Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology

NAME OF THE PROJECT: **<u>Determination of Z (Z1, Z2, and Z0) matrices for a given network (For at least 6 bus system)</u>**

Course No.: **EEE 306**
Course Name: **Power System I Lab**
Group No**.: 1B**
Level: **3**
Term: **1**
Section: **B1**

Submitted To
Apratim Roy
Assistant Professor
EEE, BUET
&
Md. Shahadat Hasan Sohel
Lecturer
EEE, BUET

Submitted By.
Mohammad Tariqul Islam, 1006071
Khalid Mahmud, 1006072
Amit Kumar Kundu, 1006073
Naimul Hasan, 1006074

Date of Submission: 24/03/2013

Objective:

The main objective of our experiment is to find the Z matrices, that is positive sequence matrix (Z1), negative sequence matrix (Z2) and zero sequence matrix (Z0) for at least six bus system. For this we take the input in Database. We use the Zbus building algorithm on the graph to develop Z matrices.

Algorithms and formula :

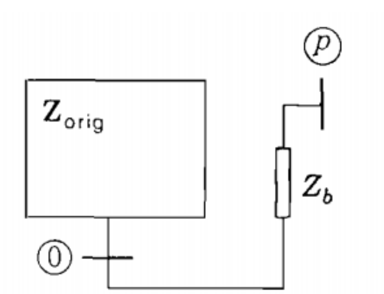
1.Assumption:

In order to make our project less clumsy we took necessary assumptions and built the code taking those assumptions in mind. These assumptions are:
1. All the elements are balanced.
2. All impedance data are given in per phase equivalent.
3. There are no regulating transformers in the network (for which Zbus is asymmetric).

2.Zbus building algorithm:

In Zbus building algorithms we can include 4 cases. These cases are :

Case1. To add a new bus ‘P’ with the reference node.
To do this we use this algorithm:
1. Add a new column and row (assuming k) in Zbus matrix.
2. All elements are zero initialized.
3. Zkk = Zb

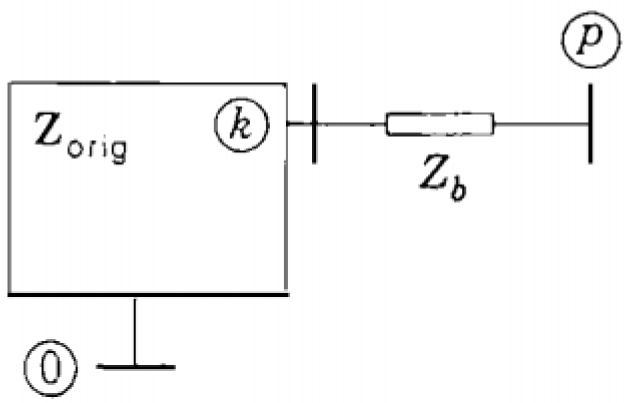


So the Zbus becomes

$$\begin{array}{cc} & \begin{array}{cc} \phantom{\mathbf{Z}_{orig}} & p \end{array} \\ p & \left[\begin{array}{c|c} \mathbf{Z}_{orig} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \quad \cdots \quad 0 & Z_b \end{array}\right] \end{array}$$

Case2. To add a new bus 'P' with an existing bus 'K'.
To do this we use this algorithm:

1. Create a new column and row in Zbus matrix
2. Copy column k and row k into new column p and row p respectively
3. Zpp = Zkk + Zb

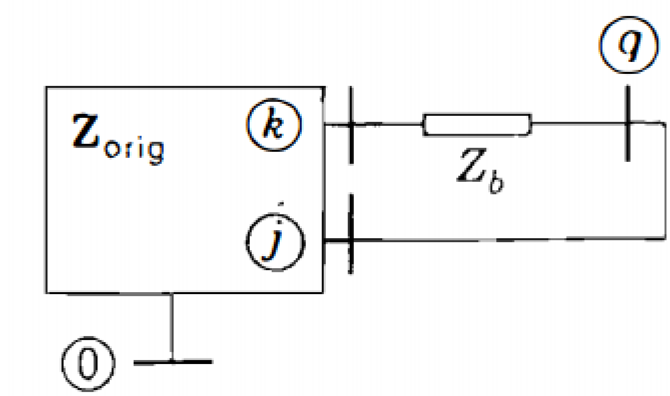


So the Zbus becomes

$$\begin{array}{cc} & \begin{array}{cc} k & p \end{array} \\ \begin{array}{c} k \\ p \end{array} & \left[\begin{array}{c|c} \mathbf{Z}_{orig} & \text{col. } k \\ \hline \text{row } k & Z_{kk} + Z_b \end{array}\right] \end{array}$$

Case3. To adding branch Zb from existing bus ‘k’ to a existing bus ‘j’
To do this we use this algorithm:

1. Add temporary node q.
2. Copy (column k-column j) to column q and copy (row k – row j).
3. Zth = Zjj + Zkk – 2 Zjk
4. Zqq = Zth + Zb
5. Use Kron reduction.
   For all index other than q:
   $$Z_{hi}(new) = Z_{hi} - Z_{hq} Z_{qi} / Z_{qq}$$
6. Delete node q.



So the Zbus becomes

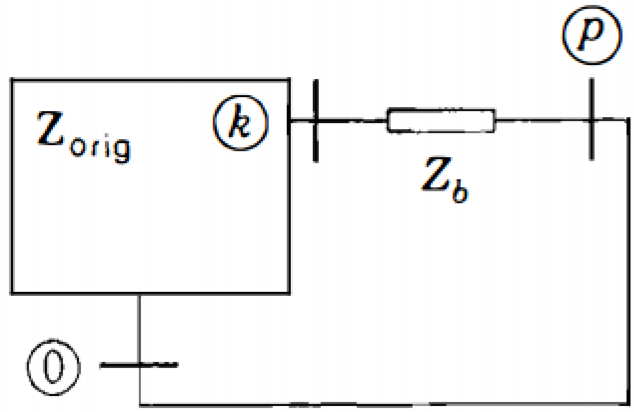
$$\begin{array}{cc} & \begin{array}{cc} \phantom{\text{row } j - \text{row } k} & q \end{array} \\ q & \left[\begin{array}{c|c} \mathbf{Z}_{orig} & \text{col. } j - \text{col. } k \\ \hline \text{row } j - \text{row } k & Z_{th,jk} + Z_b \end{array}\right] \end{array}$$

where $Z_{th,jk} = Z_{jj} + Z_{kk} - 2Z_{jk}$

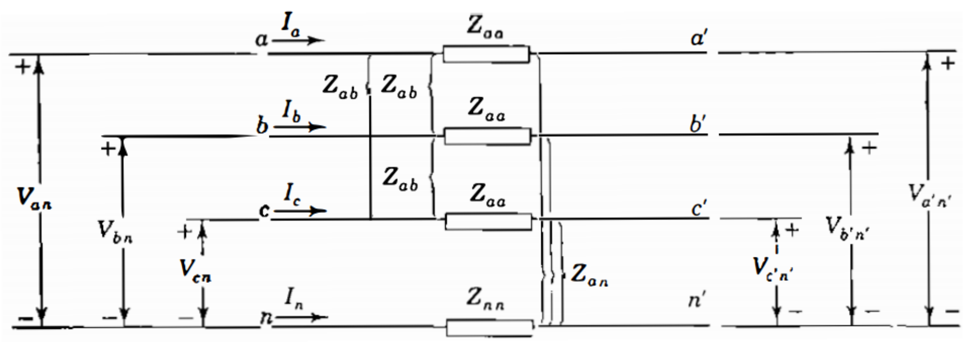
and

- Remove row q and column q by Kron reduction

Case4. To add a impedance to an existing bus from reference:
This case is not considered because we have already considered all the impedances from reference to bus in first step. For which, all such impedances are considered in case 1.

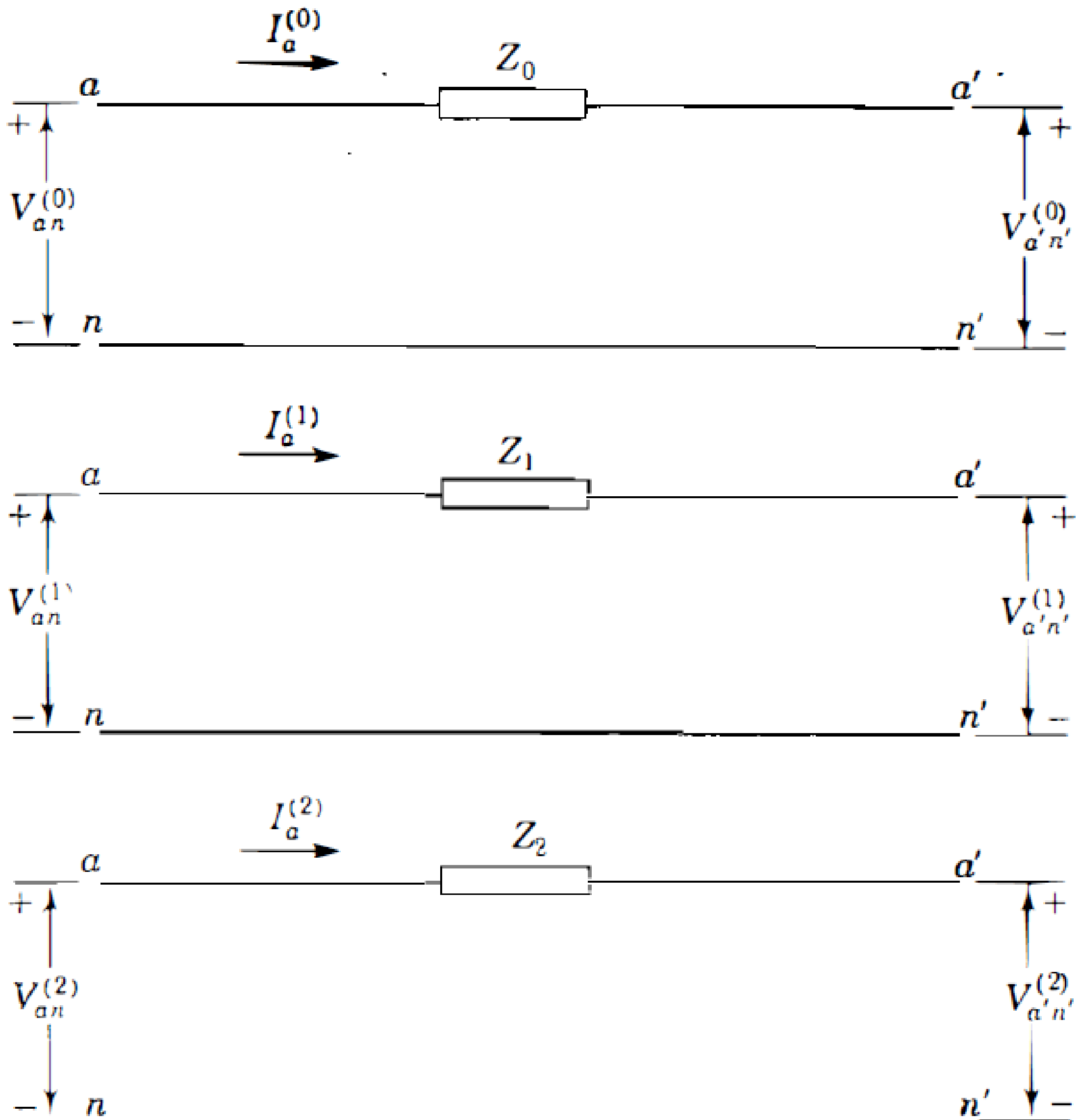


3.Sequence Circuits Of A symmetrical Transmission Line:



Zaa = Balanced transmission line impedance
Zab = Mutual Impedance between a, b, c phase
Znn = Impedance of neutral line
Zan = Mutual Impedance between a, b, c line and neutral

And the sequence circuit for the symmetrical line becomes

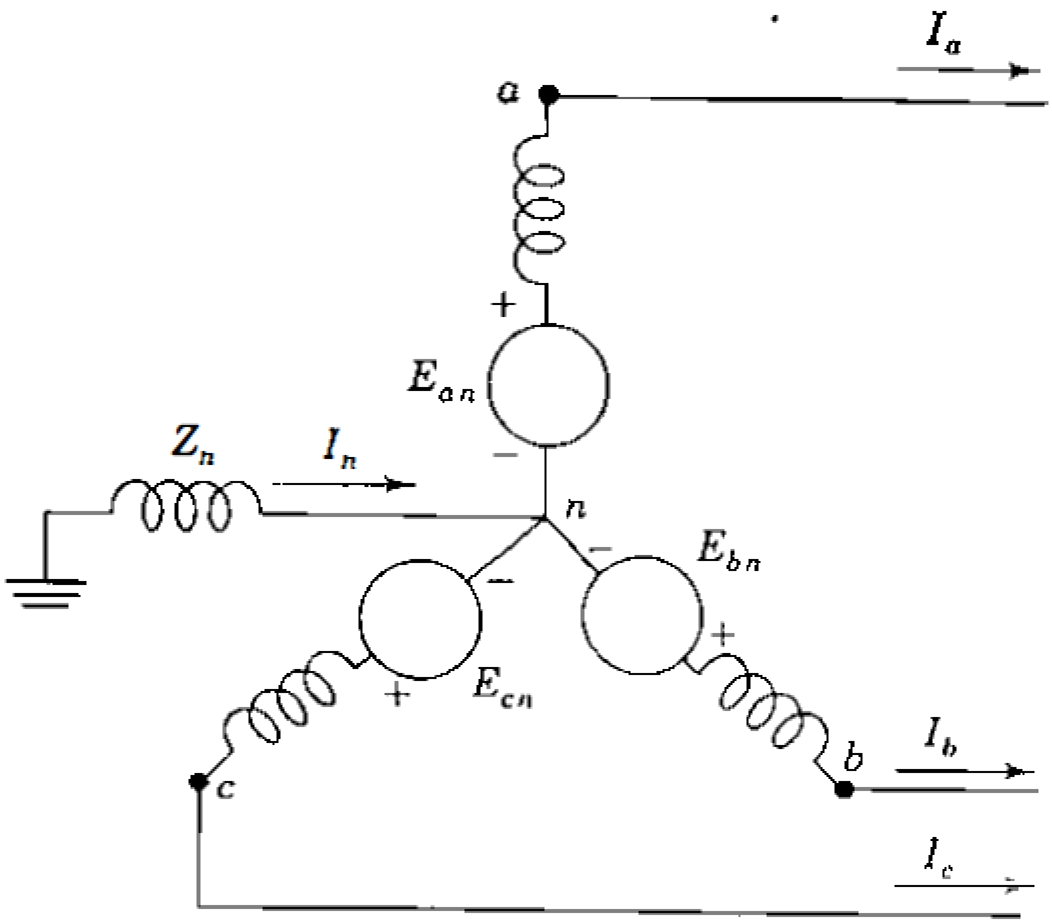


Where

$Z0 = Zaa + 2\ Zab + 3\ Znn – 6\ Zan$

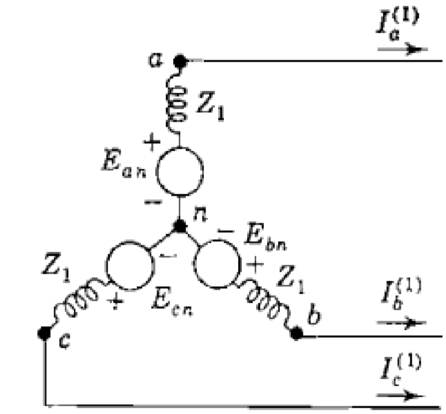
$Z1 = Z2 = Zaa - Zab$

4.Sequence Circuits Of The Synchronous Machine:

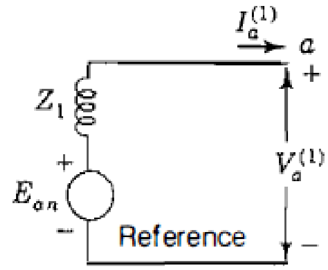


Z = Machine Impedance
Zm = Mutual Impedance
Zn = Neutral to reference impedance

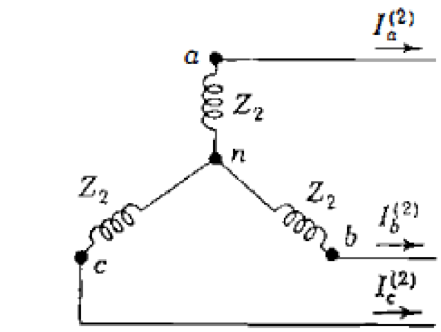
So the sequence network becomes



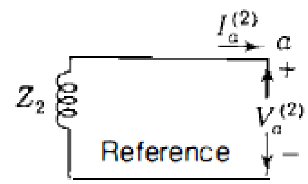
(a) Positive-sequence current paths



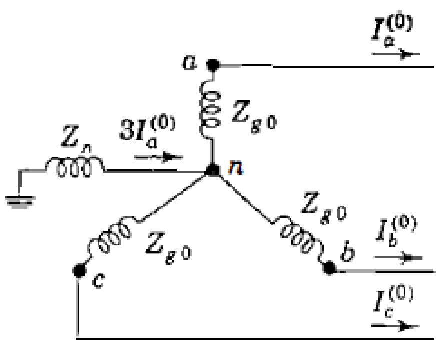
(b) Positive-sequence network



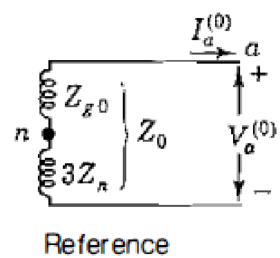
(c) Negative-sequence current paths



(d) Negative-sequence network



(e) Zero-sequence current paths

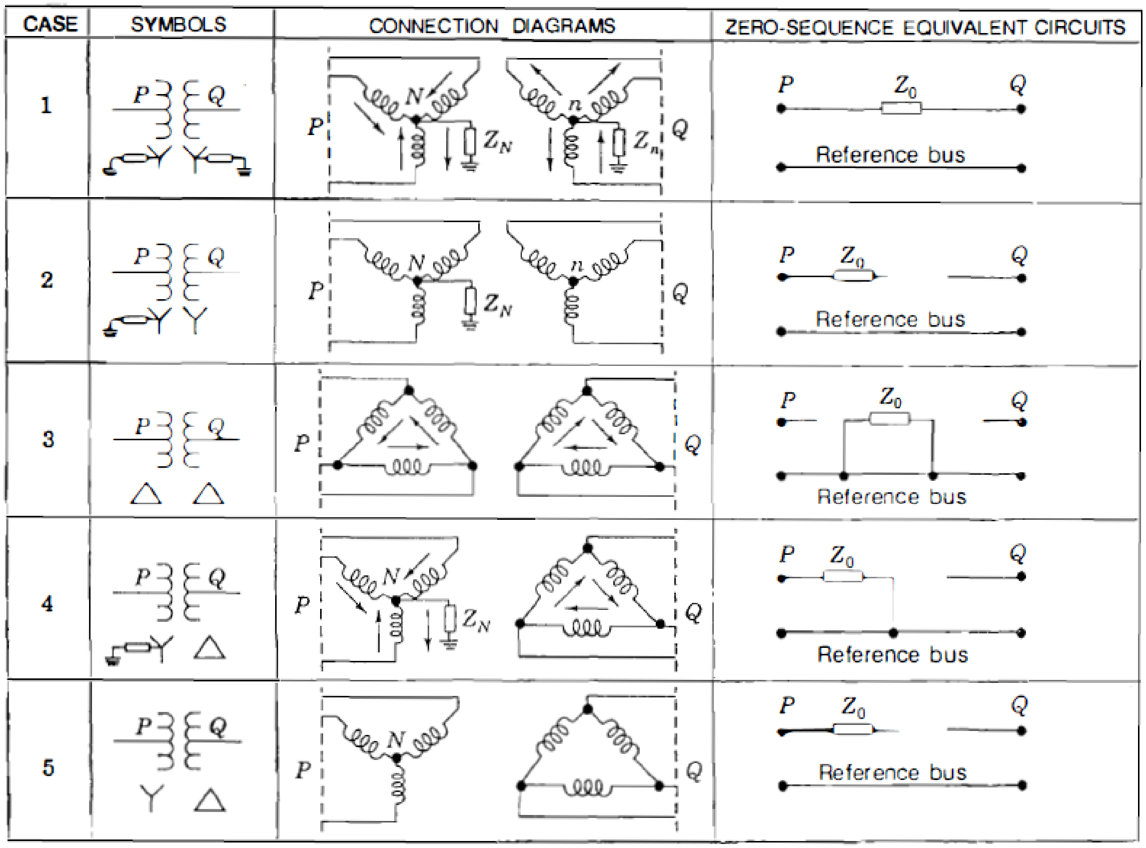


(f) Zero-sequence network

Where
Z0=3Zn+Zg0

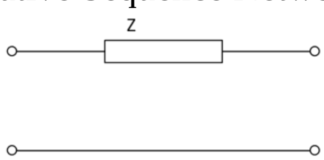
5.Sequence Circuits Of Y-D Transformers:

For circuits of Y-D transformers we mainly deal with zero-sequence equivalent circuits. They are as follows:



Where Z0=Z+3ZN+3Zn

And for Positive/Negative Sequence Network



Load data is usually given in real power. But here we have considered the per phase equivalent impedance.

Code:

The Z matrices for positive and negative sequence networks are same. Therefore Z1=Z2 . So we built oud code for zero sequence and positive sequence networks.

In our code in this project we took system information in a json file which is java script object notation. In a json file a description of various machines/ lines will be written of that particular power system. We used p_json() function to read the json files. This json files is a storing method to save bus data, line data, machine data. This p_json() function is a built in function which reads the files. json_read() function reads that file and returns those information in that file as structures.

```
function [y, len] = json_read(fname)
%
%Reads a json file using p_json() library
%USAGE: json_read(filename)
%
    fid = fopen(fname, 'rt');
    str = fscanf(fid,'%c');
    fclose(fid);
    y = p_json(str);
    if (isempty(y))
        len=0;
    else
        len=length(fieldnames(y));
    end
 end
```

In the main function we read busdata using json_read function. The length of bus data was saved. And we know the number of buses. According to that number an 2-D matrix was created using zeros function.

```
[~,len] = json_read('busdata.json'); %loading bus data from database

nG= zeros(len+1); %network graph for +ve and -ve sequence network
nG0= zeros(len+1); %network graph for zero sequence
l_0 = length(nG0); %size of zero sequence network
```

After that transformer data was read and using transformer_Z function we saved the parameters of all the transformers and updated the 2-D networks matrices for zero sequence and positive sequence according the algorithms.

```
for i=1:ll
    %getting Z1(/Z2) and Z0 data
    Z = transformer_Z(getfield(trandata,trfname{i}));

    %adding Z1(/Z2) to the network graph
    Z1 = Z.Z1;
    nG(Z1.b1,Z1.b2) = parallel_load(Z1.val,nG(Z1.b1,Z1.b2));
    nG(Z1.b2,Z1.b1) = nG(Z1.b1,Z1.b2);

    %processing for Z0
    Z0 = Z.Z0;
    if(Z0.b2=='N') %then new bus has to be created
        l_0 = l_0+1;
        nG0(l_0,:) = 0;
        nG0(:,l_0) = 0;
        Z0.b2=l_0; %adding this bus to the Z0 bus data for this transformer
    end
    %adding Z0 to the network graph
    nG0(Z0.b1,Z0.b2) = parallel_load(Z0.val,nG0(Z0.b1,Z0.b2));
    nG0(Z0.b2,Z0.b1) = nG0(Z0.b1,Z0.b2);
end
```

Next , we read generator data and using generator_Z function we saved the parameters of all the generators and updated the 2-D network matrices for zero sequence and positive sequence according the algorithms.

```
[genData,ll] = json_read('generatordata.json'); %loading generator data
if ll
    gfname = fieldnames(genData); %oading generator names
end

%for each generator doing the operations
for i=1:length(gfname)
    %getting generato's Z1(/Z2) and Z0
    Z=generator_Z(getfield(genData,gfname{i}));

    %adding Z1 to the network graph
    nG(Z.bus,1)= parallel_load(Z.Z1,nG(Z.bus,1));
    nG(1,Z.bus)= nG(Z.bus,1);

    %adding Z0 to Zero sequence network graph
    if ~Z.New
        nG0(Z.bus,1) = parallel_load(Z.Z0,nG0(Z.bus,1));
        nG0(1,Z.bus) = nG0(Z.bus,1);
    else
        l_0 = l_0+1;
        nG0(l_0,:) = 0;
        nG0(:,l_0) = 0;
        Z.bus2=l_0; %adding this bus to the Z0 bus data for this transformer

        nG0(Z.bus, Z.bus2) = parallel_load(Z.Z0,nG0(Z.bus, Z.bus2));
        nG0(Z.bus2, Z.bus) = nG0(Z.bus, Z.bus2);
    end
end
```

In the same process line data and load data was read gradually and continuously update the network matrices. When this is finished we found a complete description of the positive and negative and zero sequence reactance diagram of that specified network. Using this diagrams we can now calculate Z matrices. The zbus() function creates Zbus for positive and negative sequence networks according to the discussed algorithm.

Code of different functions:

transformer_Z:

```
function y = transformer_Z(data)
    %loading transformer parameter
    pw = data.primary_winding;
    sw = data.secondary_winding;
    pn= formatted_data(data.primary_bus)+1;

    sn = formatted_data(data.secondary_bus)+1;
```

```
    R = formatted_data(data.R1);
    X = formatted_data(data.X1);
    Z1 = R+X*1i;

    y = struct;
    y.Z0 = struct;
    y.Z1 = struct;

    y.Z1.val = Z1;
    y.Z1.b1 = pn;
    y.Z1.b2 = sn;

    %Z0 is calculated for different types of connection
    if (strcmp(pw,'Yg') && strcmp(sw,'Yg'))
        znp = formatted_data(data.primary_zn);
        zns = formatted_data(data.secondary_zn);

        y.Z0.val = Z1 + 3*znp + 3*zns;
        y.Z0.b1 = pn;
        y.Z0.b2 = sn;
    elseif (strcmp(pw,'Yg') && strcmp(sw,'D'))
        znp = formatted_data(data.primary_zn);

        y.Z0.val = Z1 + 3*znp;
        y.Z0.b1 = pn;
        y.Z0.b2 = 1;
    elseif (strcmp(pw,'D') && strcmp(sw,'Yg'))
        zns = formatted_data(data.secondary_zn);

        y.Z0.val = Z1 + 3*zns;
        y.Z0.b1 = sn;
        y.Z0.b2 = 1;
    elseif (strcmp(pw,'Yg') && strcmp(sw,'Y'))
        y.Z0.val = Z1;
        y.Z0.b1 = pn;
        y.Z0.b2 = 'N';
    elseif (strcmp(pw,'Y') && strcmp(sw,'Yg'))
        y.Z0.val = Z1;
        y.Z0.b1 = sn;
        y.Z0.b2 = 'N';
    elseif (strcmp(pw,'Y') && strcmp(sw,'D'))
        y.Z0.val = Z1;
        y.Z0.b1 = pn;
        y.Z0.b2 = 'N';
    elseif (strcmp(pw,'D') && strcmp(sw,'Y'))
        y.Z0.val = Z1;
        y.Z0.b1 = sn;
        y.Z0.b2 = 'N';
    else
        y.Z0.val = 0;
        y.Z0.b1 = 0;
        y.Z0.b2 = 0;
    end

end
```

```
function y=formatted_data(x)
    if ischar(x)
        y=str2double(x);
    else
        y=x;
    end
end
```

generator_Z:

```
function y = generator_Z(data)
    %loading generator parameter
    bus = formatted_data(data.bus)+1;
    R = formatted_data(data.sub_tr_R);
    X = formatted_data(data.sub_tr_X);
    M = formatted_data(data.sub_tr_M);

    W = data.winding;
    if (strcmp(W,'Yg'))
        if isfield(data,'Rn')
            Rn = formatted_data(data.Rn);
        else
            Rn=0;
        end

        if isfield(data,'Xn')
            Xn = 1i*formatted_data(data.Xn);
        else
            Xn = 0i;
        end
        Zn = Rn+Xn;
    else
        Zn=0;
    end

    Z1=R+(X+M)*1i;
    Z0=R+(X-2*M)*1i+3*Zn;

    %Returning appropriate data
    y=struct;
    y.Z1=Z1;
    y.Z0=Z0;
    y.bus=bus;
    if (strcmp(W,'Yg'))
        y.New = 0;
    else
        y.New = 1;
    end
end

function y=formatted_data(x)
    if ischar(x)
        y=str2double(x);
    else
```

```
        y=x;
    end
end
```

line_Z:

```
function y=line_Z(data)
    %loading line parameters from data
    b1 = formatted_data(data.bus1)+1;
    b2 = formatted_data(data.bus2)+1;

    Raa = formatted_data(data.Raa);
    Xaa = formatted_data(data.Xaa)*1i;
    Zaa = Raa+Xaa;

    if isfield(data,'Xab')
        Xab = formatted_data(data.Xab)*1i;
    else
        Xab=0;
    end

    if isfield(data,'Xan')
        Xan = formatted_data(data.Xan)*1i;
    else
        Xan=0;
    end

    if isfield(data, 'Rnn')
        Rnn = formatted_data(data.Rnn);
    else
        Rnn = 0;
    end

    if isfield(data, 'Xnn')
        Xnn = formatted_data(data.Xnn)*1i;
    else
        Xnn=0;
    end

    Znn = Rnn+Xnn;

    %calculating
    Z0 = Zaa+2*Xab+3*Znn-6*Xan;
    Z1 = Zaa-Xab;

    %returning data
    y=struct;
    y.b1 = b1;
    y.b2 = b2;
    y.Z1 = Z1;
    y.Z0 = Z0;

end
```

```
function y=formatted_data(x)
    if ischar(x)
        y=str2double(x);
    else
        y=x;
    end
end
```

load_Z:

```
function y = load_Z(data)
    bus = formatted_data(data.bus)+1;
    R = formatted_data(data.R);
    X = formatted_data(data.X)*1i;
    c=data.connection;
    if strcmp(c, 'Yg');
        if isfield(data,'Rn')
            Rn = formatted_data(data.Rn);
        else
            Rn=0;
        end

        if isfield(data,'Xn')
            Xn = formatted_data(data.Xn)*1i;
        else
            Xn=0;
        end
    end

    Z = R+X;
    Z0 = Z+Rn+Xn;

    y=struct;
    y.bus = bus;
    y.Z1 = Z;
    y.Z0 = Z0;
end

function y=formatted_data(x)
    if ischar(x)
        y=str2double(x);
    else
        y=x;
    end
end
```

parallel_load:

```
function y=parallel_load(val1,val2)
%parallel impedance calculator. 0 means impedence is infinite
    if val1==0
        y=val2;
    elseif val2==0
        y=val1;
    else
        y=val1*val2/(val1+val2);
```

```
    end
end
```

zbus:

```
function ZBUS = zbus(imp)
%this function finds the zbus of a pwer system from a impedence graph
of
%the system
%imp is a matrix that describes the graph between impedences.
%tariqul islam
%19/03/2014

    n=length(imp);
    imp2 = imp(1,2:n); %extracting node to reference impedences for
each bus
    imp = imp(2:n,2:n); %bus to bus impedences
    n=n-1;
    ZBUS = zeros(n+1); %ZBUS shell
    F=zeros(1,n); %index for knowing which bus is in which column of
ZBUS
    fi = 0; %index of F
    FUT = [0 0]'; % memory for isolating the branch which's node is not
inside the current ZBUS
    bi = 0; %index of ZBUS



    %modifies the bus matrix using Kron Reduction
    %random thoughts: why I didn't name it kronred?
    function busmod(z)
        for p=1:z-1
            for q=1:z-1
                ZBUS(p,q)=ZBUS(p,q)-ZBUS(p,z)*ZBUS(z,q)/ZBUS(z,z);
            end
        end
        bi=bi-1;
    end

    %finds whether the z node is inside ZBUS
    function y = getindexof(z)
        found=0;
        for i=1:length(F)
            if F(i)==z
                found=1;
                break;
            end
        end

        if found==1
            y=i;
        else
            y=0;
        end
    end

    %adds a new node with an existing node
```

```
%adapted from CASE 2
%Power System Analysis,
%Stevenson & Grainger, McGraw Hill India, Page: 299
function case1(j, val)
    bi=bi+1;
    ZBUS(bi,:) = ZBUS(j,:);
    ZBUS(:,bi) = ZBUS(:,j);
    ZBUS(bi,bi) = ZBUS(j,j)+val;
end

%adds impedence between two existing node
%adapted from CASE 4
%Power System Analysis,
%Stevenson & Grainger, McGraw Hill India, Page: 299
function case2(j,k,val)
    bi=bi+1;
    ZBUS(bi,:) = ZBUS(j,:) - ZBUS(k,:);
    ZBUS(:,bi) = ZBUS(:,j) - ZBUS(:,k);
    ZBUS(bi,bi) = ZBUS(j,j)+ZBUS(k,k)-2*ZBUS(j,k)+val;
end



%the general loop for the algorithm
function general_loop(a,b)
    if abs(imp(a,b))
        j=getindexof(a);
        k=getindexof(b);

        %if j=0 swaping it with k
        swapped=0;
        if j==0
            j=k;
            k=0;
            swapped=1;
        end

        %isolating non-existing nodes for future
        %if j=0 for successive checking (1st one above, 2nd one
        %below),
        %it means both j=0; k=0;
        if j==0
            fi=fi+1;
            FUT(:,fi) = [a b]';
            return;
        end

        %otherwise j=value, k=0 (CASE 2)
        if k==0
            case1(j,imp(a,b));
            if swapped
                F(bi)=a;
            else
                F(bi)=b;
            end
```

```
        %otherwise j=value, k=valuse (CASE 4)
        else
            case2(j,k,imp(a,b));
            busmod(bi);
        end
    end
end

%adding all buses connected to reference
%adapted from CASE 1
%Power System Analysis,
%Stevenson & Grainger, McGraw Hill India, Page: 299
for b=1:n
    if abs(imp2(b))
        bi=bi+1;
        ZBUS(bi,bi) = imp2(b);
        F(bi)=b;
    end
end

%adding the bues that are connected to other buses
for a=1:n
    for b=a+1:n
        general_loop(a,b);
    end
end

%now checking the nodes that were kept for future
fii=fi;
fi2=0;
while fi~=0
    a=FUT(1,1);
    b=FUT(2,1);
    FUT(:,1)=[];
    fi=fi-1;
    general_loop(a,b);

    fi2=fi2+1;
    if fi2==fii
        if fii==fi;
            break;
        else
            fi2=0;
            fii=fi
        end
    end
end

for a=1:n
    z=getindexof(a);
    if a==z
        continue;
    else
        ZBUS(n+1,:)=ZBUS(z,:);
        ZBUS(z,:)=ZBUS(a,:);
        ZBUS(a,:)=ZBUS(n+1,:);
```

```
            ZBUS(:,n+1)=ZBUS(:,z);
            ZBUS(:,z) = ZBUS(:,a);
            ZBUS(:,a) = ZBUS(:,n+1);

            F(z) = F(a);
            F(a) = a;

        end
    end

    ZBUS = ZBUS(1:n,1:n);
end
```

Features:

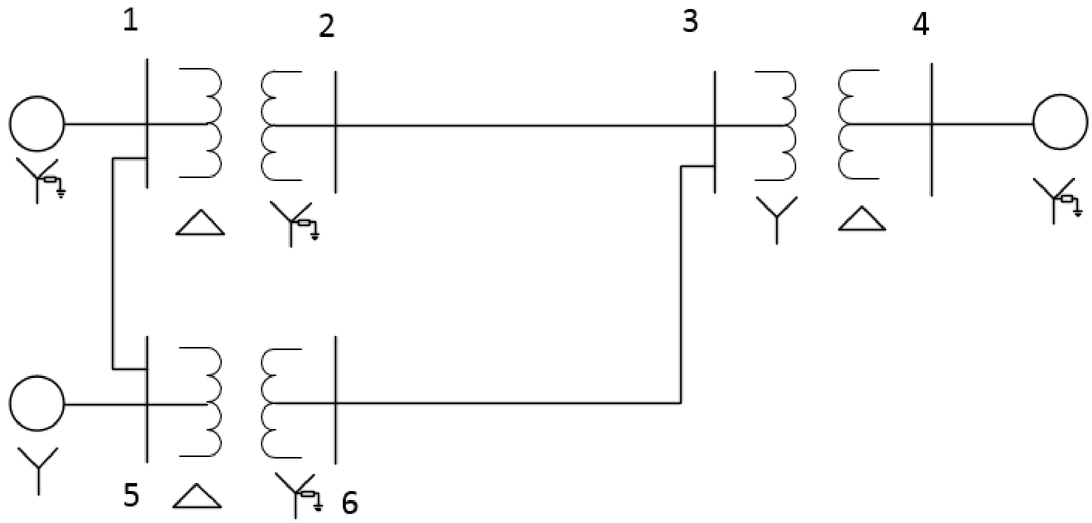
Our code works for any number of buses. It is a general code which is kept for future expansion. That is anyone can develop fault analysis or any other algorithm in power system over this code. This code is user friendly as the system information can be easily read and write using json files.

Limitations:

The effect of mutual coupled branches are not included in this code. We assumed all the impedances are balanced. So it can not work for unbalanced systems. Again there can be no regulating transformers in the network.

Test run:

We run this code for following 6 bus system:



Its generator transformer line and load information are given in json files which are as follows:

Bus data:

```
{
   "bus1":{
      "N":1,
      "V":13.8
    },
   "bus2":{
      "N":2,
      "V":69
   },
   "bus3":{
      "N":3,
      "V":69
   },
   "bus4":{
      "N":4,
      "V":13.8
   },
   "bus5":{
      "N":5,
      "V":13.8
```

},
"bus6":{
"N":6,
"V":69
}
}

Transformer data:

{
"tr1":{
"primary_kv":69,
"secondary_kv":13.8,
"primary_winding":"Yg",
"secondary_winding":"D",
"primary_zn": 0,
"primary_bus":2,
"secondary_bus":1,
"rating_mva":100,
"R1": "0.0",
"X1": "0.08",
"SLM_mva": 100
},
"tr2":{
"primary_kv":69,
"secondary_kv":13.8,
"primary_winding":"Y",
"secondary_winding":"D",
"primary_bus":3,
"secondary_bus":4,
"rating_mva":100,
"R1": "0.0",
"X1": "0.08",
"SLM_mva": 100
},
"tr3":{
"primary_kv":69,
"secondary_kv":13.8,
"primary_winding":"Yg",
"secondary_winding":"D",
"primary_zn": 0,

        "primary_bus":6,
        "secondary_bus":5,
        "rating_mva":100,
        "R1": "0.0",
        "X1": "0.08",
        "SLM_mva": 100
    }
}

Generator data:

{
    "g1":{
        "bus": 1,
        "rated_kv": 13.8,
        "rated_mva": 100,
        "sub_tr_R": 0,
        "sub_tr_X": 0.14667,
        "sub_tr_M": 0.05333,
        "tr_R": 0,
        "tr_X": 0,
        "tr_M": 0,
        "R": 0.01,
        "X": 0.12,
        "M": 0,
        "winding": "Yg",
        "Rn": 0,
        "Xn": 0.05,
        "activeGeneration_MVA": "-"
    },
    "g2":{
        "bus": 4,
        "rated_kv": 13.8,
        "rated_mva": 200,
        "sub_tr_R": 0,
        "sub_tr_X": 0.14667,
        "sub_tr_M": 0.05333,
        "tr_R": 0,
        "tr_X": 0,
        "tr_M": 0,
        "R": 0.01,

```
        "X": 0.12,
        "M": 0,
        "winding": "Yg",
        "Rn": 0,
        "Xn": 0.05,
        "activeGeneration_MVA": 100
    },
    "g3":{
        "bus": 5,
        "rated_kv": 13.8,
        "rated_mva": 200,
        "sub_tr_R": 0,
        "sub_tr_X": 0.14667,
        "sub_tr_M": 0.05333,
        "tr_R": 0,
        "tr_X": 0,
        "tr_M": 0,
        "R": 0.01,
        "X": 0.12,
        "M": 0,
        "winding": "Y",
        "activeGeneration_MVA": 100
    }
}
```


Line data:


```
{
    "L34":{
          "bus1": 2,
        "bus2": 3,
        "Raa" : 0,
        "Xaa" : 0.2,
        "Xab" : 0.05,
        "Xan" : 0.08333,
        "Rnn" : 0,
        "Xnn" : 0.1
    },
    "L15":{
          "bus1": 1,
        "bus2": 5,
```

"Raa" : 0,
"Xaa" : 0.2,
"Xab" : 0.002,
"Xan" : 0.001,
"Rnn" : 0,
"Xnn" : 0.01
},
"L63":{
"bus1": 6,
"bus2": 3,
"Raa" : 0,
"Xaa" : 0.2,
"Xab" : 0.05,
"Xan" : 0.08333,
"Rnn" : 0,
"Xnn" : 0.1
}
}

Load data:

{
}

Sample result:

Z1 =

Columns 1 through 5

0 + 0.1055i 0 + 0.0885i 0 + 0.0566i 0 + 0.0404i 0 +
0.0541i
0 + 0.0885i 0 + 0.1374i 0 + 0.0792i 0 + 0.0566i 0 +
0.0550i
0 + 0.0566i 0 + 0.0792i 0 + 0.1216i 0 + 0.0869i 0 +
0.0566i
0 + 0.0404i 0 + 0.0566i 0 + 0.0869i 0 + 0.1192i 0 +
0.0404i
0 + 0.0541i 0 + 0.0550i 0 + 0.0566i 0 + 0.0404i 0 +
0.1055i
0 + 0.0550i 0 + 0.0634i 0 + 0.0792i 0 + 0.0566i 0 +
0.0885i

Column 6

```
   0 + 0.0550i
   0 + 0.0634i
   0 + 0.0792i
   0 + 0.0566i
   0 + 0.0885i
   0 + 0.1374i
```

Z2 =

Columns 1 through 5

```
   0 + 0.1055i   0 + 0.0885i   0 + 0.0566i   0 + 0.0404i   0 + 0.0541i
   0 + 0.0885i   0 + 0.1374i   0 + 0.0792i   0 + 0.0566i   0 + 0.0550i
   0 + 0.0566i   0 + 0.0792i   0 + 0.1216i   0 + 0.0869i   0 + 0.0566i
   0 + 0.0404i   0 + 0.0566i   0 + 0.0869i   0 + 0.1192i   0 + 0.0404i
   0 + 0.0541i   0 + 0.0550i   0 + 0.0566i   0 + 0.0404i   0 + 0.1055i
   0 + 0.0550i   0 + 0.0634i   0 + 0.0792i   0 + 0.0566i   0 + 0.0885i
```

Column 6

```
   0 + 0.0550i
   0 + 0.0634i
   0 + 0.0792i
   0 + 0.0566i
   0 + 0.0885i
   0 + 0.1374i
```

```
Z0 =

  Columns 1 through 5

       0 + 0.1900i        0                  0                  0                  0 + 0.1900i
       0                  0 + 0.0622i        0 + 0.0400i        0                  0
       0                  0 + 0.0400i        0 + 0.0900i        0                  0
       0                  0                  0                  0 + 0.1900i        0
       0 + 0.1900i        0                  0                  0                  0 + 0.4180i
       0                  0 + 0.0178i        0 + 0.0400i        0                  0

  Column 6

       0
       0 + 0.0178i
       0 + 0.0400i
       0
       0
       0 + 0.0622i

>>
```

Conclusion:

This code is useful and can be edited to develop the code of power flow solutiuon. It can also be replaced as a substitute of psaf in some cases.