Chapter 6. Power Flow Analysis

6.1 Introduction

The system is assumed 3-phase balanced operating in a steady state and stable condition. It is represented by a single phase diagram on a per-unit basis, with a system wide MVA base, and a voltage base properly chosen on each side of every transformer. The base MVA and base V are specified/known everywhere in the system.

The most common way to represent such a system is to use the node-voltage method. Given the voltages of generators at all generator nodes, and knowing all impedances of machines and loads, one can solve for all the currents in the typical node voltage analysis methods using Kirchoff's current law. First the generators are replaced by equivalent *current sources* and the node equations are written in the form:

$$I = YV$$

where I is the injected current vector, Y is the admittance matrix and V is the node voltage vector. These equations are easy to write by inspection of the circuit.

The problem is not so simple in real power circuits and systems. Usually in a power system the complex power may be known at load nodes, and sometimes on generator nodes only the real power and voltage are known. Thus not enough variables are known to solve an equation of the form $I = YV$. In fact, since the power is a nonlinear function of the current and voltage, the solution of the resulting equations (while it may exist) is not easy! In fact there is no known analytical method to find the solution. As a result iterative techniques are used to find the solution (voltages, currents, etc.). The nonlinear set of equations which are generated are called *power flow equations.* The solution of such equations results in a *power flow study* or *load flow analysis.* Such studies are the backbone of power system studies, for analysis, design, control, and economic operation of the power system. They are also essential for transient analysis of the system.

6.2 Bus Admittance Matrix

1. The first step is to number all the nodes of the system from 0 to n. Node 0 is the datum or reference node (or ground node).
2. replace all generators by equivalent current sources in parallel with an admittance.
3. Replace all lines, transformers, loads to equivalent admittances whenever possible. Knowing a load in MVA but not knowing the operating voltage, makes it impossible to change the load to an admittance. The rule for this is simple: $y = 1/z$ where y and z are generally complex numbers.
4. The bus admittance matrix Y is then formed by inspection as follows (this is similar to what we learned in circuit theory):

$$y_{ii} = \text{sum of admittances connected to node } i$$

and

$$y_{ij} = y_{ji} = -\text{sum of admittances connected from node } i \text{ to node } j$$

5. The current vector is next found from the sources connected to nodes 0 to n. If no sourde is connected, the injected current would be 0.
6. The equations which result are called the *node-voltage equations* and are given the "bus" subscript in power studies thus:

$$I_{bus} = Y_{bus}V_{bus}$$

The inverse of these equations results in the set $V_{bus} = Y_{bus}^{-1} I_{bus}$. *It is emphasized that the matrix Y_{bus}^{-1} is not the same as the matrix Z which results from solving a circuit using mesh equations. To clearly show this difference we define $Z_{bus} = Y_{bus}^{-1}$. It is noted that the matrix Y_{bus} is the same as the Y matrix obtained from circuit theory. Thus $Y_{bus} = Y$ however $Z_{bus} \neq Z$.*

It is also noted that in general, solving a large set of linear equations is never done using the matrix inverse. Finding the inverse involves more work than is needed to find a solution and the resulting solutions are less accurate. Usually Gauss elimination or a similar factorization scheme is used in a direct solution. Iterative solutions are also very effective in larger systems.

It is further noted that in a real power system with hundreds of nodes, each node is rarely connected to more than two or three other nodes, thus most of the elements of the admittance matrix are zero (the admittance matrix is sparse). Special techniques exist to solve systems with sparse matrices.

In a power system the Z_{bus} is built up directly from the power system (see section 9.5) and matrix inversion is not used. Also, we have a routine to form the admittance matrix called *ybus* used as follows: `Y = ybus(zdata)` where "zdata" is a matrix of entries having n-rows and 4 columns. Column 1 and 2 indicate the nodes between which the impedance exists (nonzero), and columns 3 and 4 give the real and imaginary parts of the impedance. The result is the bus admittance matrix Y_{bus}.

Example 6.1. The emfs shown in figure 6.1 page 190 are $E_1 = 1.1\angle 0^\circ$ and $E_2 = 1.0\angle 0^\circ$. Use the function `Y = ybus(zdata)` to obtain the bus admittance matrix. Find the bus impedance matrix by inversion and solve for the bus voltages.

Tha Matlab program is shown below:

```
%     From  To    R    X
z = [ 0     1     0   1.0
      0     2     0   0.8
      1     2     0   0.4
      1     3     0   0.2
      2     3     0   0.2
      3    4    0   0.08];
 [Ybus] = ybus(z)                    % bus admittance matrix
Ibus = [-j*1.1; -j*1.25; 0; 0]; % vector of injected bus
currents
Zbus = inv(Ybus)                     % bus impedance matrix
Vbus = Zbus*Ibus
%Vbus = Ybus\Ibus
```

Note: the last statement which is commented out is the better way of doing it! Why?

Do not execute chp6ex1 from this workbook. Instead, go to the Matlab window and give the command to get the output.

6.3 Solution of Nonlinear Algebraic Equations

The most common methods for solving nonlinear algebraic equations are Gauss-Seidel, Newtow-Rahpson, and quasi-Newton-Raphson methods. We start with one dimensional equations and then generalize to n-dimensional equations.

6.3.1 Gauss-Seidel Method

This method is also known as the method of successive displacements. Consider the nonlinear equation $f(x)=0$. The equation is broken into two parts thus: $x=g(x)$. We assume $x^{(0)}$ is an initial "guess" of the solution, then "refine" the solution using:

$$x^{(1)}=g\left(x^{(0)}\right)$$

This process is repeated thus

$$x^{(2)}=g\left(x^{(1)}\right)$$

and on the n^{th} iteration we have: $x^{(n)}=g\left(x^{(n-1)}\right)$. If this process is convergent, then the successive solutions approach a value which is declared as the solution. Thus if at some step $k+1$ we have $\left|x^{(k+1)}-x^{(k)}\right|\le e$ where e is the desired "accuracy", then we claim the solution has been found to the accuracy specified.

Example 6.2 Use the Gauss-Seidel method to obtain the roots of the equation

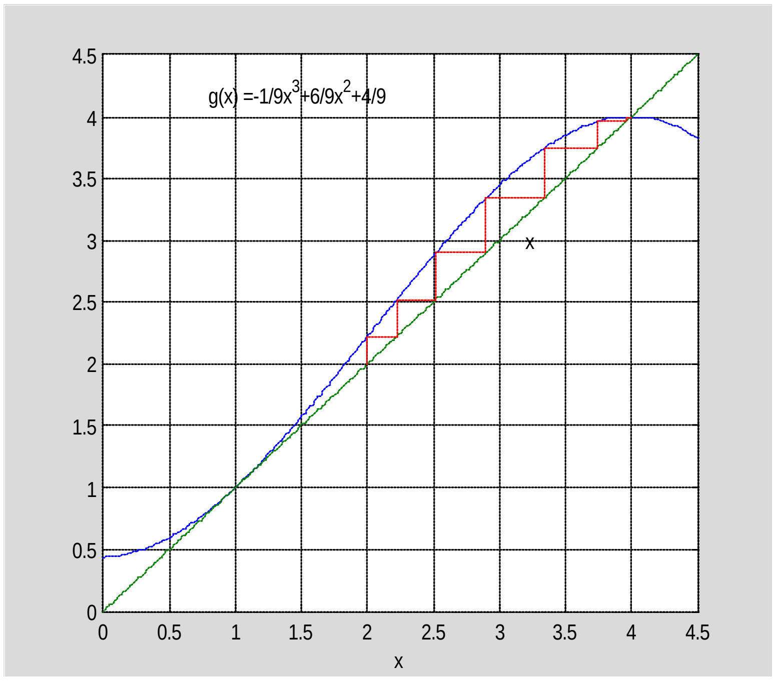
$$f\left(x\right)=x^3-6x^2+9x-4=0$$

First the equation is expressed in a different form thus

$$x=-\frac{1}{9}\left(x^3-6x^2-4\right)=g\left(x\right)$$

And the iteration can proceed. Take a good look at the shape of the iterations! Below is the program showing the process graphically (later showing how to do it iteratively).

```
clear; x=0:.01:4.5;
g=-1/9*x.^3+6/9*x.^2+4/9;
k=1;
dz=10;
z=2;
r(k)=z;
s(k)=z;
while dz >.001
  k=k+2;
  r(k-1)=z;
  p=-1/9*z^3+6/9*z^2+4/9;
  s(k-1)=p;
  dz=abs(z-p);
  z=p;
  r(k)=z; s(k)=z;
end
plot(x,g,'-',x,x,'-', r,s,'-'),grid
xlabel('x')
text(0.8, 4.2,'g(x) =-1/9x^3+6/9x^2+4/9')
text(3.2, 3.0,'x')
```

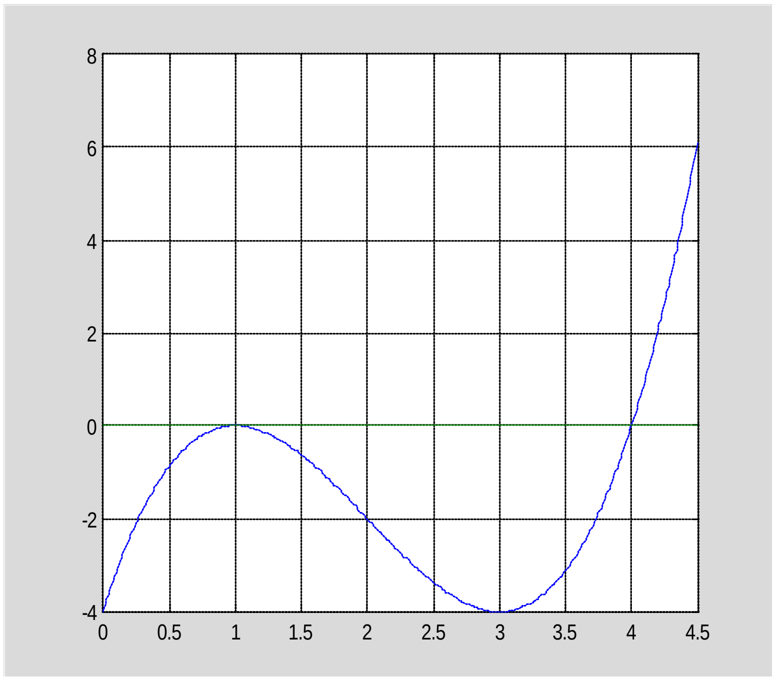


Note: try running the program above with an initial "guess" of $z = 0$ and $z = 0.8$
Does this process converge? always? to a unique solution?

Next, plot the function and find the roots using Matlab:

```
roots([1; -6; 9; -4])
x=0:0.01:4.5;
zz=zeros(size(x));    % plot the x-axis
f=x.^3-6*x.^2+9.*x-4;
plot(x,f,x,zz),grid
```

```
ans =
   4.0000
   1.0000 + 0.0000i
   1.0000 - 0.0000i
```



Note the double root at x = 1.

Now the program to do this iteratively:

```
dx=1;                   % Change in variable is set to a high value
x=2;                                     % Initial estimate
iter = 0;                                 % Iteration counter
disp('Iter    g         dx        x')      % Heading for results
while abs(dx) >= 0.001 & iter < 100       % Test for convergence
  iter = iter + 1;                        % No. of iterations
  g = -1/9*x^3+6/9*x^2+4/9 ;
  dx = g-x;                               % Change in variable
  x = x + dx;                             % Successive approximation
  fprintf('%g', iter), disp([g, dx, x])
end
```

```
Iter    g         dx        x
1    2.2222    0.2222    2.2222
2    2.5173    0.2951    2.5173
3    2.8966    0.3793    2.8966
4    3.3376    0.4410    3.3376
5    3.7398    0.4022    3.7398
6    3.9568    0.2170    3.9568
7    3.9988    0.0420    3.9988
8    4.0000    0.0012    4.0000
9    4.0000    0.0000    4.0000
```

Remarks: The roots found depend on the starting point! There may be other roots… Moreover, there is no guarantee the method is converging unless one checks on the convergence by means of some algorithm.

The Gauss-Seidel method can be expressed with a parameter a known as an acceleration factor, thus:

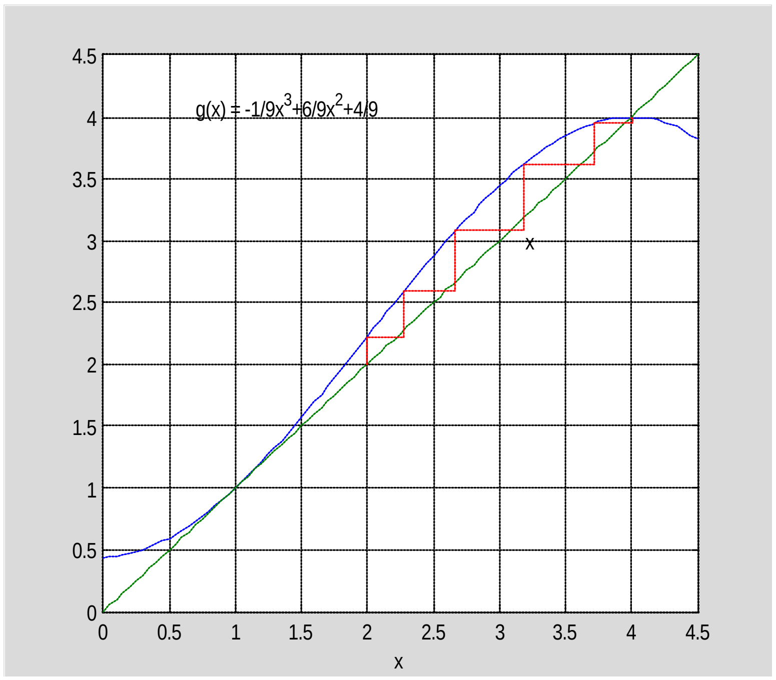
$$x^{(k+1)} = x^{(k)} + a\left[g\left(x^{(k)}\right) - x^{(k)}\right]$$

Note that if $a = 1$ the method becomes the non-accelerated method mentioned earlier.

Example 6.3. Find the roots of the equation used earlier but with an acceleration factor of 1.25.

First a graphical solution using Matlab:

```
clear; x=0:.05:4.5;
alpha=1.25;
g=-1/9*x.^3+6/9*x.^2+4/9;
k=1;
dz=10;
z=2;
r(k)=z;
s(k)=z;
while dz >.001
  k=k+2;
  r(k-1)=z;
  p=-1/9*z^3+6/9*z^2+4/9;
  s(k-1)=p;
  dz=abs(z-p);
  z=z+alpha*(p-z);
  r(k)=z; s(k)=p;
end
plot(x,g,'-',x,x,'-', r,s,'-'),grid
xlabel('x')
text(0.7, 4.1,'g(x) = -1/9x^3+6/9x^2+4/9')
text(3.2, 3.0,'x')
```



Note how at each step we choose more than "x" trying to go "further" in the right direction. This can also produce an unstable result, especially if $a \gg 1$.

Next the numerical solution using Matlab:

```
%chp6fig4                        % Graphical display for Example 6.3
clear; dx=1;                     % Change in variable is set to a high value
x=2;                                          % Initial estimate
iter = 0; alpha=1.25;                         % Iteration counter, alpha
disp('Iter    g         dx        x')         % Heading for results
while abs(dx) >= 0.001 & iter < 100           % Test for convergence
iter = iter + 1;                              % No. of iterations
g = -1/9*x^3+6/9*x^2+4/9;
dx = g-x;                                     % Change in variable
x = x +alpha*dx; % Successive approximation with alpha accel. factor
fprintf('%g', iter), disp([g, dx, x])
end
```

```
Iter    g         dx        x
1    2.2222    0.2222    2.2778
2    2.5902    0.3124    2.6683
3    3.0801    0.4118    3.1831
4    3.6157    0.4326    3.7238
5    3.9515    0.2277    4.0084
6    4.0000   -0.0085    3.9978
7    4.0000    0.0022    4.0005
8    4.0000   -0.0005    3.9999
```

Next the process is generalized for n equations in n variables:

$$\begin{aligned}
&f_1(x_1, x_2, \cdots, x_n) = c_1 \\
&f_2(x_1, x_2, \cdots, x_n) = c_2 \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&f_n(x_1, x_2, \cdots, x_n) = c_n
\end{aligned}$$

Solving for one variable from each equation, the system of equations car be represented as follows:

$$\begin{aligned}
&x_1 = c_1 + g_1(x_1, x_2, \cdots, x_n) \\
&x_2 = c_2 + g_2(x_1, x_2, \cdots, x_n) \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&x_n = c_n + g_n(x_1, x_2, \cdots, x_n)
\end{aligned}$$

First a starting solution is assumed to be $\left(x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)}\right)$. Now there are two ways to proceed:

1. Gauss iteration: use the starting estimate $\left(x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)}\right)$ in the equations to compute a first iterate $\left(x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)}\right)$ then repeat the process using the first iterate to get a second iterate $\left(x_1^{(2)}, x_2^{(2)}, \cdots, x_n^{(2)}\right)$. This is repeated till the solutions converge.

2. Gauss-Seidel iteration: use the starting estimate $\left(x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)}\right)$ in the first equation ONLY to get a new estimate for $x_1^{(1)}$. Now use $\left(x_1^{(1)}, x_2^{(0)}, \cdots, x_n^{(0)}\right)$ in the second equation to get an estimate for $x_2^{(1)}$. Now use $\left(x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(0)}\right)$ in the third equation to get an estimate for $x_3^{(1)}$ and so on till the whole set $\left(x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)}\right)$ is found. The process is repeated till the solutions converge.

An acceleration factor a can be used as before, thus at each step upgrade the estimate as follows

$$x_i^{(k+1)} = x_i^{(k)} + a\left(x_{ical}^{(k+1)} - x_i^{(k)}\right)$$

Clearly if $a = 1$ then the iteration is the same as before. Usually $a \geq 1$ but not much larger than unity.