Let knowledge grow from more to more,
But more of reverence in us dwell;
That mind and soul, according well,
May make one music as before.

— ALFRED TENNYSON

Chapter 4

Systems of nonlinear algebraic equations

In this chapter we extend the concepts developed in Chapter 2 - viz. finding the roots of a system of nonlinear algebraic *equations* of the form,

$$\boldsymbol{f}(\boldsymbol{x}) = \mathbf{o} \tag{4.1}$$

where $\boldsymbol{f}(\boldsymbol{x})$ is a vector function of $\boldsymbol{x}$ - *i.e.*, there are n equations which can be written in expanded or component form as,

$$f_1(x_1, x_2, \cdots, x_n) = 0$$

$$f_2(x_1, x_2, \cdots, x_n) = 0$$

$$\cdots$$

$$f_n(x_1, x_2, \cdots, x_n) = 0$$

As with the scalar case, the equation is satisfied only at selected values of $\boldsymbol{x} = \boldsymbol{r} = [r_1, r_2, \cdots, r_n]$, called the *roots*. The separation process model discussed in section §1.3.1 (variations 2 and 3, in particular) and the reaction sequence model of section §1.3.5 are two of the many examples in chemical engineering that give rise to such non-linear system of equations. As with the scalar case, the equations often depend on other parameters, and we will represent them as

$$\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}) = \mathbf{o} \tag{4.2}$$

where $\boldsymbol{p}$ represents a set of known parameter values. In such cases it may be required to construct solution families for ranges of values of $\boldsymbol{p}$ - *i.e.*, $\boldsymbol{x}(\boldsymbol{p})$. This task is most efficiently achieved using continuation methods. For some of the recent developments on algorithms for non-linear equations see Ortega and Rheinboldt (1970), Rabinowitz (1970), Scales (1985) and Drazin (1992).

4.1 Newton's method

For a scalar equation a geometrical interpretation of the Newton scheme is easy to develop as shown in figure 2.2d. This is difficult to visualize for higher dimensional systems. The algorithm developed in section §2.5 can, however, be generalized easily to higher dimensional systems. The basic concept of linearizing a nonlinear function remains the same as with a scalar case. We need to make use of the multivariate form of the Taylor series expansion. We will illustrate the concepts with a two-dimensional system of equations written in component form as,

$$f_1(x_1, x_2) = 0$$

$$f_2(x_1, x_2) = 0$$

Thus the vectors $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2), f_2(x_1, x_2)]$ and $\boldsymbol{x} = [x_1, x_2]$ contain two elements. Let the *roots* be represented by $\boldsymbol{r} = [r_1, r_2]$ - *i.e.*, $\boldsymbol{f}(\boldsymbol{r}) = \mathbf{o}$.

Suppose $\boldsymbol{x}^{(o)}$ be some known *initial guess* for the solution vector $\boldsymbol{x}$ and let the root be at a small displacement $\boldsymbol{\delta}$ from $\boldsymbol{x}^{(o)}$ - *i.e.*,

$$\boldsymbol{r} = \boldsymbol{x}^{(o)} + \boldsymbol{\delta}$$

If we can device a scheme to estimate $\boldsymbol{\delta}$ then we can apply such a scheme repeatedly to get closer to the root $\boldsymbol{r}$. Variations in the function value $f_1(x_1, x_2)$ can be caused by variations in either components x_1 or x_2. Recognizing this, a bi-variate Taylor series expansion around $\boldsymbol{x}^{(o)}$ can be written as,

$$f_1(x_1^{(0)} + \delta_1, x_2^{(0)} + \delta_2) = f_1(x_1^{(0)}, x_2^{(0)}) + \underbrace{\left.\frac{\partial f_1}{\partial x_1}\right|_{[x_1^{(0)}, x_2^{(0)}]} \delta_1}_{\text{variation due to } x_1} + \underbrace{\left.\frac{\partial f_1}{\partial x_2}\right|_{[x_1^{(0)}, x_2^{(0)}]} \delta_2}_{\text{variation due to } x_2} + \mathcal{O}(\boldsymbol{\delta}^2)$$

$$f_2(x_1^{(0)}+\delta_1, x_2^{(0)}+\delta_2) = f_2(x_1^{(0)}, x_2^{(0)}) + \underbrace{\left.\frac{\partial f_2}{\partial x_1}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_1}_{\text{variation due to } x_1} + \underbrace{\left.\frac{\partial f_2}{\partial x_2}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_2}_{\text{variation due to } x_2} + \mathcal{O}(\boldsymbol{\delta}^2)$$

Since $\boldsymbol{\delta}$ is supposed to be small, we can neglect higher order terms $\mathcal{O}(\boldsymbol{\delta}^2)$ in the above equations and this step is the essence of the linearization process. Since $\boldsymbol{x}^{(0)} + \boldsymbol{\delta} = \boldsymbol{r}$ and $\boldsymbol{f}(\boldsymbol{r}) = \mathbf{0}$, the left hand sides of the above equations are zero. Thus we get,

$$0 = f_1(x_1^{(0)}, x_2^{(0)}) + \left.\frac{\partial f_1}{\partial x_1}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_1 + \left.\frac{\partial f_1}{\partial x_2}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_2$$

$$0 = f_2(x_1^{(0)}, x_2^{(0)}) + \left.\frac{\partial f_2}{\partial x_1}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_1 + \left.\frac{\partial f_2}{\partial x_2}\right|_{[x_1^{(0)}, x_2^{(0)}]}\delta_2$$

These are two linear equations in two unknowns $[\delta_1, \delta_2]$. Note that the two functions $[f_1, f_2]$ and their four partial derivatives are required to be evaluated at the guess value of $[x_1^{(0)}, x_2^{(0)}]$. The above equations can be arranged into matrix form as,

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

or in symbolic form

$$\mathbf{o} = \boldsymbol{f}^{(o)} + \boldsymbol{J}^{(o)}\,\boldsymbol{\delta}$$

where $\boldsymbol{J}^{(o)}$ is called the Jacobian matrix and the superscript is a reminder that quantities are evaluated using the current guess value of $\boldsymbol{x}^{(o)}$. Thus, the displacement vector $\boldsymbol{\delta}$ is obtained by solving the linear system,

$$\boldsymbol{\delta} = -\boldsymbol{J}^{-1}\boldsymbol{f}$$

In general then, given $\boldsymbol{x}^{(o)}$, the algorithm consists of (i) evaluating the function and the Jacobian at the current iterate $\boldsymbol{x}^{(k)}$, (ii) solving the linear system for the displacement vector $\boldsymbol{\delta}^{(k)}$ and (iii) finding the new estimate for the iterate $\boldsymbol{x}^{(k+1)}$ from the equations

$$\boxed{\boldsymbol{\delta}^{(k)} = -[\boldsymbol{J}^{(k)}]^{-1}\boldsymbol{f}^{(k)} \qquad \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{\delta}^{(k)} \qquad k = 0, 1, \cdots} \quad (4.3)$$

Convergence check

The final step is to check if we are close enough to the desired root $\boldsymbol{r}$ so that we can terminate the iteration. One test might be to check if the absolute difference between two successive values of $\boldsymbol{x}$ is smaller than a specified tolerance. This can be done by computing the norm of $\boldsymbol{\delta}$

$$|\boldsymbol{\delta}| \leq \epsilon$$

Another test might be to check if the absolute value of the function $\boldsymbol{f}$ at the end of every iteration is below a certain tolerance. Since we are dealing with vectors, once again a norm of $\boldsymbol{f}$ must be calculated.

$$|\boldsymbol{f}| \leq \epsilon$$

The norm can be computed as

$$|\boldsymbol{f}| = \frac{\sqrt{\sum_{i=1}^{n} f_i^2}}{n}$$

where n is the dimension of the system.

In addition to the above convergence tests, we might wish to place a limit on the number of times the iteration is repeated. A MATLAB function, constructed in the form of a *m-file,* is shown in figure 4.1. Note that this MATLAB function requires an initial guess as well as two external fucntions for computing the function values and the Jacobian.

Example of reactors in series

This example is from section §1.3.5 and consists of a system of nonlinear equations that model a series of continuously stirred tank reactors. A sketch is shown in figure 1.6 Recall that the model equations are given by,

$$f_i := \beta a_i^2 + a_i - a_{i-1} = 0 \qquad i = 1 \cdots n \tag{4.4}$$

As discussed in section §1.3.5, there are n equations and $(n+2)$ variables in total. Hence we have two degrees of freedom. We consider a design situation where inlet and outlet concentrations are specified as, say, $a_0 = 5.0, a_n = 0.5$ mol/lit. The unknown vector consists of n variable elements,

$$\boldsymbol{x} = \{a_1, a_2 \cdots a_{n-1}, \beta\}$$

We are required to determine the volume of each reactor for a given number n. The volume is given by the expression $\beta = kV/F$. The rate constant is $k = 0.125$ lit/(mol min) and the feed rate is $F = 25$ lit/min.

```
function x=newton(Fun,Jac,x,tol,trace)
% Newton method for a system of nonlinear  equations
%       Fun   - name of the external function to compute f
%       Jac   - name of the externan function to compute J
%       x     - vector of initial guesses
%       tol   - error criterion
%       trace - print intermediate results
%
%   Usage  newton('Fun','Jac',x)

%Check inputs
if nargin < 5, trace=0; end
if nargin < 4, tol=eps; trace=0; end

max=25;
n=length(x);
count=0;
f=1;

while (norm(f)  > tol & count < max), %check convergence
  f = feval(Fun,x);      %evaluate the function
  J = feval(Jac,x);      %evaluate Jacobian
  x = x -J\f;            %update the guess
  count=count+1;
  if trace,
   fprintf(1,'Iter.# = %3i  Resid = %12.5e\n', count,norm(f));
  end
end

if (count >= max)
 fprintf(1,'Maximum iteration %3i exceeded\n',count)
 fprintf(1,'Residual is %12.5e\n ',norm(f) )
end
```

Figure 4.1: MATLAB implementation of Newton scheme

In solving the above equations we are primarily interested in β, but we also get all of the intermediate concentrations. Before we can invoke the algorithm of figure 4.1 we need to write two functions for evaluation the function values and the Jacobian. The Jacobian is given by,

$$
\begin{aligned}
J &= \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & 0 & 0 & \cdots & \frac{\partial f_1}{\partial x_n} \\
\frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & 0 & \cdots & \frac{\partial f_2}{\partial x_n} \\
0 & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\
\vdots & 0 & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \frac{\partial f_n}{\partial x_{n-1}} & \frac{\partial f_n}{\partial x_n}
\end{bmatrix} \\
&= \begin{bmatrix}
2x_n x_1 + 1 & 0 & 0 & \cdots & x_1^2 \\
-1 & 2x_n x_2 + 1 & 0 & \cdots & x_2^2 \\
0 & -1 & 2x_n x_3 + 1 & \cdots & x_3^2 \\
\vdots & 0 & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & -1 & a_n^2
\end{bmatrix}
\end{aligned}
\tag{4.5}
$$

The MATLAB functions to compute the function in equation (4.4) and the Jacobian in equation (4.5) are shown in figure 4.2. Work through the following example after ensuring that the three m-files `newton.m`, `cstrF.m` and `cstrJ.m` are the MATLAB search path.

```
»x=[1 .5 .2 .1 0]'                    % Initial guess for n=5
»r=newton('cstrF','cstrJ',x,1.e-10,1) % call Newton scheme
Iter.# = 1 Resid = 4.06325e+00
Iter.# = 2 Resid = 1.25795e+01
Iter.# = 3 Resid = 2.79982e+00
Iter.# = 4 Resid = 4.69658e-01
Iter.# = 5 Resid = 2.41737e-01
Iter.# = 6 Resid = 4.74318e-03
Iter.# = 7 Resid = 1.61759e-06
Iter.# = 8 Resid = 1.25103e-12

r =

  2.2262
  1.2919
  0.8691
  0.6399
  0.5597
```

```
function f=cstrF(x)
% Reactor in series model, the function
% x=[a(1),a(2), .., a(n-1),beta]
% f(i) = beta a(i)^2 + a(i) - a(i-1)

n=length(x);
a0=5.0; an=0.5;  %define parameters in equation

f(1) = x(n)*x(1)^2 + x(1) - a0;
for i = 2:n-1
 f(i)= x(n)*x(i)^2 + x(i) - x(i-1);
end
f(n) = x(n)*an^2 + an - x(n-1);
f=f';
```

```
function J=cstrJ(x)
% Reactor in series model, the Jacobian
% x=[a(1),a(2), .., a(n-1),beta]

n=length(x);
a0=5.0; an=0.5;  %define parameters in equation

J(1,1) = x(n)*2*x(1) + 1;
J(1,n) = x(1)^2;
for i = 2:n-1
 J(i,i) = x(n)*2*x(i) + 1;
 J(i,i-1) = -1;
 J(i,n) = x(i)^2;
end
J(n,n) = an^2;
J(n,n-1) = -1;
```

Figure 4.2: CSTR in series example - function & Jacobian evaluation

```
»V=r(5)*25/.125                    % compute V (ans:111.9427)
»x=[1:-.1:.1]'                     % repeat solution for n=10
»r=newton('cstrF','cstrJ',x,1.e-10,1) % call Newton scheme
»V=r(10)*25/.125                   % compute V (ans:44.9859)
```

In the above example, observe first that the number of reactors is defined implicitly by the length of the vector $\boldsymbol{x}$. Secondly observe the quadratic convergence of the iteration sequence - viz. the residual goes down from 10^{-3} in iteration number 6 to 10^{-6} in iteration number 7 and 10^{-12} in iteration number 8. In other words the number of significant digits of accuracy doubles with every iteration once the iteration reaches close to the root.

4.2 Euler-Newton continuation

4.3 Arc-length continuation

4.4 Quasi-Newton methods

4.4.1 Levenberg-Marquadt method

4.4.2 Steepest descent method

4.4.3 Broyden's method

4.5 Exercise problems

4.5.1 Turbulent flow through a pipeline network

Consider turbulent flow through the network shown in figure 3.8. The governing equations are the pressure drop equations for each pipe element $i-j$ and the mass balance equation at each node. The pressure drop between nodes i and j is given by,

$$p_i - p_j = \alpha_{ij} v_{ij}^2 \qquad \text{where} \qquad \alpha_{ij} = \frac{2 f \rho l_{ij}}{d_{ij}^2} \tag{4.6}$$

In general the friction factor f is given by the Moody chart or its equivalent Churchill correlation. In fully developed turbulent flow it is relatively insensitive to changes in Re. Hence take it to be a constant $f = 0.005$.

The unknown vector is,

$$\underline{x} = [p_2\ p_3\ p_4\ v_{12}\ v_{23}\ v_{24}\ v_{34}\ v_{35}\ v_{46}]$$

There will be six momentum balance equations, one for each pipe element, and three mass balance (for incompressible fluids volume balance) equations, one at each node. Arrange them as a system of nine equations in nine unknowns and solve the resulting set of equations. Take the viscosity of the fluid, $\mu = 0.1Pa \cdot s$ and the density as $\rho = 1000kg/m^3$. The dimensions of the pipes are given below.

Table 1

Element no	12	23	24	34	35	46
d_{ij} (m)	0.1	0.08	0.08	0.10	0.09	0.09
l_{ij} (m)	1000	800	800	900	1000	1000

a) Use MATLAB to solve this problem using Newton method for the specified pressures of $p_1 = 300kPa$ and $p_5 = p_6 = 100kPa$. Report the number of iterations and the flops.

b) Implement the Jacobi iteration to solve this problem. Report the number of iterations and the flops.

c) Implement the Gauss-Seidel iteration to solve this problem. Report the number of iterations and the flops.

d) Implement the Successive-relaxation iteration to solve this problem. Report the number of iterations and the flops.

e) Find out the new velocity and pressure distributions when p_6 is changed to $150kPa$.

f) Comment on how you would adopt the above problem formulation if a valve on line 34 is shut so that there is no flow in that line 34.