An efficient approach for successively perturbed groundwater models

P. Patrick Wang & Chunmiao Zheng

Department of Mathematics, University of Alabama, Tuscaloosa, AL 35487, USA
Department of Geology, University of Alabama, Tuscaloosa, AL 35487, USA

(Received 20 May 1996; revised 11 November 1996; accepted 13 January 1997)

Global optimization methods such as simulated annealing, genetic algorithms and tabu search are being increasingly used to solve groundwater remediation design and parameter identification problems. While these methods enjoy some unique advantages over traditional gradient based methods, they typically require thousands to tens of thousands of forward simulation runs before reaching optimal or near-optimal solutions. Thus, one severe limitation associated with these global optimization methods is very long computation time. To mitigate this limitation, this paper presents a new approach for obtaining, repeatedly and efficiently, the solutions of a linear forward simulation model subject to successive perturbations. The proposed approach takes advantage of the fact that successive forward simulation runs, as required by a global optimization procedure, usually involve only slight changes in the coefficient matrices of the resultant linear equations. As a result, the new solution to a system of linear equations perturbed by the changes in aquifer properties and/or sinks/sources can be obtained as the sum of a non-perturbed base solution and the solution to the perturbed portion of the linear equations. The computational efficiency of the proposed approach arises from the fact that the perturbed solution can be derived directly without solving the linear equations again. A two-dimensional test problem with 20 by 30 nodes demonstrates that the proposed approach is much more efficient than repeatedly running the simulation model, by more than 15 times after a fixed number of model evaluations. The ratio of speedup increases with the number of model evaluations and also the size of simulation model. The main limitation of the proposed approach is the large amount of computer memory required to store the inverse matrix. Effective ways for limiting the storage requirement are briefly discussed. © 1998 Elsevier Science Limited. All rights reserved.

Key words: finite-difference method, global optimization, matrix inversion, parameter identification, remediation design, sensitivity analysis.

NOMENCLATURE

Matrices

- \( A \): the coefficient matrix obtained by the finite difference method
- \( B \): inverse of matrix \( A \)
- \( C \): changes of matrix \( A \)
- \( D \): diagonal blocks in matrix \( A \)
- \( E_i \): a square matrix with a single non-zero entry equal to unity at \((i,j)\)
- \( I \): identity matrix with appropriate dimension
- \( R \): changes of matrix \( A \)
- \( T \): tridiagonal blocks in matrix \( A \)
- \( Z \): \( p \) by \( p \) matrix = \( I + RBC \)

Vectors

- \( \{a\} \): vector on the diagonal line of matrix \( A \)
- \( \{b\} \): vector on the super-diagonal line of \( A \)
- \( \{c\} \): \( = \{b\} \)
- \( \{d\} \): vector on the off diagonal line of \( A \)
If the forward equation is linear, as in the case of flow in a confined aquifer, the coefficient matrix $A_p$ and right-headside vector $f_p$ are independent of the solution $\{x\}$.

The optimization procedure search starts with an initial value $\{p_0\}$ generated either randomly or from prior information. Each typical iteration of the search consists of three components: solution of the forward equation, evaluation of the objective function, and generation of a new set of parameters $\{p\}$. First, the forward equation is solved at a fixed value of $\{p\}$. Second, the objective function is evaluated. Finally, a new value of $\{p\}$ is generated in the neighborhood of $\{p_0\}$, say $\{p\} = \{p_0\} + \{\Delta p\}$. The search continues until a stopping criterion is met. This new portion $\{\Delta p\}$ is usually smaller than the value of $\{p_0\}$ to ensure the convergence of the search procedure. Based on the strategies of selecting $\{\Delta p\}$ are different optimization techniques.

Gradient-based optimization approaches select $\{\Delta p\}$ along the gradient direction. Global optimization approaches select $\{\Delta p\}$ randomly but in favor of the direction that reduces the objective function.

In recent years, global optimization methods are being increasingly used to solve groundwater remediation design and parameter identification problems. These methods include simulated annealing, genetic algorithms, tabu search, and gradient-based local search methods. Global optimization methods do not require the objective function to be continuous, convex, or differentiable. They can achieve other attractive features such as robustness, ease of implementation, and the ability to solve many types of highly complex, nonlinear problems.

One common drawback of these global optimization methods is that many objective function evaluations (ranging from hundreds to tens of thousands) are typically required to obtain optimal or near-optimal solutions. Nearly all of the computation time required by a global optimization method is devoted to solving the forward equation. Because each objective function evaluation requires one solution of the groundwater flow and/or solute transport model, the computational cost for a three-dimensional, field-scale problem can quickly become prohibitive. It is thus imperative to develop highly efficient methods for objective function evaluation if the global optimization methods are to be accepted and used widely in the field of groundwater modeling.

The objective of this paper is to present a new method for obtaining the solutions of a linear groundwater flow model repeatedly and efficiently, each of which follows a slight perturbation in the coefficient matrix $A_x$ and/or right-headside vector $f_x$ as typically required in a global search procedure. The motivation for this study arises from the fact that it is possible to solve the system of linear equations only once; new solutions can be obtained as a sum of the old solution $\{x\}$ and a new perturbed solution $\{\Delta x\}$ corresponding to the change of $\Delta p$. Because the new perturbed solution $\{\Delta x\}$ can be derived directly without solving the large system of linear equations again, the proposed method can dramatically improve the computational efficiency of a global optimization method. Furthermore,
the proposed method is also useful in gradient based optimization methods for obtaining the sensitivity coefficients through perturbation to aquifer parameters and sinks/sources.

The remainder of this paper is organized as follows. First, the governing forward equation is approximated using the block-centered finite-difference method, resulting in a system of linear equations. For simplicity, the proposed method is illustrated by considering a two-dimensional steady-state groundwater flow model. More complex conditions, such as three-dimensional and transient flows, can be treated similarly. Next, the solution of the linear equations in response to any perturbation in hydraulic properties and the sink/source term is derived. Potential applications of the proposed method in groundwater models are discussed, followed by the conclusion. A detailed list of symbols that are used throughout this paper is provided at the beginning of the paper.

2 FINITE-DIFFERENCE FLOW MODEL

To illustrate the methodology, a two-dimensional steady-state groundwater flow model is considered. It is assumed that the aquifer is confined with a uniform thickness. The governing equation and boundary conditions of the model are given as follows,

\[ \frac{\partial}{\partial x} \left( K_x \frac{\partial h}{\partial x} \right) + \frac{\partial}{\partial y} \left( K_y \frac{\partial h}{\partial y} \right) + w = 0, \quad (x, y) \in \Omega \]  
(3)

where \( h(x, y) = h_0(x, y), \quad (x, y) \in \Gamma \)

is the groundwater flow region of interest, \( \Gamma \) the boundary of the region, \( h \) the hydraulic head, \( K_x \) and \( K_y \) the principal components of hydraulic conductivity in the x and y directions, and \( w \) the sink/source term. The flow domain is represented by a rectangle, around which the heads are specified as \( h_0(x, y) \). It should be pointed out that the approach discussed below is applicable for any type of boundary conditions. The specified-head boundary condition is used here only for the convenience of step-by-step presentation of the procedure.

Numerical methods such as finite difference or finite element are often used in the solution of eqn (3). In this paper, the block-centered finite difference method is used, and the region \( \Omega \) is discretized into a grid system as shown on Fig. 1. Specifically, the y-direction is also divided into \( n_y + 2 \) equally spaced blocks with the block-centered nodal points labeled by 0, 1, \ldots, \( n_y \), \( n_y + 1 \). Let \( n_x \) and \( n_y \) be the number of active (interior) nodal points in x and y coordinates axes, respectively and let \( \Delta x \) and \( \Delta y \) denote the grid spacings and \( h_{ij} = h(\Delta x, \Delta y) \), and \( w_{ij} = w(\Delta x, \Delta y) \). Then, the governing equation as expressed in eqn (3) is discretized as a set of difference equations:

\[
K_{i+1/2,j-1/2} h_{i+1,j-1} - h_{ij} + h_{i-1,j-1} \frac{(\Delta y)^2}{(\Delta x)^2} \\
+ K_{i+1/2,j} h_{i+1,j} - h_{ij} + h_{i-1,j} \frac{(\Delta y)^2}{(\Delta x)^2} \\
+ w_{ij} = 0
\]  
(4)

where the \( K \) values at the interface between two nodal points are estimated from those at the two neighboring nodal points as follows,

\[
K_{i+1/2,j-1/2} = \frac{2K_{ij}K_{ij+1}}{K_{ij} + K_{ij+1}}
\]  
(5a)

\[
K_{i,j-1/2} = \frac{2K_{ij}K_{ij-1}}{K_{ij} + K_{ij-1}}
\]  
(5b)

\[
K_{i+1/2,j} = \frac{2K_{ij}K_{ij+1}}{K_{ij} + K_{ij+1}}
\]  
(5c)

\[
K_{i,j+1/2} = \frac{2K_{ij}K_{ij-1}}{K_{ij} + K_{ij-1}}
\]  
(5d)

To introduce vector notation we let \( m = (i-1)n_x + j, \) \( n = n_xn_y, \) \( h_{im} = h_{ij}, \) \( i = 1,2,...,n_x, \) \( j = 1,2,...,n_y \) and \( [h] = [h_1,h_2,...,h_{n}]^T \). After rearranging the terms, the following system of linear equations is obtained,

\[
\begin{align*}
& (r_m - c_m + e_m - f_m)h_{im} + a_m h_{im} + b_m h_{im+1} + d_m h_{im+n} = f_m, \quad m = 1,2,...n
\end{align*}
\]  
(6)
where
\[ g_m = \begin{cases} \frac{2K_{ij}K_{i,j-1}}{\Delta x^2(K_{ij} + K_{i,j-1})}, & m > 1 \\ 0, & m \leq 0 \end{cases} \] (7a)

\[ c_m = \frac{2K_{ij}K_{i,j-1}}{\Delta x^2(K_{ij} + K_{i,j-1})}, \quad (i,j) \notin \Gamma \] (7b)

\[ b_m = c_m \] (7c)

\[ d_m = g_m \] (7d)

\[ a_m = -(b_m + c_{m-1} + d_m + g_{m-n}) \] (7e)

These equations can be written in a matrix form as,

\[ A(h) = f \] (8)

where \( A \) is a band matrix with dimension \( n \)

\[ A = \begin{bmatrix} a_1 & b_1 & \cdots & d_1 & 0 \\ c_1 & a_2 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1} & b_{n-1} \\ 0 & \vdots & \cdots & g_{n-n} & c_{n-1} & a_n \end{bmatrix} \] (9)

The above system of linear equations is often solved by iterative methods, including successive over-relaxation or conjugate gradient methods.\(^9\)

### 3 Perturbed Solutions

In this section, we analyze the perturbed solutions for changes in the matrix \( A \) and the vector \( f \). We will first consider the change in \( A \) caused by changes in hydraulic conductivity, keeping the vector \( f \) fixed. We will then consider the change in vector \( f \) due to changes in the sink/source term, holding matrix \( A \) constant. Finally, we will address the case where both matrix \( A \) and vector \( f \) change simultaneously. Since we deal with systems of linear equations, the solutions are additive when both \( A \) and \( f \) change.

#### 3.1 Change in matrix \( A \)

To decompose the matrix we denote by \( \{e_i\} \) the column vector whose \( i \)-th element is one and zeros otherwise and let \( E_i = [e_i][e_i]^T \) denote the \( n \) by \( n \) matrix whose only non-zero element is at location \((i,j)\) with value one. The matrix \( A \) can now be decomposed into either column or row vectors,

\[ A = \sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij}E_{ij} = \sum_{i=1}^{n} \{e_i\} \left( \sum_{j=1}^{n} a_{ij}[e_j]^T \right) \]

\[ = \sum_{i=1}^{n} \left( \sum_{j=1}^{n} a_{ij}e_i \right) [e_j]^T \] (10)

From the discussion in the previous section, one can see that when the \( K \) value is changed at one location in the grid, say, location \((i,j)\), only a small number of entries in matrix \( A \) will be changed. Let \( A' = A + \Delta A \) be the changed matrix and \( \{h'\} = \{h\} + \{\Delta h\} \) be the changed heads. However, suppose that we have a \( n \) by \( n \) non-singular matrix, \( C \) is an \( n \) by \( p \) matrix, and \( R \) is an \( n \) by \( n \) (\( p < n \)) matrix, then the disturbed matrix \( A' = A + CR \) is also non-singular and its inverse is given by

\[ (A')^{-1} = A^{-1} - A^{-1}CZ^{-1}RA^{-1} \] (11)

where \( Z = I + RA^{-1}C \) is also assumed to be a non-singular matrix.

At first glance, eqn (11) appears to be unrelated to our problem. However, suppose that we have \( A(h) = \{f\} \) and \( A'[h'] = \{f\} \), and multiply eqn (11) by \( \{f\} \) on both sides, we find that the new solution \( \{h'\} \) can be obtained from the old one \( \{h\} \) as

\[ \{h'\} = \{h\} - A'^{-1}CZ^{-1}RA^{-1} \] (12)

If \( p \) is much smaller than \( n \), the size of \( Z \) will be much smaller than the size of matrix \( A \), and computing inverse of matrix \( Z \) will be much easier and faster than computing the inverse of the entire matrix \( A' \). For example, if the value \( a_{ij} \) in \( A \) is changed from \( a_{ij} \) to \( a_{ij} + \delta \), then from eqn (10) the corresponding changes are \( \Delta A = \delta E_{ij} \), \( C = [e_j] \), \( R = [e_j]^T \), and \( Z \) reduces to a scalar, \( Z = 1 + \delta [e_j]^T[\beta] \). Thus, the disturbed solution \( \{h'\} \) would be

\[ \{h'\} = \{h\} - \delta \frac{[e_j]^T[\beta]}{1 + \delta [e_j]^T[\beta]} \] (13)

where \( \beta \) is the solution of \( A[\beta] = \{e_j\} \).

In this example, to obtain the perturbed solution the system of linear equations has to be solved twice, and thus there is no advantage of using the proposed method. However, if the problem requires changing each entry in the matrix \( A \), then multiple perturbed solutions are obtained readily by updating eqn (13).

When the \( K \) value changes at a specified node in the grid, several entries in the matrix \( A \) will be affected. Suppose that the \( K \) value at \((i,j)\) is changed. After some algebraic manipulation, it can be shown that the perturbed
Successively perturbed groundwater models

The matrix becomes \( A' = A + \Delta A \) where the perturbed portion \( \Delta A \) is given by

\[
\Delta A = \begin{bmatrix}
\Delta a_{m-n} & \Delta b_{m-n} & \cdots & \Delta d_{m-1} & 1 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\Delta a_{m-n} & \Delta b_{m-n} & \cdots & \Delta d_{m-n} & \Delta d_m \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\Delta a_m & \Delta b_m & \cdots & \Delta d_m & \Delta d_{m+n}
\end{bmatrix}
\]

(14)

Recall \( m = j + (i-1)n \). All the 13 non-zero entries are listed in the matrix. The specific changes can be calculated using eqn (7). The matrix \( \Delta A \) can be written as the product of two smaller matrices \( \Delta A = CR \) where matrices \( C \) and \( R \) are, respectively,

\[
C = \begin{bmatrix}
e_{m-n} & e_{m-1} & e_m & e_{m+1} & e_{m+n} 
\end{bmatrix}
\]

(15)

\[
R = \begin{bmatrix}
\Delta a_{m-n} \\
\Delta a_{m-n} \\
\Delta a_m \\
\Delta a_{m+n}
\end{bmatrix}
\]

(16)

with \( \Delta A_i \) defined as the \( i \)-th row of matrix \( \Delta A \). The matrix \( Z = I + RA^{-1}C \) appearing in eqn (11) has dimension of 5 by 5 with 13 non-zero entries located at

\[
Z = \begin{bmatrix}
z_{11} & 0 & z_{13} & 0 & 0 \\
0 & z_{22} & z_{23} & 0 & 0 \\
z_{31} & z_{32} & z_{33} & z_{34} & z_{35} \\
0 & 0 & z_{43} & z_{44} & 0 \\
0 & 0 & z_{53} & 0 & z_{55}
\end{bmatrix}
\]

(17)

Inversion of \( Z \) is simple and straightforward. After the inverse of \( Z \) is updated, the inverse of matrix \( A \) and the new solution \( \{ h' \} \) can be obtained readily from eqns (11) and (12), respectively.

3.2 Change in vector \( \{ f \} \)

When the sink/source term is varied at a nodal point, only the vector \( \{ f \} \) in equation \( A \{ h \} = \{ f \} \) will be affected. Let \( \{ h' \} \) be the solution of \( A \{ h' \} = \{ f' \} \) where \( \{ f' \} = \{ f \} + \{ \Delta f \} \). Since the inverse matrix \( B = A^{-1} \) has been saved, the new solution can be obtained immediately by \( \{ h' \} = B \{ f' \} = \{ h \} + B \{ \Delta f \} \).

3.3 Changes in both matrix \( A \) and vector \( \{ f \} \)

When the \( K \) values change at the boundary nodal points, both matrix \( A \) and vector \( \{ f \} \) will be affected. Suppose that \( \{ h' \} \) is the solution of \( A' \{ h' \} = \{ f' \} \). Again, we assume \( A \{ h \} = \{ f \} \) and \( A' \{ h' \} = \{ f \} \). The new solution \( \{ h'' \} \) can be split as \( \{ h'' \} = \{ h' \} + \{ \Delta h \} \) and \( \{ f'' \} \) is written as \( \{ f'' \} = \{ f' \} + \{ \Delta f \} \). Hence, we have \( A' \{ h' + \Delta h \} = \{ f' \} + \{ \Delta f \} \), which yields \( A' \{ \Delta h \} = \{ \Delta f \} \). Since the inverse of \( A' \) has been updated by eqn (11) and stored, the solution \( \{ h'' \} \) can be obtained by simply adding the two solutions \( \{ h' \} \) and \( \{ \Delta h \} \).

4 COMPUTATION OF THE INVERSE MATRIX \( B \)

There is an extensive literature on matrix inversion, including George and Liu,\(^6\) Duff et al.\(^5\) and Press et al.\(^8\) Matrices can be inverted using either direct methods or iterative methods. Direct methods are efficient and simple when the size of the matrix is small. However, they are seldom used in large-scale systems in groundwater modeling primarily due to the computer memory requirement.
4.1 Direct methods

The inverse can be determined directly by Gaussian elimination or its variants. For computing the inverse of small matrices, Gaussian elimination is probably as efficient as any other method. That is, one starts with the matrix \([A, I]\) and applies elementary row operations to change the matrix \(A\) into the identity matrix \([I, B]\) and the corresponding matrix \(B\) will be the inverse. An obvious drawback of this method is that it takes twice as much memory as is needed to store the matrix \(A\). Press et al.\(^{18}\) presented a routine in which the matrix inverse \(B\) is gradually built up in \(A\) as the original \(A\) is destroyed.

4.2 Iterative methods

Essentially, finding the inverse of matrix \(A\) is equivalent to solving the equation \(AB = I\). Iterative techniques that are used in solving \(A[h] = [f]\) can also be applied in finding the inverse. Let \([b_i]\) be the \(i\)-th column of matrix \(B\). Then one repeatedly solves the system of equations \(A[b_i] = [e_i], i = 1, \ldots, n\). Using some matrix properties that will be addressed in the following discussions, one does not have to solve the system of equations \(n\) times.

Another use of iterative methods is to clear up numerical errors. After a number of updates of the inverse matrix required for perturbed solutions, round-off errors and other numerical errors will be accumulated in the inverse matrix \(A^{-1}\). One simple but efficient iterative algorithm can be used that clears up these errors.\(^{8}\) Let \(B_i\) be the inverse of \(A\) that needs to be ‘cleared’. The iterative algorithm is given by

\[
B_{k+1} = B_k(2I - AB_k)
\]

(18)

The algorithm stops as long as \(B_{k+1}\) is close to \(B_k\). Note that this method can be used only when \(B\) is near the true values of the inverse. Otherwise the algorithm may not converge. How often eqn (8) needs to be applied to clear up the errors requires numerical experiments and also depends on the desired accuracy.

4.3 Storage of the inverse matrix

The memory requirement for storing the inverse matrix \(B\) grows rapidly with the size of the problem. It requires \((n, n)^2 \times NB\), where \(NB\) is the number of bytes of each entry in the matrix. For example, for a two-dimensional flow model with a dimension of \(n = n_x = 100\) and each entry stored with a length of four bytes, then the system memory required to store matrix \(B\) is \((100 \times 100)^2 \times 4\) bytes = 4 x 10^8 bytes = 400 MB. In this subsection, we discuss possible alternative approaches that reduce the memory requirement.

4.3.1 LU decomposition

Careful examination of the structure of matrix \(A\) reveals that it is a block tridiagonal matrix of the form

\[
A = \begin{bmatrix}
T_1 & D_1 & 0 & \cdots & 0 \\
D_1 & T_2 & \ddots & \vdots \\
0 & \ddots & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots \\
0 & \cdots & 0 & D_{n-1} & T_n \\
\end{bmatrix}
\]

(19)

where \(D_i\) are block matrices and \(T_i\) are tridiagonal matrices. Using LU decomposition, the matrix \(A\) can be decomposed as a product of two matrices, \(A = LU\) where \(L\) is a lower triangular matrix and \(U\) is an upper triangular one. The inverse \(B\) is equal to the product of inverse of \(U\) and \(L\), \(B = U^{-1}L^{-1}\). Inverting \(L\) and \(U\) is much easier than inverting matrix \(A\) due to the special structure. In addition, matrices \(L\) and \(U\) are block band matrices given by, respectively,

\[
L = \begin{bmatrix}
I & \cdots & 0 \\
L_1 & I \\
\vdots & \ddots & \vdots \\
0 & \cdots & L_{n-1} & I \\
\end{bmatrix}
\]

(20)

\[
U = \begin{bmatrix}
U_1 & D_1 & 0 & \cdots & 0 \\
0 & U_2 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots \\
0 & \cdots & D_{n-1} & U_n \\
\end{bmatrix}
\]

(21)

where the block matrices \(U_i\) can be recursively computed by the following procedure,

\[
U_i = T_i \\
\text{for } i = 2, n, \text{ do} \\
\text{Solve } L_{i-1}U_{i-1} = D_{i-1} \text{ for } L_{i-1} \\
U_i = T_i - L_{i-1}D_{i-1} \\
\text{end}
\]

If we use the LU factorization, the system memory required is \(2n^2n_x \times NB\), which could result in substantial savings in memory requirements. For example, for the 100 by 100 problem considered previously, only \(2 \times 10^8 \times 4\) bytes \(= 8\) MB of system memory is required if LU is used, i.e., a reduction of 50 times. We need to point out that this 8 MB is just for storing the LU decomposition of matrix \(A\). To obtain the inverse \(B\), it is necessary to compute \(B = U^{-1}L^{-1}\). Thus, the memory savings would be realized at the expense of longer computation time.

4.3.2 Wavelet transforms

Wavelet transforms (WT) provide another great opportunity
of compressing data. The fundamental principle of WT is that the inverse matrix $B$ is transformed into a different domain by using a set of wavelet functions called wavelet bases. Under the new domain, many coefficients that are less significant can be neglected. Only significant coefficients are stored. Since there are so many different wavelet bases, which one gives the best performance in computing time and in storage size for a matrix is problem-specific that needs to be further studied. Commonly used wavelet bases are Daubechies bases and Meyer bases.\textsuperscript{16}

WT has been applied successfully in image processing and signal analysis\textsuperscript{19} and in other applications where a substantial amount of data needs to be transferred or stored. One interesting and promising application of WT is numerical analysis proposed by Beylkin\textsuperscript{1,2}. A larger matrix (an integral operator) is thought of as a digital image. Suppose that the operator compresses well under a two-dimensional wavelet transform, i.e., that a large fraction of its wavelet coefficients are so small as to be negligible, then any linear system involving the operator becomes a sparse system in the wavelet bases.

4.3.3 Use of auxiliary memory

If the memory requirements are still excessive after other memory-reducing techniques have been applied, one can use auxiliary memory such as hard disks or tapes. Computation of inverse of $A$ can be broken into a number of steps. In each step, only portion of the matrix is inverted and the results are stored. For example, instead of computing $AB = I$, we can split matrix $B$ as $B_1$ and $B_2$, and then solve,

$$AB_1 = I_1, AB_2 = I_2$$

(23)

The identity matrix $I = [I_1, I_2]$ can be further split into a number of smaller portions.

5 OTHER COMPUTATIONAL CONSIDERATIONS

Reductions in computational time and system memory requirements can be achieved if one carefully implements the proposed solution procedure. We discuss some computational aspects that may be helpful in implementing the proposed method. First, once the inverse of the $A$ matrix is obtained, most of the computation time that the proposed method spends is on updating the matrix $B' = B - BCZ^{-1}RB$ where $Z = I + RBC$. Note that $Y = CZ^{-1}R$ is a sparse matrix with only 25 non-zero entries. The matrix $YB$ is also a sparse matrix containing only 5 non-zero rows. The procedure of computing $B'$ should start with $Y, YB$, and then $B Y B$. Careful arrangement of computational sequence can lead to a substantial reduction in computation time. Second, matrix $A$ is symmetric and so is its inverse $B$. Thus, not only can the required memory be halved, the time in computing and updating the inverse can also be reduced accordingly. Golub and Van Loan\textsuperscript{7} provide more detailed information on efficient storage and manipulation of symmetric matrices.

6 NUMERICAL EXAMPLES

We consider a two-dimensional flow system as shown in Fig. 2. The size of the aquifer is 2000 by 3000 m. The aquifer is discretized into regularly spaced 20 rows and 30 columns. The hydraulic conductivity of the aquifer is assumed to be constant initially with $K = 100$ m/day. The head values at the specified-head boundaries are 10 m on the east, 9.1 m on the west, and on the north and south, the heads vary linearly between 10 and 9.1 m from the west border to the east border.

Two tests are conducted using the model. In the first test, the $K$ value at one nodal point inside the boundary is changed during each model evaluation. The change of the $K$ value is by a constant of 2 m/day. The sink/source term $w$ is set to zero everywhere. In the second test, the $K$ value is held constant at 100 m/day, but the $w$ term is assigned a non-zero value sequentially at each nodal point inside the boundary during each model evaluation. With the proposed method, it is only necessary to compute the inverse matrix once. New solutions of hydraulic heads are derived on the basis of the old solution and the inverse.

The results obtained by the proposed method are compared with those from the finite-difference flow model, MODFLOW.\textsuperscript{14} Both methods give exactly the same hydraulic heads. The standard MODFLOW code is implemented in an efficient subroutine form, eliminating all disk input/output during successive runs. The system of linear equations is solved by the Slice Successive Over-relaxation (SSOR) solver, which uses Gaussian elimination to obtain the solution in each two-dimensional ‘slice’. The test example is posed in such a way that the flow model is aligned with a slice in the SSOR procedure and the resulting system of linear equations is solved using the direct method. A new solution is recalculated by calling the MODFLOW subroutine once in response to any change in the hydraulic conductivity distribution and the sink/source term.

All computations were performed on an IBM-compatible computer.
personal computer with a 33 mHz 80486 CPU. Table 1 presents the total CPU times (in seconds) required for the two test cases. For the proposed method, the CPU times of computing the inverse and the CPU times of updating the perturbed solutions are reported separately. The break-even number, which is the minimum number of model evaluations required for the proposed method to be more efficient than running the MODFLOW subroutine repeatedly, is also listed in the table. The time of updating solutions in the proposed method is so small that the break-even number can be approximated by the ratio of the time needed to invert the matrix to the time needed to obtain one solution by MODFLOW.

6.1 Test case 1: Changing hydraulic conductivity

The first example solves for the new head distribution repeatedly when the hydraulic conductivity at each model node is changed one at a time, as usually required in applying a global optimization procedure such as simulated annealing to parameter identification problems. For this example, the numerical model is solved 504 times as the hydraulic conductivity value is changed successively at each of the 504 active model nodes. In this case, the break-even number is 23, meaning that the proposed method is faster if the simulation model is evaluated more than 23 times during the global optimization process. If the number of model evaluations required is fewer than 23, then repeatedly calling the MODFLOW subroutine is faster. The proposed method is about 15 times faster than the conventional method of calling MODFLOW repeatedly after 504 model evaluations, i.e., after the hydraulic conductivity value at each of the 504 active model nodes has been perturbed once. The break-even number and CPU speedup ratio depend on the problem size and the number of model evaluations required for a specific global optimization procedure.

The CPU times versus the number of model evaluations are shown in Fig. 3. The CPU time required for calling MODFLOW increases linearly at approximately 0.6 s per call. In contrast, the CPU time required for the proposed method has a constant time of 14 s for matrix inversion in the first evaluation, and a much smaller time (approximately 0.01 s) for solution updating for subsequent evaluations.

6.2 Test case 2: Changing sinks/sources

The second example can be related to a remediation design problem in which the optimal locations of injection/extraction wells are sought so that the total operating cost is minimized. In this case, the sink/source term is changed at each active model node, one node at a time, and the new heads are recalculated by using the proposed method and by calling the MODFLOW subroutine repeatedly. The CPU time for MODFLOW remains the same as in test case 1. However, the CPU time for the proposed method decreases considerably because the inverse matrix does not need to be updated. It is about 19 times faster than running the MODFLOW subroutine repeatedly after a total of 504 model evaluations, each of which corresponds to the change of the sink/source term at one of the 504 active model nodes.

7 POSSIBLE APPLICATIONS

The discussion of the proposed method has been based on a two-dimensional steady-state flow model. However, the proposed method can be readily extended to three-dimensional and transient models, both flow and transport, as long as they are linear. For example, for a three-dimensional steady-state flow model, there are 7 non-zero diagonal lines in the coefficient matrix A. The resulting matrix Z would be a 7 by 7 matrix when the hydraulic conductivity value changes at a model cell. Inverting a 7 by 7 matrix in the three-dimensional case is as simple and

| Table 1. Comparison of computational times for tests 1 and 2 |
|----------------------------------|---|---|
| Test 1                          | Test 2 |
| Total number of model evaluations | 504 | 504 |
| Total CPU time of calling MODFLOW | 310 | 310 |
| Total CPU time of the proposed method | 20  | 16  |
| Matrix inversion                | 14  | 14  |
| Solution updating               | 6   | 2   |
| Break-even number               | 23  | 23  |
| Speedup ratio for the proposed method | 15.5| 19.4|
easy as inverting a 5 by 5 matrix in the two-dimensional case. As for two- or three-dimensional transient flow models in which there is a system of equations, \( A(h) = \{f\} \) for each time step \( k \), more computer memory may be needed to store the inverse matrix for each of the time steps. This is because the coefficient matrices contain the time step size \( \Delta t \) used in transient solutions. Thus, if the time step size \( \Delta t \) is not uniform, the inverse matrix needs to be stored each time step. Except for this difference, all other procedures as used in the steady-state case are directly applicable in the transient case.

The proposed method can be directly used in the framework of an integrated global optimization and simulation model for identification of optimal aquifer parameters. In a global optimization procedure, such as simulated annealing and tabu search, the hydraulic conductivity value may be altered at only one location for each time the objective function is evaluated. Given that tens of thousands of objective function evaluations may be required in an optimization step, a tremendous saving in computation time would be realized by using the proposed method as opposed to running the simulation code repeatedly.

The proposed method is not only beneficial to global optimization methods, but also useful for gradient based optimization methods. For example, for parameter identification problems, parameter sensitivities related to the hydraulic conductivity values at observation points are evaluated by using either the adjoint-state method or the perturbation method. Instead of repeatedly evaluating the forward simulation code, the proposed method can be applied to obtain the sensitivities. If the number of observations is large enough, the proposed method is expected to be much faster.

The second direct application of the proposed method is in remediation designs in which optimal well locations and pumping rates are sought so that the total capital and operating costs associated with the remedial measures are minimized. When the pumping rates and/or well locations change in a confined aquifer, the corresponding changes in the system of linear equations \( A(h) = \{f\} \) are limited to the elements in vector \( \{f\} \) under confined flow conditions. The matrix \( A \) and its inverse remain unchanged. Therefore, once the inverse of \( A \) is computed, it will not be affected by the changes in well locations and pumping rates. Updating the new solution can be done readily and efficiently since the inverse matrix has been calculated and stored.

There are two limitations associated with the new proposed method. First, it only works for linear systems such as flow in confined aquifers. For flow in unconfined aquifers, the coefficients of the matrix \( A \) depend upon the solution \( \{h\} \). As a result, the proposed method cannot be used directly. The second limitation of the proposed method is the large amount of computer memory required. It may be possible to overcome this limitation by using new, innovative techniques for array storage such as wavelet transforms.

8 CONCLUSIONS

Global optimization models are being increasingly used in groundwater remediation designs (for both well locations and pumping rates) and in parameter identification (for both parameter structures and values). Both types of applications require repeated solutions of matrix equations in the form of eqn (8) for each change in the coefficient matrices. Based on the assumption that the forward simulation model is linear and that only a very small portion of the matrices are perturbed during successive simulation runs, we have presented, in this paper, a new solution methodology that is much more efficient than solving the system of equations directly and repeatedly for each change in the coefficient matrices. The proposed solution methodology obtains the new solution as the sum of a base solution and a solution to the perturbed portion of the coefficient matrices. The computational efficiency of the proposed method arises from the fact that the solution corresponding to the perturbed coefficient matrices can be easily derived without solving the full system of linear equations again.

The proposed method is tested using two relatively simple examples in which the finite-difference solutions to a two-dimensional steady-state flow model are evaluated repeatedly in response to successive changes in the hydraulic conductivity distribution and sink/source term. The hydraulic conductivity value and the sink/source term at each of the model active nodes is perturbed one at a time, similar to that required by repeated objective function evaluations in a global optimization procedure. The test results show that the proposed method leads to a reduction in the CPU times by more than 15 times over the method of calling the simulation code repeatedly. The reduction becomes even more significant as the number of model evaluations increases. Although only a two-dimensional steady-state flow model is used in this study, the fundamental principles can be extended to three-dimensional, transient flow and solute transport problems. The methodology can also be employed in finite element solutions. It is believed that the proposed method would make the global optimization methods computationally more competitive, thus removing the single most formidable limitation of the global optimization methods.

ACKNOWLEDGEMENTS

This research is supported in part by the U. S. Environmental Protection Agency through grant R823136-01. We are grateful to T.-Z. Mai for the helpful suggestions and references, and Y. Ke for the excellent programming work. We also wish to express our gratitude to C. J. Neville who reviewed an earlier draft of this manuscript and provided many valuable comments, and to the three anonymous reviewers whose constructive comments and suggestions helped us clarify our presentation.
REFERENCES