Chapter 5. Solution of a System of Nonlinear Algebraic Equations

5.1. Introduction

Not only is life nonlinear, but its variegated phenomena are typically coupled to each other. The dynamic evolution of a system or a material cannot be described by independent solutions of a material balance, a momentum balance and an energy balance because each equation depends upon the variables solved for in the other equations. Thus we end up with systems of nonlinear equations to describe interesting phenomena. Fear not! Just as was the case in the solution of single nonlinear algebraic equations, today there exist reliable tools to methodically solve systems of nonlinear algebraic equations.

As illustrated in the previous chapter, the challenge in solving a single nonlinear algebraic equation is finding a reasonable initial guess. If you think mucking around in one-dimensional space, looking for an initial guess that will allow the method to converge, is painful, then you can imagine that wandering in *n*-dimensional space looking for an *n*-dimensional starting point is significantly more difficult. Nevertheless, it can be done. What is required is access to the appropriate numerical tool coupled with the understanding of the physical system that is provided by the other courses in the undergraduate curriculum.

5.2. Multivariate Newton-Raphson Method

Not surprisingly, the Multivariate Newton-Raphson method is a direct extension of the single variable Newton-Raphson method. Where the single variable Newton-Raphson method solved $f(x)=0$, the multivariate version will solve a system of n equations of the form

$$\begin{aligned} &f_1(x_1, x_2, x_3, \ldots x_{n-1}, x_n) = 0 \\ &f_2(x_1, x_2, x_3, \ldots x_{n-1}, x_n) = 0 \\ &f_3(x_1, x_2, x_3, \ldots x_{n-1}, x_n) = 0 \\ &\ldots \\ &f_{n-1}(x_1, x_2, x_3, \ldots x_{n-1}, x_n) = 0 \\ &f_n(x_1, x_2, x_3, \ldots x_{n-1}, x_n) = 0 \end{aligned} \tag{5.1}$$

We will adopt the short-hand notation for equation (5.1)

$$\underline{f}(\underline{x}) = 0 \tag{5.2}$$

Note that this short hand notation, which was used for linear algebra, does not here imply anything about the linearity of any of the equations in $\underline{f}(\underline{x})$.

The basis of the single variable Newton-Raphson method lay in the fact that we approximate the derivative of $f(x)$ numerically using a forward finite difference formula based on a truncated Taylor series,

$$f'(x_1) = \left.\frac{df}{dx}\right|_{x_1} \approx \frac{f(x_1) - f(x_2)}{x_1 - x_2} \tag{4.8}$$

Although they were not presented in Chapter 3, multivariate Taylor series also exist. The idea behind the a multivariate Taylor series lies in the definition of the total derivate of a multivariate function. For a function of two variable we can write,

$$df(x_1, x_2) = \left(\frac{\partial f}{\partial x_1}\right)_{x_2} dx_1 + \left(\frac{\partial f}{\partial x_2}\right)_{x_1} dx_2 \tag{5.3}$$

where $\left(\frac{\partial f}{\partial x_1}\right)_{x_2}$ is called the partial derivative of the function, *f*, with respect to variable, x_1. The subscript outside the parentheses in a partial derivative indicates variables that were treated as constants during the differentiation. For a function of n variables, we have

$$df(\underline{x}) = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i}\right)_{x_{m \neq i}} dx_i \tag{5.4}$$

The multivariate Taylor series expansion, truncated after the first derivative is thus

$$f\left(\underline{x}^{(k)}\right) - f\left(\underline{x}^{(k+1)}\right) = \sum_{i=1}^{n} \left. \left(\frac{\partial f}{\partial x_i}\right)_{x_{m \neq i}} \right|_{\underline{x}^{(k)}} \left(x_i^{(k)} - x_i^{(k+1)}\right) \tag{5.5}$$

Let it be clear that the *i* subscript on the variable *x* indicates a different independent variable. The superscript (*k*) on the *x* is not an exponent. The parentheses are included to make clear it is a notation that does not signify a mathematical operation. Instead, the superscript (*k*) indicates a different value of *x*, which we shall soon see can be associated with the *k* and *k*+1 iterations of the Newton-Raphson method. The subscript that now appears outside the parentheses, $x_{m \neq i}$, indicates that all variables except x_m are held constant in the differentiation. The final subscript $\underline{x}^{(k)}$ of the partial derivative indicates the value of $\underline{x}$ where the derivative is evaluated.

If n=1, then equation (5.5) simplifies to

$$f\left(x_1^{(k)}\right) - f\left(x_1^{(k+1)}\right) = \left.\frac{df}{dx_1}\right|_{\underline{x}^{(k)}} \left(x_1^{(k)} - x_1^{(k+1)}\right) \tag{5.6}$$

Equation (5.6) can be rearranged for $x_1^{(k+1)}$

$$x_1^{(k+1)} = x_1^{(k)} - \frac{f\left(x_1^{(k)}\right) - f\left(x_1^{(k+1)}\right)}{\left.\frac{df}{dx_1}\right|_{x_1^{(k)}}} \tag{5.7}$$

which is precisely the Newton-Raphson method provided in equation (4.11) once we set $f\left(x_1^{(k+1)}\right) = 0$. Similarly for the multivariate case, in which we have *n* equation and *n* unknowns, we write equation (5.5) for every function

$$f_j\left(\underline{x}^{(k)}\right) - f_j\left(\underline{x}^{(k+1)}\right) = \sum_{i=1}^{n} \left. \left(\frac{\partial f}{\partial x_i}\right)_{x_{m \neq i}} \right|_{\underline{x}^{(k)}} \left(x_i^{(k)} - x_i^{(k+1)}\right) \tag{5.8}$$

where all that has been done is to add a subscript *j* to *f*., identifying each equation from *j* = 1 to n. Equation (5.8) is a system of nonlinear algebraic equations. The n unknowns are the next iteration of *x*, $x_1^{(k+1)}$. Following the Newton-Raphson procedure, we set $f_j\left(\underline{x}^{(k+1)}\right) = 0$ for all *j*. Again, this choice is based on the fact that we intend for our next estimate to be a better approximation of the root, at which the functions are zero. By convention, we express equation (5.8) in matrix notation as

$$\underline{\underline{J}}^{(k)}\underline{\delta x}^{(k)} = -\underline{R}^{(k)} \tag{5.9}$$

where $\underline{R}^{(k)}$ is called the residual vector at the kth iteration and is defined as

$$\underline{R}^{(k)} \equiv \underline{f}\left(\underline{x}^{(k)}\right) \tag{5.10}$$

In other words, the residual is simply a vector of the values of the functions evaluated at the current guess. The Jacobian matrix at the kth iteration, $\underline{\underline{J}}^{(k)}$, defined as

$$J_{j,i}^{(k)} \equiv \left(\frac{\partial f_j}{\partial x_i}\right)_{x_{m \neq i}}\Bigg|_{\underline{x}^{(k)}} \tag{5.11}$$

Thus the Jacobian matrix is an *n*x*n* matrix of all the possible pairwise combinations of partial first derivatives between *n* unknown variables, x_i, and *n* functions, f_j. The difference vector, $\underline{\delta x}^{(k)}$, is defined as

$$\underline{\delta x}^{(k)} \equiv \underline{x}^{(k+1)} - \underline{x}^{(k)} \tag{5.12}$$

The difference vector can be rearranged for the value of $\underline{x}$ at the new iteration,

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} + \underline{\delta x}^{(k)} \tag{5.13}$$

The algorithm for solving a system of nonlinear algebraic equations via the multivariate Newton-Raphson method follows analogously from the single variable version. The steps are as follows:

1. Make an initial guess for $\underline{x}$.
2. Calculate the Jacobian and the Residual at the current value of $\underline{x}$.
3. Solve equation (5.9) for $\underline{\delta x}^{(k)}$.
4. Calculate $\underline{x}^{(k+1)}$ from equation (5.13).
5. If the solution has not converged, loop back to step 2.

The multivariate Newton-Raphson Method suffers from the same short-comings as the single-variable Newton-Raphson Method. Specifically, as with all methods for solving nonlinear algebraic equations, you need a good initial guess. Second, the method does provide fast (quadratic) convergence until you are close to the solution. Third, if the determinant of the Jacobian is zero, the method fails. This last constraint is the multi-dimensional analogue of the fact that the single variable Newton-Raphson method diverged when the derivative was zero.

The determination of convergence of a system that has multiple variables requires a tolerance. One could use a tolerance on each variable. That is the relative error on x_i must be less than tol$_i$. Alternatively, one can use something like the root mean square (RMS) error,

$$err_{RMS}^{(k)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i^{(k)} - x_i^{(k-1)}}{x_i^{(k-1)}}\right)^2} \tag{5.14}$$

to provide a single error for the entire system. In this case, even with an RMS relative error on x of 10^{-m}, you are not guaranteed that every variable has ^{m} good significant digits. You are only guaranteed that the RMS error is less than the acceptable tolerance.

Let's work two examples.

Example 5.1. Multivariate Newton-Raphson Method

Consider the system of two nonlinear algebraic equations.

$$f_1(x_1, x_2) = (x_1)^2 + (x_2)^2 - 4 = 0$$
$$f_2(x_1, x_2) = (x_1)^2 - (x_2) + 1 = 0$$

The first equation is that of a circle with radius 2 centered at the origin. The second equation is that of a parabola. In Figure 5.1. We plot the solution to the two equations independently. Since there are two variables in each equation, there are an infinite number of solutions for the equations treated independently. The solution to the system of nonlinear algebraic equations corresponds to ordered pairs of (x_1, x_2) that satisfy both equations. In Figure 5.1., this corresponds to the intersection between the two curves.

Any solution technique finds only one root at a time. From the plot we can estimate that one of the roots is near $(x_1, x_2) = (1,2)$.

In order to use the multivariate Newton-Raphson method, we must first

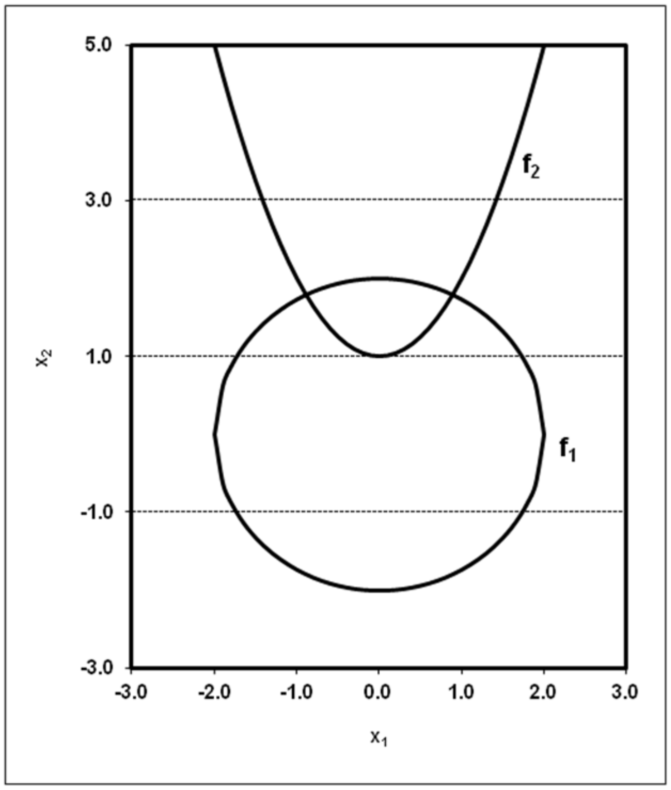

Figure 5.1. Plot of $f_j(x_1,x_2) = 0$ for j = 1 and 2.

determine the functional form of the n^2 partial derivatives analytically. For this small system, we have

$$J_{1,1} = \left(\frac{\partial f_1}{\partial x_1}\right) = 2x_1 \qquad J_{1,2} = \left(\frac{\partial f_1}{\partial x_2}\right) = 2x_2$$
$$J_{2,1} = \left(\frac{\partial f_2}{\partial x_1}\right) = 2x_1 \qquad J_{2,2} = \left(\frac{\partial f_2}{\partial x_2}\right) = -1$$

The residual is composed simply of the functions,

$$\underline{R} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} (x_1)^2 + (x_2)^2 - 4 \\ (x_1)^2 - (x_2) + 1 \end{bmatrix}$$

We now follow the algorithm outlined above.

Step One. Make an initial guess. $(x_1, x_2) = (1,2)$
Step Two. Using that initial guess, calculate the residual and the Jacobian.

$$\underline{\underline{J}}^{(1)} = \begin{bmatrix} 2 & 4 \\ 2 & -1 \end{bmatrix} \text{ and } \underline{R}^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Step Three. Solve equation (5.9) for $\underline{\delta x}^{(k)}$. $\qquad \underline{\delta x}^{(1)} = \begin{bmatrix} -0.1 \\ -0.2 \end{bmatrix}$

Step 4. Calculate $\underline{x}^{(k+1)}$ from equation (5.13). $\qquad \underline{x}^{(2)} = \begin{bmatrix} 0.9 \\ 1.8 \end{bmatrix}$

Step 5. If the solution has not converged, loop back to step 2.

Further iterations yield

	$\underline{x}$	$\underline{\underline{J}}$	$\underline{R}$	$\underline{\delta x}$
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 \\ 2 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -0.1 \\ -0.2 \end{bmatrix}$
2	$\begin{bmatrix} 0.9 \\ 1.8 \end{bmatrix}$	$\begin{bmatrix} 1.8 & 3.6 \\ 1.8 & -1 \end{bmatrix}$	$\begin{bmatrix} 0.05 \\ 0.01 \end{bmatrix}$	$\begin{bmatrix} -0.0104 \\ -0.0087 \end{bmatrix}$

3	$\begin{bmatrix}0.8896\\1.7913\end{bmatrix}$	$\begin{bmatrix}1.7792 & 3.5826\\1.7792 & -1.0000\end{bmatrix}$	$\begin{bmatrix}0.1835e-3\\0.1079e-3\end{bmatrix}$	$\begin{bmatrix}-0.6991\text{e}-4\\-0.1650\text{e}-4\end{bmatrix}$
4	$\begin{bmatrix}0.8895\\1.7913\end{bmatrix}$	$\begin{bmatrix}1.7791 & 3.5826\\1.7791 & -1.0000\end{bmatrix}$	$\begin{bmatrix}0.5159\text{e}-8\\0.4887\text{e}-8\end{bmatrix}$	$\begin{bmatrix}-0.2780\text{e}-8\\-0.0059\text{e}-8\end{bmatrix}$

So one of the two roots is located at $(x_1, x_2) = (0.8895,1.7913)$. (The other is located at $(x_1, x_2) = (-0.8895,1.7913)$ by symmetry. From an examination of the final value of $\underline{\delta x}$ both solutions have converged to an absolute error on x of less than 10^{-8}.

Example 5.2. Multivariate Newton-Raphson Method: Linear Systems

Linear systems are a subset of non-linear systems. The multivariate Newton-Raphson solve linear systems exactly in one iteration, just as was the case in the single-variable problem.

Consider the system of linear equations:

$$f_1(x_1, x_2) = 5(x_1) + (x_2) - 4 = 0$$
$$f_2(x_1, x_2) = (x_1) - 3(x_2) + 1 = 0$$

The solutions of the two independent equations is plotted in Figure 5.2. The solution of the system of equations is the intersection of the two lines. For linear equations, the Jacobian is a constant matrix,

$$\underline{\underline{J}}^{(k)} = \begin{bmatrix}5 & 1\\1 & -3\end{bmatrix}$$

The residual is composed simply of the functions,

$$\underline{R} = \begin{bmatrix}f_1\\f_2\end{bmatrix} = \begin{bmatrix}5x_1 + x_2 - 4\\x_1 - 3x_2 + 1\end{bmatrix}$$

We now follow the algorithm outlined above.

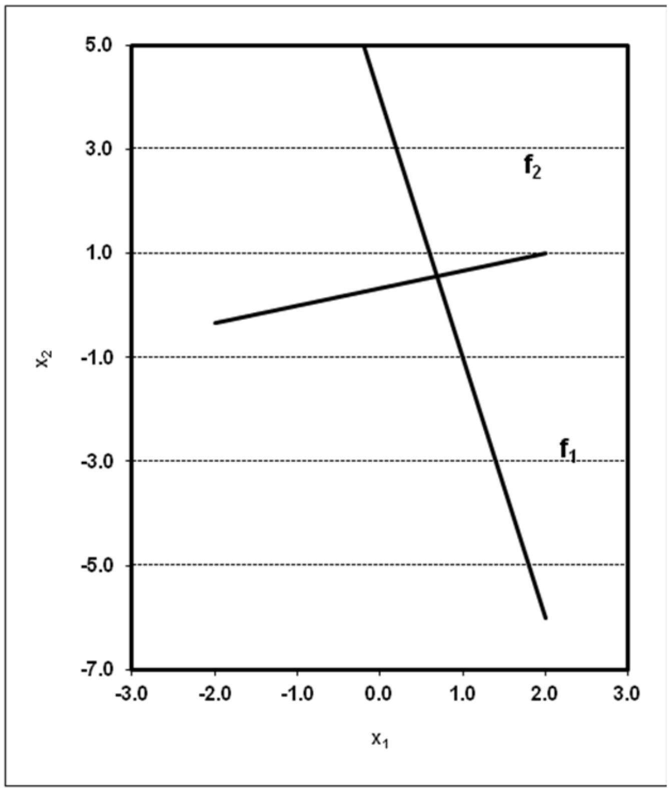

Figure 5.2. Plot of $f_j(x_1,x_2) = 0$ for $j = 1$ and 2.

Step One. Make an initial guess.
$(x_1, x_2) = (2,2)$

Step Two. Using that initial guess, calculate the residual and the Jacobian.

$$\underline{\underline{J}}^{(1)} = \begin{bmatrix} 5 & 1 \\ 1 & -3 \end{bmatrix} \quad \text{and} \quad \underline{R}^{(1)} = \begin{bmatrix} 8 \\ -3 \end{bmatrix}$$

Step Three. Solve equation (5.9) for $\underline{\delta x}^{(k)}$. $\underline{\delta x}^{(1)} = \begin{bmatrix} -1.3125 \\ -1.4375 \end{bmatrix}$

Step 4. Calculate $\underline{x}^{(k+1)}$ from equation (5.13). $\underline{x}^{(2)} = \begin{bmatrix} 0.6875 \\ 0.5625 \end{bmatrix}$

A second iteration will show that this is the exact solution.

5.3. Multivariate Newton-Raphson Method with Numerical Derivatives

There is a pretty obvious drawback to the Multivariate Newton-Raphson method, namely that one must provide the analytical form of *n*x*n* partial derivatives. If n is small, it can be sometimes be done. If *n* is large, this process is simply not practical. Therefore, just as was the case for the single variable Newton-Raphson method, we turn to numerical methods to provide estimates of the partial derivatives. The second-order centered-finite difference formula for the first derivative was derived in Chapter 3,

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \tag{3.7}$$

This can be directly extended to partial derivatives as follows,

$$\left(\left.\frac{df_j}{dx_i}\right|_{x_i}\right)_{x_{m\neq i}} = \frac{f_j(x_1, x_2, \ldots x_i + h_i, \ldots x_n) - f_j(x_1, x_2, \ldots x_i - h_i, \ldots x_n)}{2h_i} + O(h^2) \tag{5.15}$$

Of note, only the value of x_i changes when the differentiation is with respect to x_i. Furthermore, the value of discretization, h_i, may be different for each variable, x_i. Each iteration of a multivariate Newton-Raphson method with analytical derivatives requires *n* function evaluations and n^2 derivative evaluations. Each iteration of a multivariate Newton-Raphson method with numerical derivatives requires $n+2n^2$ function evaluations. So there is more computational effort in the numerical scheme, but potentially a much reduced effort in developing the code since only one function must be entered.

A MATLAB code which implements the multivariate Newton-Raphson method with numerical derivatives is provided later in this chapter.

5.4. Subroutine Codes

In this section, we provide a routine for implementing the multivariate Newton-Raphson method with numerical derivatives. Note that these codes correspond to the theory and notation exactly as laid out in this book. These codes do not contain extensive error checking, which would complicate the coding and defeat their purpose as learning tools. That said, these codes work and can be used to solve problems.

As before, on the course website, two entirely equivalent versions of this code are provided and are titled *code.m* and *code_short.m*. The short version is presented here. The longer version, containing instructions and serving more as a learning tool, is not presented here. The numerical mechanics of the two versions of the code are identical.

Code 5.1. Multivariate Newton-Raphson with Numerical derivatives (nrndn_short)

```
function [x,err,f] = nrndn(x0,tol,iprint)
%
% inputs:
% x0 = initial guess of x, column vector of length n
% tol = RMS tolerance on relative error on x, scalar
% iprint = value of 1 requests iteration information
% f(x) entered in the function "funkeval" at the bottom of this file
%
% outputs:
%
% x = converged solution, column vector of length n
% f = RMS value of f, scalar
% err = RMS value of relative error on x, scalar
%
maxit = 1000;
n = max(size(x0));
Residual = zeros(n,1);
Jacobian = zeros(n,n);
InvJ = zeros(n,n);
dx = zeros(n,1);
x = zeros(n,1);
xold = zeros(n,1);
xeval = zeros(n,1);
xp = zeros(2);
fp = zeros(n,n,2);
dxcon = zeros(n,1);
dxcon(1:n) = 0.01;
x = x0;
err = 100.0;
iter = 0;
while ( err > tol )
   for j = 1:1:n
```

```
      dx(j) = min(dxcon(j)*x(j),dxcon(j));
      i2dx(j) = 1.0/(2.0*dx(j));
   end
   Residual = funkeval(x);
   for j = 1:1:n
      for i = 1:1:n
         xeval(i) = x(i);
      end
      xp(1) = x(j) - dx(j);
      xp(2) = x(j) + dx(j);
      for k = 1:1:2
         xeval(j) = xp(k);
         fp(:,j,k) = funkeval(xeval);
      end
   end
   for i = 1:1:n
      for j = 1:1:n
         Jacobian(i,j) = i2dx(j)*( fp(i,j,2) - fp(i,j,1) );
      end
   end
   xold = x;
   invJ = inv(Jacobian);
   deltax = -invJ*Residual;
   for j = 1:1:n
      x(j) = xold(j) + deltax(j);
   end
   iter = iter +1;
   err = sqrt( sum(deltax.^2) /n  ) ;
   f = sqrt(sum(Residual.*Residual)/n);
   if (iprint == 1)
      fprintf (1,'iter = %4i, err = %9.2e f = %9.2e \n ', iter, err, f);
   end
   if ( iter > maxit)
      Residual
      error ('maximum number of iterations exceeded');
   end
end

function f = funkeval(x)
n = max(size(x));
f = zeros(n,1);
f(1) = x(1)^2 + x(2)^2 - 4;
f(2) = x(1)^2 - x(2) + 1;
```

An example of using nrndn_short is given below.

```
» [x,err,f] = NRNDN_short([2,2],1.0e-6,1)
iter =    1, err = 5.83e-001 f = 3.54e+000
 iter =    2, err = 1.91e-001 f = 6.60e-001
 iter =    3, err = 2.78e-002 f = 7.31e-002
 iter =    4, err = 6.13e-004 f = 1.54e-003
 iter =    5, err = 2.99e-007 f = 7.52e-007
```

```
x =    0.8895    1.7913
err =  2.9893e-007
f =  7.5211e-007
```

The root is at $(x_1, x_2) = (0.8895, 1.7913)$ and the error is less than the tolerance of 10^{-6}.

5.5. Problems

Problems are located on course website.