Solution of Large Sparse Systems of Non-Linear Equations[*]

R. P. Tewarson[+] and J. L. Stephenson[++]

Abstract:

Three algorithms for solving large sparse systems of non-linear equations are given. The algorithms are particularly suitable for handling equations that can be partitioned into two sets, such that the first set is large and easy to solve for most of the variables as functions of the remaining variables and the second set is small. This partitioning is done by using graph theoretic methods and/or the available information about the model structure. The algorithms are based on the Gaussian elimination, the Implicit Function Theorem and a Quasi-Newton Method. They require only a fraction of the storage and computing time that would be required to solve the complete system together without using sparse matrix methods; this feature makes them attractive in handling large problems.

[*]This Research was supported in part by the National Institute of Arthritis, Metabolism and Digestive Diseases, National Institutes of Health, Bethesda, Md Grant No. RM01 AM 17593-02.

[+]Applied Mathematics and Statistics Department, State University of New York at Stony Brook, New York 11794.

[++]Section on Theoretical Biophysics, National Heart and Lung Institute and Mathematical Research Branch, National Institute of Arthritis, Metabolism and Digestive Diseases, NIH, Bethesda, Maryland 20014.

Solution of Large Sparse Systems of Non-Linear Equations

R. P. Tewarson and J. L. Stephenson

I. Introduction

Solving a large sparse system of non-linear equations is often one of the important steps in handling large non-linear electrical networks, flow networks in physiological models, finite analogs of various other non-linear boundary value problems, and certain econometric models, etc. In dealing with boundary value problems, it is possible to avoid solving large systems of equations by making use of the shooting (or multiple shooting) methods [16]; but this leads to slow convergence and unstable behavior in many cases. As an example, we can cite the mathematical models of kidney function [18], where shooting (or multiple shooting) methods turned out to be completely unsatisfactory [25]. In view of the above facts, we have, in many cases, no choice but to solve a large system of non-linear equations. Fortunately such equations tend to be sparse (most equations involve only a few variables) and it is therefore possible to develop algorithms which require only a small subset of equations at a time for the solution of the whole system. A description of some efficient methods for solving such problems constitutes the primary basis of this paper.

We will assume that all the known information about the desired solution of a given system of non-linear equations that can be reasonably quantified has already been incorporated in the system. For example, quite often it is known that the solution vector x is an exponential function of the distance t from some origin (this is true of many biological models), then $x = x(t)$ and the given system should be rewritten as a function of t rather than x. Various other types of information e.g., smoothing, can also be incorporated either 'priori' in the system or in the solution process [22, 26]. This generally increases the size of the given system.

In the design and selection of methods for solving large systems of non-linear equations, many problems have to be considered, e.g., large storage requirements, slow rate of convergence, enlarging the domain of attraction of a root and choosing the initial approximation in such domain, computational instability and large programming costs. We will now briefly discuss each of these problems.

1. Large storage requirement. The algorithm for the solution of the given problem may require an amount of computer storage which is not readily available in the primary high speed storage of the computer. In this case some auxiliary slow storage must be used and the overall solution time increases by several orders of magnitude. This is especially true when algorithmic languages like Fortran are used. The situation can be somewhat improved by using some assembly language and exploiting the input-output features of the machine but this involves a major programming effort and the inevitable debugging problems.

Even if the problem is such that it can be accommodated in the primary storage and no auxiliary storage is required, the cost for the large amount of primary storage is significant. For example, in each Newton step, it is well known that the usual Gaussian elimination method for solving a non-sparse system of linear equations requires an order of n^3 or $O(n^3)$ multiplications and uses $O(n^2)$ cells for storage [27]. For large amounts of storage one is usually charged for storage in addition to the charge for the run time, and therefore in the Gaussian elimination, we use $O(n^2)$ storage for $O(n^3)$ time units and the total charge will be $O(n^5)$. From this it is evident that any saving in the storage requirements which does not lead to an increase of the same order in the number of arithmetical operations (primarily multiplications) is very worthwhile.

If the cost of solving the problem is not the primary consideration, then the user may still want to decrease the storage requirements for the sake of a quick turn around time. Because, in most computer installations, particularly those using time sharing, problems requiring an inordinately large amount of storage need a long turn around time; generally such problems can only be run overnight.

2. Slow rate of convergence. This involves the cost per iteration step and the total number of iterations required for a specified accuracy. These two factors generally work in opposite directions, e.g., the Gauss-Seidel method is fast per iteration but requires a large number of iterations and Newton's method has exactly the opposite characteristics [12]. In our experiments with flow network problems of kidney transport models, the block Gauss-Seidel method turned out to be quite unsuitable, not due to the cost per iteration but due to the very large number of iterations required for convergence. Even acceleration techniques did not significantly

improve the situation.

3. Enlarging the domain of attraction of a root and choosing the initial approximation in such a domain. For most methods the initial choice of an approximation is extremely critical.' It must lie in the domain of attraction of one of the roots, otherwise convergence will not take place to this particular root. The process may converge to some other undesirable root or may even diverge. It is known that this domain shrinks with the increase in the size of problems [13]. The above facts lead to the following important questions: How to enlarge the domain? How to choose an initial approximation for the root and desensitise the solution algorithm to this choice? Partial answers to these problems will now be given.

We can make use of the continuation methods [6, 14]. These essentially involve the solution of a series of problems, starting with a problem whose solution is easy to find and slowly perturbing it, a step at a time to get to the desired problem. At each step the solution of the previous problem is used as an initial approximation to the solution of the new problem. For example, in the case of flow network problems [20] the continuation method can be realized by varying the transverse permeabilities from zero to their final values in several small steps.

We can also proceed as follows [13]. If the system of non-linear equations was a result of discritizing a continuous problem, then first we choose the discretization parameter (or parameters) large and solve the resulting small sized problem, then we solve a series of problems of increasing sizes by successively decreasing the discretization parameter, until it is of the desired size. At each stage, the solution of the smaller problem (with the intermediate values obtained by interpolation) is used as a starting solution for the next (bigger) problem.

A simpler way than either of the above two procedures is to use simple iterative methods, like the Gauss-Seidel or the Successive Over-relaxation,' in the beginning and then switch over to rapidly convergent iterative methods like Newton's method near the root. This turns out to be useful in practice, because it is well known that the former methods have a slow convergence rate but large domains of attraction and the latter have fast convergence but smaller domains of attraction. The Levenberg-Marquardt damping [6] used in Newton's method, to some extent, utilizes the above mentioned hybrid technique. Because in the beginning a large value for the damping factor is used which makes the method closer to the steepest descent method and near the root a small damping factor makes the method like the Gauss-Newton method [6].

4. Computational Instability. The rounding errors may have a significant effect on the stability of the solution algorithm. For example, in each step of Newton's method a system of linear equations must be solved. For large sparse systems, the choice of pivots is critical. Reasonable sized pivots which do not create too many additional nonzero elements during the elimination process are generally desirable [9, 10, 23]. Small pivots should be avoided. It is our experience that in flow network problems [25], if a whole row has small pivots, setting that row and the corresponding right hand size to zero, does not significantly affect the convergence in Newton's method.

5. Large Programming costs. To implement any method or a set of methods for solving a given class of problems requires a large amount of investmentin programming and debugging. If problems of a similar structure are being repeatedly solved, then such an investment is usually justified. Another programming consideration is the amount of storage used by the program itself; it should be reasonably small relative to the size of the problems it is designed to handle.

It is evident that all of the above mentioned facts must be kept in mind when choosing an algorithm for solving a large system of non-linear equations. In the next section, we will describe some methods which have been found especially useful in practice for flow network problems. We will focus our attention on algorithms based on discretized Newton method which do not require the full approximate Jacobian in storage but use only parts of it at a time.

II. Methods Requiring a Small Amount of Storage

We shall first briefly describe a discretized Newton method for the solution of the set of non-linear equations

$$H(w) = 0, \tag{2.1}$$

where H and w are p dimensional column vectors.

If w^0 is the initial approximation to a root w^* of the system (2.1) and z^0 is the correction vector such that $w^0 - z^0 = w^*$, then $H(w^0 - z^0) = 0$ and the use of Taylor's theorem leads to the equation

$$H(w^0) - H'(w^0)\, z^0 + \cdots = 0, \tag{2.2}$$

where $H'(w^0)$ is the Jacobian of H evaluated at w^0. If the second derivatives (Hessians) of H are bounded near w^0 and $\|z^0\|$, the norm of z^0, is small, then the solution of the linear system

$$H'(w^0)\, z^0 = H(w^0)$$

gives an approximate value for z^0 and the next approximation for w^* is w^1, which is given by $w^1 = w^0 - z^0$. In general, if w^k is the kth approximation for w^*, then the next approximation is given by

$$w^{k+1} = w^k - z^k, \tag{2.3}$$

where z^k is the solution of the linear system

$$H'(w^k) z^k = H(w^k). \tag{2.4}$$

The iterative process is terminated when a suitable norm of $H(w^k)$ and/or z^k is reasonably small.

The Jacobian $H'(w^k)$ is generally determined by using numerical techniques, because analytic differentiation is difficult to carry out for a large number of equations, which are often not explicitly given, but only a subroutine for computing $H(w)$ for any given w is available. The Jacobian $H'(w^k)$ can be computed a column at a time by the formula

$$H'(w^k)e_j = \frac{H(w^k+h_k e_j)-H(w^k)}{h_k}, \quad j = 1, 2, \cdots, p, \tag{2.5}$$

where e_j is the jth column of the pth order identity matrix I. The parameter h_k

is generally chosen in a heuristic manner, e.g.,

$$h_k = \min (\alpha\|H(w^k)\|, \beta\|w^k\|, \tau) \qquad (2.6)$$

where τ is the machine tolerance and α and β are suitable weight factors [2]. If $H'(w^k)$ is computed by (2.5) or some other discrete numerical method, then this method is called a discretized Newton's method.

The main work at each step k of the discretized Newton's method given by (2.3), (2.4) and (2.5) is the solution of the linear system (2.4). In this paper we will be primarily concerned with techniques which reduce the amount of storage required by the Jacobian $H'(w^k)$ at each stage k of Newton's method. We will now describe three algorithm which require only parts of $H'(w^k)$ in storage at a time.

Let P and Q be two permutation matrices such that

$$PH(Qw) = \begin{bmatrix} f(x,y) \\ \\ g(x,y) \end{bmatrix} = 0, \qquad (2.7)$$

where $f(x,y) = 0$ is 'easy' to solve-for x if y is given. Note that if w is thought of as a row vector, then in (2.7) Qw should be replaced by wQ. It is easy to see that the vector x can be expressed as a function of y by using $f(x,y) = o$. Once this is done, then $g(x,y) = o$ can be written as $g(x(y),y)$ and it can now be thought of as a function of y only. If g and y are both m dimensional and f and x are n dimensional vectors such that $m << p$, where $p = m + n$, then $g(x(y), y) = o$ is a small system of equations. Its solution will require only the storage of a small sized m x m Jacobian $\frac{dg}{dy}$. Thus instead of a p x p Jacobian $H(w^k)$, we only store an m x m Jacobian. This is a large saving in storage. But, as we shall see later, there is a price to be paid for this saving.

In many cases it is possible to find P and Q such that in the system $f(x,y) = o$, if y is given, the first equation can be solved for x_1 (the first component of x), then the second one for x_2, and so on. For example, the set of equations

$$\begin{aligned} &f_1(x_1, y) = o \\ &f_2(x_1, x_2, y) = o \\ &\cdots \quad \cdots \\ &f_n(x_1, x_2, \cdots, x_n, y) = o \end{aligned} \tag{2.8}$$

can be solved in sequence for $x_1, x_2, \ldots, x_n$.

Equations like the above are said to be lower triangular in x. In order to fully discuss the structure of the original non-linear system $H(w) = o$, we define an occurrence matrix M of $H(w)$ such that the i^{th} row j^{th} column element of M is unity, if the variable w_j occurs in the equation H_i and is zero otherwise. In equations (2.8), if y is a scalar, then the occurrence matrix associated with $f(x,y)$ is

$$\begin{bmatrix} 1 & 0 & 0 & \cdot & \cdot & \cdot & 1 \\ 1 & 1 & 0 & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \end{bmatrix} .$$

The occurrence matrix M for the system $H(w) = o$ can be obtained from $H'(w^o)$, provided that for all i and j, $\frac{\partial H_i(w)}{\partial w_j}$ at w^o is zero if and only if H_i is independent of w_j (there are no horizontal tangents). Quite often M can be obtained from the model structure. The components of the model that interact with each other directly lead to a one in the associated matrix entry, the rest of the elements of M are zero.

The rearrangement of the equations and variables (which determines P and Q) such that the 'difficult' equations and variables can be isolated is not an easy problem, unless this information can be obtained from the structure of the model. For example, if a number of semi-permeable flow tubes interact transversly with a common bath, but not directly with each other, then the mass balance equations, and concentrationsand other variables associated with the bath are $g = o$ and y, respectively, and the rest of the system is $f = o$. Therefore, given y (the bath variables), the equations for all the other tubes $f(x,y) = o$ can be easily solved for x in terms of y in the flow direction [11, 19, 24].

If it is not possible to identify g and y from the model structure or if the model is not known to the equation solver, then we proceed as follows [23]. First we determine the occurrence matrix M, then permute it to get a set of ones on the main diagonal and call it $\hat{M}$. (In the literature on graph theory this process is called getting the maximum transversal.) Then we determine the'points of attachment' of the graph associated with $\hat{M}$. Various methods for doing this are available and are often called 'tearing and partitioning' methods [7, 23]. Essentially, they involve looking at the associated directed graph (or bipartite graph) of $\hat{M}$ and breaking larger loops which identify a set of rows and columns of M, such that after the removal of this set, the rest of the rows and columns of M can be permuted to a 'desirable form' for a particular algorithm. For a list of desirable forms for the Gaussian elimination see [23].

We have already mentioned the fact that a bordered lower triangular matrix can be made lower triangular if the variables and equations associated with the border are removed. Generally speaking, such 'partitioning and tearing' methods for identifying and removing the border are computationally slow and difficult to implement. We will not burden the reader with additional details as this is still an active area of research. The interested reader is referred to [23].

Let us now direct our attention to the solution of the equations $g(x(y), y) = o$ for y. For this purpose, we will require the evaluation of the small Jacobian $\frac{dg}{dy}$ at the point $(x(y^k), y^k)$. We will describe three methods that can be used for this purpose. The aim of this article is to describe how the storage requirements can be minimized, and not to burden the reader with mathematical rigor. Therefore, conditions of continuity, differentiability and non-singularity will be implicitely assumed whenever they are required in the derivation of a particular equation. For rigorous mathematical analyses of the various methods for solving a system of non-linear equations, the reader is referred to [12].

To compute the small Jacobian $\frac{dg}{dy}$ from the equations $f(x,y) = o$ and $g(x,y) = o$ we proceed as follows.

Let the partial derivatives of f and g with respect to x and y be denoted by f_x, g_x and f_y, g_y, respectively. Then differentiating $f(x,y) = o$, we have

$$\frac{df}{dy} = f_x \frac{dx}{dy} + f_y = o,$$

or

$$\frac{dx}{dy} = - f_x^{-1} f_y. \tag{2.9}$$

Now from g(x,y) we have

$$\frac{dg}{dy} = g_x \frac{dx}{dy} + g_y$$

and using (2.9) we get

$$\frac{dg}{dy} = g_y - g_x f_x^{-1} f_y \tag{2.10}$$

In order to make use of the above equation, the Gaussian elimination can be utilized as follows.

If the occurrence matrix for $\begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix}$ is of bordered lower triangular form,

then $\begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}$ is also of the same form (but **it** may have more zeroes than the occurrence matrix, due to the parallel tangents) and f_x is a n x n lower triangular matrix, the p x m block column $\begin{bmatrix} f_y \\ g_y \end{bmatrix}$ comprises the border.

Now to compute $\frac{dg}{dy}$ according to (2.10) we first generate and store the block border $\begin{bmatrix} f_y \\ g_y \end{bmatrix}$ in a p x m storage, say W, and then generate the columns of $\begin{bmatrix} f_x \\ g_x \end{bmatrix}$ one at a time (this requires a small amount of storage) starting from the top left hand corner and transform them by elementary raw operations (Gaussian elimination) to become unit vectors. The same transformations are applied to W. At the termination of this procedure, g_Y will have been replaced by $\frac{dg}{dx}$. This can be seen easily from (2.10) and the following equation

$$\begin{bmatrix} f_x^{-1} & 0 \\ -g_x f_x^{-1} & I \end{bmatrix} \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix} = \begin{bmatrix} I & f_x^{-1} f_y \\ 0 & g_y - g_x f_x^{-1} f_y \end{bmatrix} . \tag{2.11}$$

If f_x is a lower block triangular matrix, but its diagonal blocks are small relative to its overall size n, then the elimination is performed in several steps, starting with the top left hand diagonal block and proceeding down the main diagonal a block at a time. In each step, first the relevant diagonal block is generated in a storage, say V, and then transformed to an identity matrix of the appropriate order by the forward'and backward courses of the Gaussian elimination. The same operations are performed on W. Then the rest of the elements of the columns having elements in that particular diagonal block are generated, one column at a time in V plus any more storage if needed, and the forward course of the Gaussian elimination is used on V and W to make V zero. If any element on the diagonal of a particular diagonal block is small, then either partial or complete pivoting [23] can be used within the block. If even this does not lead to an acceptable pivot, then the rows of W which correspond to the elements in the diagonal block can be considered for pivoting. If a suitable pivot is available in W, then we have to interchange a column of W with a column of f_x having elements in the diagonal block. If a whole row of W and the corresponding row of the diagonal block under consideration have small elements, then the whole row can be set to zero. This creates no problems in practice and is theoretically justified [1,25].

In any case, if d is the size of the largest diagonal block in f_x, then the total storage S, needed for the above method to compute $\frac{dg}{dy}$ is given by S, = mp + max $(d^2, p-d^2)$. We know that the storage S for the whole Jacobian $H'(w^k)$ is given by $S = p^2$, therefore

$$\frac{S_1}{S} = \frac{m}{p} + \max\left(\frac{d^2}{p^2}, \frac{1}{p} - \frac{d}{p^2}\right) \approx \frac{m+1}{p}, \text{ if } d << p.$$

Thus there is a considerable amount of saving in the storage requirements if the set of difficult equations $g(x,y)$ has a small dimension m relative to p.

Once $\frac{dg}{dy}$ is available only m^2 cells of storage are required to store **it** and the same $\frac{dg}{dy}$ can be used for several steps to compute the new approximations for the root. This makes the method somewhat like the fixed slope method and the convergence is not quadratic as in Newton's method, but the saving in storage is more than before, as only m^2 cells of storage are now needed for $\frac{dg}{dy}$. **If** we let $\hat{S}_1 = m^2$, then $\hat{S}_1/S = \frac{m^2}{P} < \frac{m+1}{P} = \frac{S_1}{S}$. Thus we have shown that in comparison with the usual Newton method the above method requires only a fraction $\frac{m}{P}$ of cells for the first stage (when $\frac{dg}{dy}$ is being computed) and $(\frac{m}{P})^2$ in the subsequent stages, during which the storage used by f_Y for computing $\frac{dg}{dy}$ is not needed.

Prior to a discussion of two other methods which require only m^2 storage cells throughout their operation, we briefly describe how $\frac{dg}{dy}$ can be used to compute the next approximation (x^{k+1}, y^{k+1}) from (x^k, y^k). We consider g as a function of y and y alone. For this purpose, we express x as a function of y such that $f(x(y^k), y^k) = o$. In the following discussion we assume that all functions are evaluated at (x^k, y^k), unless indicated otherwise. Let

$$x(y^k) = x^k - \hat{\delta x}^k, \tag{2.12}$$

then $f(x^k - \hat{\delta x}^k, y^k) = o$ and from Taylor's theorem we have $f - f_x \hat{dx}^k + \cdots = o$. Neglecting higer order terms in $\hat{\delta x}^k$, we have

$$f_x \hat{\delta x}^k \approx f, \tag{2.13}$$

now since g is a function of y alone, the **k**th step of Newton's method applied to $g(x(y),y) = o$ is

$$y^{k+1} = y^k - \left(\frac{dg}{dy}\right)^{-1} g(x(y^k), y^k).$$

But in view of (2.12) and (2.13), we have

$$g(x(y^k),y^k) = g(x^k - \hat{d}x^k, y^k) \approx g - g_x\hat{d}x^k \approx g - g_x f_x^{-1} f.$$

Now, from the above equation, (2.11) and the fact that

$$\begin{bmatrix} f_x^{-1} & 0 \\ -g_x f_x^{-1} & I \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} f_x^{-1} f \\ g - g_x f_x^{-1} f \end{bmatrix},$$

it follows that if the column vector $\begin{bmatrix} f \\ g \end{bmatrix}$ was appended to the matrix $\begin{bmatrix} f_y \\ g_y \end{bmatrix}$ in W and the same operations were performed on it as on $\begin{bmatrix} f_y \\ g_y \end{bmatrix}$, then g would be transformed to $g(x(y^k), y^k)$. This requires only a small amount of increase in the storage W. Once y^{k+1} is known, x^{k+1} is given by

$$x^{k+1} = x^k - f_x^{-1}(x^k, y^{k+1})\, f(x^k, y^{k+1}).$$

We will now describe the other two methods for computing $\frac{dg}{dy}$. Both methods require m^2 storage cells but take longer than the method we have just described above. The first one makes use of the implicit function theorem and the second is based on the Quasi-Newton updates to transform g_Y to $\frac{dg}{dy}$.

The small Jacobian $\frac{dg}{dy}$ can be evaluated, as in (2.5), a column at a time. Its j^{th} column is given by

$$\frac{dg}{dy} e_j \approx \frac{g(x(y^k+h_k e_j), y^k+h_k e_j) - g(x(y^k), y^k)}{h_k}, \qquad (2.14)$$

where e_j is the j^{th} column of an identity matrix of order m and h_k is a suitably chosen small positive number as in (2.6). To evaluate $x(y^k+h_k e_j)$, we must use

$f(x,y) = o$. Therefore $f(x(y^k + h_k e_j), y^k + h_k e_j) = o$ and if we let $x(y^k + h_k e_j) = x^k - \hat{\delta x}^k$, then we have $f(x^k - \delta x^k, y^k + h_k e_j) = o$ and the use of Taylor's theorem gives

$$f(x^k, y^k + h_k e_j) - f_x(x^k, y^k + h_k e_j)\, \hat{\delta x}^k + \cdots = o.$$

If $\hat{\delta x}^k$ is small, then an approximation for it can be obtained by solving the linear system

$$f_x \hat{d x}^k = f, \tag{2.15}$$

where f_x and f are evaluated at $(x^k, y^k + h_k e_j)$. Note that if for $j = o$ we let $e_j = o$, then $y^k + h_k e_o = y^k$ and the evaluation of $g(x(y^k), y^k)$ is a particular case of the computation of $g(x(y^k + h_k e_j), y^k + h_k e_j)$. It is clear that for each j the linear system (2.15) has to be solved. Therefore, the evaluation of $\frac{dg}{dy}$ according to (2.14) will require the solution of (2.15) $m + 1$ times ($j = 0,1, \cdots, m$; where for $j = o, e_o = o$ and $x(y^k)$ is computed).

The above method will turn out to be very slow unless $f(x,y) = o$ is an easy and fast system to solve for x, given y. However, in all cases, if m is small, the situation is not bad.

Note that the columns of f_x can be generated one at a time, if f_x is lower triangular (or some other desirable form; see [23] for other desirable forms). This requires only p cells. In case f_x is block lower triangular then, as in the first method, the storage for solving equation (2.15) will be max $(d^2, p\text{-}d^2)$. Of course, $\frac{dg}{dy}$ requires m^2 cells. Thus the total storage for this method is given by $S_2 = m^2 + \max(d^2, p\text{-}d^2)$.

In the third method, we update g_Y to get $\frac{dg}{dy}$ and at the same time update the approximations (x^k, y^k). We can use any one of the rank one or rank two Quasi-Newton methods for this purpose [12, 15]. Essentially, we find a zero of the functions $g(x(y^k), y^k)$, where $x(y^k)$ is chosen such that $f(x(y^k), y^k) \equiv o$ for all y^k.

As in the second method, if (x^k, y^k) is given, then $x(y^k)$ is obtained by solving (2.13) for $\ddot{\delta x}^k$ and then using (2.12). We are now in a position to describe a Quasi-Newton method [3] for the solution of the system

$$g(x(y), y) = 0$$

with the condition that for a given y, x(y) will always be obtained by using (2.13) and (2.12). It is convenient' to describe this method in an algorithmic form as follows.

Given (x^0, y^0).

Compute $x(y^0) = x^0 - f_x^{-1}f$, where f_x and f are evaluated at (x^0, y^0).

Let $H^0 = (\frac{\partial g}{\partial y})^{-1}$ at $(x(y^0), y^0)$.

For k = 0, 1, 2, $\cdots$ perform the following steps

1. $y^{k+1} = y^k - H^k g^k$, where $g^k = g(x(y^k), y^k)$.
2. $x(y^{k+1}) = x(y^k) - f_x^{-1}f$; f_x and f are evaluated at $(x(y^k), y^{k+1})$.
3. $\delta g^k = g^{k+1} - g^k$, where $g^{k+1} = g(x(y^{k+1}), y^{k+1}$.
4. $\delta y^k = y^{k+1} - Y^k$
5. $H^{k+1} = H^k + \frac{(\delta y^k - H^k \delta g^k)\delta g^{k^T}}{\delta g^{k^T} \delta g^k}$

Stop when $\|g^k\|$ and/or $\|\delta y^k\|$ is small.

Note that in computing H^0, we take

$$\frac{\partial g}{\partial y} e_j \approx \frac{g(x(y^0), y^0 + h_0 e_j) - g(x(y^0), y^0)}{h_0}, \; j = 1, 2, \cdots, m.$$

In Step 5 of the above algorithm, it is possible to update H^k in many ways; we have used a rank one correction. Various other rank one and rank two uptdate formulas are given in [4,5,8,15,17,21].

For faster convergence, Step 1 of the algorithm can be replaced by $y^{k+1} = y^k - \alpha H^k g^k$, where the scalar a is chosen such that $\|g^{k+1}\| < \|g^k\|$, but this increases the computation time per step. It was pointed out by Broyden [6] that in many cases $a = 1$ is a good choice. This was found to be the case in practice when we solved our flow network problems.

III Concluding Remarks

In this paper we have discussed some of the important problems that one faces when attempting to solve a large system of non-linear equations. If the system is sparse, then it is possible to save storage and computing time. We have shown that if the equations and variables can be rearranged in such a manner that in the resulting system

$$f(x,y) = o$$
$$g(x,y) = o ,$$

the first set $f(x,y) = o$ can be solved 'easily' for x in a sequential or other manner for any given y in some neighborhood of a 'desirable' root and in comparison with f the set g consists of only a small number of equations, then it is possible to solve g as a function y alone. This requires only a small amount. of storage as compared with that required to solve the whole problem together.

We have described how the computation of the small Jacobian $\frac{dg}{dy}$, which is required for the solution of the small system $g(x(y), y) = o$ for y, can be done by any one of three methods: the Gaussian elimination, use of the Implicit Function Theorem and a rank one Quasi-Newton method. For the first two methods, we have shown that it is not necessary to compute $\frac{dg}{dy}$ at every iteration, but the same value can be used for several iterations to save the computation time.

We have applied all three methods to the flow network problems arising in the mathematical modelling of mammalian kidneys. We found that the computation of the small Jacobian $\frac{dg}{dy}$ was much faster in the first method than in the other two, but this advantage was offset by the relatively large storage requirement of this method. On the average in our problems $p \geq 6\ m$, and therefore the first method, needed more than six times the storage required by the other two. Due to this fact, the first method could not be used for solving many large problems. In terms of the overall cost all the three methods were competitive with each other.

Acknowledgements: The authors would like to thank Mrs. L.L. Juang, Mr. R.H. Mejia, Dr. A. H. Kydes and Professor B. Kellogg for many helpful comments and suggestions.

References

1. A. Ben-Israel, "A Newton-Raphson method for the solution of systems of equations", J. Math. Anal. Appl., Vol. 15, pp. 243-252, 1966.

2. K. M. Brown, "computer-oriented algorithms for solving systems of simultaneous nonlinear algebraic equations", pp. 281-348 in Numerical Solution of Systems of Nonlinear Algebraic Equations, G. Byrne and C. Hall, Eds., New York: Academic Press, 1973.

3. C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations", Math. Comp., Vol. 19, pp. 577-593, 1965.

4. C. G. Broyden, "Quasi-Newton methods and their applications to function minimization", Math. Comp., Vol. 21, pp. 368-381, 1967.

5. C. G. Broyden, "A new method for solving nonlinear simultaneous equations", Comput. J., Vol. 12, pp. 94-99, 1969.

6. C. G. Broyden, "Recent developments in solving nonlinear algebraic systems", in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, Ed., New York: Gordon and Breach, pp. 61-74, 1970.

7. L. K. Cheung and E. S. Kuh, "The bordered block triangular matrix and minimum essential sets of a digraph". Memorandum No. ELR-M375. Electronic Research Lab., Univ. of Calif. at Berkeley, Berkeley, Feb, 1973.

8. R. Fletcher, "A new approach to variable metric algorithms", Computer J., Vol. 13, pp. 317-322, 1970.

9. H. Y. Hsieh and M. S. Ghausi, "On optimal pivoting algorithms in sparse matrices", IEEE Trans. on Cir. Th., Vol., 19, pp. 93-96, Jan, 1972.

10. H. Y. Hsieh and M. S. Ghausi, "A probabilistic approach to optimal pivoting and prediction of fill-in for random sparse matrices", IEEE Trans. on Cir. Th., Vol. 19, No. 4, pp. 329-336, July, 1972.

11. L. L. Juang, J. L. Stephenson and R. P. Tewarson, "Solution of bordered type nonlinear systems of equations", submitted for publication, 1975.

12. J. Ortega and W. C. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables. New York: Academic Press, 1970.

13. W. C. Rheinboldt, "On the solution of large, sparse sets of nonlinear equations" in Computational Mechanics-Lecture Notes in Mathematics, Vol. 461, A. Dold and B. Eckmann, Eds., New York: Springer-Verlag, pp. 169-194, 1975.

14. W. C. Rheinboldt, "An adaptive continuation process for solving systems of nonlinear equations", Technical Report TR-393, Comp. Sc. Dept., Univ. of Maryland, July, 1975.

15. W. C. Rheinboldt and J. S. Vandergraft, "On local convergence of update methods", SIAM J. on Numer. Anal., Vol. 11, pp. 1069-1085, 1974.

16. S. M. Roberts and J. S. Shipman, Two Point Boundary Value Problems: Shooting Methods. New York: American Elsevier, 1972.

17. D. Shanno, "conditioning of Quasi-newton methods for function minimization", Math. Comp., Vol. 24, pp. 647-656, 1970.

18. J. L. Stephenson, "The mathematical theory of renal function", in Engineering Principles in Physiology, J.H.U. Brown and D. S. Gann, Eds., Vol. 2, pp. 283-320, New York: Academic Press, 1973.

19. J. L. Stephenson, "Equations for solute and water transport in models of biological flow systems", to be submitted for publication, 1975.

20. J. L. Stephenson, R. P. Tewarson and R. Mejia, "Quantitative Analysis of mass and energy balance in non-ideal models of the renal counterflow system", Proc. Nat. Acad. Sci. USA, Vol. 71, pp. 1618-1622, 1974.

21. R. P. Tewarson, "On the use of generalized inverses in function minimization", Computing, Vol. 6, pp. 241-248, 1970.

22. R. P. Tewarson, "solution of linear equations in remote sensing and picture reconstruction", Computing, Vol. 10, pp. 221-230, 1972.

23. R. P. Tewarson, Sparse Matrices, New York: Academic Press, 1973.

24. R. P. Tewarson and L. L. Juang, "Kidney modelling and sparse matrices", submitted for publication, 1974

25. R. P. Tewarson, J. L. Stephenson, A. Kydes and R. Mejia, "Use of sparse matrix techniques in numerical solution of differential equations for renal counterflow systems", (Abstract 74T-C34), Notices Am. Math. Soc. 21(1974), p. A498.

26. R. P. Tewarson, "Use of Smoothing and damping techniques in the solution of nonlinear equations", submitted for publication, 1975.

27. J. R. Westlake, A Handbook of Numerical Matrix Inversion and Solution of Linear Equations. New York: Wiley, 1968.