	70.95	60.64	54.69	---	2929.07
29x29x29	183.51	179.15	199.90	---	---

Table 9 Approximate necessary memory (mega-byte) for conjugate-gradient-like and Sky-Line methods

Method grid	IGCR(10)	IGCR(20)	IGCR(30)	DGCR(30)	SL
15x15x15	1.7	2.2	2.8	2.4	7.7
22x22x22	5.7	7.3	9.0	7.7	93.5
29x29x29	13.5	17.2	21.0	17.9	312.0

6. Conclusions

(1) This general index method is an efficient for storing a sparse symmetric or non-symmetric matrix. In general, the general index method has the following advantages, especially for the computations in the generalized curvilinear coordinate.

a. adaptable to various governing and boundary conditions
b. flexible for higher order approximation
c. independent of problem dimension
d. efficient for complex problems when global matrix is not symmetric
e. convenient for general sparse matrices
f. computationally efficient in the most time consuming procedure of matrix multiplication (since only non-zero terms are stored)
g. flexible and applicable to any developed matrix solver

(2) Its application to conjugate gradient like methods showed a significant reduction in computations and memory requirement compared to the SK for three-dimensional problems.

Appendix I

```
      subroutine unsol1
     (nx,ny,nlu,nll,ll,lll,lu,llu,el,eu,ed,ndf,id)
      implicit none
      integer*4nx,ny,nlu,nll,i,ii1,ii2,j,ndf,jj,j1,j2,k,
     kk,i1,i2
      integer*4lu(nx*ny),ll(nx*ny),llu(nlu),lll(nll),
     id(nx*ny)
      real*8 ed(nx*ny),el(nll),eu(nlu)
      real*8 tmp
c**
c**   ed = ts
c**   eu = tsu
c**   el = tsl
c**
c**   incompleted Choleski decomposition
c**
      do i=2,ndf
        ii1=ll(i-1)+1
        ii2=ll(i)
c**
        do j=ii1,ii2
          id(lll(j))=j
        enddo
c**
c**     find el
c**
        do j=ii1,ii2
          jj=lll(j)
```

```
            if ((jj-1).eq.0) then
              j1=1
            else
              j1=lu(jj-1)+1
            endif
            j2=lu(jj)
            tmp=0.0d0
c**
            do k=j1,j2
               kk=id(llu(k))
               if(kk.ne.0)tmp=tmp+eu(k)*el(kk)
            enddo
c**
            el(j)=(el(j)-tmp)/ed(jj)
          enddo
c**
c**       find eu
c**
          i1=lu(i-1)+1
          i2=lu(i)
c**
          do j=ii1,ii2
             id(lll(j))=0
          enddo
          do j=i1,i2
             id(llu(j))=j
          enddo
c**
          do j=i1,i2
             jj=llu(j)
             if ((jj-1).eq.0) then
               j1=1
             else
               j1=ll(jj-1)+1
             endif
             j2=ll(jj)
             tmp=0.0d0
c**
             do k=j1,j2
                kk=id(lll(k))
                if(kk.ne.0)tmp=tmp+el(k)*eu(kk)
             enddo
             eu(j)=eu(j)-tmp
          enddo
c**
c**       find ed
c**
          tmp=0.0d0
          do j=ii1,ii2
             kk=id(lll(j))
             if(kk.ne.0) tmp=tmp+el(j)*eu(kk)
          enddo
          ed(i)=ed(i)-tmp
          do j=i1,i2
             id(llu(j))=0
          enddo
c**
        enddo
c**
        return
        end
```

Appendix II

```
        subroutine unsol2(nx,ny,ll,lll,lu,llu,ed,el,eu,
        r,nlu,nll)
        implicit none
        integer*4 i,j,jj,nx,ny,j1,j2
        integer*4 ll(nx*ny),lll(nll),lu(nx*ny),llu(nlu)
        real*8 tmp
        real*8 el(nll),eu(nlu),ed(nx*ny),r(nx*ny)
c**
c**     forward substitution
c**
        do i=2,nx*ny
         j1=ll(i-1)+1
         j2=ll(i)
         tmp=0.0d0
         do j=j1,j2
           tmp=tmp+el(j)*r(lll(j))
         enddo
         r(i)=r(i)-tmp
        enddo
c**
c**     backward substitution
c**
        do i=nx*ny,2,-1
         r(i)=r(i)/ed(i)
         j1=lu(i-1)+1
         j2=lu(i)
         tmp=r(i)
         do j=j1,j2
```

```
          jj=llu(j)
          r(jj)=r(jj)-eu(j)*tmp
        enddo
      enddo
c**
      return
      end
```

Appendix III

Iteration scheme of ground water mounding for the steady-state problem:

step 1: compute initial guess of the water table profile by Dupuit assumption ($H_{I,J}$)

step 2: construct grid for computation and calculate transformation relations based on the geometry of aquifer domain

step 3: solve the flow domain potential ($\phi_{I,J,K}$) according to the governing equation and boundary conditions

step 4: find the maximum difference between $H_{I,J}$ and $\phi_{I,J,NY}$ (i.e., $\mathrm{Max}\left|\phi_{I,J,NZ} - H_{I,J}\right|$) and check convergence. If it is satisfied, go to step 5; if not, compute $H_{I,J}^{new} = \left(H_{I,J}^{old} + \phi_{I,J,NZ}\right)/2$ and go to step 2

step 5: output results

References

1. Bercobier, M. and Rosenthal, A., 1986, Using the conjugate gradient method with preconditioning for solving FEM approximations of elasticity problems: Engineering Computation, v.3, p. 77-80.
2. Crank, J., 1984, Free and moving boundary problems: Oxford Science Publications, New York, 425 p.
3. George, A. and Liu, J. W-H, 1981, Computer solution of large sparse positive definite system: Prentice-Hall, Inc..
4. Obeysekare, U. R. B.; Allen, M. B.; Ewing, R. E.; George, J. H., 1987, Application of conjugate-gradient-like methods to a hyperbolic problem in porous-media flow: International Journal for Numerical Methods in Fluids, v. 7, no. 6, p. 551-566.
5. Suetomi, E. and Sekimoto, H., 1989, Conjugate gradient like methods and their application to fixed source neutron diffusion problem: Journal of Nuclear Science and Technology, v. 26, no. 10, p. 899-912.
6. Tsay, T. S., 1995 Numerical simulation of ground water mounding: Ph.D. Thesis, Department of Civil and Environmental Engineering, University of Wisconsin-Madison.

收稿日期：民國 87 年 9 月 9 日
修正日期：民國 87 年 11 月 3 日
接受日期：民國 87 年 11 月 5 日