Implementation of the Solution Algorithm for Index and Near Index Problems

Yonsoo Chung, Arthur W. Westerberg

EDRC 06-104-91

Implementation of the Solution Algorithm for Index and Near Index Problems

May, 1991

Yonsoo Chung
Arthur W. Westerberg

**Engineering Design Research Center
and Department of Chemical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213**

ABSTRACT

This paper deals with implementing the solution algorithm for index and near index problems proposed by Chung and Westerberg. A numerical singularity checking method is suggested to detect the singularity or the near singularity of a matrix and to identify the equations responsible for the singularity or the near singularity. An exact numerical differentiation method is also suggested to generate elements of the Jacobian matrix and the equations needed. The potential use of the existing algebraic equation and ODE/DAE solvers in constructing a solver for index and near index problems is also studied. With the numerical method suggested in this paper, index and near index problems can be numerically and automatically solved.

Introduction

For the last decade, there has been a large literature building on the solution of index problems. Several papers [1,2,3,5, 8] propose solution algorithms for index problems. Chung and Westerberg [3,4] review these papers. Chung and Westerberg [4] also introduce a new class of DAEs - near index problems, and study its relation to stiff differential equations. They present a numerical algorithm for solving near index problems. In this paper, we shall discuss implementation of the solution algorithm for index and near index problems of Chung and Westerberg [3,4]. We shall first discuss issues in implementing their algorithm for a large problem involving thousands of equations and then present numerical methods that make the issues straight forward to solve.

To help one understand the background for this paper, this paper begins with review and summary of the solution algorithm for index and near index problems proposed by Chung and Westerberg [3,4].

Solving Index and Near Index Problems

Chung and Westerberg [3,4] propose a numerical algorithm for detecting and solving index and near index problems. The algorithm starts by finding whether the problem is well posed or not. We can demonstrate that the problem is not well posed if the equations are singular with respect to all the variables in the equations or if both a state variable and its derivative must be solved for using the equations. If not well posed, we would need to start over by reposing a correct set of equations for the problem at hand. To implement this step, we need a Jacobian matrix with respect to all the variables in the equations and a method to check the singularity of the Jacobian matrix.

For a well posed problem, the algorithm detects the rank of the Jacobian matrix with respect to derivatives of the state and algebraic variables to find whether the system has an index or a near index problem. If the Jacobian matrix is singular, the problem has an index problem; if the Jacobian is near singular, the problem has a near index problem. This step needs the Jacobian matrix with respect to derivatives of state variables and algebraic variables and a method to check the singularity or the near singularity of the Jacobian matrix.

For an index or a near index problem, the algorithm analyzes the Jacobian matrix used for detecting an index or a near index problem to find equations responsible for the singularity or the near singularity. Here, we must identify the equations responsible for the singularity or the near singularity of the Jacobian matrix.

Equations responsible for the singularity or the near singularity are differentiated to generate the equations needed to reduce the index of the system. In Chung and Westerberg [3,4], symbolic differentiation has been used as Gear and Petzold [6], Gear [5], and Bachmann et al. [1,2] have also done. An exact numerical differentiation method [7,9] is a good candidate to replace the symbolic differentiation method. We shall discuss an exact numerical differentiation method in detail in this chapter.

For the original equations plus the additional ones generated from differentiation, Chung and Westerberg [3,41 detect the rank of the new Jacobian matrix to find whether the system still has an index or a near index problem. The new Jacobian matrix is generated with respect to derivatives of state variables, algebraic variables, and new variables generated from differentiation. To implement this step, we need to expand the Jacobian matrix.

If the expanded Jacobian matrix is still singular or near singular, The algorithm detects and differentiates equations responsible for its singularity again. Singularity checking is carried out again. This procedure continues until the expanded Jacobian matrix becomes nonsingular. When the expanded Jacobian matrix is nonsingular, the algorithm obtains a solution from the new set of equations: the original plus the additional equations. The new set of equations which has a nonsingular Jacobian matrix is called as the augmented set. The solution algorithm for index problems terminates with a nonsingular Jacobian matrix. The remaining step is to solve the augmented set of equations as a set of algebraic equations and to integrate only a part of the original differential equations. The solution of index problems has only integration error which can be easily controlled. The algorithm for near index problems needs some more steps to improve the solution accuracy obtained from the augmented set of equations.

The algorithm for near index problems updates the augmented nonpivot variables to improve solution accuracy. Chung and Westerberg [3,4] generate a set of derivative equations by differentiating the highest order equations in the augmented set The final set of equations to solve consists of the augmented set and the additional sets of augmenting equations. The sets of augmenting equations are obtained by repeatedly differentiating equations in the augmented set which have never been differentiated according to the number of the augmented nonpivot variables and the specified error tolerance. The algorithm can improve the solution accuracy up to the specified error tolerance just with solving the final set of equations as a set of algebraic equations and integrating only a part of the original differential equations. To implement the updating procedure, we need to differentiate equations.

The Summary of the Algorithm

The following is the summary of an algorithm for identifying variables to be integrated numerically and obtaining their time derivatives at any time [4].

1. Given known or guessed values x, *xf* and *y*, make the Jacobian matrix of the original equations.
2. Detect the rank of the Jacobian matrix with respect to all the variables. Detect this rank where no variable x_L and its time derivative *x{* are simultaneously pivoted. If rank deficient, the problem is not well posed. Quit with a warning. (One has only guessed the solution point for the equations and has not yet converged them. Actual rank deficiency can only be detected at a solution point so this test is ambiguous at first.)
3. Detect the rank of the original equations with respect only to variables *xf* and *y*.
4. If rank deficient with respect to *xf* and *y*, the equations have an apparent index problem. Continue with the next step. If not, the equations can have a near index problem. Follow step 15.
5. Identify the equations responsible for the rank deficiency of the Jacobian.
6. Differentiate the detected equations. Note, the Jacobian matrix for the differentiated equations is obtained in this step.
7. Pivot through the Jacobian matrix of the differentiated equations only in the columns under newly generated variables until it can no longer be pivoted using only these variables.
8. Pivot through the Jacobian matrix of all the equations with respect to variables *x!* and *y* plus the pivoted variables found in the previous step.

9. If all these variables cannot be pivoted, repeat steps 5 to 8. Else, continue with the next step.
10. Assign arbitrary but convenient fixed values to those variables in the set of the newly generated variables not pivoted but appearing in the new equations.
11. For equations not yet pivoted in step 8, allow pivoting among the state variables, *x*.
•12. Solve the expanded set of equations.
13. Repeat steps 1 to 12 until the equations are converged at a solution.
14. Return with identity of state variables not solved in step 1 to 13 and with their time derivatives.
15. Check whether the Jacobian with respect to variables *xf* and *y* is nearly singular. If nearly singular, continue with the next step. If not, solve the equations for variables *:i* and *y*. Return with all state variables identified to be integrated and with their time derivatives, *ii*.
16. Treat the problem as if it were an index problem. Find the augmented set of equations with the algorithm used in steps 5 to 9.
17. Determine the near index of the system and locate augmented nonpivot variables.
18. To update augmented nonpivot variables, make a set of augmenting equations by differentiating the highest order equations in the augmented set.
19. Differentiate the set of augmenting equations found in the previous step as many times as one less than the near index of the system. Repeat this step if higher accuracy required.
20. Solve all equations found in steps 1 to 19.
21. Return with identity of state variables not solved in step 1 to 20 and with their time derivatives.

To implement their algorithm for a large problem involving thousands of equations, we need

1. to check the singularity or the near singularity of the Jacobian matrix and to identify the equations responsible for the singularity or the near singularity of the Jacobian matrix,
2. to differentiate equations and to generate the Jacobian matrix,
3. to solve the augmented set or the final set of equations and propagate it along time.

The first issue is the singularity checking problem, and the second is the differentiation problem. These are not simple for large nonlinear problems. The third issue is to solve a set of algebraic equations at each time step and integrate a part of differential equations between two adjacent time steps. Solution of a set of algebraic equations and integration of differential equations are well understood. We will study the potential use of existing algebraic equation and ODE/DAE solvers. The following sections discuss how to make the above issues straight forward to carry out.

Checking Numerical Singularity

The algorithm needs to detect if a matrix is singular or nearly singular. From the arguments in Chung and Westerberg [4], we see that in fact we only need to decide if a pivot is small as the solution is expanded in terms of this small pivot. We can therefore sidestep the issues of a matrix being singular and only ask if a pivot is small. The following discussion is to argue that we need only look at the magnitude of a pivot to decide if it is small.

We assume we have a set of equations *f(x,x^,y)* =/(JC,Z) = 0 which are well scaled. Each variable has a nominal value near to unity and the largest term in each equation has a magnitude near unity. We form the Newton equations for the equations at a guessed point for their solution and wish to detect if these equations have a small pivot when variables z are used as the pivots. To detect if the magnitude of a pivot is small, we should examine the Jacobian matrix in the larger context of having all the variables and the right-hand-side present during the forward elimination process, including those for the variables *x* which we do not intend to pivot. After the forward elimination process of a Gaussian elimination procedure, the equations would appear as shown in Figure 1.

The last equation is

$$\sum_i a_i \Delta x_i + \varepsilon \Delta z_{n+m} = b \qquad 0)$$

There is an interesting interpretation for *b*. Assume that Az_{n+W} is exactly zero. Then solve the remaining equations for their pivoted variables. Substitute these values into the original equation to which this unused last equation corresponds, *b* would be exactly the residual resulting.

There are three cases to consider in trying to decide if e is really small.

1. All a_{g9} e and *b* are really small in magnitude, say of order 10"^{8}. The equation is essentially stating that zero equals zero. However, we could divide through by e and have an equation with reasonable coefficients everywhere. It is here that the interpretation of *b* given above is useful, *b* (not *bit)* is the error in satisfying the last equation for a zero value for Az_{n+fn}. A small value of *b* says the original equation is in fact well satisfied locally with this assumption. We can claim this last equation is a locally dependent equation.
2. e only is really small in magnitude. One of the variables Ax; with a large coefficient $a_{\{}$ should really be selected as a pivot variable for this equation. Choosing Az_{rt+m} as a pivot would require it to have a value with an enormous magnitude. Since all variables are well scaled, none should have a perturbation with a magnitude much larger than unity.
3. All a_{x} and e are really small in magnitude and *b* is of order unity in magnitude. Then the

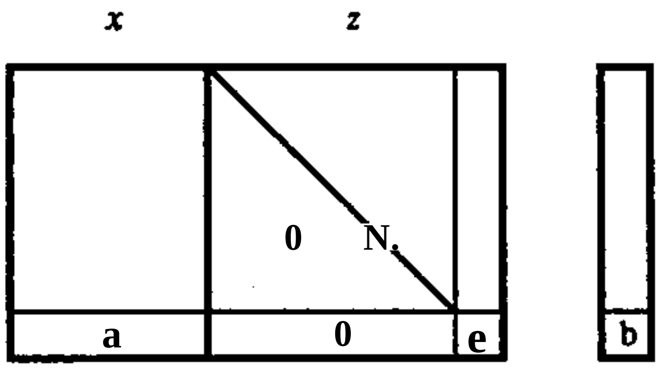

Figure 1: Jacobian Matrix after the Forward Gaussian Elimination

equation has to be inconsistent. Any variable selected as a pivot would have a value of enormous magnitude which is inconsistent with having a well scaled set of equations and variables.

In all of these cases, the coefficient e is small if and only if it is small in an absolute sense, provided the problem is well scaled. It can be detected during Gaussian elimination.

Discovering the Equations Causing the Singularity

Consider an *n* x *n* matrix *A*. Let us assume that matrix *A* is rank deficient by 1. Gaussian elimination of matrix *A* results in the echelon form in which the last row is empty. With information of row exchange from Gaussian elimination, matrix *A* can be rearranged as follows:

$$\begin{vmatrix} A_n & A_{l2} \\ A_{2l} & A_{22} \end{vmatrix} \tag{2}$$

where A_{2l} and $A22$ correspond to the empty row of the echelon form, and the dimension of submatrix A_n is (n-1) x (n-1). Gaussian elimination of matrix (2) converts it to

$$\begin{vmatrix} U_{11} & L_n{}^{-1} A_n \\ 0 & A_{22}-A_{21}U_{11}{}^{-1}L_{1f}{}^{1A}{}_{\backslash 2} \end{vmatrix} \tag{3}$$

where L_n and U_n are the LU factors of submatrix A_n. We note here that

$$A_{22} \ - \ A_{21}U_n{}^{-1}L_n{}^{-1}A_u \ = \ 0 \tag{4}$$

because the last row of matrix (2) becomes empty with Gaussian elimination. The last row is a linear combination of the first n-1 rows; we write therefore that

$$A_{11}{}^T \gamma \tag{5}$$

$$A_{12}{}^T \gamma \ \text{-V''} \tag{6}$$

where *y* is an n-1-dimensional vector. Using the LU factors of submatrix A_n, equation (5) becomes

$$U_{11}{}^{T}L_{11}{}^{T}\mathrm{Y}\text{-}\wedge 21^{\mathrm{T}} \qquad (7)$$

Premultiplying equation (7) with $U_{X}\{{}^{\prime\prime T}$ converts it to the form

$$\wedge \mathrm{li}^{\mathrm{r}}\mathrm{Y}=\mathrm{tflf}^{\mathrm{r}}\mathrm{V}* \qquad (8)$$

Therefore,

$$L_{u}{}^{T}y \ = \ (A_{2l}U_{n}{\sim}b^{T} \qquad \textbf{(9)}$$

From equations (4) and (9),

$$A_{22} \ = \ A_{2l}U_{u}{\sim}^{l}L_{u}\text{-}^{l}A_{n} \quad = \ i^{r}L_{n}L_{l}f^{l}A_{l2} \quad = \ fA_{n} \qquad \textbf{(10)}$$

Transposition of equation (10) is

$$A_{22}{}^{T} = A_{12}{}^{T}\gamma \qquad \textbf{(11)}$$

which is exactly the same as equation (6). This means that equation (6) is dependent on equation (5). Only equation (5) is required to determine the independent linear combination.

When equation (5) is solved for y, nonzero elements of *y* indicate rows involved in the linear combination. In other words, the nonzero elements of *y* indicate the rows responsible for the singularity of matrix *A*. Among the identities of equation (5), equation (9) is the easiest to solve for *y* because it can be simply solved by back-substitution.

To solve equation (9), we have to know A_{2l} $U_{n}{\sim}^{l}$ and L_{n}. We note here that submatrix A_{2l} and the lower triangular **part of** submatrix A_{n} have zero elements after Gaussian elimination, and their sizes are the same as those of $A_{2}\wedge$ f/n"^{1} and L$_{n}$. This situation allows us to store A_{2l} C/$_{11}{\sim}^{1}$ and L^ in the zero elements in matrix (3) to save memory and to supply information for equation (9). With this "overloading," matrix (3) becomes

$$\left|\begin{array}{cc} L_{n}\backslash U_{n} & L_{u}{}^{-1}A_{n} \\ A_{21}U_{11}{}^{-1}\backslash 0 & A_{22}\text{–}A_{21}U_{11}\ L_{11}{}^{-1}A_{12} \end{array}\right| \qquad (12)$$

Of course, $A_2r^{mA}2i^{u}ii^{\prime IL}n$ Ti4 i2 is zero-

Matrix (12) includes everything needed to detect the singularity of matrix A and to identify the rows responsible for the singularity. The zero value of ^22~"^{A}21^if^ n " ^{1} ^ indicates the singularity of matrix *A*, and the solution of equation (9) for y identifies the rows responsible for the singularity with A_2i $Ui\backslash\sim^l$ and L_n appearing in the leftmost part of the matrix.

The following example shows how the above numerical method detects the singularity of a square matrix and identifies the rows responsible for the singularity.

Example 1

Suppose we have a 4 x 4 matrix *A* for which the last row is a linear combination of other rows.

$$A = \begin{vmatrix} 2 & 1 & 1 & 4 \\ 4 & 1 & 0 & 5 \\ -2 & 2 & 1 & 3 \\ -6 & 3 & 1 & 2 \end{vmatrix} \qquad (13)$$

Submatrices of matrix (13) are

$$A_{11} = \begin{vmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{vmatrix}, \quad A_{12} = \begin{vmatrix} 4 \\ 5 \\ 3 \end{vmatrix} \qquad (14)$$
$$A_{21} = \begin{vmatrix} -6 & 3 & 1 \end{vmatrix}, \quad 4_{22} = \begin{vmatrix} 2 \end{vmatrix}$$

The LU factors of submatrix A_lj are

$$hi' \begin{vmatrix} 1 & & \\ 2 & 1 & \\ -1 & -3 & 1 \end{vmatrix}, \quad {}^uu = \begin{vmatrix} 2 & 1 & 1 \\ & -1 & -2 \\ & & -4 \end{vmatrix} \qquad (15)$$

From (14) and (15), the following information can be obtained.

$$L_{11}^{-1}A_{12} = \begin{vmatrix} 4 \\ -3 \\ -2 \end{vmatrix} \tag{16}$$

$$A_{21}U_{11}^{-1} = \begin{vmatrix} -3 & -6 & 2 \end{vmatrix}$$

$$A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} = 0$$

The overloaded matrix is

$$\begin{vmatrix} 2 & 1 & 1 & 4 \\ 2 & -1 & -2 & -3 \\ -1 & -3 & -4 & -2 \\ -3 & -6 & 2 & 0 \end{vmatrix} \tag{17}$$

where the diagonal elements of L_n do not appear because there is a lack of space and they are all 1's. The zero value of $A_22\text{-}A_2\backslash U_n\sim^l L_u\sim^l A_n$ shows that matrix (13) is singular.

For this example, equation (9) becomes

$$\begin{vmatrix} 1 & 2 & -1 \\ & 1 & -3 \\ & & 1 \end{vmatrix} \begin{vmatrix} Yi \\ Y_2 \\ Y_3 \end{vmatrix} = \begin{vmatrix} -3 \\ -6 \\ 2 \end{vmatrix} \tag{18}$$

By simple back-substitution, the solution of equation (18) is

$$Yi = -1. \quad Y_2 = 0. \quad Y3 = {}^2 \tag{19}$$

This indicates that the first, the third and the fourth rows of matrix (13) are responsible for the singularity.

When the rank deficiency of matrix *A* is greater than 1 and less than n, the above method can be also applied. In this case, the zero values of the submatrix ^ " ^ l ^ i T ^ i r ^ n shows the singularity, *y* becomes a set of vectors whose solutions identify the rows responsible for the singularity.

Discovering Equations Responsible for a Near Singularity

The Newton formula for a set of DAEs, /(x, x', *y*, *t*) = 0 , is

~~#~~Ax + - ^ Ax' + ^ Ay + *%At* = -/(x,V,;y, *t*) (20)
dx dxf dy dt

In the above equation, Ax' and *Ay* terms are unknowns, and others are known from previous steps. Therefore, Ax and *At* are zero, and equation (20) reduces to:

∂f *AX?* + *&Ay* = -/(u'j,r) (21)
dx? 3y

If we introduce a new variable z for variables x' and *y*, equation (21) becomes

^ A z = *RHS* (22)
dz

Let *U* be the echelon form of the Jacobian and *b* be the vector of *RHS* after Gaussian elimination, then

UAz = *b* (23)

When *U* has a very small pivot, the system has a near index problem. Let us assume that the last pivot of *U* has a very small value, e. In this case,

$\wedge 22 - A_{2l}U_n"\%f^dA_n = e$ (24)

where

0 £ |e| « 1

The linear combination for the near singular matrix *A* is

$A_u^T y + e,_ = \wedge^r$ (25)

$$A_{12}^T \gamma + \varepsilon_2 = A_{22}^T \qquad (26)$$

where γ is an n-1-dimensional vector and ε_2 is a scalar with very small values. If we treat ε in equation (24) as if it were zero, ε_1 and ε_2 would become zero. This makes equation (26) dependent on equation (25) which is the same as equation (5). The rows responsible for the near singularity can be identified by solving equations (9) with an error tolerance which is a little larger than the magnitude of ε.

The following example shows how the above method works on a near singular matrix.

Example 2

Consider the following matrix which is perturbed from matrix (13).

$$A = \begin{vmatrix} 2 & 1 & 1 & 4 \\ 4 & 1 & 0 & 5 \\ -2 & 2 & 1 & 3 \\ -6 & 3 & 1+\varepsilon & 2 \end{vmatrix} \qquad (27)$$

If we treat ε in matrix (27) as if it were zero, we can apply the above method for this example. Submatrices of matrix (27) are

$$A_{11} = \begin{vmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{vmatrix}, \quad A_{12} = \begin{vmatrix} 4 \\ 5 \\ 3 \end{vmatrix} \qquad (28)$$

$$A_{21} = \begin{vmatrix} -6 & 3 & 1+\varepsilon \end{vmatrix}, \quad A_{22} = \begin{vmatrix} 2 \end{vmatrix}$$

The overloaded matrix is

$$\begin{vmatrix} 2 & 1 & 1 & 4 \\ 2 & -1 & -2 & -3 \\ -1 & -3 & -4 & -2 \\ -3 & -6 & 2-\frac{\varepsilon}{4} & -\frac{\varepsilon}{2} \end{vmatrix} \qquad (29)$$

The value of $A_{22} - A_{21} U_{11}^{-1} L_{11}^{-1} A_{12}$ is -1. It shows that matrix (27) is near singular.

For this example, equation (9) becomes

$$\begin{vmatrix} 1 & 2 & -1 \\ & 1 & -3 \\ & & 1 \end{vmatrix} \begin{vmatrix} Y_i \\ Y_2 \\ Y_3 \end{vmatrix} = \begin{vmatrix} -3 \\ -6 \\ 2-\frac{\varepsilon}{4} \end{vmatrix} \tag{30}$$

The solution of equation (30) is

$$\gamma_1 \;--\text{«}+*\overset{5}{\text{«}}\bullet \qquad \gamma_2 = -\frac{3}{4}\varepsilon, \qquad \gamma_3 = 2-\frac{\varepsilon}{4} \tag{31}$$

If we specify an error tolerance of 2e for the solution of equation (30), y_2 can be treated as zero, while y_x and y_3 are nonzero. These values indicate that the first, the third and **the** fourth rows of matrix (27) are responsible for the near singularity.

The algorithm for solving index and near index problems investigates two kinds of Jacobian matrices: a square Jacobian matrix and a rectangular one. The square Jacobian matrix is used for detecting equations to differentiate and the rectangular one is used for detecting whether the system is well posed or not. The above numerical method is well suited to the case for the square Jacobian matrix. **However,** it can be also applied for the case of the rectangular Jacobian matrix. When we detect the singularity of the rectangular Jacobian matrix to check whether a system is well posed or not, we do not need to identify the equations responsible for the singularity. We just need to know whether it is singular or not. This can be easily carried out by Gaussian elimination.

An Exact Numerical Differentiation Method

Many equation solvers and optimizers have used symbolic differentiation or finite difference approximation to obtain derivatives of a function. Most symbolic differentiation packages are large and cumbersome because it is difficult to manipulate symbolic expressions, and the algebraic form of the derivatives of nonlinear functions is commonly more complicated than the original expression. Finite difference approximation includes errors in itself, and we cannot guarantee that there is no accumulation of the error. We will discuss an exact numerical differentiation method to use instead of a symbolic differentiation method or a finite difference approximation approach.

Ponton [9] states that if the derivative obtained with symbolic differentiation is more complex, and hence takes longer to evaluate than the function, then numerical differentiation would be faster. He presents a computational scheme for obtaining the numerical value of the analytical derivatives of a function without the generation of computer code representing the differentiated expression. The original function is supplied as a reverse Polish string, and values of the function and partial derivatives with respect to all the variables are obtained simultaneously using multiple stacks.

Griewank [7] shows that a reverse mode of automatic differentiation is far superior to symbolic differentiation or finite difference approximation. He states that the symbolic differentiation approach is not only inefficient but unmaintainable because it generates separate expressions for the function and each gradient component directly in terms of the independent variables and it has difficulties in inspecting and correcting programming errors. He insists that one should write a program for evaluating the function efficiently and then generate an extended program that evaluate the function and gradient simultaneously. He uses the chain rule and composite functions to evaluate the function and each gradient component simultaneously. He demonstrates that the work ratio of the reverse mode of automatic differentiation is bounded by a constant number: five, while the bounds of the forward mode of automatic differentiation, symbolic differentiation, and finite difference approximation grow linearly with the number of variables. The woric ratio is defined as the ratio of the work required to evaluate the function and each gradient component to the work required to evaluate the function only.

Based on the reverse mode of automatic exact numerical differentiation in Griewank [7], we will develop a numerical differentiation method to evaluate the function and its partial derivatives up to any order. We will carry out this discussion with an example problem to aid in making the points more clearly.

The Function Evaluation

Suppose we have a function to evaluate:

$$>>=/(*) \qquad 02)$$

where x e˙ R^n. Evaluation of a function can be expressed by a computational graph [7]. Figure 2 shows a computational graph for a function $f_1[f_6\{f_5(x_v x_4),x_2),x_l]$. Nodes x_5, x_6 and x_7 correspond to elementary functions/$_5$,/$_6$ and/$_7$, respectively.

From a view of **the** computational graph, a function can be interpreted as a set of composite functions as follows;

For i = n+1, n+2, ... , m (33)

End i

$y = {}^{x}m$

where an elementary function f_i depends only on the already computed quantities Xp where node j is directly connected to node i in the computational graph. In other words, / is the composition of m-n elementary functions *frs.* The procedure (33) evaluates a function. The function of Figure 2 can be evaluated with the composition of 3 elementary functions:/$_5$,/$_6$ and/$_7$.

The First Order Partial Derivative

To obtain gradients of function (32), we need to calculate the first order partial derivatives:

$$\frac{\partial f_m}{dx_k} \tag{34}$$

where f_m is the top node of the computational graph and x_k is one of the independent variables. We note that f_m has the value of the function with procedure (33). Because a function is the composite of elementary functions, the first order partial derivative (34) is obtained by the chain rule.

$$\frac{\partial f_m}{\partial x_j} = \sum_i \frac{\partial f_m}{\partial x_i} \frac{\partial f_i}{\partial x_j} \tag{35}$$

where $\frac{df.}{}$ is nonzero only if node j is directly connected to node /; in other words, the right hand side term is effective only when node j is directly connected to node i in the computational graph. If the index j is iterated in the reverse order, $\frac{a/_m}{\partial x_i}$ is always previously calculated. Here, node j is not necessarily an independent variable, but can be an elementary function. The first order partial derivative (34) is obtained

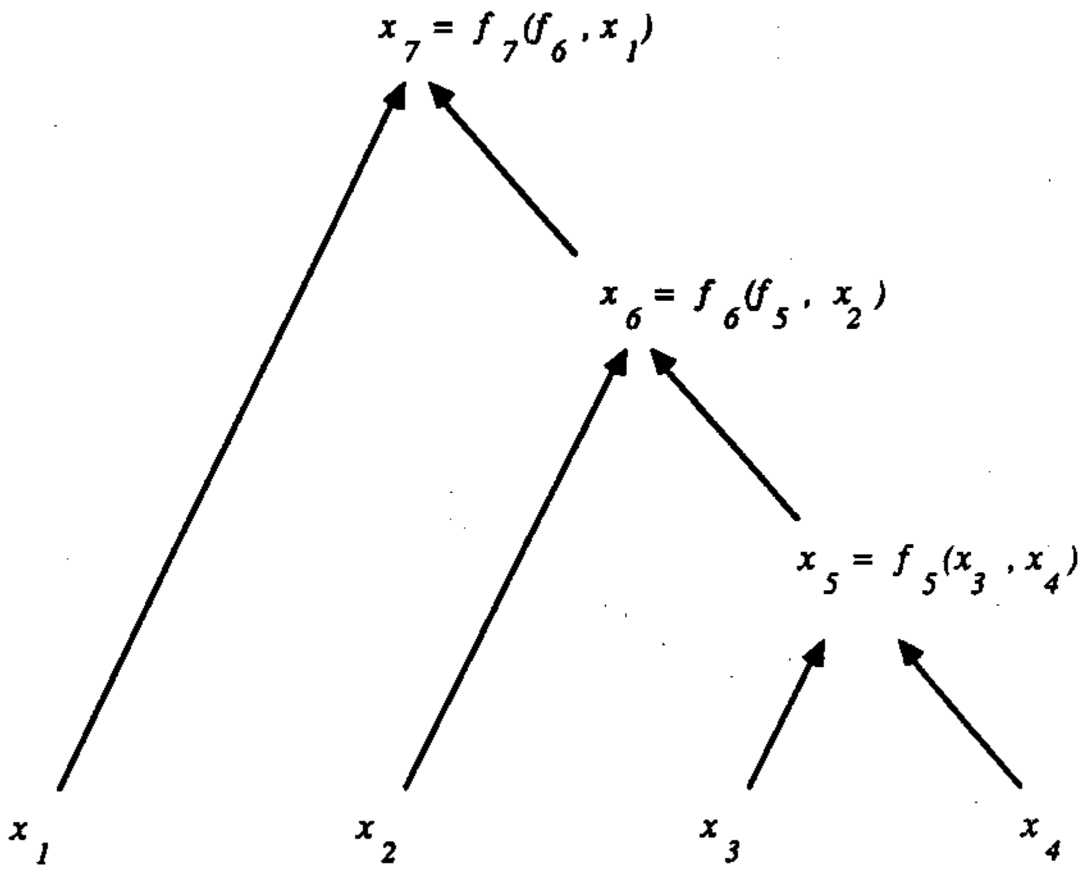

Figure 2: **A** Computational Graph for Function $/_7$ [$/_6$ {$/_5$ (x_3, x_4), X2}, x_x]

when; in equation (35) corresponds to one of the independent variables.

If we use the following notation,

$$ij_J = T''\ dx_j \qquad (36)$$

the procedure to evaluate gradients of function (32) is as follows.

```
For i = m-1, m-2 . . . . 1                                   (37)
    m_j = X^{m,1} h        where j e C(i) and j € C(m)
End i
```

where *C(f)* is the set of the children of node i_9 i.e., the set of elements directly connected to node *i*. We note that *j € C(m)* because my for *j* e *Cim)* is assumed to be computable. We also note that this procedure is in "reverse mode" because the index *i* is decremented from m-1 to 1.

We consider a simple example to understand how procedures (33) and (37) evaluate the function and its first order partial derivatives.

Example 3

$$y = *!X2^2*3*4 \qquad (38)$$

where it is known that JCJ = \,x_2 = 2,JC_3 = 3andx$_4$ = 4.

It is possible to construa many computational graphs for this example problem. All of the possible structures generate the same result. Figure 3 shows one possible structure. Let us consider the computational graph in Figure 3. There are four elementary functions: JC_5, x_6, JC_7 and *x%*. The first order partial derivatives of each elementary function with respect to its own variables are easily determined. They are noted on the links connecting the two nodes. Function (38) can be evaluated by procedure (33) as follows:

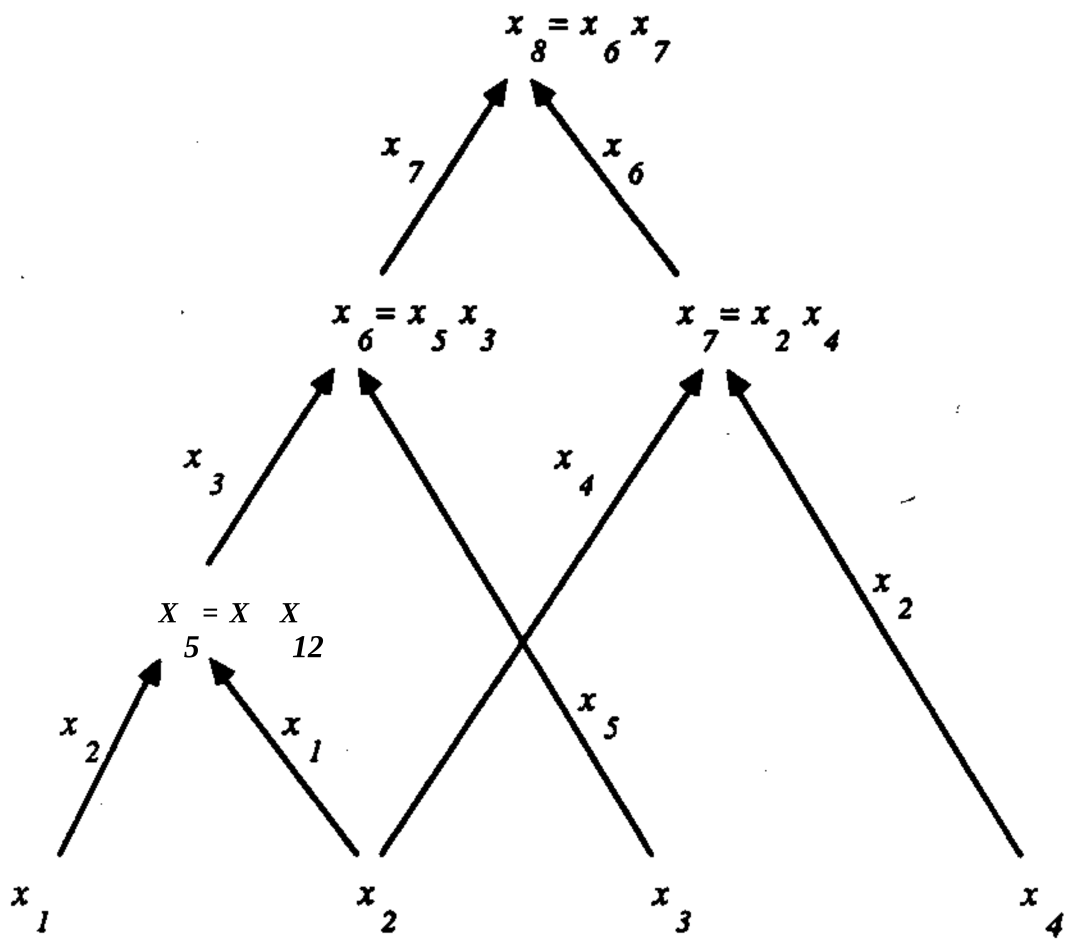

Figure 3: A Computational Graph of Example 3

$$x_5 = x_1 x_2 = 2 \tag{39}$$
$$x_6 = x_5 x_3 = 6$$
$$x_7 = x_2 x_4 = 8$$
$$x_8 = x_6 x_7 = 48$$

Function (38) has the value of 48 which is exact with the given values of ***xfs*****.**

The sets of the children are

C(8) = {7,6} (40)

CO) = {4,2}

C(6)= {5,3}

C(5) = {2,1}

The first order partial derivatives of elementary functions are as follows, using the numbers computed by (39).

$8_7 = x_6 = 6$ (41)

$8_6 = xj = 8$

7_4 = *2 = 2

$7_2 = x_4 = 4$

65 = *3 = 3

$63 = x_5 = 2$

5_2 - X! - 1

5i = *2 = 2

These **are the exact** numeric values for the first **order partial derivatives of the elementary functions. With this information,** procedure (37) proceeds as follows:

$$8_5 = 8_6 6_5 = 24 \tag{42}$$

$$8_4 = 8_7 7_4 = 12$$

$$83 = 8563 = 16$$

$$8_2 = 8_7 7_2 + 85\,5_2 = 48$$

$$8_X = 855! = 48$$

The gradient components of function (38): 8_{1t} 8_2, 83 and 8_4 have **exact** values **with the** given values of **Vs.**

The Second Order Partial Derivative

In the above, we have discussed and showed how to evaluate the function and its first order partial derivatives with the reverse mode of automatic differentiation. The **second order partial** derivatives can also be obtained with a similar method. For the second order partial derivatives for **the** function (32), we need to calculate

$$\frac{\partial^2 f_m}{\text{T-T-}_{\iota}\,{}^{\wedge}{}_n} \tag{43}$$

where f_m is the same as in (34), and $x_{\{}$ and x_n are the independent variables. These can be obtained by differentiating the first order partial derivatives (35).

$$\frac{\partial^2 f_m}{\partial x_i \partial x_j} = \frac{\partial}{dxi}\frac{\partial f_m}{dx_j} = \frac{\partial}{dXiT}\sum \frac{\partial f_m}{dx_k}\frac{\partial f_k}{dx_j} \tag{44}$$

Using notation (36) and

$${}^kU_7 = \text{J} \;\; \frac{\wedge}{dxfaj} \tag{45}$$

the chain rule generates

$$m_{ij} = \sum_k \{m\wedge kj + m_k ky) \tag{46}$$

where **J̶c̶** and m_k are known from the earlier evaluation of the first order partial derivatives, and $k\wedge$ is nonzero only if both nodes i and j are directly connected to node k. So, the term $m_k k\wedge$ is included only when both nodes i and; are directly connected to node k in the computational graph. If indices i and; are decremented from m-1 to 1, m^ is always previously calculated. Here again, nodes i and j are not necessarily the independent variables but may be elementary functions. Second order partial derivatives (43) are obtained when both i and; in equation (46) correspond to the independent variables.

The following procedure evaluates the second order partial derivatives of function (32).

```
For j = m-1, m-2, ... , 1                                    (47)
  For i = m-1, m-2, ... , j
      $m_{ij} = \sum_k \{ m_{ik} k_j + m_k k_{ij} \}$
              where ij e C(k) and ij € C(m)
  End i
End j
```

Again, indices i and j are decremented. We note that index i is decremented from m-1 to j, not to 1, because indices i andy are exchangeable for the second order partial derivatives.

Let us consider Example 3 again to obtain the second order partial derivatives of function (38). The second order partial derivatives of each elementary function with respect to its own variables (its children) are easily determined. These are

$$^{8}7.7 = \circ> \quad ^{8}7{,}6 = !' \quad ^{8}6{,}6 = 0, \tag{48}$$

$$?4{,}4 = \circ' \quad ^{7}4{,}2 = 1. \quad 7_{2,2} = 0,$$

$$\wedge 5{,}5 \wedge 0, \quad 653 = 1, \quad 633 = 0,$$

$$5_{2,2} = 0, \quad 5_{2J} = 1, \quad 5_{U} = 0$$

With the second order partial derivatives of elementary functions and the results from the evaluation of the function and its gradient components - equations (48), (39), (41) and (42), procedure (47) can be applied to obtain **the** second order partial derivatives of function (38). Table 1 shows the sequential procedure for the second order partial derivatives. The three columns show the calculated partial derivative, the contents of calculation and the value of the derivative, respectively.

The second order partial derivatives of function (38): 8_U, 8_{21}, 8_{3j}, $8_{4>1}$, $8\wedge_2$, 8_{32}, 8_{42}, 8_{33}, 8_{43} and 8_{4f4} have exact values for the given values of the *xfs.*

Higher Order Partial Derivative

Similarly, we can construct the procedure for the third order partial derivatives of function (32) as follows;

```
For k = m-1, m-2, ... , 1                                         (49)
  For j = m-1, m-2, ... , &
    For i 35 m-1, m-2, ... , j
       mijk = X * mVl lfc + m,d lJk + mJl lik + m/ liJk *
              where ijjc e C(0 and ij\k & C(m)
    End i
  End]
End k
```

The above numerical scheme can be easily extended up to any order of partial derivatives. If we know partial derivatives of the elementary functions up to the n-th order, we can obtain the n-th order partial derivatives of a composite ftinction. Prerequisite for the above method is only information about relations between elementary functions and their partial derivatives.

Differentiating Equations and Generating the Jacobian Matrix

In the previous section, we discussed and presented an exact numerical differentiation method which evaluates a function and its partial derivatives up to any order. In this section, we will discuss how the

Table 1: Procedure for the Second Order Partial Derivatives of Example 3

derivative	contents	value
87,5	*he*65	3
$8_{6(5}$	86.565	o
85.5	$^{8}{}_{5,6}6_{5}$	o
$8_{7,4}$	$8_{7,7}7_{4}$	0
$8_{6}.4$	$^{8}6.7^{7}4$	2
85,4	$^{8}{}_{5,7}7_{4}$	6
$8_{4,4}$	$8_{4>7}7_{4}$	0
$8_{7>3}$	$8_{7>6}6_{3}$	2
$8_{6,3}$	$8_{6,6}\,6_{3}$	0
85,3	$8_{5>6}6_{3}$ + $8_{6}6_{5,3}$	8
$8_{4>3}$	84,663	4
833	$8_{3>6}6_{3}$	0
$8_{7,2}$	$8_{7>7}7_{2}$ + $8_{7,5}5_{2}$	3
$8_{6>2}$	$8_{6>7}7_{2}$ + $8_{6>5}5_{2}$	4
$8_{5,2}$	$8_{5J}7_{2}$ + $8_{5>5}5_{2}$	12
$8_{4>2}$	$8_{4J}7_{2}$ + $8_{7}7_{4>2}$ + $8_{4(5}5_{2}$	12
$8_{3>2}$	$8_{3>7}7_{2}$ + $8_{3>5}5_{2}$	16
$8_{2>2}$	82,772+8^52	24
87,1	$^{8}7.5\ ^{5}1$	6
8_{64}	$8_{6>55l}$	0
85,1	85,5$,	0
$8_{4>1}$	$84._{5}5i$	12
83.1	8^5!	16
$8_{2)1}$	$8_{2},5\ 5i$ + $8_{5}{}^{5}2,l$	48
8_{U}	8^5!	0

numerical differentiation method can be used in differentiating equations and generating the Jacobian matrix.

Consider the following equation

$$f(z) = 0 \tag{50}$$

where z is the n-dimensional vector of variables and f is a scalar function which is assumed to be sufficiently differentiable. The Newton formula of equation (50) is

$$\frac{\partial f}{\partial z} \Delta z = -f(z) \tag{51}$$

In the above equation, $\frac{\partial f}{\partial z}$ is the first order partial derivatives of equation (50). It can be easily obtained with the numerical differentiation method: procedure (37). The right hand side term can be also evaluated by procedure (33). The only unknown variable in equation (51) is Δz. The coefficient of the unknown variable Δz in equation (51) is the same as elements of the Jacobian matrix of equation (50).

Differentiating equation (50) generates

$$f'(z, z') = \frac{d}{dz} f(z) = \frac{\partial f}{\partial z} = 0 \tag{52}$$

The Newton formula of equation (52) is

$$\frac{\partial f'}{\partial z} \Delta z + \frac{\partial f'}{\partial z'} \Delta z' = -f'(z, z') \tag{53}$$

From equation (52),

$$\frac{\partial f'}{\partial z} = \frac{\partial}{\partial z} \left(\frac{\partial f}{\partial z} \right) = \frac{\partial^2 f}{\partial z^2} \tag{54}$$

and

<£ = £{£/}=£ (55)
3/ 3/ dz dz

If variable / is known, equation (53) can be solved for unknown variables Az and *Az'* because the coefficients *%-* %* and the right hand side term of equation (53) can be calculated from equations (54),
at at
(55) and (52) respectively. We note that the first and the second partial derivatives in those equations can be obtained using procedures (37) and (47). In equation (55), ^ is the same as the first order partial derivatives $\frac{\partial f}{ds}$. This resemblance has been observed in previous chapters. Again, the coefficients of the unknown variables Az and Az' in equation (53) are the same as elements of the Jacobian matrix of equation (52).

Differentiating equation (52) generates

$$f''(z,z',z'') = \frac{d}{dt}f'(z,z') = \frac{\partial f'}{dz}z' + \frac{\partial f'}{dz'}z'' = 0 \tag{56}$$

where ^ and ^ are known from equations (54) and (55). The Newton formula of equation (56) is

$$\frac{\partial f''}{dz}\Delta z + \frac{\partial f''}{d/}\Delta z' + \frac{\partial f''}{\partial z''} \text{[illegible]} \quad (57)$$

From equations (56), (54) and (55),

$$\frac{\partial f''}{\partial z} = \frac{\partial}{\partial z}\{\frac{\partial f'}{\partial z}z' + \frac{\partial f'}{\partial z'}z''\} = \frac{\partial}{\partial z}\{\frac{\partial^2 f}{\partial z^2}z'^2 + \frac{\partial f}{\partial z}z''\} \tag{58}$$

$$- \frac{dv\,\_y?}{\partial z^3} \quad \frac{df}{dz*} ,,$$

$$\frac{\partial f''}{\partial z'} = \frac{\partial}{\partial z'}\{\frac{\partial^2 f}{\partial z^2}n + \frac{<l_z'')}{\partial z} = 2 \wedge z' \tag{59}$$

and

$$\frac{\partial f''}{\partial z''} = \frac{\partial}{\partial z''}\{\frac{\partial^2 f}{\partial z^2}z'^2 + \frac{\partial f}{\partial z}z''\} = \frac{\partial f}{\partial z} \qquad (60)$$

With given or known z", everything except unknown variables Az, A/ and A/' can be calculated with the results from the previous steps and the numerical differentiation method: procedure (49). As expected, the coefficients of the unknown variables Az_9 Az' and Az" in equation (57) are also the same as elements of the Jacobian matrix of equation (56). In the above, values for / and V are assumed to be given or known. Actually, they are calculated from the newton equations through iteration.

In this section, we have discussed and showed how an equation is differentiated and how its newton equation is generated. To carry out the above method, only the results from numerical differentiation are needed. Using this method, we can differentiate an equation as many times as we want, and we can get the newton equations of the differentiated equations. We can also obtain the Jacobian matrix of the differentiated equations because the coefficients of the unknown variables in the newton equations of the differentiated equations are always the same as elements of the Jacobian matrix.

Example 4

Let us consider the following equation:

$$x_1 x_2^2 x_3 x_4 = 0 \qquad (61)$$

In Example 3, the function value and partial derivatives of equation (61) were evaluated when x_1 = l_t *2 « 2,*3 = 3andx_4 = 4.

Using equation (51) and the results of Example 3, the newton formula of equation (61) at the same point as in Example 3 can be obtained as follows.

$$48\ Ax_x + 48\ AX2 + 16\ Ax_3 + 12\ Ax_4 = -48 \qquad (62)$$

The Jacobian matrix of equation (61) for variables x_v x_v x_3 and x_4 is [48,48,16,12] whose elements are the same as coefficients of the Ax^s here.

When equation (61) is differentiated with respect to time, its newton equation can be obtained from equation (53). Guessing *x{* = 1, *x{* = -1, x_3' = 1 and x_4' = 1, equation (53) for this example becomes

$$-20\,\Delta x_1 + 52\,\Delta x_2 + 4\,\Delta x_3 + 4\,\Delta x_4 \tag{63}$$
+ 48 Ax/ + 48 Ax/ + I6AX3' + 12Ax$_4$' » -28

The Jacobian matrix of equation (61) after the first differentiation for variables z$_{1f}$ *x^$_{\%}$* x$_3$, JC$_4$, *x$_x$\ x{$_{\%}$ xj* and *x$_4$'* is [-20* 52, 4, 4, 48, 48, 16, 12] whose elements are also the same as coefficients of Ax/s and Ax/*s. The values for variables *x$_x$\ x{, x{* and x$_4$' are calculated firom the newton equation, equation (63), through iteration.

The newton formula and the Jacobian matrix of equation (61) after the second differentiation can be also obtained from equation (57).

Solving and Propagating Index and Near Index Problems

We have discussed and presented a numerical method which can (1) detect the singularity or the near singularity of the Jacobian matrix and identify the equations responsible for the singularity or the near singularity of the Jacobian matrix and (2) differentiate equations and generate their Jacobian matrix and newton equations. The remaining issue for implementing the solution algorithm for index and near index problems is how to solve the augmented set or the final set of equations and how to propagate values along time.

The solution algorithm for index and near index problems solves the augmented set or the final set of equations as a set of algebraic equations. Solving a set of algebraic equations is usually carried out by iteratively solving the corresponding set of the newton equations. When an index or a near index problem is given, the numerical method discussed above results in a set of the newton equations for the augmented or final set of equations. Using the exact numerical differentiation method, all the coefficients of the unknown variables and the right hand side term in the newton equations have numeric values. In addition to this numerical characteristic, the set of the newton equations always nonsingular with respect to the unknown variables because the algorithm guarantees the nonsingularity of the augmented or the final set of equations. We can obtain the solution of the index or the near index problem just by iteratively solving the set of the newton equations and propagating a part of the original differential equations.

Suppose an augmented set of equations consists of *n* equations and *m* variables in solving an index

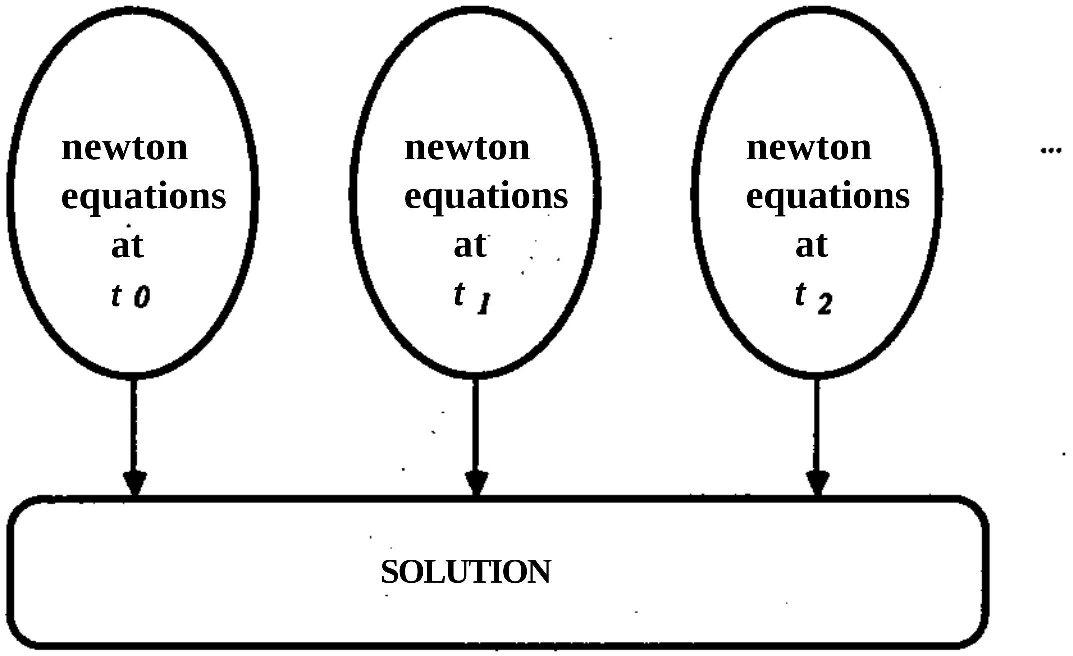
newton equations at t_0
newton equations at t_1
newton equations at t_2
...
SOLUTION

Figure 4: Solution of an Index Problem with Zero Degrees of Freedom

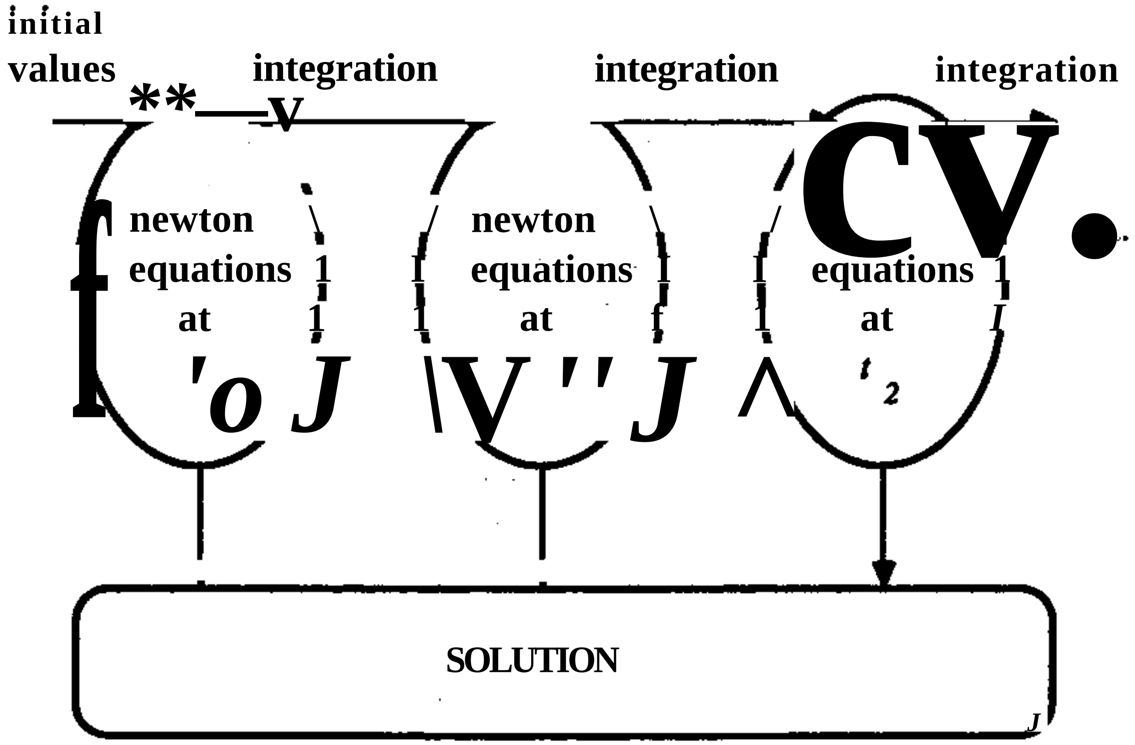

Figure 5: Solution of an Index Problem with Nonzero Degrees of Freedom

problem. The corresponding set of the newton equations also has *n* equations and *m* unknown variables. According to the solution algorithm for index and near index problems, *m* is always greater than or equal to *n* if it is a well posed problem. When *m* is equal to n, that is, when the degrees of freedom is zero, a nonsingular set of the newton equations has to be solved at each time step including the initial point. There will be no need to integrate any differential equations. The nonsingular set of the newton equations can be easily solved by existing algebraic equation solvers. Figure 4 shows the solution procedure for an index problem for which the augmented set has zero degrees of freedom. When *m* is greater than *n,* the degrees of freedom is not zero. At the initial point, *m-n* initial values have to be assigned. At each time step during propagation, *m-n* values must be given to solve the augmented set of equations. This requires integration of *m-n* differential equations to reduce the degrees of freedom to zero. Existing ODE/DAE solvers can be used to integrate the differential equations. We note that the algorithm presented in this thesis can be used with any integration scheme. The remaining *n* equations are treated as algebraic equations and solved by an existing algebraic equation solver. Figure 5 shows the solution procedure when there are one or more degrees of freedom.

To solve a near index problem, a final set of equations is generated according to the user specified error tolerance. The solution procedure for the final set of equations is almost the same as that for the augmented set of an index problem . The only difference is that values must be assigned to a set of nonpivot variables for the final set of equations at each time step. Figure 6 shows the solution procedure for a near index problem with *m-n* degrees of freedom.

Conclusion

In this paper, we have discussed implementation of the solution algorithm for index and near index problems proposed by Chung and Westerberg [3,4]. We have presented a numerical singularity checking method to detea the singularity or the near singularity of the Jacobian matrix and to identify the equations to be differentiated. We have also presented an exact numerical differentiation method to differentiate equations and to generate the Jacobian matrix. The numerical method presented in this thesis results in a set of the newton equations for the augmented or final set of equations. Existing algebraic equation or ODE/DAE solvers can be used to solve the resulting set of the newton equations at each time step or to integrate only a part of the differential equations between two adjacent time steps. With the numerical method suggested in this thesis, index and near index problems can be numerically and automatically solved.

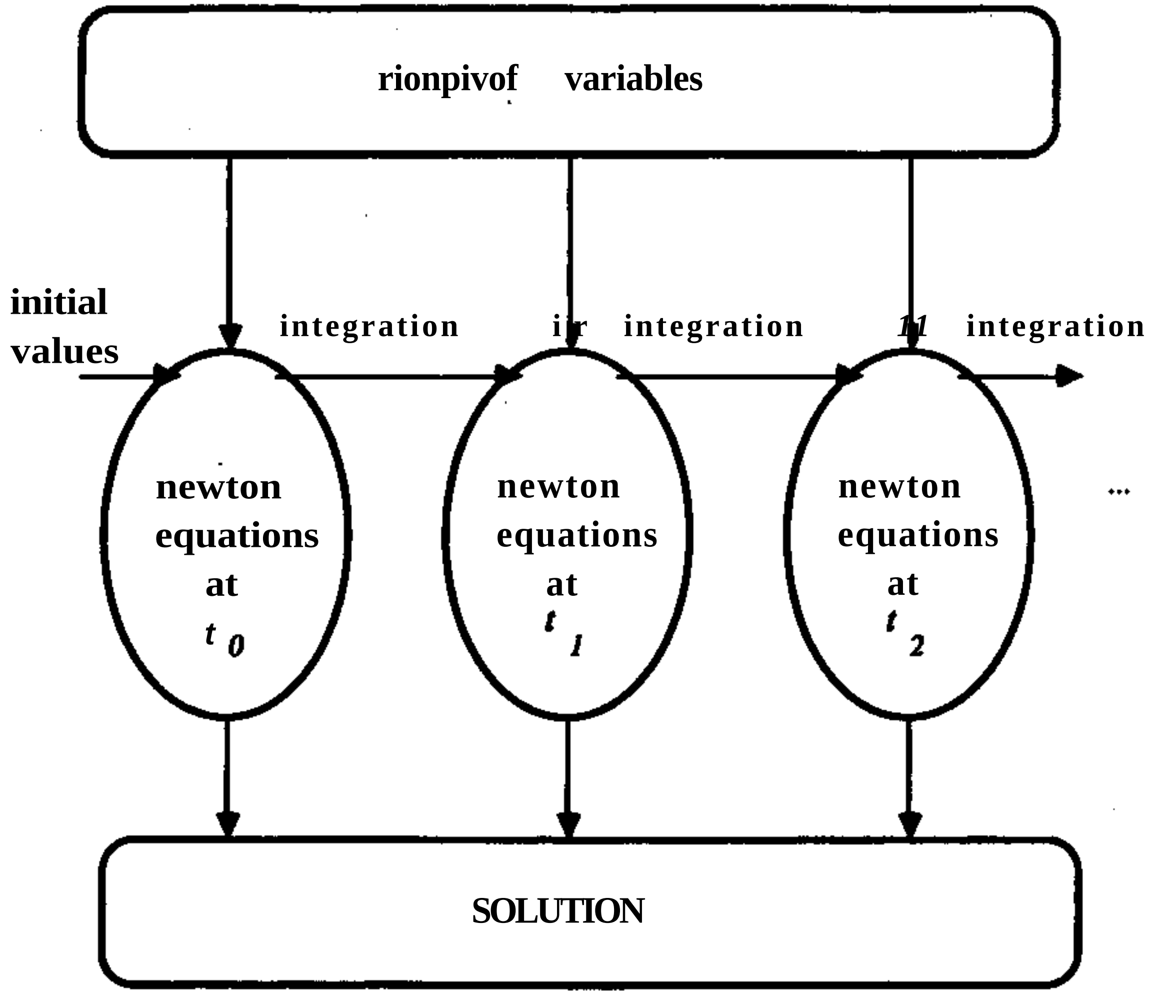

Figure 6: Solution of a Near Index Problem with Nonzero Degrees of Freedom

Acknowledgement

Support for this work comes from the National Science Foundation through its support for Design Research Center, NSF Grant EDC-8943164.

References

[1] Bachmaim, R., L. Brifll, U. Pallaske, and Th. Mrziglod.
A Contribution to The Numerical Treatment of Differential Algebraic Equations Arising in **Chemical Engineering.**
***Dechema-Monographs vol.116 - VCH Verlagsgesellschcft* :343-349,1989.**

[2] Bachmann, R., L. Brüll, Th. Mrziglod, and U. Pallaske.
On Methods for Reducing the Index of Differential Algebraic Equations.
Comput. Chem. Engng. 14(11):1271-1273,1990.

[3] Chung, Y. and A.W. Westerberg.
A Proposed Numerical Algorithm for Solving Nonlinear Index Problems.
Ind. Eng. Chem. Res. 29(7): 1234-1239,1990.

[4] Chung, Y. and A.W. Westerberg.
Solving Stiff DAE Systems as 'Near' Index Problems.
***Submitted to Ind. Eng. Chem. Res.* 0:. 1991.**

[5] Gear, C.W.
Differential-Algebraic Equation Index Transformation.
SIAMJ. SCI. STAT. COMPUT. 9(l):39-47, January, 1988.

[6] Gear, C.W. and L.R. Petzold.
ODE Methods for The Solution of Differential/Algebraic Systems.
SIAMJ. NUMER. ANAL. 21(4):716-728, August, 1984.

[7] Griewank, A.
On Automatic Differentiation.
***Preprint MCS-P10-1088. Argonne National Laboratory,* October, 1988.**

[8] Pantelfdes, C.C.
The Consistent Initialization of Differential-Algebraic Systems.
***SIAMJ. SCI. STAT. COMPUT.* 9(2):213-231,1988.**

[9] Ponton, J.W.
The Numerical Evaluation of Analytical Derivatives.
Comput. Chem. Engng. 6(4):331-333,1982.