AN EFFICIENT ALGORITHM USING HOUSEHOLDER'S FORMULAS FOR THE SOLUTION OF FAULTED POWER SYSTEMS

by

ARMANDO ALTAMIRANO CHAVEZ

A MASTERS REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by :

Dr. Anil Pahwa
Major Professor

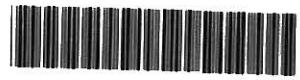

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I wish to express deep gratitude to Dr. Anil Pahwa for his guidance and great help throughout this Master's report and my graduate studies.

My appreciation is also extended to Dr. Janice Honeyman for her generous time and assistance with this work.

I also want to thank Dr. Gary Johnson and Dr. Richard Greechie for agreeing to be members on my commitee.

CHAPTER 1

INTRODUCTION

One of the purposes of a protection scheme of a power system is to ensure the maximum possible service continuity with minimum system disconnection. When a fault occurs the power system goes temporarily out of its normal balanced condition until the corresponding protective devices act. If the faulted equipment is not promptly disconnected from the rest of the system, damage may result to other parts of operating equipment. The fault conditions are detected by continuous monitoring of current, voltage, power, frequency, and impedance at various points of the system. The most significant ones are voltages and currents. It is important to determine the values of system voltages and currents during fault conditions so that protective devices may be set to detect and minimize the harmful effects of such contingencies [1].

Several methods for computer analysis of faulted systems have been developed since the advent of computers [1,2,3,4]. A detailed list of references is available in [1,2,3]. However, there has been a need for continuous development of new and more efficient methods to keep up with the computer technology. This has been more true in the last few years because of rapid advances in personal

computer technology. The objective of this report is to develop an algorithm for calculating voltages and currents of a faulted system for implementation on a personal computer. Simplicity and efficiency are the two main features of the proposed algorithm.

The faults to be considered are the most common ones: balanced three phase, line to line, double line to ground, and single line to ground. The method developed in this report is based on Householder's work on matrix methods [5] and later application of these methods for fault computation in analog circuits by Pahwa and Rohrer [6]. The formulas presented in reference [6] for open a short circuit between two nodes are used to simulate connections among the three sequence networks in order to get the voltages and currents in the faulted power system. The connections and modifications made over the sequence networks are illustrated in a simple and logical manner. A computer program to generate the solution for a faulted system based on the proposed algorithm is developed. The program is written in Turbo Pascal, which is a dialect of Pascal and very popular on personal computers. Pascal facilitates writing structured programs and Turbo Pascal provides several extensions that make it easy to use [7]. Following the program an example is considered as an illustration of the method. The results are discussed and suggestions for some applications of the method are made.

CHAPTER 2

ALGORITHM FOR FAULT CALCULATIONS

Faults of many types and causes may appear on electric power systems. Moreover, these faults are unpredictable with regards to both time of occurrence and location. In order to predict the performance of a protective scheme it is necessary to know what the fault conditions will be. When a study over a power system is made to observe its behavior during a fault, it is necessary to assume the location of the fault and to know the impedance of the lines, transformers, and generators of the system. The results of a study provide the engineer all the information needed for setting the protective devices.

2.1 Prefault Conditions.

The algorithm developed in this report assumes standard simplifications with respect to prefault conditions. These simplifications do not significantly affect the accuracy of the results and, at the same time, they avoid unnecessary complications.

i) Series resistances are neglected: Although an impedance consists of a resistance and a reactance, it is sufficient to take only the reactance into consideration in fault conditions.

ii) Shunt elements are neglected: This includes shunt capacitances of the transmission line model and leakage conductances.

iii) Loads are neglected: This implies that the voltage at all points in the system can be considered to be 1 p.u.

Thus, summarizing the prefault conditions to be uitlized:

$$V(i) = 1 \angle 0 \qquad \text{for all } i$$

$$I(i,j) = 0 \qquad \text{for all } i, j$$

where V (i) represents the phasor voltage in each bus of the system and I (i,j) is the phasor current flowing from bus i to bus j.

2.2 Types of faults.

It is useful to distinguish between shunt faults and series faults. A shunt fault is an unbalance between phases or between phase and neutral. A series fault is an

unbalance in the line impedances and does not involve the neutral or ground, nor does it involve any interconnection between phases [2]. This report will study only shunt faults, which are balanced three phase, line to line, double line to ground, and single line to ground. The description of each fault and its analysis is given in the following chapter. It is assumed that the fault impedance is zero. However, extension of the algorithm to include non zero fault impedance will not be very difficult. It is also assumed that the faults occur on the buses. Faults elsewhere, such as the middle of a the transmission line are not considered. Again it is not very difficult to extend the algorithm to include such faults.

2.3 The Sequence Impedance Matrices.

The report shall use the bus impedance matrices for the fault analysis problem. The bus admittance matrix which is easily formed will be created first and then Shipley's Inversion Method [3] will be used to invert it and get the bus impedance matrix for each sequence network. In Shipley's Inversion Method an operation that is referred to as pivoting is performed on each major diagonal in any sequence. When the process has been completed, the inverse replaces the original matrix [3]. Instead of using this method one could form the impedance matrices directly using

any Zbus building algorithm [1,2,3]. For convenience +j has been taken common out of all the entries of the impedance matrix.

Thus the first step is formation of the bus admittance matrix for the three sequence networks under prefault conditions. For a general case these three networks appear as shown in figure 1, where n is the number of buses in the system and r1, r2 and r0 are the reference buses for positive, negative and zero sequence networks, respectively. It is considered here that all reference buses are connected to ground. Hence, we get three n x n sequence impedance matrices with ground as the reference. It is also assumed that the system configuration is such that its zero sequence network does not have any buses without a path to the reference. This path could be through any number of branches of the system. If there are buses which do not have a path to the reference then the Ybus matrix turns out to be singular and thus its Zbus matrix does not exist. This situation is similar to that of connecting an impedance between two new buses in Zbus building algorithm as is explained in reference [1]. Such a situation is very rare for practical systems and therefore it will not be dealt with in this report. However, it can be handled by sub-dividing the zero sequence network into several networks based on connectivity and representing each sub- network by a sub-matrix.

Note that under prefault conditions all the buses in the positive sequence network have voltage of 1 p.u., whereas the buses of negative and zero sequence networks have zero voltage. Also, note that capital letters represent matrices and lowercase ones represent elements. So, Vl is a voltage vector of positive sequence, Z2 is the negative sequence impedance matrix, Z0 is the zero sequence impedance matrix, rl is the reference for positive sequence, r2 is the reference for negative sequence, r0 is the reference for zero sequence, z0 (2,5) represents the entry at location (2,5) in the zero sequence network, zl (1,2) represents entry at location (1,2) in the positive sequence matrix, and so on.

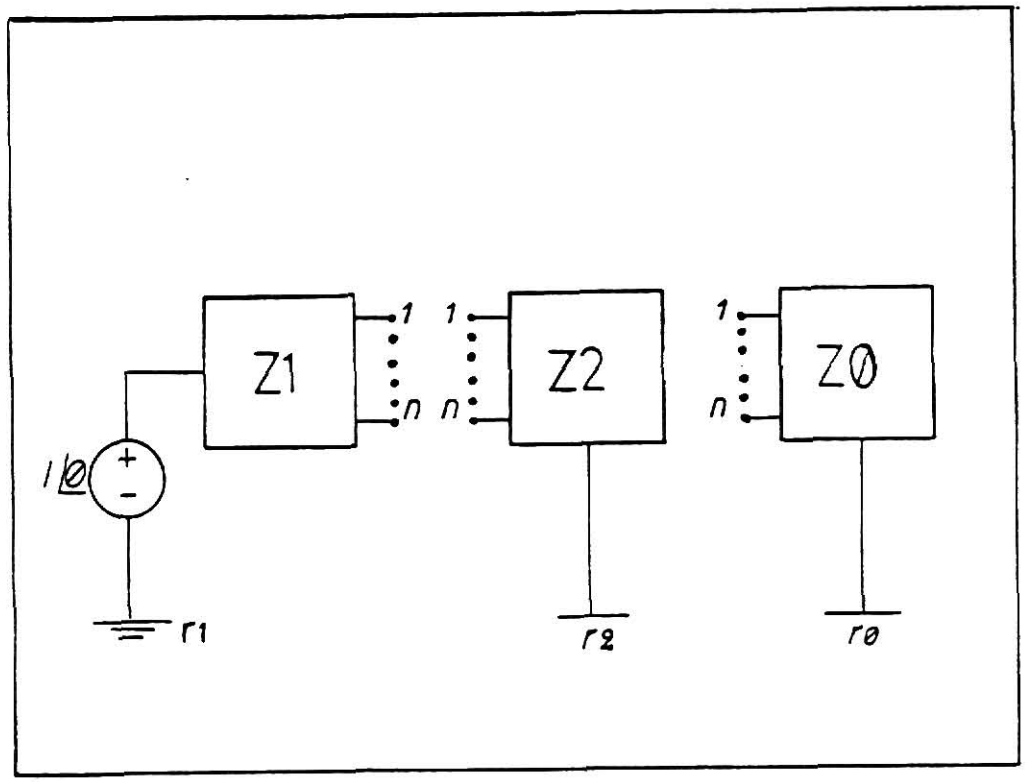

Figure 1.- System sequence networks before their interconnections to simulate faults.

2.4 The Householder's Formulas for Fault Simulation.

In the analysis of faults, the interconnections to be done with the sequence networks is the key for the solution of a faulted system. For a general circuit, application of Householder's formula gives a direct method of simulating open and short circuit faults in elements of the circuit [6]. According to this method if prefault voltages are known then the post-fault voltages can be calculated from:

$$Vopen = [\, Id + G\, Z\, X\, X^{t} / [\, 1 - G\, X^{t}\, Z\, X\,]\,]\, V \qquad (1)$$

$$= A\ V$$

$$Vshort = [\, Id - Z\, X\, X^{t} / X^{t}\, Z\, X\,]\ V \qquad (2)$$

$$= A\ V$$

where,

V is the voltage vector before the fault,

Vopen is the voltage vector after an open circuit occurs,

Vshort is the voltage vector after a short circuit occurs,

Id is an identity matrix,

Z is the impedance matrix of the pre-fault circuit,

G is the admittance of the open line,

X is the connectivity vector having in general its 'a' entry equal to 1 and its 'b' entry equal -1

consideringthat the fault is between nodes 'a' and 'b'. The rest of the terms are zero. If node 'b' is ground then the 'b' entry has value 0.

Note that under pre-fault conditions for any circuit V = Z I, where I is the vector of currents or equivalent currents entering the buses. Therefore, if I stays the same for the faulted case, then:

$$V_{fault} = A Z I = Z' I \qquad (3)$$

Hence, Z' is the impedance matrix of the modified circuit.

In power systems the faulted condition is analyzed with the help of proper connections among the sequence networks. Thus,a given power system fault can be simulated using the Householder's formulas repeatedly starting from the pre-fault circuit shown in figure 1. To achieve the faulted circuit configuration certain points in the circuits of figure 1 are shorted and opened in a proper order, which is dependent on the type of fault and it is selected in such a way that the computations are minimized.

Now, under pre-fault conditions the composite system of figure 1 can be represented by a block diagonal matrix of size three times (or three times plus one in a special

case, which will be discussed later) the size of each sequence network.

This matrix is:

$$Z = \begin{bmatrix} Z1 & 0 & 0 \\ 0 & Z2 & 0 \\ 0 & 0 & Z0 \end{bmatrix} \qquad (4)$$

Thus, in the composite system, node numbers 1 to n represent the system buses in the positive sequence network, n + 1 to 2n represent the system buses in the negative sequence network, and 2n + 1 to 3n represent the system buses in the zero sequence network. Thus, if one is considering fault on bus k, then it will mean faulted nodes in the composite system are k, n + k, and 2n + k. For the presentation of the method it is necessary to start with this composite matrix but it will be shown later that because of the decoupled nature of the problem the implementation of the method does not require formation of this composite matrix.

Upon application of Householder's formulas to simulate connection between the sequence networks this composite matrix will get modified in each step. Note from figure 1 that zero and negative sequence networks do not have sources and therefore connections between them will not result in flow of currents. Flow of currents takes place only when connection to positive sequence network is made. Also note

that until connection to positive sequence network is made the voltages in zero and negative sequence networks remain zero and similarly the voltages at the nodes of positive sequence network remain $1 \angle 0$. Therefore, to reduce computation, connection to the positive sequence network is done as the last step.

Thus, the procedure for fault computation is as follows:

i) If only one step is needed for fault simulation then matrix A is calculated for that fault.

ii) If more than one step is needed for simulation then Z' for intermediate steps is calculated and in the final step matrix A is calculated.

Recall that the last step will always be connection to the positive sequence network of the rest of the network. Therefore, computation of intermediate voltage vectors is not necessary because they will have 1's in their first n rows (n is the number of buses of the system), and 0's in the remainder of their rows until the connection to positive sequence network is done, which is the last step.

Now, the calculation of the sequence voltages after the occurrence of a fault requires multiplication of A with V. Because of the nature of the vector V, the multiplication process can be accomplished by simple addition of the first 'n' terms of each row of matrix A.

CHAPTER 3

THE SEQUENCE NETWORKS AND THEIR CONNECTIONS

In this section the power system faults will be considered individually and method for connection of sequence networks to simulate these faults will be presented. Also, steps used in developing the computer program are presented. It will be assumed that the system has 'n' buses.

3.1 Three Phase Fault.

Here, only the positive sequence network is needed. For a fault on bus ' k ' figure 2 represents the faulted network.

It must be noted that the composite matrix Z is only Z1 for this case, so the size of it is n x n . Moreover, the column vector X has only a value of 1 in its k row and the rest of its rows have value 0. Now, using equation (2):

$$X^{t} \; Z \; X = z1\,(k,k) \qquad (5)$$

and

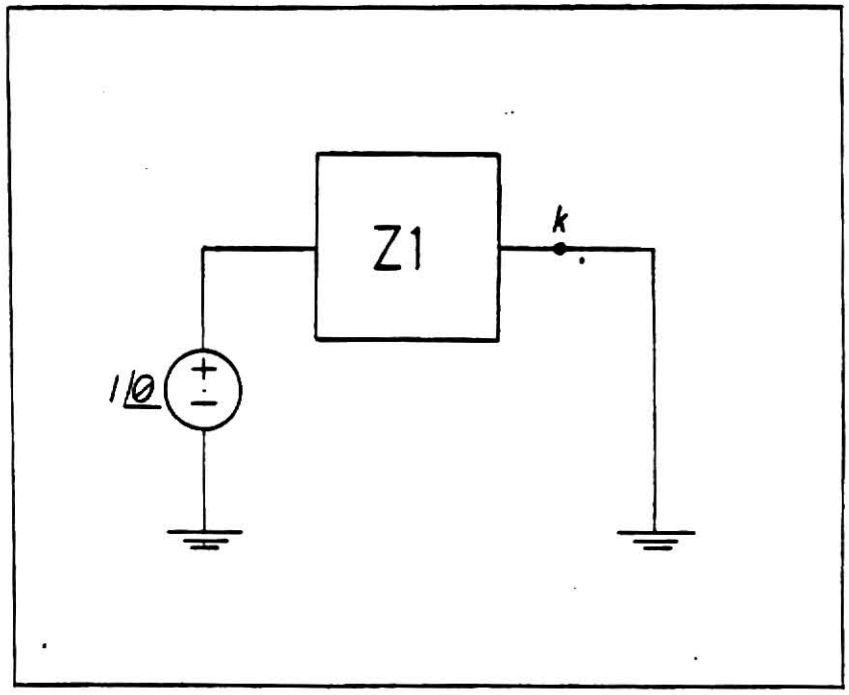

Figure 2.- Equivalent circuit for a three phase fault.

$$Z\,X\,X^{t} = \begin{bmatrix} 0 & \cdots & z1(1,k) & \cdots & 0 \\ 0 & \cdots & z1(2,k) & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & & z1(n,k) & \cdots & 0 \end{bmatrix} \quad (6)$$

column k

Hence,

$$A = [\; Id - Z\,X\,X^{t} / X^{t}\,Z\,X \;] \quad (7)$$

$$= \begin{bmatrix} 1 & 0 & \cdots & -z1(1,k)/z1(k,k) & \cdots & 0 \\ 0 & 1 & \cdots & -z1(2,k)/z1(k,k) & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \cdots & \cdots & -z1(n,k)/z1(k,k) & \cdots & 1 \end{bmatrix}$$

column k

Note that this matrix has 1 in all the diagonal entries except kth entry, which is zero. Other non zero entries are in column k as indicated.

Then, the post-fault voltages can be calculated which are:

$$vl(i) = 1 - zl(i,k) / zl(k,k) ; \qquad i = 1,....n \tag{8}$$

$$vl(k) = 0$$

Thus the standard formula for the computation of the post-fault voltages for a three phase fault has been obtained.

3.2 Two Line Fault.

In this case the simulation of the fault involves two sequence networks, positive and negative. The equivalent circuit for a two line fault is presented by figure 3. Remember, i1, i2, v1, and v2 represent sequence currents and voltages. Let k1 and k2 represent the faulted bus in each sequence network. Thus, if k1 = k then k2 = n + k . The composite impedance matrix in this case contains only Z1 and Z2 and before any modifications it is :

$$Z = \begin{bmatrix} Z1 & 0 \\ 0 & Z2 \end{bmatrix} \tag{9}$$

Now, to short k1 and k2 equation (2) is used. The column vector X is of dimension (2n x 1) and has 1 on its k th row and -1 on its (n + k) th row. The rest of its row have value 0.

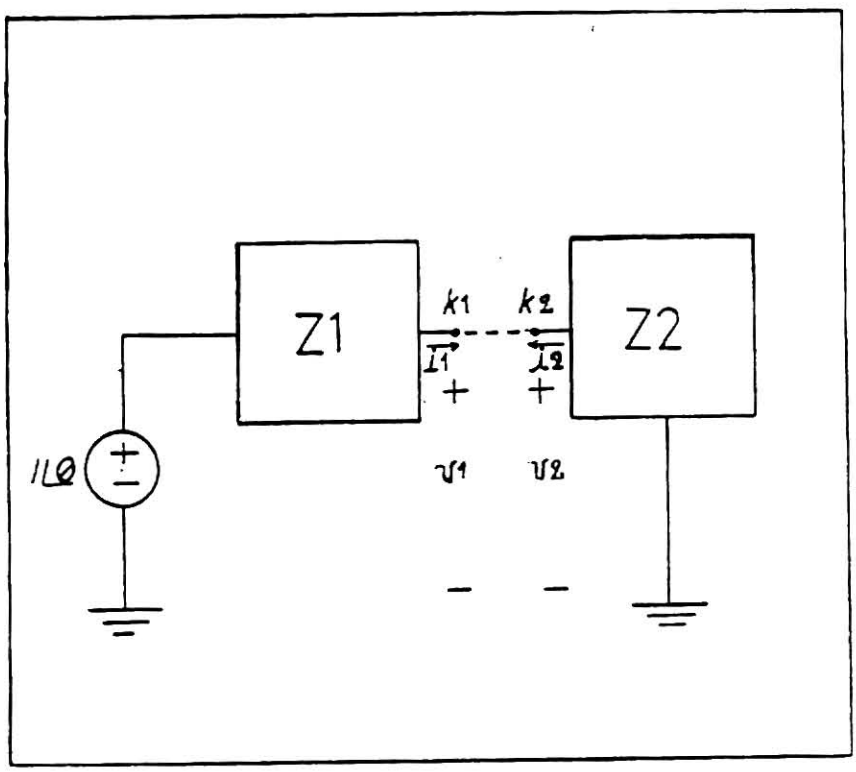

Figure 3.- Equivalent circuit for a double line fault.

With all the information above:

$$X^{t} Z X = z1(k,k) + z2(k,k) \tag{10}$$

and

$$Z X X^{t} = \begin{bmatrix} 0 & \cdots & z1(1,k) & \cdots & -z1(1,k) & \cdots & 0 \\ 0 & \cdots & z1(2,k) & \cdots & -z1(2,k) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & & z1(n,k) & \cdots & -z1(n,k) & \cdots & 0 \\ \vdots & & -z2(1,k) & \cdots & z2(1,k) & \cdots & 0 \\ \vdots & & -z2(2,k) & \cdots & z2(2,k) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & & -z2(n,k) & \cdots & z2(n,k) & \cdots & 0 \end{bmatrix} \tag{11}$$

column k column n + k

$$V^{t} = [\ 1\ 1\ 1\ \cdots\ \underset{\text{column n}}{1}\ 0\ 0\ 0\ \cdots\ 0\ ] \quad (1 \times 2n) \tag{12}$$

Now, upon making appropriate substitutions in (2) the elements of Vshort are obtained. The first n entries of Vshort are the positive sequence voltages and the rest are the negative sequence voltages, such that:

$$V\,short^{t} = [\ V1short^{t}\ \ V2short^{t}\ ] \tag{13}$$

Therefore, the elements of V1short and V2short are:

$$v1(i) = 1 - z1(i,k) / [\, z1(k,k) + z2(k,k) \,] \qquad (14)$$

$$v2(i) = z2(i,k) / [\, z1(k,k) + z2(k,k) \,] \qquad (15)$$

where, $i = 1, \ldots, n.$

Again, the standard formulas are obtained for the post-fault voltages for a line to line fault.

3.3 Two Line to Ground fault.

The two line to ground fault involves the three sequence networks and so the size of the composite matrix is 3n x 3n. However, to minimize computation the steps for the connection of the three sequence networks are done in such a way that the use of the three sequence networks together is left as the last step.

In the first step the negative and zero sequence networks are connected in parallel as shown in figure 4 and then this set is connected in series with the positive sequence network.

Because of this first step, the initial composite matrix Z contains only Z2 and Z0. Thus, the prefault Z matrix looks like:

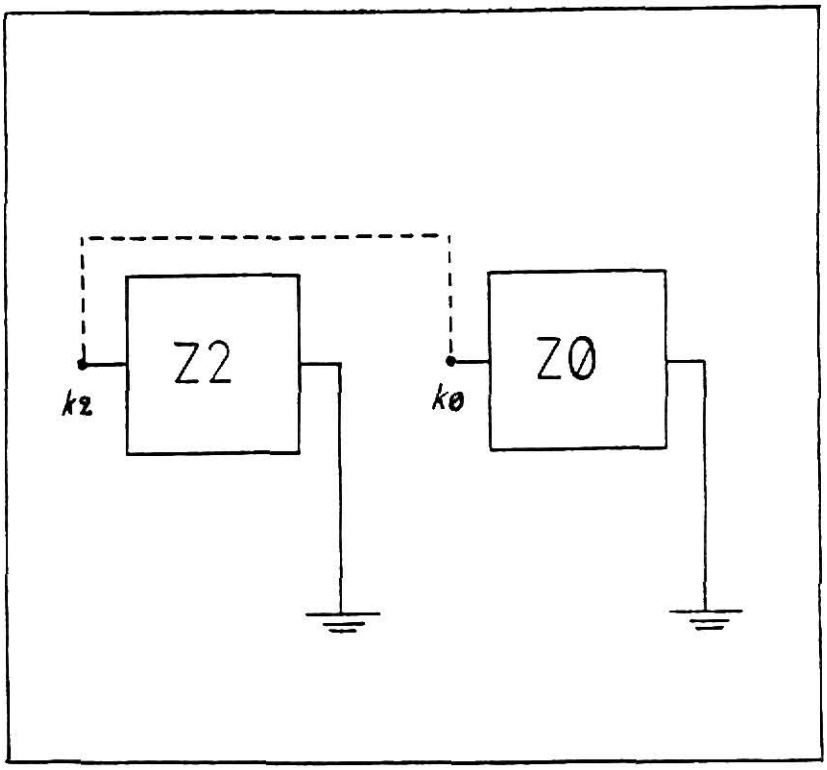

Figure 4.- First step for a two line to ground fault.

$$Z = \begin{bmatrix} Z2 & 0 \\ 0 & Z0 \end{bmatrix} \qquad (16)$$

Now, the connection between k2 and k0 (the faulted buses on negative and zero sequence respectively) is done using equation (2). This results in modifications in matrix Z and let the modified matrix be called Z'.

Note that from equation (3) Z' can be found,which is:

$$Z' = Z - Z X X^t Z / [X^t Z X] \qquad (17)$$

where X is a column vector 2n x 1 in which k row and (n + k) row have values 1 and -1 respectively and rest of the entries are 0.

To find Z' manipulation of (17) can be done in several steps. First, consider that the resulting matrix $Z X X^t Z$ is called Ztemp. Upon expansion the entries of Ztemp are found to be :

$$ztemp(i, j) = z2(i,k) * z2(k,j) \qquad (18)$$

$$ztemp(i, n + j) = - z2(i,k) * z0(k,j) \qquad (19)$$

$$ztemp(n + i, j) = - z0(i,k) * z2(k,j) \qquad (20)$$

$$ztemp(n+i, n+j) = z0(i,k) * z0(k,j) \qquad (21)$$

where i and j are integers and have values from 1 to n.

Besides, as was found in (10) the calculation of the value $X^t Z X$ is simple. In this case, this value involves negative and zero sequence values as is presented by (22).

$$X^t Z X = z2(k,k) + z0(k,k) \qquad (22)$$

Thus upon substitution of proper terms in equation (17) Z' is found. Note that this connection has not changed the pre-fault voltages.

Figure 5 represents the final circuit for a two line to ground fault. The implementation of this step is simple. This case looks like a two line fault if Z2 is replaced by Z' in figure 3. Figure 6 represents this similarity. In this case Z2 and Z0 are replaced by Z'. Of course, the size of Z' is 2n x 2n.

Thus, the sequence voltages after fault are calculated directly applying (2), (10), (11), and (12) . Making the simplifications it is found that :

$$v1(i) = 1 - z1(i,k) / [\, z1(k,k) + z'(k,k) \,] \qquad (23)$$

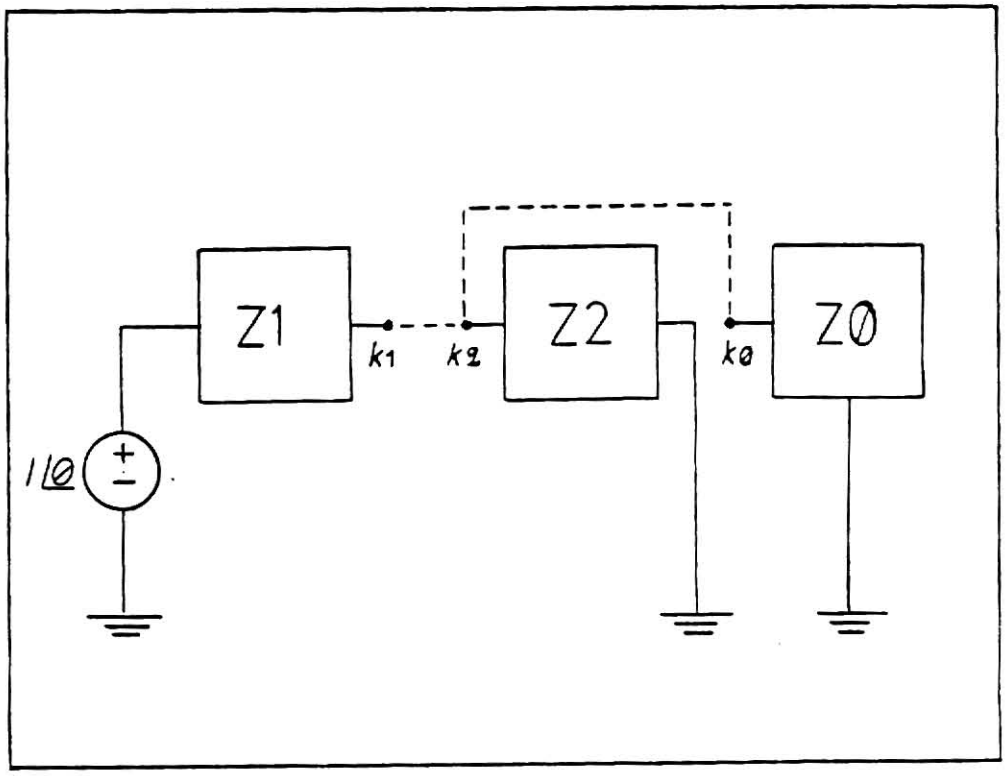

Figure 5.- Equivalent circuit for a two line to ground fault.

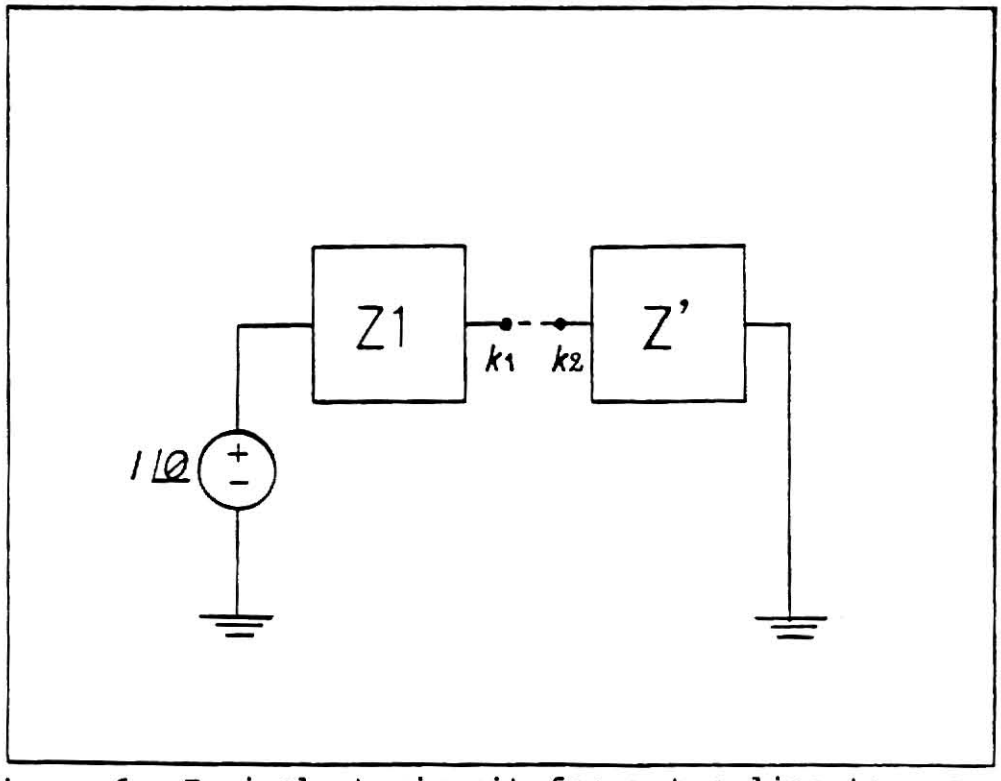

Figure 6.- Equivalent circuit for a two line to ground fault using Z'.

$$v2(i) = z'(i,k) / [\, zl(k,k) + z'(k,k) \,] \quad (24)$$

$$v0(i) = z'(n + i,k) / [\, zl(k,k) + z'(k,k) \,] \quad (25)$$

Note that the final formulas for calculating the sequence voltages of a two line to ground fault have a similarity with those for calculating three phase and line to line fault.

3.4 Single Line to Ground Fault.

For this type of fault the three sequence networks are connected in series through the faulted bus for each sequence as shown in Figure 7. Note that in this figure the references for negative and zero sequence networks are not connected to the ground. However, as far as series connections are concerned, they can be arranged as shown in figure 8 without affecting the magnitude of the currents and voltages of the system. The only difference that arises out of this rearrangement is a phase shift of 180 degrees in the currents and voltages in the negative and zero sequence circuits.

With the above mentioned rearrangement the reference node of zero sequence network gets connected to ground but

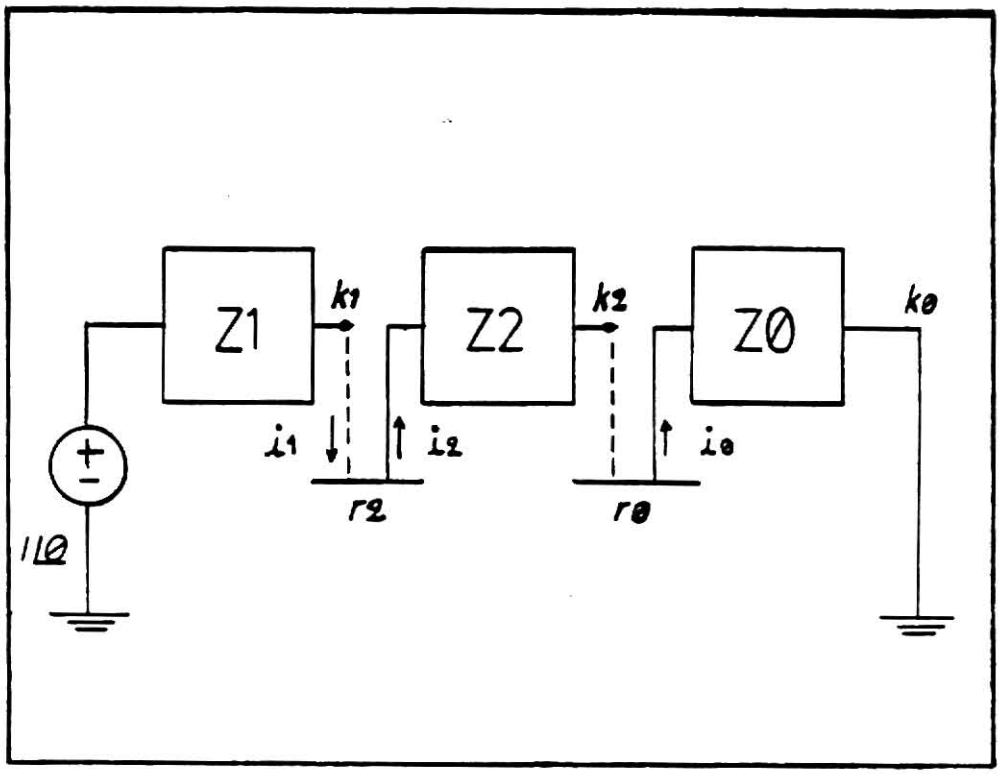

Figure 7.- Equivalent circuit for a single line to ground fault.

that of negative sequence network is still not connected to ground. Recall that under pre-fault conditions all the references are considered to be connected to ground. Also, these references remained connected to ground under fault conditions for the faults considered so far. Therefore, for a single line to ground fault a step for open circuit between r2 and ground will be incorporated in the algorithm. However, this step must be selected carefully to avoid excessive computation.

Hence, the first step is to create a new negative sequence network (Z2') such that its reference is connected to ground through a 1 p.u. dummy reactance. Introduction of this dummy reactance is necessary because the algorithm can not simulate open circuits between two points with zero impedance, such as the short circuit between r2 and ground. A value of 1 p.u. for the dummy reactance is chosen because it is easy to manage during the calculations. Figure 9 presents this variation.

The matrix Z2' involves creation of a new node. So, the size of Z2' is (n + 1) x (n + 1) but it is not much different from Z2. All the values of the (n + 1) column and (n + 1) row of Z2' are 1. The remaining elements of Z2' are the same elements of Z2 plus 1.

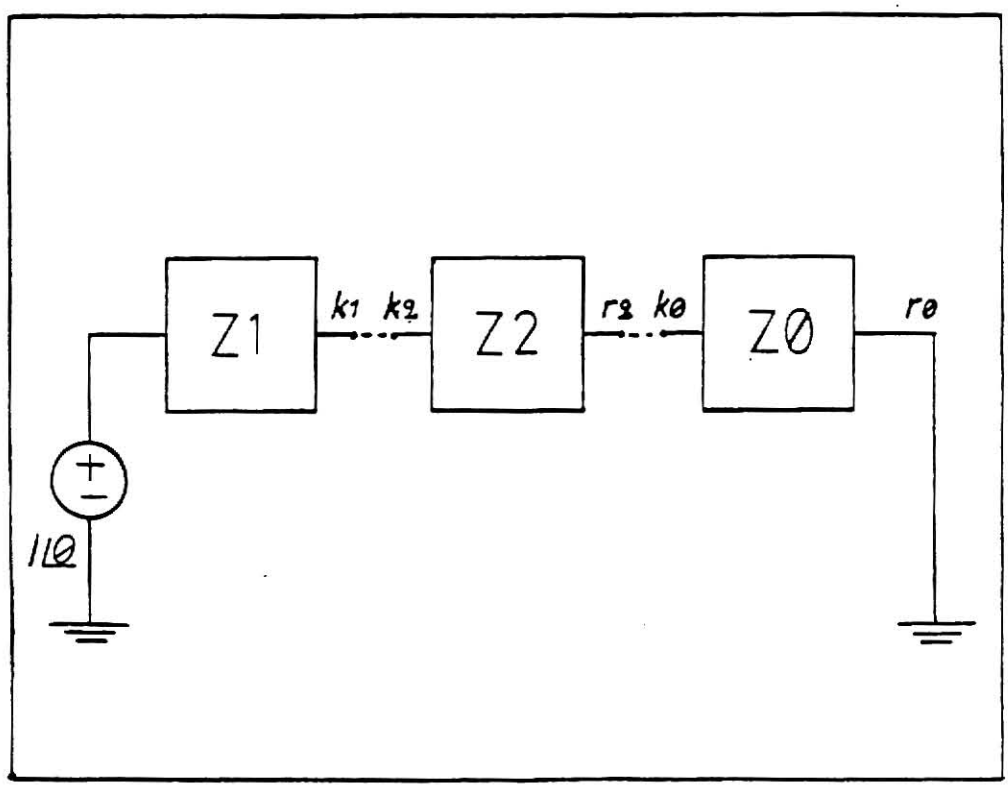

Figure 8.- Modification to circuit of figure 7.

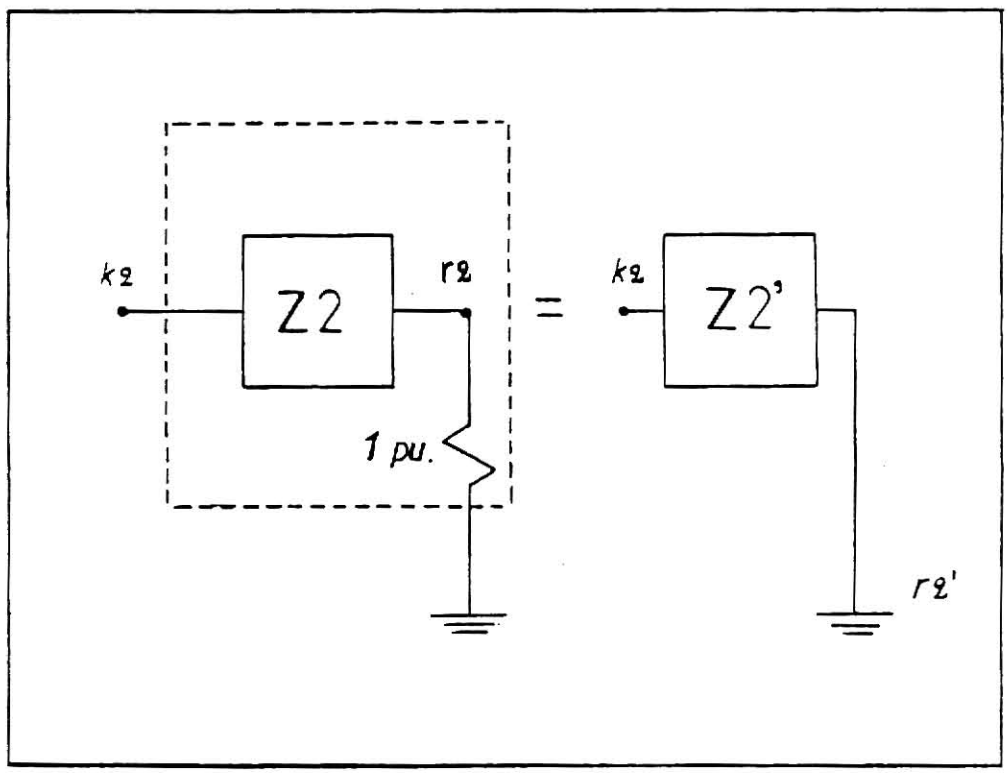

Figure 9.- Variation of Z2 before considering the fault.

Now, a composite matrix Z (2n+1 x 2n +1) is formed with Z2' and Z0 , which is :

$$Z = \begin{bmatrix} Z2' & 0 \\ 0 & Z0 \end{bmatrix} \qquad (26)$$

The next step is to connect negative and zero sequence networks by simulating the connection between r2 and k0 as shown in figure 10. Thus, Householder's formula is used and Z' is obtained. In this case Z' represents the modified matrix resulting due to connection between r2 and k0, which is

$$Z' = Z - Z X X^{t} Z / [X^{t} Z X] \qquad (27)$$

where, the vector X (2n + 1 x 1) has all its values equal to 0 except the (n +1) and (n + 1 + k) rows that have values 1 and -1, respectively. Note that r2 = n + 1 , which is the new node that was created and k0 = (n + 1 + k), which is the new number for the faulted bus in the zero sequence network.

Now, for computer implementation, as in the two line to ground fault, $Z X X^{t} Z$ is called Ztemp and all its values are calculated using (28), (29), (30), and (31), which are

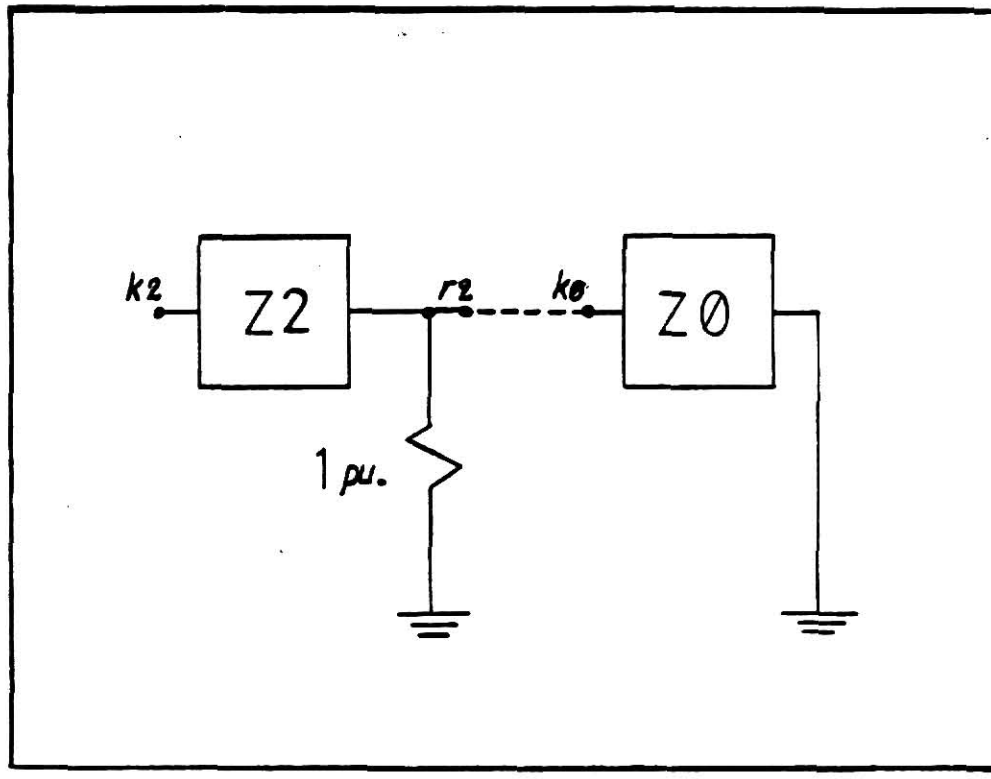

Figure 10.- Connection between negative and zero sequence.

$$ztemp(i, j) = 1.0 \quad ; \qquad i,j = 1,\ldots,(n+1) \tag{28}$$

$$ztemp(i, n+j+1) = -z0(k,j) \quad ; \qquad i = 1,\ldots,(n+1) \quad j = 1,\ldots,n \tag{29}$$

$$ztemp(n+i+1, j) = -z0(i,k) \quad ; \qquad i = 1,\ldots,n \quad j = 1,\ldots,(n+1) \tag{30}$$

$$ztemp(n+i+1, n+j+1) = z0(i,k) * z0(k,j) \quad ; \qquad i = 1,\ldots,n \quad j = 1,\ldots,n \tag{31}$$

And, in this case :

$$X^t Z X = 1 + z0(k,k) \tag{32}$$

Then, upon proper substitution in (27) the elements of Z' are easily obtained.

After Z' is formed another modification must be done.

The reactance between r2 and ground needs to be removed because this was introduced as a dummy for putting together the negative and zero sequence networks avoiding the complication of having the reference for negative sequence grounded. To achieve this, equation (1) for open circuit is used. The new modified matrix is called Z''. Z'' represents the system formed as a result of removing the dummy reactance in figure 10. The resulting circuit is shown in figure 11.

Therefore :

$$Z'' = Z' + Z' X X^{t} Z' / [1 - X^{t} Z' X] \quad \ldots\ldots\ldots (33)$$

In (33) the vector X (2n+1 x 1) has all its values equal to zero except the (n + 1) row that has value 1. The computation of Z'' is done using equation (27). Now, the matrix $Z' X X^{t} Z'$ is called Ztemp. So, as far as memory of the computer is concerned no more memory is required for this new Ztemp because the new values of Ztemp replace the old ones. The new values of Ztemp are calculated as follows.

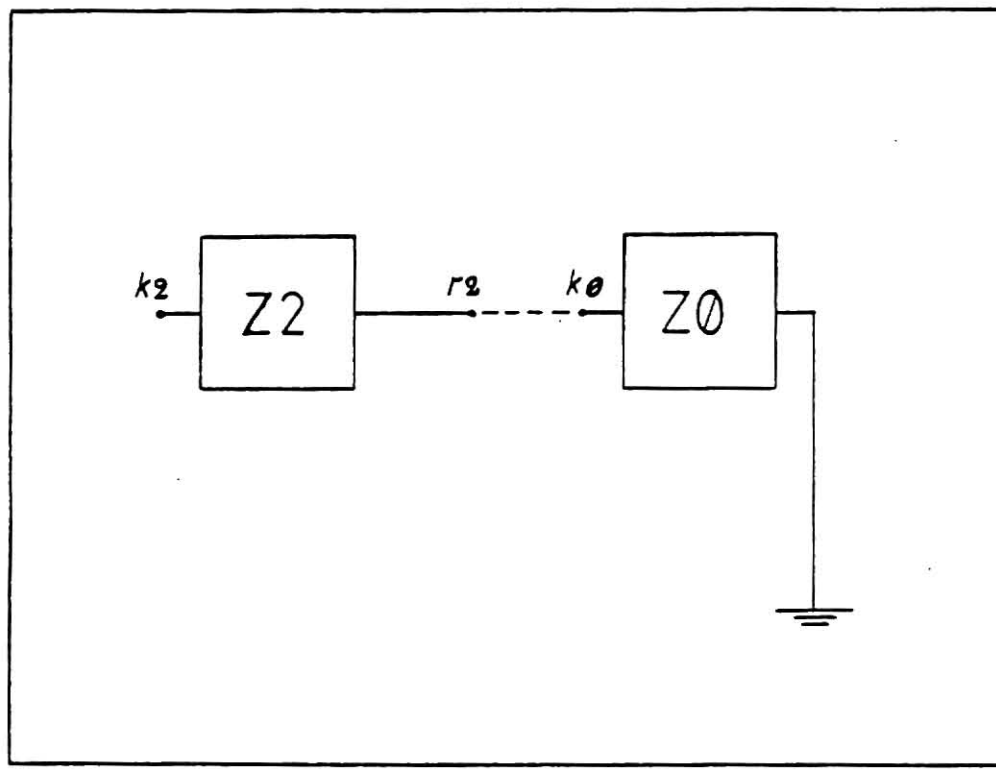

Figure 11.- Circuit of figure 10 without the dummy reactance.

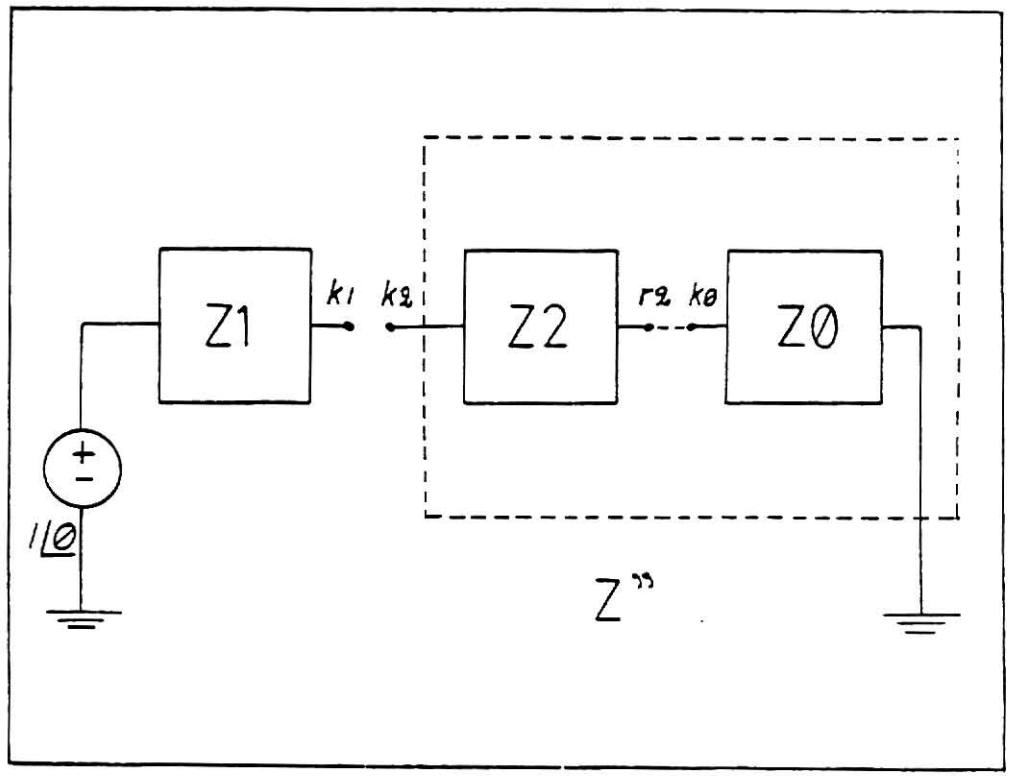

Figure 12.- Final connection for single line to ground fault

$$ztemp(i,j) = z'(i,n+1) * z'(n+1,j) \quad (34)$$

$$i = 1,.....,(2n+1)$$

$$j = 1,.....,(2n+1)$$

and,

$$X^{t} Z' X = z'(n+1,n+1) \quad (35)$$

The values of Z'' are then easily found using (34) and (35) in (33).

Now, the sequence networks are ready to be connected in series, so the positive sequence network must be included. This last connection is represented by figure (12).

It should be noted that this last step is similar to those for double line and double line to ground faults. The simplification now is the same. Note that there is no need to create the composite matrix that involves all the three sequence networks. Using the Householder's formula for the last connection the sequence voltages after fault are obtained to be:

$$v1(i) = 1 - zl(i,k) / [zl(k,k) + z''(k,k)] \quad (36)$$

$$v2(i) = z''(i,k) / [zl(k,k) + z''(k,k)] \quad (37)$$

$$v0(i) = z''(n+i+1,k) / [zl(k,k) + z''(k,k)] \quad (38)$$

Again, note that these expressions have a similarity to the expressions obtained for the other faults.

3.5 Fault Currents.

So far the method has obtained all sequence voltages of the system upon the occurrence of faults. If the voltages at the end points of a impedance are known, the current across the impedance can be calculated. This information is necessary but not enough to calculate the sequence currents in all elements of the system. There is still a small complication if the system includes transformers with wye - delta connections. Then, the computer program must be developed to recognize such a connection and do the necessary corrections on angles for the sequence voltages as well as the sequence currents. These connections are recognized establishing a special code for each bus as is explained in the following section.

3.6 Corrections for wye - delta transformer connections.

A procedure needs to be established for calculating the phase relationship of all voltages and currents for all types of transformer connections. According to ASA

(American Standards Association) convention it is assumed that the voltages on the high voltage side lead the corresponding voltages on the low voltage side by 30 degrees for all wye-delta or delta-wye transformers. The actual phase relationship depends upon the labeling of the three phases and may be positive or negative values of 30, 150, and 90 degrees [2].

To take account of the phase shift a code is used for the buses which is as follows. A reference bus is chosen and it is assigned a code of 0. To assign a code to other buses a path from the reference bus to the bus under consideration is traversed. A count of +1 indicates a 30 degree shift and a count of -1 indicates a -30 degree shift. If there are wye - delta connections then counts of +1 and -1 are used to count the number of 30 degree shifts in the path traversed. The integer value thus obtained indicates the number of 30 degree shifts in phase of that bus with respect to the reference bus and the sign indicates the direction of shift. Thus, +2 will mean a shift of +60 degrees in phase between the voltage of the bus under consideration with respect to the reference bus. Figure 13 [1] shows an example of a power system with the respective shift codes (between parenthesis) for the buses. The reference bus in this case is bus number 1.

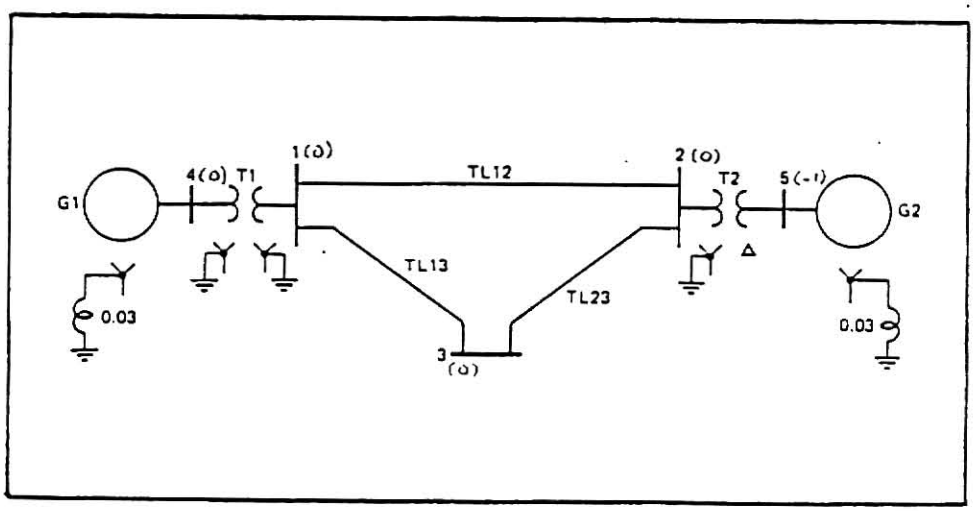

Figure 13.- One line diagram of a power system including shift code for the buses.

Although selection of reference bus is arbitrary, it is customary to consider the faulted bus as the reference bus. Thus, during fault study if the faulted bus has to be considered as the new reference, then the codes for the buses are changed to reflect this change according to the following equation.

$$shift[i] = shift[i] - shift[k] \qquad (39)$$

where, k is the faulted bus,
i is a bus of the system, and
shift is the name for the array that keeps all the codes for the buses of the system.

In this way it is easy to keep track of the angles of the buses with respect to one another.

The new codes thus obtained are with respect to a new reference. Now, if another bus has to be faulted then equation (39) is used again to get shift codes with respect to next reference. Note that it is a moving reference situation and it is not necessary to preserve original or old shift codes. The latest shift codes are maintained in any one session of running this program. After quitting if one reruns the program then the shift codes will be the ones that have been specified in the data.

Chapter 4

COMPUTER PROGRAM

In this chapter, some details on how to use the computer program and the way the data of the system is entered are discussed. As it was mentioned at the beginning of this report the program is written in Turbo Pascal, which facilitates writing programs that are relatively easy to read, understand, and maintain. So, the program can be run on any personal computer that has the Turbo Pascal System. This system includes an editor for creating and modifying programs and a compiler and loader. It also has the capability to save and retrieve files on a disk [7]. Moreover, the program itself is a file that is saved on a disk under the name PRUEBA. After the program is run, the results are also in a permanent file that is generated and saved by the program. The only file that the user must create and save is the input file which includes all the necessary data for the solution of the faulted system. The chapter will explain the organization of the input file, the main program block, and the output file.

4.1 The Input File

Complete information about the configuration of the system must be given to the computer before any fault study can be applied. This data must be saved on a disk and created as follows.

The first line of the input file has four integers which represent the number of buses, generators, lines and transformers in the system, respectively. Following this, the data for the generators is entered. Each of these lines contains information about each generator of the system, which includes the bus number to which the generator is connected followed by the positive, negative, and zero sequence reactance, and finally the neutral reactance. All reactance values must be in percent. If the generator is connected in wye ungrounded or in delta, then the zero sequence reactance and the neutral reactance must be set to zero. Note that each line of generator data has five numbers.

After the information for the generators is entered the one for the lines of the system follows. In this case, each line of the input file includes the two terminating bus numbers, and the positive, negative, and zero sequence reactances. Again, the values for the reactances are in

percent.

The transformer data, which is entered next, is a little more complicated but it is taken care of in the following manner. Since the transformers can be connected in different ways a code for the connections is implemented, which is given in table 1. Each line for transformer data has eight entries in the following order; the first terminating bus number, the second terminating bus number, the connection code, the positive, negative, and zero sequence reactance, the neutral reactance for the first terminating bus, and finally the neutral reactance for the second one. If any side of the transformer is connected in delta or wye ungrounded the neutral reactance must be entered as 0 for that terminating bus. Once again, the reactance values must be in percent.

Connection		Connection Code
bus P	bus Q	
wye grounded	wye ungrounded	1
wye grounded	wye grounded	2
wye ungrounded	wye ungrounded	3
wye grounded	delta	4
wye ungrounded	delta	5
delta	wye grounded	6
delta	delta	7

Table 1.- Connection code for transformer connections.

Note that this code is different from the shift code used for phase angles between buses.

So far, all the information about generators, lines, and transformers of the system have been entered. Now, the shift code for the buses that tells the computer the differents levels of angles must be entered.

Thus, in the next line a number is entered which indicates the number of buses which have non zero code. If there are no wye - delta transformers in the system then all the buses will have the same phase and so in this line a zero is entered to indicate that no buses require a shift in angle. Note that the shift code will be determined by the user based on selected reference, the high voltage side of the system, and the presence of wye - delta transformers. Thus, the following lines have two integers, one represents the bus number and the other the shift code assigned for that bus.

With the last step completed, the input file is saved and ready to be processed with the main program. The name for this input file depends on the user. Any legal name can be chosen as the file name.

4.2 The Main Program.

The program PRUEBA for the solution of the power system is run in an interactive mode. The instructions in the main program block are in such a sequence that it permits the user to know exactly what the program is processing. Questions are asked during the running of the program to which the user must respond in order for the computer to continue doing different types of calculations. The questions that the user will be asked are the following in order.

Enter the name of your input file.

Enter the name of your output file.

Enter the bus number to be faulted.

Do you want a three phase fault ? : Y / N

Do you want a two line fault ? : Y / N

Do you want a two line to ground fault ? : Y / N

Do you want a single line to ground fault ? : Y / N

Do you want to repeat the faults for another bus ? : Y/N

Each question must be answered according to what the user wants. Once the type of fault is selected the execution starts. After the computation for the selected fault has been completed the program displays a message on the screen and then prompts by asking the user next question in the sequence.

4.3 The Output File.

The output file containing the results is created by the program and saved on the same disk as a permanent file and it can be accessed as often as needed. The computer will use the selected name of the output file for creating a file type TEXT that is predefined in Pascal [7].

At the beginning of this output file the information entered as data for the configuration of the system is found. After this, the solution for the faulted system is presented. This include the voltages for each bus of the system and the currents for all the branches. The results are in per unit for magnitudes and degrees for angles. The type of fault and the bus number is listed. The user can repeat the faults for different buses of the system as many times as he wishes and the output file gets appended with each run. Thus after completion of the selected runs the output file will contain complete results of all the runs.

CHAPTER 5

EXAMPLE

In order to illustrate the algorithm presented in this report for the solution of faulted power system, an example is presented. The example system is taken from reference [1]. The data is filled out in the input file named PROOFDAT.PAS as explained in section 4.1. After this, the computer program PRUEBA.PAS is run considering the bus 3 of the sample system as the faulted bus and results for the four different types of faults mentioned above are obtained. The results obtained as an output of the program execution are compared with the actual results of reference [1] which uses the fault analysis program entitled FALTCALC.

5.1 Sample System.

The sample system under consideration is presented in figure 14 which corresponds to figure 9-3 in reference [1].

The sample system data is given in table 2 [1].

ITEM	MVA RATING	VOLTAGE RATING (KV)	X1	X2	X0
			(per unit)		
G1	100	25	0.2	0.2	0.05
G2	100	138	0.2	0.2	0.05
T1	100	25/230	0.05	0.05	0.05
T2	100	13.8/230	0.05	0.05	0.05
TL12	100	230	0.1	0.1	0.3
TL23	100	230	0.1	0.1	0.3
TL13	100	230	0.1	0.1	0.3

Table 2.- Sample System Data.

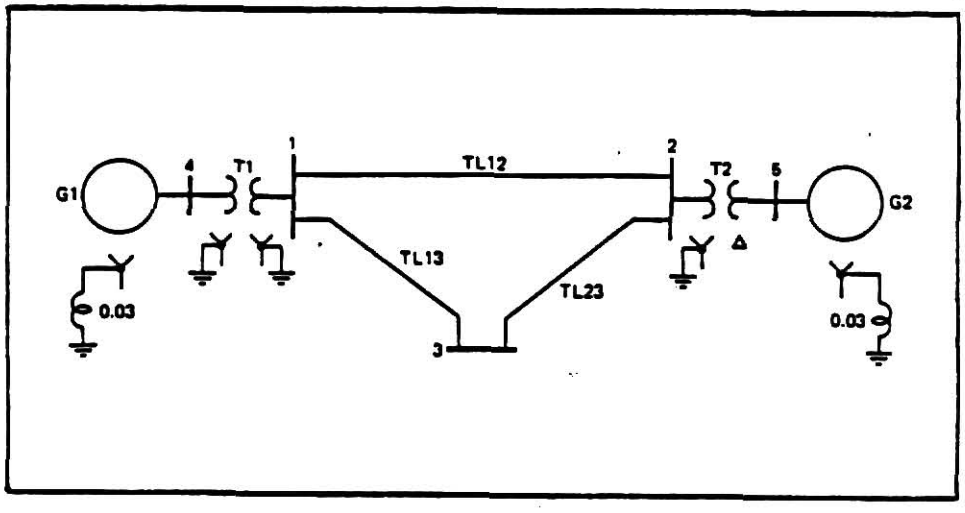

Figure 14.- Example System.

Taking the system data, the input file named PROOFDAT.PAS is created as mentioned earlier, which is presented in table 3.

```
5     2     3     2
4     20    20    5     3
5     20    20    5     3
3     1     10    10    30
2     1     10    10    30
2     3     10    10    30
2     5     4     5     5     5     0     0
1     4     2     5     5     5     0     0
1
5    -1
```

Table 3.- Input File Data.

Let us take a look to this input file. In this case, it consists of 10 lines. The first line tells that the system under consideration has 5 buses, 2 generators, 3 transmission lines, and 2 transformers. So, with this information in mind the two following data lines are for generator data, followed by three data lines for the transmission lines, and two for transformer data. The nineth line holds the number of changes that must be done

over the shift code of the buses. In this case there is only one change that corresponds to bus number 5 with a shift code equal to -1. This change in shift code is because of the presence of wye - delta connection transformer. Note that the shift code for the buses 1 to 4 remains 0.

5.2 Results.

The output for the above mentioned runs was named SOLUTION. All the system data and the calculated values for the different faults appear in SOLUTION in a sequential order. The print out of the file SOLUTION is presented in the following pages.

SYSTEM INFORMATION
****

number of buses : 5 number of generators : 2

number of lines : 3 number of transformers : 2

DATA FOR GENERATORS
****

bus	+ react	- react	0 react	neutral
4	20.00	20.00	5.00	3.00
5	20.00	20.00	5.00	3.00

DATA FOR THE LINES
****

from bus	to bus	+ react	- react	0 react
3	1	10.00	10.00	30.00
2	1	10.00	10.00	30.00
2	3	10.00	10.00	30.00

DATA FOR THE TRANSFORMERS
****

The connection code refers to the differents

ways the transformers can be connected.

The following is applied:

code	connection
1	wye grounded - wye ungrounded
2	wye grounded - wye grounded
3	wye ungrounded - wye ungrounded
4	wye grounded - delta
5	wye ungrounded - delta
6	delta - wye grounded
7	delta - delta

bus P	to	bus Q	connection code	x1	x2	x0	zn_p	zn_q
2		5	4	5.00	5.00	5.00	0.00	0.00
1		4	2	5.00	5.00	5.00	0.00	0.00

SEQUENCE IMPEDANCE MATRICES

****

POSITIVE SEQUENCE MATRIX

(1,1)	0.13971	(1,2)	0.11029	(1,3)	0.12500	(1,4)	0.11176	(1,5)	0.08824
(2,1)	0.11029	(2,2)	0.13971	(2,3)	0.12500	(2,4)	0.08824	(2,5)	0.11176
(3,1)	0.12500	(3,2)	0.12500	(3,3)	0.17500	(3,4)	0.10000	(3,5)	0.10000
(4,1)	0.11176	(4,2)	0.08824	(4,3)	0.10000	(4,4)	0.12941	(4,5)	0.07059
(5,1)	0.08824	(5,2)	0.11176	(5,3)	0.10000	(5,4)	0.07059	(5,5)	0.12941

NEGATIVE SEQUENCE MATRIX

(1,1)	0.13971	(1,2)	0.11029	(1,3)	0.12500	(1,4)	0.11176	(1,5)	0.08824
(2,1)	0.11029	(2,2)	0.13971	(2,3)	0.12500	(2,4)	0.08824	(2,5)	0.11176
(3,1)	0.12500	(3,2)	0.12500	(3,3)	0.17500	(3,4)	0.10000	(3,5)	0.10000
(4,1)	0.11176	(4,2)	0.08824	(4,3)	0.10000	(4,4)	0.12941	(4,5)	0.07059
(5,1)	0.08824	(5,2)	0.11176	(5,3)	0.10000	(5,4)	0.07059	(5,5)	0.12941

ZERO SEQUENCE MATRIX

(1,1)	0.10795	(1,2)	0.02159	(1,3)	0.06477	(1,4)	0.07955	(1,5)	0.00000
(2,1)	0.02159	(2,2)	0.04432	(2,3)	0.03295	(2,4)	0.01591	(2,5)	0.00000
(3,1)	0.06477	(3,2)	0.03295	(3,3)	0.19886	(3,4)	0.04773	(3,5)	0.00000
(4,1)	0.07955	(4,2)	0.01591	(4,3)	0.04773	(4,4)	0.09545	(4,5)	0.00000
(5,1)	0.00000	(5,2)	0.00000	(5,3)	0.00000	(5,4)	0.00000	(5,5)	0.14000

RESULTS FOR THREE PHASE FAULT

faulted bus is : 3

sequence voltages (positive, negative, zero) : magnitude (per unit) and angle (degrees)

bus	1	0.28571	0.000	0.00000	0.000	0.00000	0.000
bus	2	0.28571	0.000	0.00000	0.000	0.00000	0.000
bus	3	0.00000	0.000	0.00000	0.000	0.00000	0.000
bus	4	0.42857	0.000	0.00000	0.000	0.00000	0.000
bus	5	0.42857	-30.000	0.00000	0.000	0.00000	0.000

phase voltages (A, B, C): magnitude (per unit) and angle (degrees)

bus	1	0.28571	0.000	0.28571	240.000	0.28571	120.000
bus	2	0.28571	0.000	0.28571	240.000	0.28571	120.000
bus	3	0.00000	0.000	0.00000	0.000	0.00000	0.000
bus	4	0.42857	0.000	0.42857	240.000	0.42857	120.000
bus	5	0.42857	-30.000	0.42857	210.000	0.42857	90.000

sequence currents (positive, negative, zero) : magnitude (per unit) and angle (degrees)

from	0	to	4	2.85714	-90.00000	0.00000	0.00000	0.00000	0.00000
from	0	to	5	2.85714	-120.00000	0.00000	0.00000	0.00000	0.00000
from	1	to	3	2.85714	-90.00000	0.00000	0.00000	0.00000	0.00000
from	1	to	2	0.00000	90.00000	0.00000	0.00000	0.00000	0.00000
from	2	to	3	2.85714	-90.00000	0.00000	0.00000	0.00000	0.00000
from	2	to	5	2.85714	90.00000	0.00000	0.00000	0.00000	0.00000
from	1	to	4	2.85714	90.00000	0.00000	0.00000	0.00000	0.00000
from	5	to	2	2.85714	-120.00000	0.00000	0.00000	0.00000	0.00000
from	4	to	1	2.85714	-90.00000	0.00000	0.00000	0.00000	0.00000

phase currents (A ,B ,C) : magnitude (per unit) and angle (degrees)

from	0	to	4	2.85714	-90.00000	2.85714	150.00000	2.85714	30.00000
from	0	to	5	2.85714	240.00000	2.85714	120.00000	2.85714	0.00000
from	1	to	3	2.85714	-90.00000	2.85714	150.00000	2.85714	30.00000
from	1	to	2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
from	2	to	3	2.85714	-90.00000	2.85714	150.00000	2.85714	30.00000
from	2	to	5	2.85714	90.00000	2.85714	-30.00001	2.85714	210.00000
from	1	to	4	2.85714	90.00000	2.85714	-30.00001	2.85714	210.00000
from	5	to	2	2.85714	240.00000	2.85714	120.00000	2.85714	0.00000
from	4	to	1	2.85714	-90.00000	2.85714	150.00000	2.85714	30.00000

RESULTS FOR DOUBLE LINE FAULT

faulted bus is : 3

sequence voltages (positive, negative, zero) : magnitude (per unit) and angle (degrees)

bus	1	0.64286	0.000	0.35714	0.000	0.00000	0.000
bus	2	0.64286	0.000	0.35714	0.000	0.00000	0.000
bus	3	0.50000	0.000	0.50000	0.000	0.00000	0.000
bus	4	0.71429	0.000	0.28571	0.000	0.00000	0.000
bus	5	0.71429	-30.000	0.28571	30.000	0.00000	0.000

phase voltages (A, B, C): magnitude (per unit) and angle (degrees)

bus	1	1.00000	0.000	0.55787	206.329	0.55787	153.670
bus	2	1.00000	0.000	0.55787	206.329	0.55787	153.670
bus	3	1.00000	0.000	0.50000	180.000	0.50000	180.000
bus	4	1.00000	0.000	0.62270	216.587	0.62270	143.413
bus	5	0.89214	-13.898	0.89214	193.898	0.42857	90.000

sequence currents (positive, negative, zero) : magnitude (per unit) and angle (degrees)

from	0	to	4	1.42857	-90.00000	1.42857	90.00000	0.00000	0.00000
from	0	to	5	1.42857	-120.00000	1.42857	120.00000	0.00000	0.00000
from	1	to	3	1.42857	-90.00000	1.42857	90.00000	0.00000	0.00000
from	1	to	2	0.00000	90.00000	0.00000	-90.00000	0.00000	0.00000
from	2	to	3	1.42857	-90.00000	1.42857	90.00000	0.00000	0.00000
from	2	to	5	1.42857	90.00000	1.42857	-90.00000	0.00000	0.00000
from	1	to	4	1.42857	90.00000	1.42857	-90.00000	0.00000	0.00000
from	5	to	2	1.42857	-120.00000	1.42857	120.00000	0.00000	0.00000
from	4	to	1	1.42857	-90.00000	1.42857	90.00000	0.00000	0.00000

phase currents (A ,B ,C) : magnitude (per unit) and angle (degrees)

from	0	to	4	0.00000	0.00000	2.47436	180.00000	2.47436	0.00000
from	0	to	5	1.42857	180.00000	1.42857	180.00000	2.85714	0.00000
from	1	to	3	0.00000	0.00000	2.47436	180.00000	2.47436	0.00000
from	1	to	2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
from	2	to	3	0.00000	0.00000	2.47436	180.00000	2.47436	0.00000
from	2	to	5	0.00000	0.00000	2.47436	0.00000	2.47436	180.00000
from	1	to	4	0.00000	0.00000	2.47436	0.00000	2.47436	180.00000
from	5	to	2	1.42857	180.00000	1.42857	180.00000	2.85714	0.00000
from	4	to	1	0.00000	0.00000	2.47436	180.00000	2.47436	0.00000

RESULTS FOR DOUBLE LINE TO GROUND FAULT

faulted bus is : 3

sequence voltages (positive, negative, zero) : magnitude (per unit) and angle (degrees)

bus	1	0.53373	0.000	0.24802	0.000	0.11310	0.000
bus	2	0.53373	0.000	0.24802	0.000	0.05754	0.000
bus	3	0.34722	0.000	0.34722	0.000	0.34722	0.000
bus	4	0.62698	0.000	0.19841	0.000	0.08333	0.000
bus	5	0.62698	-30.000	0.19841	30.000	0.00000	0.000

phase voltages (A, B, C): magnitude (per unit) and angle (degrees)

bus	1	0.89484	0.000	0.37200	221.694	0.37200	138.306
bus	2	0.83929	0.000	0.41513	216.587	0.41513	143.413
bus	3	1.04167	0.000	0.00000	0.000	0.00000	0.000
bus	4	0.90873	0.000	0.49622	228.414	0.49622	131.586
bus	5	0.74624	-16.688	0.74624	196.688	0.42857	90.000

sequence currents (positive, negative, zero) : magnitude (per unit) and angle (degrees)

from	0	to	4	1.86508	-90.00000	0.99206	90.00000	0.59524	90.00000
from	0	to	5	1.86508	-120.00000	0.99206	120.00000	0.00000	0.00000
from	1	to	3	1.86508	-90.00000	0.99206	90.00000	0.78042	90.00000
from	1	to	2	0.00000	90.00000	0.00000	-90.00000	0.18519	-90.00000
from	2	to	3	1.86508	-90.00000	0.99206	90.00000	0.96561	90.00000
from	2	to	5	1.86508	90.00000	0.99206	-90.00000	1.15079	-90.00000
from	1	to	4	1.86508	90.00000	0.99206	-90.00000	0.59524	-90.00000
from	5	to	2	1.86508	-120.00000	0.99206	120.00000	0.00000	0.00000
from	4	to	1	1.86508	-90.00000	0.99206	90.00000	0.59524	90.00000

phase currents (A ,B ,C) : magnitude (per unit) and angle (degrees)

from	0	to	4	0.27778	-90.00000	2.68085	157.36507	2.68085	22.63492
from	0	to	5	1.61630	207.88949	1.61630	152.11051	2.85714	0.00000
from	1	to	3	0.09259	-90.00000	2.75742	153.81126	2.75742	26.18874
from	1	to	2	0.18519	-90.00000	0.18519	-90.00000	0.18519	-90.00000
from	2	to	3	0.09259	90.00000	2.84401	150.46156	2.84401	29.53843
from	2	to	5	0.27778	-90.00000	2.93972	-32.68019	2.93972	212.68018
from	1	to	4	0.27778	90.00000	2.68085	-22.63493	2.68085	202.63492
from	5	to	2	1.61630	207.88949	1.61630	152.11051	2.85714	0.00000
from	4	to	1	0.27778	-90.00000	2.68085	157.36507	2.68085	22.63492

RESULTS FOR SINGLE LINE TO GROUND FAULT

faulted bus is : 3

sequence voltages (positive, negative, zero) : magnitude (per unit) and angle (degrees)

bus	1	0.77226	0.000	0.22774	180.000	0.11801	180.000
bus	2	0.77226	0.000	0.22774	180.000	0.06004	180.000
bus	3	0.68116	0.000	0.31884	180.000	0.36232	180.000
bus	4	0.81781	0.000	0.18219	180.000	0.08696	180.000
bus	5	0.81781	-30.000	0.18219	210.000	0.00000	0.000

phase voltages (A, B, C): magnitude (per unit) and angle (degrees)

bus	1	0.42650	0.000	0.94990	245.742	0.94990	114.258
bus	2	0.48447	0.000	0.92759	249.008	0.92759	110.992
bus	3	0.00000	0.000	1.02243	237.889	1.02243	122.111
bus	4	0.54865	0.000	0.95595	244.950	0.95595	115.050
bus	5	0.74364	-42.250	0.74364	222.250	1.00000	90.000

sequence currents (positive, negative, zero) : magnitude (per unit) and angle (degrees)

from	0	to	4	0.91097	-90.00000	0.91097	270.00000	0.62112	-90.00000
from	0	to	5	0.91097	-120.00000	0.91097	300.00000	0.00000	0.00000
from	1	to	3	0.91097	-90.00000	0.91097	-90.00000	0.81435	-90.00000
from	1	to	2	0.00000	0.00000	0.00000	0.00000	0.19324	90.00000
from	2	to	3	0.91097	-90.00000	0.91097	-90.00000	1.00759	-90.00000
from	2	to	5	0.91097	90.00000	0.91097	90.00000	1.20083	90.00000
from	1	to	4	0.91097	90.00000	0.91097	90.00000	0.62112	90.00000
from	5	to	2	0.91097	-120.00000	0.91097	-60.00000	0.00000	0.00000
from	4	to	1	0.91097	-90.00000	0.91097	-90.00000	0.62112	-90.00000

phase currents (A ,B ,C) : magnitude (per unit) and angle (degrees)

from	0	to	4	2.44306	-90.00000	0.28986	90.00000	0.28986	90.00000
from	0	to	5	1.57785	-90.00000	1.57785	90.00000	0.00000	0.00000
from	1	to	3	2.63630	-90.00000	0.09662	90.00000	0.09662	90.00000
from	1	to	2	0.19324	90.00000	0.19324	90.00000	0.19324	90.00000
from	2	to	3	2.82954	-90.00000	0.09662	-90.00000	0.09662	-90.00000
from	2	to	5	3.02277	90.00000	0.28986	90.00000	0.28986	90.00000
from	1	to	4	2.44306	90.00000	0.28986	-90.00000	0.28986	-90.00000
from	5	to	2	1.57785	-90.00000	1.57785	90.00000	0.00000	0.00000
from	4	to	1	2.44306	-90.00000	0.28986	90.00000	0.28986	90.00000

Comparing the results obtained after running the computer program PRUEBA.PAS with those of reference 1 it is quite satisfactory to observe that these are the same in each one of the fault types.

Moreover, the output file is now saved as a permanent file on disk and it can be accessed as often as needed. Thus, it can be printed out as many times as the user likes without running the program PRUEBA.PAS again. If this program is run again it can use another input file (for another sample system) and create another output file. Therefore, the disk can have more than one input and output file depending on the user. Each sample system will have its input and output permanent files independently.

CHAPTER 6

PRACTICAL CONSIDERATIONS AND ENHANCEMENT OF THE ALGORITHM

In this chapter possibilities of enhancing the algorithm to include some practical situations in power systems are considered. The implementation of the modifications to the algorithm presented depends on the additional situations the user wants the computer program to solve. Some new ideas will be proposed in this chapter for further work on this topic.

All the faults studied in the algorithm developed in this report have been considered as direct short circuits between lines or lines and ground. If the faults are considered through a impedance the modifications to be done in the computer program PRUEBA.PAS can be easily achieved. The first step is augmentation of the proper impedance matrix to include the fault impedance. In the next step proper points are short circuited using the Householder's formula.

If the computer program is implemented for the solution of faulted power systems including the faults through an impedance, there are other cases that can be dealt with

easily. For example, consider figure 15 as a part of a power system : the transmission line with bus 1 and 2 at the two ends, and the circuits breakers CB1 and CB2 for protection of the line. One situation that can be considered is that CB2 is opened and there is a fault at y or there is a fault at x. To take care of this situation first an open circuit between 1 and 2 is simulated using Householder's formula. Then, if the fault is at x, bus 1 can be considered faulted without impedance. However, if the fault is at y with CB2 open, then this can be simulated like a fault through impedance and the fault impedance is equal to the impedance of the transmission line. A similar situation is present if CB1 is opened and the fault occurs at points y and x respectively.

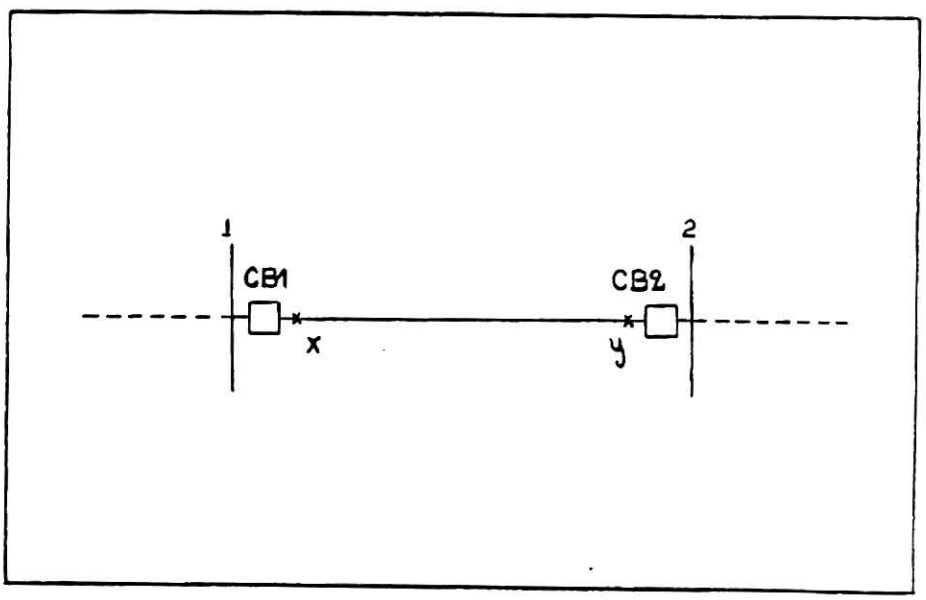

Figure 15.- One line diagram for a transmission line showing the protection of it.

Another problem that deserves to be implemented is the consideration of a power system that does not have a path to reference for one or more buses in the zero sequence network. For this case, the impedance matrix of the zero sequence network does not exist. The idea is to develop an algorithm for the zero sequence network which recognizes this situation and gives as a result the bus number of the bus or buses that have no path to reference. Not only is this necessary but also the knowledge about the buses which are connected together. Let us suppose for example it is found that in a sample power system the buses 7, 8, 10, 12, 15, 22, 23, and 25 have no path to ground, and among these the buses 7, 8, 10, and 12 are connected together, and buses 15, 22, 23, 25 are too. Thus, the sample system can be visualized in its zero sequence network like three unconnected sub_networks such that one of them has path to ground and the other two do not. So, the new algorithm to be developed must recognize this situation and be able to simulate the shunt faults by creating proper connections between the sequence networks.

CHAPTER 7

C O N C L U S I O N S

The main purpose of this report was to develop an efficient algorithm for solving power systems under shunt faults. This algorithm uses the Householder's formulas to simulate the connection among the sequence networks for the analysis of the shunt faults. The algorithm also takes into consideration the phase shifts because of wye-delta transformer connections. An interactive computer program in PASCAL suitable for a personal computer has been written based on the developed algorithm. Some examples were tried and the results were found to be exactly as found using other algorithms.

The algorithm is very simple in nature and understandable. Thus, it can be used as a teaching tool in power systems courses at the senior and graduate levels. Also with certain enhancements the computer program can be upgraded to commercial category. Some of these enhancements have been suggested in the previous chapter.

REFERENCES

1. Gross,Charles A., Power System Analysis, 2nd edition, John Wiley and Sons, Inc., New York, 1986.

2. Anderson, Paul M., Analysis of Faulted Power Systems, Iowa State Press, Ames, Iowa, 1973.

3. Brown, Homer E., Solution of Large Networks by Matrix Methods, 2nd edition, John Wiley and Sons, Inc., New York, 1985.

4. Brandwajn, V., and Tinney, W.F., "Generalized method of fault analysis,"IEEE Transactions on Power Apparatus and Systems, Vol.PAS-104, No.6, pp.1301-1306, June 1985.

5. Householder, A.S., "A survey of some closed methods for inverting matrices,"SIAM J. Appl. Math., Vol.5, pp.155-159, 1957.

6. Pahwa, A., and Rohrer, R., " Band Faults: efficient approximations to fault bands for the simulation before fault diagnosis of linear circuits,"IEEE Transactions onCircuits and Systems, Vol.CAS-29, No.2, February 1982.

7. Koffman, Elliot B., Turbo Pascal A Problem Solving Approach, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1986.

APPENDIX A.

COMPUTER PROGRAM PRUEBA.PAS

```
program prueba ;

{ CONSTANT DECLARATION }

const

 node_lim = 30 ;  {* set the maximum number of nodes of
                                        the system       *}

 nodeliml = 31 ;  {* maximum number of nodes  +  1       *}

 doublel  = 61 ;  {* (maximum number of nodes  * 2) + 1 *}

 limitdat = 80 ;  {* maximum  #  of lines in input file *}

 trflim   = 20  ; {* maximum number of transformers      *}

 maximum  = 100 ; {*  limitdat  +  trflim                *}


{ TYPE  DECLARATION  }

type

   ingresos    = record
                 al,bl       : integer;
                 cl          : real   ;
                 end;

   impedan     = array [ 1..limitdat ] of ingresos;


   linedat     = record
                 x      : real;
                 done   : boolean;
                 end;


   { size of each sequence network. maximum : nodelim }
    matrix      = array [ 1..node_lim, 1..node_lim ] of
                                                 linedat;


  { array that holds the shift code for the buses }
  class       = array [ 1..node_lim ] of integer ;

  vector      = record
                magn  : real;
                angle : real ;
                end;
```

```
   { array for the nodes of the system }
   volt = array [ 1..nodeliml ] of vector ;


  { array  for the composite matrix that holds }
  { the negative and zero sequence impedances  }
  compose = array [ 1..doublel, 1..doublel ] of real ;



{ VAR DECLARATION }


var

   positive, negative, zero   : impedan ; { input data for
                                            sequences    }
   infile, outfile            : text    ;
   inproof, outproof          : string[20];{input / output
                                            files names  }
   numbus                     : integer;      { number of
                                              buses of the
                                                    system}

   zbusl , zbus2 , zbus0      : matrix  ;        { sequence
                                                  impedance
                                                  matrices}
   total                      : integer ;      { number of
                                                 line data }
   i,j                        : integer ;      { counters }
   faultbus                   : integer ;  { faulted bus }
   shift                      : class   ;  { shift code
                                             for  buses  }

  vpos, vneg, vzero           : volt    ;  { sequence
                                               voltages }
  gener, lines, transf        : integer ;
  ftype                       : integer ;     { fault type
                                                  code  }
  answer                      : char    ;
  response                    : boolean ;
```

```
{*********************************************************}
{*                                                       *}
{*              P R O C E D U R E S                      *}
{*                                                       *}
{*********************************************************}


{*********************************************************}
{   getting   information    and   writing   it   out      }
{*********************************************************}


procedure getdata ( var positive, negative, zero : impedan;
                    var  numbus : integer; var  total : integer;
                    var gener, lines, transf  : integer ;
                    var shift : class );
var


 temp                                 : integer;

 i , j                                : integer;
 connec                               : integer;
 bus,fbus,tbus,bus_p,bus_q,hv,lv      : integer;
 x1,x2,x0,xn,zn_p,zn_q                : real   ;
 modif                                : integer;


begin  { procedure get-data }


writeln ( outfile, 'SYSTEM    INFORMATION    ' );
writeln ( outfile, '********************** ' );
writeln ( outfile );
writeln ( outfile );
writeln( outfile);

readln ( infile,numbus, gener, lines, transf );
total :=  gener + lines + transf ;

writeln ( outfile,'number of buses :  ', numbus:4,
     '         number of generators   :  ',gener:4 );
writeln ( outfile );
writeln(outfile,'number of lines :  ',lines:4,
     '         number of transformers :  ',transf:4 );
writeln(outfile, '   ');
writeln(outfile,'  ');
writeln(outfile, ' DATA FOR GENERATORS ' );
writeln(outfile, ' ******************* ' );
writeln(outfile);
writeln (outfile,
  'bus       + react    - react    0 react     neutral');
writeln ( outfile);
```

```
for i := 1 to gener do
 begin
    readln( infile, bus, x1, x2, x0, xn );
    writeln( outfile,bus:3,'    ', x1:10:2,x2:11:2,x0:10:2,
                                                  xn:13:2 );
    writeln( outfile, '     ' );
    positive[i].a1 := 0;
    positive[i].b1 := bus;
    positive[i].c1 := x1;
    negative[i].a1 := 0;
    negative[i].b1 := bus;
    negative[i].c1 := x2;

    { for generators in Y ungrounded or Delta }
    if ( x0 = 0.0 ) then
       begin
        zero[i].a1 := 0;
        zero[i].b1 := 0;
        zero[i].c1 := 0;
       end
       else
        begin
          zero[i].a1      := 0 ;
          zero[i].b1      := bus;
          zero[i].c1      := x0  + ( 3 * xn );
        end;

end;   { all data of generators are entered }


writeln( outfile );
writeln( outfile );
writeln (outfile, ' DATA FOR THE LINES ');
writeln (outfile, ' ****************** ');
writeln( outfile );
writeln ( outfile,
  'from bus  to  bus      + react    - react    0 react');
writeln ( outfile );
for i :=  ( gener + 1 ) to ( gener + lines ) do
begin
readln ( infile, fbus, tbus, x1, x2, x0 );
writeln ( outfile, fbus:7, tbus:9,'    ', x1:9:2, x2:12:2,
                                                  x0:10:2 );
positive[i].a1  := fbus;
positive[i].b1  := tbus;
positive[i].c1  := x1  ;
negative[i].a1  := fbus;
negative[i].b1  := tbus;
negative[i].c1  := x2  ;
zero[i].a1      := fbus;
zero[i].b1      := tbus;
zero[i].c1      := x0  ;
end;   { all data for lines are entered }
```

```
{* DATA FOR THE TRANSFORMERS *}

IF ( TRANSF > 0 ) THEN

BEGIN


writeln( outfile );
writeln( outfile );
writeln(outfile, ' DATA FOR THE TRANSFORMERS ' );
writeln(outfile, ' ************************** ' );
writeln(outfile);
writeln(outfile);


writeln(outfile,
'The connection code refers to the differents');
writeln(outfile,
'ways the transformers can be connected.');
writeln(outfile);
writeln(outfile,'The following is applied:              ');
writeln(outfile);
writeln(outfile,'     code                  connection   ');
writeln(outfile);
writeln(outfile,
 '      1        wye grounded   - wye ungrounded ');
writeln(outfile,
 '      2        wye grounded   - wye grounded   ');
writeln(outfile,
 '      3        wye ungrounded - wye ungrounded ');
writeln(outfile,
 '      4        wye grounded   - delta          ');
writeln(outfile,
 '      5        wye ungrounded - delta          ');
writeln(outfile,
 '      6        delta          - wye grounded   ');
writeln(outfile,
 '      7        delta          - delta          ');
writeln(outfile);
writeln(outfile);
writeln(outfile);
write(outfile,' bus  P   to   bus Q      connection code ');
write(outfile,'        xl     x2     x0 ' );
writeln( outfile,'    zn_p   zn_q');



writeln ( outfile );
for  i := (gener + lines + 1) to (gener + lines + transf) do
begin

readln (infile, bus_p, bus_q,connec, xl,x2, x0, zn_p, zn_q);
```

```
write ( outfile, bus_p:7,'    ', bus_q:8,'           ',
                                        connec:4,'   ' );

   writeln(outfile,'          ', x1:7:2, x2:7:2, x0:7:2,
                                zn_p:7:2,zn_q:7:2);

   positive[i].al := bus_p;
   positive[i].bl := bus_q;
   positive[i].cl := x1   ;
   negative[i].al := bus_p;
   negative[i].bl := bus_q;
   negative[i].cl := x2   ;


case  connec of

1, 3, 5, 7 :     begin
                 hv:= 1;
                 lv:= 1;
                 end;

         2 :     begin
                 hv:= 0;
                 lv:= 0;
                 end;

        4 :      begin
                 hv:= 0;
                 lv:= 1;
                 end;

       6 :       begin
                 hv:= 1;
                 lv:= 0;
                 end;

end;       {*  case  connec  *}




  if  ( hv = 0 ) and ( lv = 0 ) then
      begin
      zero[i].al := bus_p;
      zero[i].bl := bus_q;
      zero[i].cl := x0  +  ( 3 * ( zn_p + zn_q ) ) ;
      end;  { connection is star - star }

  if ( hv = 0 ) and ( lv = 1 ) then
     begin
     zero[i].al := 0;
```

```
      zero[i].bl := bus_p;
      zero[i].cl := x0 + ( 3 * zn_p );
      end;  { connection is  star - delta }

   if ( hv = 1 ) and ( lv = 0 ) then
      begin
      zero[i].al := 0;
      zero[i].bl := bus_q;
      zero[i].cl := x0 + ( 3 * zn_q );
      end; { connection is delta - star }
   if ( hv = 1 ) and ( lv = 1 ) then
      begin
      zero[i].al := 0;
      zero[i].bl := 0;
      zero[i].cl := x0 ;
      end;  { connection is delta - delta }

 end;   { all data for transformers are entered }


END; { IF TRANSFORMERS EXIST }


{ changing positions of buses if the second one is greater }

for i:= 1 to ( gener + lines + transf ) do
begin
if positive[i].al > positive[i].bl then
 begin
      temp := positive[i].al;
      positive[i].al := positive[i].bl;
      positive[i].bl := temp;
 end;

if negative[i].al > negative[i].bl then
  begin
       temp := negative[i].al;
       negative[i].al := negative[i].bl;
       negative[i].bl := temp;
  end;
if zero[i].al > zero[i].bl then
  begin
       temp := zero[i].al ;
       zero[i].al := zero[i].bl;
       zero[i].bl := temp ;
  end;

end;

writeln ( outfile );
writeln ( outfile );
write   ( outfile,'**************************************');
writeln ( outfile, '**********************************    ' );
```

```
writeln ( outfile );
writeln ( outfile );
writeln ( outfile );


{********* information for shift *************}
{ initialize the code for each bus to 0 as a reference }
for i := 1 to numbus do
shift[i] := 0 ;
readln ( infile, modif );
if modif > 0 then
 for i := 1 to modif do
 readln ( infile, j , shift [j] );


end; { procedure get-data. The sequence networks are formed}
```

```
{*********************************************************}
{ Procedure for forming the sequence impedances matrices   }
{ one at a time .   Inside there is a nested procedure for }
{ inverting matrices.                                      }
{*********************************************************}


procedure formzbus ( positive : impedan ;
                     numbus, total : integer;
                     var  zbus : matrix       );

var

   i, j  : integer  ;
   sum   : real     ;
   first : integer  ;
   inv   : real     ;
   k     : integer  ;


{ **Nested procedure for inverting a matrix *********}

procedure changes ( var z : matrix; numbus, k: integer);

var

 i,j : integer;


begin


   z[k,k].done := true;
   z[k,k].x    := 1.0 / z[k,k].x ;

   for i:= 1 to numbus do
   if  i <> k then

   z[i,k].x := z[i,k].x * z[k,k].x ;

       for i:= 1 to numbus do
        for j:= 1 to numbus do
        if ( i <> k ) and ( j <> k ) then
        z[i,j].x := z[i,j].x  -  ( z[i,k].x * z[k,j].x );
       for j:= 1 to numbus do
        if j <> k then
        z[k,j].x := - z[k,j].x * z[k,k].x ;
end;  { procedure changes }
```

```
{ ***    end  of the nested procedure      ********** }


begin

   for i := 1 to  numbus do
       for j := 1 to numbus do
       begin
       zbus[i,j].done := false ;
       zbus[i,j].x := 0.0;
       end;


for i := 1 to total do

if positive[i].bl <> 0 then   { to avoid connection
                                delta-delta          }

begin  { forming mutual impedances }

   first := positive[i].al;

  if  first  >  0  then
      begin
        zbus[positive[i].al,positive[i].bl].x :=
                              1.0  /  positive[i].cl ;

        inv := zbus[positive[i].al,positive[i].bl].x;

        zbus[positive[i].bl,positive[i].al].x := inv;
      end;

   if first = 0 then
      zbus[positive[i].bl,positive[i].bl].x :=
                              1.0 /  positive[i].cl ;

end;  { forming mutual admitances }

  { forming y(i,i) }
  for i := 1 to numbus do
  begin
       sum := 0.0;
       for j := 1 to numbus do
       sum := sum + zbus[i,j].x;
       zbus[i,i].x := - sum;
  end;
  { now, we invert the matrix }


  for i := 1 to numbus do
  begin
```

```
        k := 1;
  while ((zbus[k,k].done = true) or ( zbus[k,k].x = 0.0 ))
           and  ( k <= numbus ) do

        k := k + 1 ;

  { clue for inverting the matrix }
  changes ( zbus, numbus, k );

  end; { all changes are done . zbus is formed }
 writeln( ' matrix is being inverted' );
 writeln;
 for i := 1 to numbus do
       for j := 1 to numbus do
       zbus[i,j].x := - zbus[i,j].x / 100 ;

  { check if any Zbus[k,k].done := false }
  for i:= 1 to numbus do
      if ( zbus[i,i].done = false ) then
       begin
       writeln ( ' WARNING ' );
       write( ' IT IS NOT POSSIBLE TO GET Zbus    ' );
       writeln ('IN THIS SEQUENCE ' );
       writeln;
       end;


end; { procedure formzbus }
```

```
{*********************************************************}
{* procedure for writing the sequence impedance matrices  *}
{*********************************************************}


procedure networks ( zbus1, zbus2, zbus0 : matrix  ;
                     numbus              : integer  );

var
  i, j    : integer ;


begin


writeln( outfile,' SEQUENCE      IMPEDANCE      MATRICES ');
writeln( outfile,' ************************************ ');
writeln( outfile );
writeln( outfile );
writeln( outfile,' POSITIVE SEQUENCE MATRIX ' );
writeln( outfile );

for i:= 1 to numbus do
  for j := 1 to numbus do
    begin
    write( outfile,'(',i,',',j,')  ',
                    zbus1[i,j].x:8:5,'        ') ;
    if ( j mod 5 = 0 ) then  writeln ( outfile );
    if ( j = numbus ) then writeln ( outfile );
    end;

writeln( outfile );
writeln( outfile );
writeln( outfile,' NEGATIVE SEQUENCE MATRIX ' );
writeln( outfile );

for i:= 1 to numbus do
  for j := 1 to numbus do
  begin
  write( outfile,'(',i,',',j,')  ',
                  zbus2[i,j].x:8:5,'        ' );
  if ( j mod 5 = 0 ) then writeln ( outfile );
  if ( j = numbus )  then writeln ( outfile );
  end;

writeln( outfile );
writeln( outfile );
writeln( outfile,' ZERO SEQUENCE MATRIX ' );
writeln( outfile );

for i:= 1 to numbus do
  for j := 1 to numbus do
```

```
  begin
  write( outfile,'(',i,',',j,')  ',
                 zbus0[i,j].x:8:5,'        ' );
  if ( j mod 5 = 0 ) then writeln ( outfile );
  if ( j = numbus  ) then writeln ( outfile );
  end;


writeln( outfile );
write( outfile,'*****************************************');
writeln ( outfile,'*****************************************');
writeln( outfile );
writeln ( outfile );
writeln ( outfile );
writeln ( outfile );
writeln ( outfile,
' ********************************************************');
writeln ( outfile,
' *     R     E     S     U     L     T     S           *');
writeln ( outfile,
' ********************************************************');
writeln ( outfile );
writeln ( outfile );
writeln ( outfile );


end;    { procedure for writing sequence matrices }
```

```
{*********************************************************}
{ general procedure to get phase values from sequence ones }
{*********************************************************}


procedure seqtophase (po, poang, ne, neang, ze, zeang: real;
                  var iorva, iorvang, iorvb, iorvbang,
                                      iorvc, iorvcang : real );

const

    pi = 3.1415926 ;


var

   xa, ya, xb, yb, xc, yc : real ; { real,imag for phase }
   anpos, anneg, anzer    : real ; { angles in radians for
                                                  sequence }


{ general procedure inside for converting a complex }
{ number from rectangular to polar                  }

procedure transform ( xr, yi : real;
               var  mgn, angles : real );

     const
          pi = 3.1415926 ;
          epsilon = 0.0001;

     begin

          if  abs(xr) <= epsilon  then  xr := 0.0;
          if  abs(yi) <= epsilon  then  yi := 0.0;

          if xr = 0.0 then
          begin
               if yi = 0.0 then
               begin
               mgn   := 0.0 ;
               angles := 0.0;
               end;   { values at origen }
               if yi > 0.0  then
               begin
               mgn   := yi;
               angles := 90;
               end;    { values over axe y positive }
               if yi < 0.0 then
               begin
               mgn   := abs ( yi );
```

```
                angles := -90 ;
                end;    { values over axe y negative }
            end; { when x is always zero }

            if yi = 0.0 then
            begin
                if xr > 0.0 then
                begin
                mgn    := xr ;
                angles := 0.0;
                end;    { values over axe x positive }
                if xr < 0.0 then
                begin
                mgn    :=  abs ( xr );
                angles := 180 ;
                end;    { values over axe x negative }
            end;   { when y is always zero }
            if xr > 0.0 then
                if yi <> 0.0 then
                begin
                mgn    := sqrt ( sqr(xr)  +  sqr(yi) );
                angles := ( arctan(yi/xr) ) * 180 / pi ;
                end;{this covers first and fourth quadrant}
            if xr < 0.0 then
                if yi <> 0.0 then
                begin
                mgn := sqrt ( sqr(xr) + sqr(yi) );
                angles := 180 + (arctan(yi/xr) * 180 / pi);
                end;{this covers second and third quadrant}
        end; { procedure transform }


        {***** end  of nested procedure   ********}



begin { for procedure seqtophase }

    { angles in radians for each sequence }
    anpos := poang * pi / 180 ;
    anneg := neang * pi / 180 ;
    anzer := zeang * pi / 180 ;

    { real and imaginary part for phase A }
    xa := ( ze * cos(anzer)) + ( po * cos(anpos))
                                 + ( ne * cos(anneg) ) ;
    ya := ze * sin(anzer) + po * sin(anpos)
                                 + ne * sin(anneg);

    {calling the procedure for getting magnitude and angle
     for phase A }
     transform ( xa, ya, iorva, iorvang );
```

```
    {**********************************}

    { now real and imaginary part for phase B.
      First change angles }
    anpos := ( poang + 240 ) * pi / 180 ;
    anneg := ( neang + 120 ) * pi / 180 ;


    xb := ze * cos(anzer) + po * cos(anpos)
                               + ne * cos(anneg);

    yb := ze * sin(anzer) + po * sin(anpos)
                               + ne * sin(anneg);

    { getting magnitud and angle for phase B }
    transform ( xb, yb, iorvb, iorvbang ) ;
    {**********************************}

    { now phase C }
    anpos := ( poang + 120 ) * pi / 180 ;
    anneg := ( neang + 240 ) * pi / 180 ;

    xc := ze * cos (anzer) + po * cos(anpos)
                               + ne * cos(anneg);

    yc := ze * sin (anzer) + po * sin(anpos)
                               + ne * sin(anneg);

    transform ( xc, yc, iorvc, iorvcang );
    {**********************************}

end;{end - procedure to get phase values from sequence ones}
```

```
{*************************************************************}
{**** the following gives the sequence and phase currents ****}
{*************************************************************}


procedure seqcurr ( vpos, vneg, vzero          : volt    ;
                    positive, negative, zero   : impedan ;
                    gener, lines, transf,numbus,
                                          ftype : integer);

const

   epsilon = 0.0001;


type

  linecurr = record
             frombus, tobus : integer;
             pos, posangle,
             neg, negangle,
             zero, zeroangle : real;
             end;

     { array must match with maximum number of lines
       for data plus the maximum number of transformers.}


  datacurr = array [ 1..maximum ] of linecurr ;


var

  current : datacurr ;
  i       : integer;
  fb      : integer; { current from bus }
  tb      : integer; { current to bus   }
  ia, iang, ib, ibang, ic, icang : real ;  {phase values}


begin { for procedure seqcurr }

  { corrections - aproximations }
  for i := 1 to numbus do
  begin
       if vpos[i].magn <= epsilon then
       begin
       vpos[i].magn := 0.0;
       vpos[i].angle := 0.0;
       end;
```

```
        if vneg[i].magn <= epsilon then
        begin
        vneg[i].magn  := 0.0;
        vneg[i].angle := 0.0;
       end;
       if vzero[i].magn <= epsilon then
       begin
       vzero[i].magn  := 0.0;
       vzero[i].angle := 0.0;
       end;
   end; { corrections - aproximations }


 { initialize values for frombus and tobus  }
 for i := 1 to ( gener + lines + transf ) do
 begin
 current[i].frombus  := positive[i].al ;
 current[i].tobus    := positive[i].bl ;

 if i > ( gener + lines ) then
      begin
      current[ i + transf ].frombus := positive[i].bl;
      current[ i + transf ].tobus   := positive[i].al;
      end;
 end; { loop for initializing values frombus and tobus }


{ currents for + - and 0 seq. for generators }
for i := 1 to gener do
begin
tb := current[i].tobus ;
current[i].pos := ( 1 - vpos[tb].magn )
                              * 100.0 / positive[i].cl ;

current[i].posangle := vpos[tb].angle - 90 ;

current[i].neg := vneg[tb].magn * 100 / negative[i].cl ;

if current[i].neg <> 0.0 then

           current[i].negangle := vneg[tb].angle + 90
                      else  current[i].negangle := 0.0 ;
if zero[i].cl <> 0.0 then
current[i].zero := vzero[tb].magn * 100.0 / zero[i].cl
                      else current[i].zero := 0.0 ;

if current[i].zero <> 0.0 then
                      current[i].zeroangle := -90
                      else current[i].zeroangle := 0.0 ;
end; { currents for the generators are assigned }
```

```
{ now the currents for the lines }
{*******************************}
for i := ( gener + 1 ) to ( gener + lines ) do
begin
fb := current[i].frombus ;
tb := current[i].tobus;
current[i].pos :=(vpos[fb].magn - vpos[tb].magn)
                                * 100.0 / positive[i].cl   ;

if current[i].pos = 0.0 then current[i].posangle := 0.0  ;
if current[i].pos > 0.0 then current[i].posangle := -90  ;
if current[i].pos < 0.0 then current[i].posangle :=  90  ;
current[i].pos := abs ( current[i].pos );

current[i].neg := (vneg[fb].magn - vneg[tb].magn)
                                 * 100.0 / negative[i].cl  ;
if current[i].neg = 0.0 then current[i].negangle := 0.0  ;
if current[i].neg > 0.0 then current[i].negangle :=  90  ;
if current[i].neg < 0.0 then current[i].negangle := -90  ;
current[i].neg := abs ( current[i].neg );

current[i].zero := (vzero[fb].magn - vzero[tb].magn)
                                      * 100.0 / zero[i].cl   ;
if current[i].zero = 0.0 then current[i].zeroangle := 0.0;
if current[i].zero > 0.0 then current[i].zeroangle :=  90;
if current[i].zero < 0.0 then current[i].zeroangle := -90;
current[i].zero := abs ( current[i].zero );

end; { all currents for the lines are assignned }


IF ( TRANSF > 0 ) THEN

BEGIN


{ now working with transformers }
for i:= (gener + lines + 1) to (gener + lines + transf) do

begin   { initial for transformers in general }
    fb := current[i].frombus ;
    tb := current[i].tobus   ;
  { transformers positive sequence }
  current[i].pos :=(vpos[fb].magn - vpos[tb].magn)
                                    * 100 / positive[i].cl;
    current[ i + transf ].pos := abs ( current[i].pos ) ;
    if current[i].pos = 0.0 then
                  begin
                  current[i].posangle := 0.0;
                  current[i + transf].posangle :=0.0;
                  end;
    if current[i].pos > 0.0 then
                     begin
```

```
                        current[i].posangle := -90;
                        current[i + transf].posangle := 90;
                       end;
    if current[i].pos < 0.0 then
                  begin
                  current[i].posangle := 90;
                  current[i + transf].posangle := -90;
                  end;
    current[i].pos := abs ( current[i].pos );


{ transformers negative sequence }
    current[i].neg := (vneg[fb].magn - vneg[tb].magn)
                                    * 100 / negative[i].cl;
    current[i + transf].neg := abs ( current[i].neg )    ;
    if current[i].neg = 0.0 then
                      begin
                      current[i].negangle := 0.0;
                      current[i + transf].negangle := 0.0;
                      end;
    if current[i].neg > 0.0 then
                      begin
                      current[i].negangle :=  90;
                      current[i + transf].negangle := -90;
                      end;
    if current[i].neg < 0.0 then
                      begin
                      current[i].negangle :=  -90;
                      current[i + transf].negangle := 90;
                      end;
    current[i].neg := abs ( current[i].neg ) ;


{ corrections for angle currents when delta-star }
{ correction is for positive and negative sequence }

 if ( vpos[fb].angle - vpos[tb].angle ) < 0.0 then
       begin
           current[i].posangle := current[i].posangle +
                      (vpos[fb].angle - vpos[tb].angle) ;
           if current[i].neg <> 0.0 then
            current[i].negangle := current[i].posangle -
                      (vpos[fb].angle - vpos[tb].angle) ;
       end;

 if ( vpos[fb].angle - vpos[tb].angle ) > 0.0 then
      begin
          current[i + transf].posangle :=
                       current[i + transf].posangle -
                       (vpos[fb].angle - vpos[tb].angle);

          if current[i + transf].neg <> 0.0 then
```

```
            current[i + transf].negangle :=
                                current[i + transf].negangle +
                              (vpos[fb].angle - vpos[tb].angle);
      end;
{ corrections for star delta are done }


{ analysis for zero sequence delta star transformers }
{*************************************************}

if ( positive[i].al = zero[i].al ) and
                    ( positive[i].bl = zero[i].bl ) then
begin
  current[i].zero := (vzero[fb].magn - vzero[tb].magn)
                                        * 100 / zero[i].cl ;
  if current[i].zero = 0.0 then
                  begin
                    current[i].zeroangle := 0.0 ;
                    current[i + transf].zeroangle := 0.0 ;
                  end;

  if current[i].zero > 0.0 then
                 begin
                   current[i].zeroangle :=  90 ;
                   current[i + transf].zeroangle := -90;
                 end;

  if current[i].zero < 0.0 then
                 begin
                   current[i].zeroangle :=  -90;
                   current[i + transf].zeroangle := 90;
                 end;

  current[i].zero := abs ( current[i].zero ) ;
  current[i + transf].zero := current[i].zero ;
end; { in this case connection is star star both grounded }

if  ( zero[i].al = 0 ) and  ( zero[i].bl = 0 ) then
                 begin
                   current[i].zero := 0.0 ;
                   current[i + transf].zero := 0.0 ;
                   current[i].zeroangle := 0.0;
                   current[i + transf].zeroangle := 0.0 ;
                 end;
{ this was for connection delta delta }


{ now continue with star delta or  delta star }

if (zero[i].al = 0) and (zero[i].bl = positive[i].al) then
  begin
   current[i].zero := vzero[fb].magn * 100 / zero[i].cl ;
   if current[i].zero <> 0.0 then
```

```
                    current[i].zeroangle := 90
                    else
                       current[i].zeroangle := 0.0;;

    current[i + transf].zero := 0.0;
    current[i + transf].zeroangle := 0.0;
   end;

if (zero[i].al = 0) and (zero[i].bl = positive[i].bl) then
   begin
    current[i].zero := 0.0;
    current[i].zeroangle := 0.0;
    current[i + transf].zero := vzero[tb].magn
                                    * 100 / zero[i].cl ;
    if current[i + transf].zero <> 0.0 then
                  current[i + transf].zeroangle := 90
                  else
                     current[i + transf].zeroangle := 0.0;
   end;
{all connections are checked }



end; { for the loop of transformers in general }


END;


{ last modifications to include all faults }

{ this will include corrections for all faults }
if  ftype  >  2   then
 begin
   for i :=  ( gener + 1 ) to
              ( gener + lines +  ( 2 * transf ))  do

     if current[i].neg <> 0.0 then
      if current[i].negangle  > 0.0 then
       current[i].negangle :=  current[i].negangle - 180.0
        else
         current[i].negangle:=current[i].negangle+ 180.0;

  for i := 1 to ( gener + lines + ( 2 * transf ) ) do

    if current[i].zero <> 0.0 then
     if current[i].zeroangle > 0.0 then
      current[i].zeroangle := current[i].zeroangle - 180.0
       else
       current[i].zeroangle:=current[i].zeroangle + 180.0;

 end;
```

```
writeln ( outfile );
writeln ( outfile );
write ( outfile,
   ' sequence currents ( positive, negative, zero ) :');

writeln(outfile,
        '  magnitude (per unit) and angle (degrees) ' );
writeln ( outfile );

for i := 1 to ( gener + lines +   2 * transf  ) do
with current[i] do
writeln(outfile,'from',frombus:5,
                        '     to ', tobus:5,'     ',
            pos:10:5,'      ', posangle:10:5,'    ',
            neg:10:5,'      ', negangle:10:5,'    ',
            zero:10:5,'      ', zeroangle:10:5 );
writeln ( outfile );
writeln ( outfile );
write ( outfile,' phase currents ( A ,B ,C ) :');
writeln(outfile,'magnitude (per unit) and angle (degrees)');
writeln ( outfile );


for i := 1 to ( gener + lines + 2 * transf ) do
   begin
   seqtophase ( current[i].pos,  current[i].posangle,
                current[i].neg, current[i].negangle,
                current[i].zero, current[i].zeroangle,
                ia, iang, ib, ibang, ic, icang );

  writeln(outfile, 'from',current[i].frombus:5,'     to ',
                              current[i].tobus:5,'     ',
            ia:10:5,'     ',iang:10:5,'     ',
            ib:10:5,'     ',ibang:10:5,'     ',
            ic:10:5,'     ',icang:10:5 );
   end;


writeln( outfile);
writeln( outfile);
writeln( outfile);
write( outfile,'*****************************************');
writeln( outfile,'*****************************************');
writeln(outfile);
writeln(outfile);
writeln(outfile);

end; { end of procedure seqcurr ***************** }
```

```
{*********************************************************}
{ the following procedure writes sequence and phase voltage}
{ using as input sequence values .    Inside it calls the  }
{ procedure transform to get the phase values.             }
{*********************************************************}


procedure volwrite (vpos, vneg, vzero : volt ;
                              numbus : integer);


const
  epsilon = 0.0001;

var

  i : integer;
  va, vang, vb, vbang, vc, vcang : real ; { phase values }


begin

   { corrections - aproximations }
   for i := 1 to numbus do
   begin
   if vpos[i].magn <= epsilon then
      begin
      vpos[i].magn   := 0.0;
      vpos[i].angle := 0.0;
      end;
   if vneg[i].magn <= epsilon then
      begin
      vneg[i].magn  := 0.0;
      vneg[i].angle := 0.0;
      end;
   if vzero[i].magn <= epsilon then
      begin
      vzero[i].magn  := 0.0;
      vzero[i].angle := 0.0;
      end;

  end; { aproximations }


  write ( outfile, 'sequence voltages');
  write ( outfile,' ( positive, negative, zero ) :   ');
  writeln(outfile,
          'magnitude (per unit) and angle (degrees)');
  writeln ( outfile );
  for i := 1 to numbus do
      writeln ( outfile, 'bus  ',i:3,'    ',
      vpos[i].magn:9:5, '     ',vpos[i].angle:9:3,'     ',
```

```
      vneg[i].magn:9:5,'  ',vneg[i].angle:9:3,'      ',
      vzero[i].magn:9:5, '     ',vzero[i].angle:9:3 );
   writeln( outfile);
   writeln( outfile);

   write ( outfile,'phase voltages ( A, B, C ):' );
   writeln( outfile,
          '  magnitude (per unit) and angle (degrees)');
   writeln (outfile);
   for i := 1 to numbus do
   begin
   seqtophase( vpos[i].magn,  vpos[i].angle,
               vneg[i].magn, vneg[i].angle,
              vzero[i].magn,vzero[i].angle,
              va, vang, vb, vbang, vc, vcang );
   writeln ( outfile,'bus  ',i:3,'    ',
             va:10:5,'     ',vang:10:3,'     ',
             vb:10:5,'     ',vbang:10:3,'     ',
             vc:10:5,'     ',vcang:10:3 );
   end;
   writeln( outfile );
   writeln ( outfile );

end; { of procedure to write sequence and phase voltages }



{ *** PROCEDURES  FOR FAULTS ARE FOLLOWING  *** }
```

```
{*********************************************************}
{      PROCEDURE FOR SINGLE LINE TO GROUND FAULT          }
{*********************************************************}


procedure  slgfault  ( zbus1, zbus2, zbus0 : matrix ;
                       numbus, faultbus    : integer;
                                    shift : class  ;
                 var   vpos, vneg, vzero   : volt ) ;


var

 zold, ztemp   : compose;
 i,j,k         : integer;
 divisor       : real   ;


begin { procedure slgfault }

k := faultbus;    { clue for knowing which bus is faulted }


{initialize matrix that put
 together negative and zero sequence}

for i := 1 to ( 2 * numbus  + 1 ) do
   for j := 1 to ( 2 * numbus  + 1 ) do

       zold[i,j] := 0.0 ;


{ introducing zbus2  }
for i := 1 to numbus do
 for j := 1 to numbus do

      zold[i,j] := zbus2[i,j].x  + 1.0 ;


for  i := 1 to  ( numbus + 1 ) do
      zold[i, numbus + 1 ] :=  1.0;


for  j := 1 to numbus + 1  do
      zold[ numbus + 1, j ] := 1.0;


{ introducing zbus0 }
for i := 1 to numbus do
 for j := 1 to numbus do
   zold[numbus + 1 + i, numbus + 1 + j] := zbus0[i,j].x ;
```

```
 { now, introducing changes for this old bus }

 { forming  z  x   xt   z  }
for i := 1 to ( numbus + 1 )  do
   for j := 1 to ( numbus + 1 ) do

     ztemp[i,j] := 1.0 ;

for i := 1 to ( numbus + 1 ) do
 for j := 1 to numbus do

     ztemp[i, numbus + j + 1 ]   :=   - zbus0[k,j].x ;

for i := 1 to numbus do
 for j := 1 to ( numbus + 1 ) do

     ztemp[ numbus + i + 1 , j ] :=   - zbus0[i,k].x ;

for i := 1 to numbus do
 for j := 1 to numbus do
  ztemp[numbus + i + 1, numbus + j + 1] :=
                    zbus0[i,k].x   *   zbus0[k,j].x ;

{ we have completed zold and ztemp }
{ now we need zold - ztemp }
 divisor := 1 + zbus0[k,k].x ;

for i := 1 to ( 2 * numbus + 1 ) do
 for j := 1 to ( 2 * numbus + 1 ) do

 zold[i,j] := zold[i,j] - ( ztemp[i,j] / divisor ) ;

{ we have completed the union of zbus2 znd zbus0 }

{ now we need to remove the old reference of zbus2  }
divisor := 1 - zold[ numbus + 1, numbus + 1 ];
{ new ztemp }
for i := 1 to ( 2 * numbus + 1 ) do
 for j := 1 to ( 2 * numbus + 1 ) do
  ztemp[i,j]:= zold[i, numbus + 1] * zold[numbus + 1, j] ;

{ zold + ztemp/  divisor }

for i := 1 to ( 2 * numbus + 1 ) do
 for j := 1 to ( 2 * numbus + 1 ) do
  zold[i,j] := zold[i,j]  +  ( ztemp[i,j] /  divisor ) ;

{ ready for connecting with zbus1 }

divisor := zbus1[k,k].x  + zold[k,k]  ;
{ solution for slgfault }

for i:= 1 to numbus do
```

```
  vpos[i].magn := 1 - ( zbusl[i,k].x / divisor ) ;

for i := 1 to ( numbus + 1 ) do
 vneg[i].magn :=  zold[i,k] / divisor  ;

for  i := 1 to numbus do
 vzero[i].magn :=  zold[ numbus + i + 1 , k ] / divisor  ;

{  all results  can be written }
writeln ( outfile,
   '   ********************************************');
writeln ( outfile,
  '   *  RESULTS FOR SINGLE LINE TO GROUND FAULT   *' );
writeln ( outfile,
 '   ********************************************* ' );
writeln ( outfile );
writeln ( outfile, '     faulted bus is : ',k:4 );
writeln ( outfile );
writeln ( outfile );
{*** correction of values for negative sequence ********}
for i := 1 to numbus do
 vneg[i].magn := vneg[i].magn - vneg[ numbus + 1 ].magn ;

{ working now on the angles for the voltages }
for i := 1 to numbus do
 begin
 vpos[i].angle := 0.0 ;
 vneg[i].angle := 180.0;
 vzero[i].angle := 180.0;
 end;

 { corrections for the angles }
 for i := 1 to numbus do
 begin
 vpos[i].angle := vpos[i].angle  +  ( shift[i] * 30.0 ) ;
 vneg[i].angle := vneg[i].angle  -  ( shift[i] * 30.0 ) ;
 end;

 { writing results }

 volwrite ( vpos, vneg, vzero, numbus );


end; { procedure slgfault }
```

```
{*******************************************************}
{***     PROCEDURE FOR THREE PHASE FAULT             *****}
{*******************************************************}


procedure fault30 ( zbusl : matrix; numbus,
                    faultbus: integer  ;
                      shift : class;
                    var   vpos, vneg, vzero : volt );

var

   i, k : integer;


begin

    k := faultbus ; { clue for knowing fault bus }

    for i := 1 to numbus do
    begin
    vpos[i].magn := 1 - ( zbusl[i,k].x / zbusl[k,k].x ) ;
    vneg[i].magn := 0.0 ;
    vzero[i].magn := 0.0 ;
    vpos[i].angle := shift[i] * 30.0 ;
    vneg[i].angle := 0.0 ;
    vzero[i].angle := 0.0 ;
    end;

 { writing results for 30 fault . voltages  }

  writeln( outfile,'************************************');
  writeln( outfile,
   '   * RESULTS   FOR   THREE   PHASE   FAULT  * ' );
  writeln( outfile,
   '   ******************************************* ' );
  writeln( outfile);
  writeln( outfile, '     faulted bus is : ',k:4 );
  writeln ( outfile );
  writeln ( outfile );
  volwrite ( vpos, vneg, vzero, numbus );

end; { procedure for fault 30 }
```

```
{*****************************************************}
{      PROCEDURE FOR DOUBLE LINE FAULT                }
{*****************************************************}


procedure fault21 ( zbus1, zbus2                : matrix ;
                    numbus, faultbus            : integer;
                    shift                       : class  ;
                var vpos, vneg, vzero           : volt ) ;

var

 i,k     : integer ;
 divisor : real    ;


begin
  k := faultbus ; { faulted bus }
  divisor := zbus1[k,k].x  +  zbus2[k,k].x ;

  { assignning magnitudes for the voltages }
  for i := 1 to numbus do
  begin
   vpos[i].magn    :=  1 - ( zbus1[i,k].x / divisor ) ;
   vneg[i].magn    :=  zbus2[i,k].x / divisor ;
   vzero[i].magn   :=  0.0 ;
  end;

  { now working on the angles for the voltages }
  for i := 1 to numbus do
  begin
       vpos[i].angle  := 0.0;
       vneg[i].angle  := 0.0;
       vzero[i].angle := 0.0;
  end;

  { corrections for the angles }
  for i := 1 to numbus do
  begin
   vpos[i].angle := vpos[i].angle  +  ( shift[i] * 30 );
   vneg[i].angle := vneg[i].angle  -  ( shift[i] * 30 );
  end;

  { writing results for double line fault }
  writeln ( outfile,
   '   ********************************************');
  writeln ( outfile,
   '   *  RESULTS    FOR    DOUBLE    LINE    FAULT  *');
  writeln ( outfile,
   '   ********************************************');
  writeln ( outfile );
  writeln ( outfile, '     faulted bus is : ',k:4 );
```

```
  writeln ( outfile );
  writeln ( outfile );
  volwrite ( vpos, vneg, vzero, numbus );

end;  { end procedure for double line fault }
```

```
{*****************************************************}
{     PROCEDURE FOR DOUBLE LINE TO GROUND             }
{*****************************************************}

procedure fault21g ( zbus1, zbus2, zbus0          : matrix ;
                     numbus, faultbus             : integer;
                     shift                        : class  ;
                var  vpos, vneg, vzero            : volt ) ;


var

zold, ztemp : compose;
i,j,k       : integer;
divisor     : real   ;


begin    { begin of procedure double line to ground }

  k := faultbus;

  { initialize matrix that put together negative
                                  and zero sequence }
  for i := 1 to  ( 2 * numbus ) do
        for j := 1 to ( 2 * numbus ) do
        zold[i,j] := 0.0 ;

  { introducing zbus2 }
  for i := 1 to numbus do
        for j := 1 to numbus do
        zold[i,j] := zbus2[i,j].x ;

  { introducing zbus0 }
  for i := 1 to numbus do
        for j := 1 to numbus do
        zold[ numbus + i, numbus + j ] := zbus0[i,j].x ;

{ now we need zold  - ( zold * x * xt * zold  / divisor ) }
{ working with ztemp = ( zold * x * xt *  zold )    }

  for i := 1 to numbus do
                for j := 1 to numbus do
        ztemp[i,j] := zbus2[i,k].x  *  zbus2[k,j].x ;


{now rows from 1to numbus,columns from numb + 1 to 2 * numb}
  for i:= 1 to numbus do
    for j := 1 to numbus do
 ztemp[ i, numbus + j ] := - zbus2[i,k].x * zbus0[k,j].x;


{now rows from numbus + 1 to 2 numbus,columns 1 to numbus}
  for i := 1 to numbus do
```

```
    for j := 1 to numbus do
 ztemp[ numbus + i, j ] := - zbus0[i,k].x * zbus2[k,j].x;


{ now rows and columns from numbus + 1 to 2 numbus }
  for i := 1 to numbus do
    for j := 1 to numbus do
     ztemp[ numbus + i,numbus + j ] :=
                          zbus0[i,k].x * zbus0[k,j].x ;


{ now zold - ztemp / divisor }
  divisor := zbus2[k,k].x + zbus0[k,k].x ;
  for i := 1 to ( 2 * numbus ) do
    for j := 1 to ( 2 * numbus ) do
     zold[i,j] := zold[i,j] - ( ztemp[i,j] / divisor ) ;


{ we have completed the union of zbus2 and zbus0 }
{ now put this with zbus1 and calculate the voltages }
  divisor := zbus1[k,k].x + zold[k,k] ;

  { solution for doble line to ground is following }
  for i := 1 to numbus do
  begin
  vpos[i].magn   := 1 - ( zbus1[i,k].x / divisor ) ;
  vneg[i].magn   := zold[i,k] / divisor ;
  vzero[i].magn  := zold[ numbus + i, k ] / divisor ;
  vpos[i].angle  := shift[i] * 30 ;
  vneg[i].angle  := - shift[i] * 30 ;
  vzero[i].angle := 0.0;
  end;


{ writing results for double line to ground fault }
  writeln(outfile,
'  ************************************************');
  writeln(outfile,
'  * RESULTS  FOR  DOUBLE  LINE  TO  GROUND  FAULT   *');
  writeln(outfile,
'  ************************************************');
  writeln( outfile );

  writeln( outfile );
  writeln(outfile,'     faulted bus is : ',k:4);
  writeln ( outfile );
  writeln( outfile );

  volwrite ( vpos, vneg, vzero, numbus );


end;  { procedure doble line to ground fault }
```

```
{ ******************************************************}
{ ***          M A I N     P R O G R A M             ****}
{**********************************************************}


begin   { main program }


 writeln ( ' Enter the name of your input file ' );
 readln  ( inproof );
 writeln;

 writeln ( ' Enter the name of your output file ' );
 readln  ( outproof );
 writeln;

 assign  ( infile, inproof );
 assign  ( outfile, outproof );
 reset   ( infile );
 rewrite ( outfile );


 getdata (positive,negative,zero , numbus,total, gener,
                                  lines, transf,shift );

 writeln  ( ' positive sequence admittance' );
 formzbus ( positive, numbus, total, zbus1 );

 writeln  ( ' negative sequence admittance' );
 formzbus ( negative, numbus, total, zbus2 );

 writeln  ( ' zero sequence admittance' );
 formzbus ( zero, numbus, total, zbus0 );

 networks ( zbus1, zbus2, zbus0, numbus );

 writeln;



response := true ;
while  response do
begin

writeln ( ' Enter the bus number to be faulted ' );
readln( faultbus );

{now change shift of all buses
             according to that of faultbus}
for i := 1 to numbus do
    shift[i] := shift[i] - shift[ faultbus ] ;
```

```
 {******   THREE PHASE FAULT   *********************}

writeln( ' Do you want a  three phase fault ? : Y / N ' );
readln ( answer );

if ( answer = 'Y' ) or ( answer = 'y' ) then

begin
fault30 ( zbusl, numbus, faultbus, shift,
          vpos, vneg, vzero  );
ftype := 2 ; { this means fault is 30 }
seqcurr ( vpos, vneg, vzero, positive, negative, zero,
          gener, lines, transf,numbus, ftype );

writeln ('three phase fault was completed ' );
writeln;

end;


{********   DOUBLE LINE  FAULT    **********************}


writeln ( ' Do you want a two line fault ? : Y / N  ' ) ;
readln ( answer );

if  ( answer = 'Y' ) or ( answer = 'y' ) then

begin

fault21 ( zbusl, zbus2, numbus, faultbus,shift,
          vpos, vneg, vzero  );
ftype := 3 ; { this means fault is 21 }
seqcurr ( vpos, vneg, vzero, positive, negative, zero,
          gener, lines, transf,numbus, ftype );

writeln ( 'double line fault was done ' );
writeln ;

end;


{******   DOUBLE LINE TO GROUND FAULT    ***************}


write ( ' Do you want a two line to ground fault  ? ');
writeln ('   : Y / N ' );
readln ( answer );

if ( answer = 'Y' ) or ( answer = 'y' ) then
```

```
begin

fault2lg ( zbus1, zbus2, zbus0, numbus, faultbus, shift,
           vpos, vneg, vzero );
ftype := 4 ; { means fault is 2lg }
seqcurr ( vpos, vneg, vzero, positive, negative, zero,
          gener, lines, transf,numbus, ftype );

writeln ( ' double line ground fault was done ' );
writeln ;

end;

 {***     SINGLE LINE TO GROUND FAULT     ***************}


write( ' Do you want a single line to ground fault ?');
writeln('    : Y / N ' );
readln ( answer ) ;

if ( answer = 'Y') or ( answer = 'y' ) then

begin

slgfault ( zbus1, zbus2, zbus0, numbus, faultbus, shift,
           vpos, vneg, vzero  );
ftype := 1;  { this mean fault is single line to ground }
seqcurr ( vpos, vneg, vzero, positive, negative, zero,
          gener, lines, transf,numbus, ftype );


writeln ( ' single line to ground fault was done ' );
writeln;

end;


{*****************************************************}

write   ( outfile,
  '******** END ****** FOR *************** FAULTS ****');
writeln ( outfile,
 '********** OF ********** BUS **** ',faultbus:3,' ***');
writeln ( outfile );
writeln ( outfile );
writeln ( outfile );
writeln (outfile ,
'*************************************************');
writeln ( outfile );
writeln ( outfile );
writeln ( outfile );
```

```
write (' Do you want to repeat the
                                    faults for another bus ?');
writeln('  : Y / N ');
readln ( answer );

if ( answer = 'Y' ) or ( answer = 'y' ) then
response := true
else  response := false ;
end; { for repeating the faults }


{******************  *************************  *******}


close ( infile );
close ( outfile );


END.      { END OF MAIN PROGRAM }
```

A EFFICIENT ALGORITHM USING HOUSEHOLDER'S FORMULAS FOR THE SOLUTION OF FAULTED POWER SYSTEMS

by

ARMANDO ALTAMIRANO CHAVEZ

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

AN EFFICIENT ALGORITHM USING HOUSEHOLDER'S FORMULAS FOR THE SOLUTION OF FAULTED POWER SYSTEMS

ABSTRACT

An efficient algorithm for solving faulted power systems using the Householder's formulas is presented. The faults under consideration are three phase, single line to ground, double line, and double line to ground fault. The formulas for open and short circuit between two nodes are used to simulate connection among the three sequence networks in order to get voltages and currents in the faulted power system. A computer program based on the developed method is written in TURBO PASCAL for running on a personal computer. Results are found to be in an excellent agreement with those computed using other algorithms. The limitations of the algorithm are discussed. Suggestions and recomendations for enhancing the method are pointed out.