

Nonlinear Equations

W. E. Schiesser
Iacocca D307
111 Research Drive
Lehigh University
Bethlehem, PA 18015

(610) 758-4264 (office)
(610) 758-5057 (fax)

wes1@lehigh.edu

[http://www.lehigh.edu/~ wes1/wes1.html](http://www.lehigh.edu/~wes1/wes1.html)

Nonlinear Equations

[http://www.lehigh.edu/~ wes1/apci/24feb00-b.ps](http://www.lehigh.edu/~wes1/apci/24feb00-b.ps)
<http://www.lehigh.edu/~ wes1/apci/24feb00-b.pdf>

<http://www.lehigh.edu/~ wes1/apci/24feb00-s.tex>
<http://www.lehigh.edu/~ wes1/apci/24feb00-s.ps>
<http://www.lehigh.edu/~ wes1/apci/24feb00-s.pdf>

Previous lectures are in:

<http://www.lehigh.edu/~ wes1/apci/28jan00.tex>
<http://www.lehigh.edu/~ wes1/apci/28jan00.ppt>
<http://www.lehigh.edu/~ wes1/apci/28jan00.ps>
<http://www.lehigh.edu/~ wes1/apci/28jan00.pdf>

<http://www.lehigh.edu/~ wes1/apci/18feb00.ps>
<http://www.lehigh.edu/~ wes1/apci/18feb00.pdf>

A Table of Contents is in:

<http://www.lehigh.edu/~ wes1/apci/files.txt>

1. Newton's method and quasilinearization (WES)
2. Obtaining derivatives (LTB)
3. Broyden's method (LTB)
4. Continuation methods (WES)
5. Davidenko's equation (WES)

6. Levenberg-Marquardt method (WES)
7. Tearing algorithms (LTB)
8. Direct substitution (LTB)
9. Wegstein method (LTB)
10. References and software (LTB/WES)

The General Nonlinear Problem

The $n \times n$ nonlinear problem is:

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \tag{1}$$

which can be summarized as

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{2}$$

where we use bold face to denote vectors:

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Two points:

- Nonlinear equations are generally solved numerically by the iterative solution of linear equations
- John Rice states (approximately) that the solution of nonlinear equations is the most difficult problem in scientific computation

Newton's Method and Quasilinearization

To obtain a form of Newton's method for systems of equations, we start with the Taylor series in n variables, e.g., $n = 2$, we want to obtain the values of x_1, x_2 that satisfy the 2 x 2 system

$$\begin{aligned} f_1(x_1, x_2) &= 0 \\ f_2(x_1, x_2) &= 0 \end{aligned} \tag{7}$$

If f_1 and f_2 are expanded in two Taylor series in two dimensions (with respect to the iteration counter k)

$$\begin{aligned} f_1(x_1^{k+1}, x_2^{k+1}) &= f_1(x_1^k, x_2^k) + \frac{\partial f_1(x_1^k, x_2^k)}{\partial x_1}(x_1^{k+1} - x_1^k) + \frac{\partial f_1(x_1^k, x_2^k)}{\partial x_2}(x_2^{k+1} - x_2^k) + \dots \\ f_2(x_1^{k+1}, x_2^{k+1}) &= f_2(x_1^k, x_2^k) + \frac{\partial f_2(x_1^k, x_2^k)}{\partial x_1}(x_1^{k+1} - x_1^k) + \frac{\partial f_2(x_1^k, x_2^k)}{\partial x_2}(x_2^{k+1} - x_2^k) + \dots \end{aligned} \tag{8}$$

truncation after the first derivative terms, with the resulting equations expressed in terms of Newton corrections

$$\begin{aligned} \Delta x_1^k &= (x_1^{k+1} - x_1^k) \\ \Delta x_2^k &= (x_2^{k+1} - x_2^k) \end{aligned}$$

and with just the right Newton corrections so that the LHS's of eqs. (8) are zero, gives

$$\begin{aligned} \frac{\partial f_1(x_1^k, x_2^k)}{\partial x_1} \Delta x_1^k + \frac{\partial f_1(x_1^k, x_2^k)}{\partial x_2} \Delta x_2^k &= -f_1(x_1^k, x_2^k) \\ \frac{\partial f_2(x_1^k, x_2^k)}{\partial x_1} \Delta x_1^k + \frac{\partial f_2(x_1^k, x_2^k)}{\partial x_2} \Delta x_2^k &= -f_2(x_1^k, x_2^k) \end{aligned} \tag{9}$$

Eqs. (9) are a 2 x 2 linear system in the two Newton corrections, $\Delta x_1^k, \Delta x_2^k$ then can be solved by any standard method for linear equations, e.g., Gauss row reduction. The unknowns can then be updated iteratively by

$$\begin{aligned} x_1^{k+1} &= \Delta x_1^k + x_1^k \\ x_2^{k+1} &= \Delta x_2^k + x_2^k \end{aligned} \quad k = 0, 1, 2, \dots \quad (10)$$

Eq. (9) can be generalized to the $n \times n$ case by writing it in matrix form

$$J \Delta \mathbf{x} = -\mathbf{f} \quad (11)$$

where

$$J = \begin{bmatrix} \frac{\partial f_1(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_1} & \frac{\partial f_1(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_2} & \dots & \frac{\partial f_1(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_n} \\ \frac{\partial f_2(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_1} & \frac{\partial f_2(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_2} & \dots & \frac{\partial f_2(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_1} & \frac{\partial f_n(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_2} & \dots & \frac{\partial f_n(x_1^k, x_2^k, \dots, x_n^k)}{\partial x_n} \end{bmatrix}, \Delta x = \begin{bmatrix} \Delta x_1^k \\ \Delta x_2^k \\ \vdots \\ \Delta x_n^k \end{bmatrix} \quad (12)(13)$$

Eq. (11) is Newton's method for an $n \times n$ system (a remarkably compact and useful equation).

We now consider three small nonlinear problems:

(1) The third order polynomial

$$f(x) = x^3 - 2x - 5 = 0$$

As noted on 28jan00, this polynomial may be the first application of Newton's method, since Newton used it to illustrate his method

If we apply Newton's method, eq. (9), for the scalar ($n = 1$) case

$$\frac{df(x^k)}{dx} (x^{k+1} - x^k) = -f(x^k)$$

or the familiar form

$$x^{k+1} = x^k - \frac{f(x^k)}{df(x^k)/dx}, k = 0, 1, \dots$$

where k is an iteration index or counter.

We can make some observations about the application of Newton's method to Newton's polynomial and the generalization of Newton's method to the $n \times n$ problem (eq. (11)):

- The polynomial has three roots (three reals or one real and two complex conjugates - conjugates are required for the polynomial to have real coefficients). This conclusion is from a famous proof by C. F. Gauss; in general, *nonlinear problems will have multiple (nonunique) roots*.
- If we make the choice of the initial starting point at a root of

$$\frac{df(x)}{dx} = 3x^2 - 2 = 0$$

or

$$x^0 = \pm\sqrt{2/3}$$

the method will fail because this is a *singular point* (the *Jacobian matrix*, or in this case, the first derivative, is singular).

- Also, if we choose a starting point near the singular point, the method will most likely fail, i.e., the system is *near-singular* (this is clear if we consider Newton's method as a projection along the tangent to $f(x)$).

- The choice of an initial point that is not singular or near-singular becomes increasingly difficult as the size of the problem increases (more equations), and we should have a test in our software for the *condition* of the Jacobian matrix.
- The Jacobian matrix, \mathbf{J} (in eq. (11)), is $n \times n$, and therefore grows quickly with the problem size.
- The Jacobian matrix is not constant (a property of nonlinear systems), and therefore must be updated as the iterations proceed; updating at every iteration may not be required since all that is required is convergence of the iterations.
- The right hand side (RHS) vector, \mathbf{f} in eq. (11), must also be updated as the iterations proceed, but this will be less costly than for the Jacobian matrix because of the smaller size of \mathbf{f} .
- A stopping criterion is required for the iterations, e.g.,

$$|\Delta x_i^k| < \varepsilon_i, i = 1, 2, \dots, n \text{ (relative or absolute)}$$

$$|\Delta x_i^{k+1} - \Delta x_i^k| < \varepsilon_i, i = 1, 2, \dots, n \text{ (relative or absolute)}$$

$$|f_i(\mathbf{x}_i^k)| < \varepsilon_i, i = 1, 2, \dots, n$$

$$|f_i(\mathbf{x}_i^{k+1}) - f_i(\mathbf{x}_i^k)| < \varepsilon_i, i = 1, 2, \dots, n$$

- Quasilinearization is the characterization of eq. (11) denoting the linearization with respect to the iteration index or counter, k .

- Eq. (11) is almost never solved as

$$\Delta \mathbf{x} = -J^{-1} \mathbf{f}$$

Rather, J is factored (LU, QR factorization), and eq. (11) is then solved as a system of linear algebraic equations (for the vector of Newton corrections Δx).

- For large nonlinear systems, e.g., $n > 100$, the structure of the Jacobian matrix in eq. (11) can often be exploited to good advantage, e.g., as a banded or sparse matrix.

(2) A 2 x 2 system provided by L. T. Biegler (1991):

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + x_2^2 - 17 \\ f_2(x_1, x_2) &= 2x_1^{1/3} + x_2^{1/2} - 4 \end{aligned} \quad (14)$$

Eqs. (14) have the solutions $\mathbf{x} = (1, 4)^T$ and $\mathbf{x} = (4.07150, 0.65027)^T$.

Note the nonlinearity of this system. This is manifest in the Jacobian matrix,

$$\mathbf{J} = \begin{bmatrix} \partial f_1 / \partial x_1 = 2x_1 & \partial f_1 / \partial x_2 = 2x_2 \\ \partial f_2 / \partial x_1 = (2/3)x_1^{-2/3} & \partial f_2 / \partial x_2 = (1/2)x_2^{-1/2} \end{bmatrix} \quad (15)$$

since the elements of J are functions of x_1, x_2 (for a linear system, the Jacobian matrix is a constant matrix).

J is singular if

$$|\mathbf{J}| = \begin{vmatrix} 2x_1 & 2x_2 \\ (2/3)x_1^{-2/3} & (1/2)x_2^{-1/2} \end{vmatrix} = 0$$

or

$$(2x_1)(1/2)x_2^{-1/2} - (2x_2)(2/3)x_1^{-2/3} = 0 \quad (16)$$

Eq. (16) is a relationship between x_1 and x_2 that defines a locus of singular points for system (14). Later we will start a numerical solution at a singular point given by eq. (16) and observe what happens with the calculation of the solution.

Applying eq. (11) to eqs. (14), we have

$$\begin{bmatrix} 2x_1 & 2x_2 \\ (2/3)x_1^{-2/3} & (1/2)x_2^{-1/2} \end{bmatrix}^k \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix}^k = - \begin{bmatrix} x_1^2 + x_2^2 - 17 \\ 2x_1^{1/3} + x_2^{1/2} - 4 \end{bmatrix}^k$$

or

$$\begin{aligned} 2x_1^k \Delta x_1^k + 2x_2^k \Delta x_2^k &= -(x_1^k)^2 - (x_2^k)^2 + 17 \\ (2/3)(x_1^k)^{-2/3} \Delta x_1^k + (1/2)(x_2^k)^{-1/2} \Delta x_2^k &= -2(x_1^k)^{1/3} - (x_2^k)^{1/2} + 4 \end{aligned} \quad (17)$$

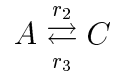
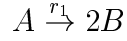
A basic algorithm for solving eqs. (14) is therefore

- 1 Select $x_1^k = x_1^0, x_2^k = x_2^0$
- 2 Substitute x_1^k, x_2^k in eqs. (17)
- 3 Solve eqs. (17) for $\Delta x_1^k, \Delta x_2^k$
- 4 $|\Delta x_1^k| < \varepsilon_1$ and $|\Delta x_2^k| < \varepsilon_2$?; yes - output, quit; no - continue
- 5 $x_1^{k+1} = x_1^k + \Delta x_1^k, x_2^{k+1} = x_2^k + \Delta x_2^k$
- 6 Let $x_1^k = x_1^{k+1}, x_2^k = x_2^{k+1}$; go to 2

Note that all that is required to implement this algorithm is a linear equation solver (to compute the Newton corrections, $\Delta x_1^k, \Delta x_2^k$).

(3) A 4 x 4 system from Riggs, J. B. (1994), "An Introduction to Numerical Methods for Chemical Engineers", 2nd ed., Texas Tech University Press, pp 82-89:

A continuous stirred tank reactor (CSTR) is used for the following reaction scheme



where

$$\begin{aligned} r_1 &= k_1 C_A \\ r_2 &= k_2 C_A^{3/2} \\ r_3 &= k_3 C_C^2 \\ r_4 &= k_4 C_B^2 \\ k_1 &= 1.0 \text{ sec}^{-1} \\ k_2 &= 0.2 \text{ liter}^{1/2}/\text{gm mol}^{1/2}\text{-sec} \\ k_3 &= 0.05 \text{ liter/gm mol-sec} \\ k_4 &= 0.4 \text{ liter/gm mol-sec} \end{aligned}$$

Each reaction rate (r_1 to r_4) is in gm mols/liter-sec. If the volume of the reactor is $V_r = 100$ liters, the volumetric flow rate through the reactor is $Q = 50$ liters/sec, and the feed to the reactor is $C_{A0} = 1.0$ gm mol/liter, the steady state material balances for the reactor are:

$$f_1(C_A, C_B, C_C, C_D) = -QC_A + QC_{A0} + V_R(-k_1 C_A - k_2 C_A^{3/2} + k_3 C_C^2) = 0 \quad (17a)$$

$$f_2(C_A, C_B, C_C, C_D) = -QC_B + V_R(2k_1 C_A - k_4 C_B^2) = 0 \quad (17b)$$

$$f_3(C_A, C_B, C_C, C_D) = -QC_C + V_R(k_2 C_A^{3/2} - k_3 C_C^2 + k_4 C_B^2) = 0 \quad (17c)$$

$$f_4(C_A, C_B, C_C, C_D) = -QC_D + V_R(k_4C_B^2) = 0 \quad (17d)$$

We will subsequently consider a solution to eqs. (17) to three significant figures, i.e., the values of C_A, C_B, C_C, C_D that satisfy eqs. (17).

One possible approach to computing a solution would be to convert eqs. (17) to a system of ordinary differential equations (ODEs):

$$V \frac{dC_A}{dt} = -QC_A + QC_{A0} + V_R(-k_1C_A - k_2C_A^{3/2} + k_3C_C^2) = 0, C_A(0) = C_A^0 \quad (18a)$$

$$V \frac{dC_B}{dt} = -QC_B + V_R(2k_1C_A - k_4C_B^2) = 0, C_B(0) = C_B^0 \quad (18b)$$

$$V \frac{dC_C}{dt} = -QC_C + V_R(k_2C_A^{3/2} - k_3C_C^2 + k_4C_B^2) = 0, C_C(0) = C_C^0 \quad (18c)$$

$$V \frac{dC_D}{dt} = -QC_D + V_R(k_4C_B^2) = 0, C_D(0) = C_D^0 \quad (18d)$$

where: (a) V is the volume of the reactor (if we are interested in the true dynamic response of the CSTR) or (b) V is a “pseudo” time constant if all we are interested in is the final steady state with $\frac{dC_A}{dt} \approx \frac{dC_B}{dt} \approx \frac{dC_C}{dt} \approx \frac{dC_D}{dt} \approx 0$.

Method (b) is a perfectly satisfactory way of solving eqs. (14) called the method of “false transients”. However, care must be given to adding the derivatives to the nonlinear equations with the correct sign to ensure a stable steady state or equilibrium solution (which may not be obvious for large systems of equations). Later we will consider a systematic procedure for adding the derivatives.

Continuation Methods

Consider again the 2 x 2 system of eq. (7). If we assume two related functions,

$$\begin{aligned} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{aligned}$$

with known solutions based on arbitrary values $x_1 = x_1^0, x_2 = x_2^0$, i.e.,

$$\begin{aligned} g_1(x_1^0, x_2^0) &= 0 \\ g_2(x_1^0, x_2^0) &= 0 \end{aligned}$$

then we can form a vector of *homotopy functions*

$$\begin{aligned} h_1(x_1, x_2, t) &= tf_1(x_1, x_2) + (1 - t)g_1(x_1, x_2) \\ h_2(x_1, x_2, t) &= tf_2(x_1, x_2) + (1 - t)g_2(x_1, x_2) \end{aligned} \tag{19}$$

where t is an *embedded homotopy continuation parameter*.

Now we vary t over the interval $0 \leq t \leq 1$ while always setting the homotopy functions to zero. This can be easily visualized using a table

t	$h_1(x_1, x_2, t) = 0$ $h_2(x_1, x_2, t) = 0$	x_1, x_2	Comments
0	$g_1(x_1, x_2) = 0$ $g_2(x_1, x_2) = 0$	x_1^0 x_2^0	Assumed initial solution
0.1	$0.1f_1(x_1, x_2) + 0.9g_1(x_1, x_2) = 0$ $0.1f_2(x_1, x_2) + 0.9g_2(x_1, x_2) = 0$	$x_1^{0.1}$ $x_2^{0.1}$	$x_1^{0.1}$ close to x_1^0 $x_2^{0.1}$ close to x_2^0
0.2	$0.2f_1(x_1, x_2) + 0.8g_1(x_1, x_2) = 0$ $0.2f_2(x_1, x_2) + 0.8g_2(x_1, x_2) = 0$	$x_1^{0.2}$ $x_2^{0.2}$	Root finder required at each step
0.3	$0.3f_1(x_1, x_2) + 0.7g_1(x_1, x_2) = 0$ $0.3f_2(x_1, x_2) + 0.7g_2(x_1, x_2) = 0$	$x_1^{0.3}$ $x_2^{0.3}$	
\vdots	\vdots	\vdots	
0.9	$0.9f_1(x_1, x_2) + 0.9g_1(x_1, x_2) = 0$ $0.9f_2(x_1, x_2) + 0.9g_2(x_1, x_2) = 0$	$x_1^{0.9}$ $x_2^{0.9}$	
1	$f_1(x_1, x_2) = 0$ $f_2(x_1, x_2) = 0$	x_1^1 x_2^1	Required solution

The essential features of this continuation method are:

- The solution is *continued* from the known solution
- The closer $g_1(x_1, x_2)$ and $g_2(x_1, x_2)$ are to $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$, the better chances of this procedure working.
- However, the success of the method is generally not particularly sensitive to the selection of $g_1(x_1, x_2)$ and $g_2(x_1, x_2)$. For example, we could try linear functions

$$\begin{aligned}
g_1(x_1, x_2) &= (x_1 - x_1^0) + (x_2 - x_2^0) \\
g_2(x_1, x_2) &= (x_1 - x_1^0) + (x_2 - x_2^0)
\end{aligned}$$

- A root finder is required at each step (each value of t).
- However, the initial guess for the root finder can be the solution of the preceding step (since the change in the solution will be small from one step to the next).
- If the method fails, the increment in t can be reduced and the calculation repeated (the calculations in the preceding table are easily programmed in a loop, and the loop can be repeated more often to improve the successful continuation from the assumed initial condition to the final solution).
- The extension to the $n \times n$ problem is straightforward.

The homotopy functions of eq. (19) are just one possibility with one particular form of embedding of the continuation parameter t . Here are some other possibilities:

(1)

$$\begin{aligned}
h_1(x_1, x_2, t) &= f_1(x_1, x_2) - (1 - t)f_1(x_1^0, x_2^0) \\
h_2(x_1, x_2, t) &= f_2(x_1, x_2) - (1 - t)f_2(x_1^0, x_2^0)
\end{aligned} \tag{20}$$

with the properties

$$\begin{aligned}
h_1(x_1, x_2, 0) &= 0 \\
h_2(x_1, x_2, 0) &= 0
\end{aligned}$$

$$\begin{aligned}h_1(x_1, x_2, 1) &= f_1(x_1, x_2) \\h_2(x_1, x_2, 1) &= f_2(x_1, x_2)\end{aligned}$$

as required.

(2)

$$\begin{aligned}h_1(x_1, x_2, t) &= f_1(x_1, x_2) - e^{-t}f_1(x_1^0, x_2^0) \\h_2(x_1, x_2, t) &= f_2(x_1, x_2) - e^{-t}f_2(x_1^0, x_2^0)\end{aligned}\tag{21}$$

with the properties

$$\begin{aligned}h_1(x_1, x_2, 0) &= 0 \\h_2(x_1, x_2, 0) &= 0\end{aligned}$$

$$\begin{aligned}h_1(x_1, x_2, \infty) &= f_1(x_1, x_2) \\h_2(x_1, x_2, \infty) &= f_2(x_1, x_2)\end{aligned}$$

as required (note that the interval in t is now $0 \leq t \leq \infty$).

Dauidenko's Equation

If we consider again the embedding of eqs. (21), we can analyze it by considering the differentials of the homotopy functions

$$\begin{aligned} dh_1 &= \frac{\partial h_1}{\partial x_1} dx_1 + \frac{\partial h_1}{\partial x_2} dx_2 + \frac{\partial h_1}{\partial t} dt = 0 \\ dh_2 &= \frac{\partial h_2}{\partial x_1} dx_1 + \frac{\partial h_2}{\partial x_2} dx_2 + \frac{\partial h_2}{\partial t} dt = 0 \end{aligned} \quad (22)$$

(these differentials are zero since $h_1(x_1, x_2, t) = 0, h_2(x_1, x_2, t) = 0$). From eqs. (21) and (22), we have

$$\begin{aligned} dh_1 &= \frac{\partial f_1}{\partial x_1} dx_1 + \frac{\partial f_1}{\partial x_2} dx_2 + e^{-t} f_1(x_1^0, x_2^0) dt = 0 \\ dh_2 &= \frac{\partial f_2}{\partial x_1} dx_1 + \frac{\partial f_2}{\partial x_2} dx_2 + e^{-t} f_2(x_1^0, x_2^0) dt = 0 \end{aligned}$$

or

$$\begin{aligned} \frac{\partial f_1}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial f_1}{\partial x_2} \frac{dx_2}{dt} &= -e^{-t} f_1(x_1^0, x_2^0) = -f_1(x_1, x_2) \\ \frac{\partial f_2}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial f_2}{\partial x_2} \frac{dx_2}{dt} &= -e^{-t} f_2(x_1^0, x_2^0) = -f_2(x_1, x_2) \end{aligned} \quad (23)$$

Eqs. (23), a differential form of Newton's method, are the *Dauidenko differential equation*, that can be written in matrix form for the $n \times n$ problem as

$$J \frac{d\mathbf{x}}{dt} = -\mathbf{f} \quad (24)$$

where again J and \mathbf{f} are the Jacobian matrix and function vector of the nonlinear system, respectively, and $\frac{d\mathbf{x}}{dt}$ is the derivative of the solution vector with respect to the continuation parameter, t (compare eq. (11) with eq. (24) to see why Dauidenko's ODE is a differential form of Newton's method).

Eq. (24) defines an initial value problem, requiring an initial condition vector that serves as the assumed initial or starting solution

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{25}$$

The procedure for the solution of the nonlinear system is to integrate eq. (24) until

$$J \frac{d\mathbf{x}}{dt} \approx 0 \text{ which implies } \mathbf{f} \approx \mathbf{0} \tag{26}$$

This is the required solution of the nonlinear system, i.e., whatever the solution vector, $\mathbf{x}(t)$, of eq. (24) is when condition (26) occurs is the required solution of the original nonlinear system.

Note that form of ODEs (23); there is more than one derivative in each ODE, and they are termed *linearly implicit ODEs*. Special integrators are available for ODEs of this form, e.g., LSODI, DASSL.

Since eqs. (23) are linear in the derivatives, another approach to integrating them is first use a linear algebraic equation solver to compute the derivative vector, $\frac{d\mathbf{x}}{dt}$, then use a conventional ODE integrator for the *explicit ODEs* to compute a solution. In other words, eq. (24) can be written as

$$\frac{d\mathbf{x}}{dt} = -J^{-1}\mathbf{f} \tag{27}$$

This method works well for relatively low order systems.

The advantages of using eq. (24) or eq. (27) relative to the conventional finite stepping Newton method, eq. (11), are

- The solution of eqs. (24) or (27) proceeds by differential steps rather than the finite steps of Newton's method. Thus, the convergence to a solution is often more reliable, i.e., Davidenko's method may work when Newton's method fails.

- In other words, we are relying on the error monitoring and step size control of an ODE integrator (rather than the use of damping in Newton's method).
- The implementation of eq. (24) requires only an integrator for linearly implicit ODEs, e.g., LSODI, DASSL.
- Eq. (27) requires only a linear equation solver plus an integrator for explicit ODEs.

Thus, the differential approach to Newton's method can be implemented with readily available software.

To conclude this section, consider the application of eqs. (24) and (27) to problem (2). For eq. (24)

$$\begin{bmatrix} 2x_1 & 2x_2 \\ (2/3)x_1^{-2/3} & (1/2)x_2^{-1/2} \end{bmatrix} \begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \end{bmatrix} = - \begin{bmatrix} x_1^2 + x_2^2 - 17 \\ 2x_1^{1/3} + x_2^{1/2} - 4 \end{bmatrix}$$

or

$$\begin{aligned} 2x_1 \frac{dx_1}{dt} + 2x_2 \frac{dx_2}{dt} &= -(x_1^2 + x_2^2 - 17) \\ (2/3)x_1^{-2/3} \frac{dx_1}{dt} + (1/2)x_2^{-1/2} \frac{dx_2}{dt} &= -(2x_1^{1/3} + x_2^{1/2} - 4) \end{aligned} \quad (28)$$

For eq. (27)

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \end{bmatrix} = - \begin{bmatrix} 2x_1 & 2x_2 \\ (2/3)x_1^{-2/3} & (1/2)x_2^{-1/2} \end{bmatrix}^{-1} \begin{bmatrix} x_1^2 + x_2^2 - 17 \\ 2x_1^{1/3} + x_2^{1/2} - 4 \end{bmatrix} \quad (29)$$

A Fortran program for the solution of eqs. (29) is given in Schiesser (1994), pp 174-193.

Finally, we can analyze the convergence of solutions to Davidenko's ODE, eq. (24). For the scalar case, eq. (24) becomes

$$\frac{df}{dx} \frac{dx}{dt} = -f$$

which rearranges to

$$\frac{df}{f} = -dt$$

Integration of each side of this ODE, subject to the initial condition

$$f(t_0) = f_0$$

gives

$$\ln(f/f_0) = -(t - t_0)$$

or

$$f = f_0 e^{-(t-t_0)} \tag{30}$$

Eq. (30) indicates that the convergence of Davidenko's method is exponential in t . This result also carries over to the $n \times n$ case. This exponential convergence is demonstrated numerically for problem (2) in Schiesser (1994).

Levenberg-Marquardt Method

A major difficulty with Newton's method, eq. (11), or Davidenko's method, eq. (24), is failure caused by a singular or near singular Jacobian matrix (note that both methods require the inverse Jacobian matrix). To circumvent this problem, we consider the discrete and differential (continuous) Levenberg-Marquardt methods:

$$\{(1 - \mu)J^T J + \mu I\} \Delta x = -J^T \mathbf{f} \quad (31)$$

$$\{(1 - \mu)J^T J + \mu I\} \frac{dx}{dt} = -J^T \mathbf{f} \quad (32)$$

When $\mu = 0$, eqs. (31) and (32) reduce to

$$J^T J \Delta x = -J^T \mathbf{f} \text{ or } J \Delta x = -\mathbf{f} \quad (33)$$

$$J^T J \frac{dx}{dt} = -J^T \mathbf{f} \text{ or } J \frac{dx}{dt} = -\mathbf{f} \quad (34)$$

Eqs. (33) and (34) are just Newton's method and Davidenko's method, respectively (and both will fail if J is singular).

When $\mu = 1$, eqs. (31) and (32) reduce to

$$\Delta x = -J^T \mathbf{f} \quad (35)$$

$$\frac{dx}{dt} = -J^T \mathbf{f} \quad (36)$$

Eqs. (35) and (36) express the *discrete and differential steepest descent*. Note that neither eq. (35) nor eq. (36) requires the inverse Jacobian. However, these equations reflect the conditions for a maximum or minimum in

$$ss = \sum_i^N f_i^2 \quad (37)$$

Thus, eqs. (35) and (36) can “get stuck” at a local maximum or minimum of ss of eq. (37). Also, when they do approach the solution $f_1 = f_2 = \dots f_N = 0$, they do so relatively slowly (compared to Newton’s method).

Therefore, the approach to avoiding a singular system and still approach the solution $f_1 = f_2 = \dots f_N = 0$, is to use $0 < \mu < 1$, and to possibly vary μ as the solution proceeds by monitoring the condition of J .

Equations (31) and (32) have been coded in library software. In particular, eq. (32) is available as a Fortran program from WES. We conclude this discussion with the application of eq. (32) to problem (3).

The initial conditions for eq. (32) are

```
C...
C... First starting point
      IF(NORUN.EQ.1)THEN
          X(1)=0.5D0
          X(2)=0.5D0
          X(3)=0.5D0
          X(4)=0.5D0
      END IF
C...
C... LM parameter in eq. (1). Note: the calculation fails with
C... L = 0 for a singular system (determinant of the Jacobian
C... matrix is zero)
      L=0.5D0
```

The programming of the RHS of eqs. (32) is then

```
C...
```

```

C... Once past the singular point, the DLM parameter is reset to
C... zero to give the differential Newton's method (and thereby
C... enhance the rate of convergence)
      IF(T.GT.1.0D0)L=0.0D0
C...
C... Vector of functions (f in eq. (1))
      CALL FUNC(N,X,F)
C...
C... Jacobian matrix (J in eq. (1))
      CALL JAC(N,X,J)
C...
C...
C... Jacobian matrix transpose (J in eq. (1))
      CALL TRANS(N,J,JT)
C...
C...
C...
C... Matrix product (J*J in eq. (1))
      CALL MULT(N,N,N,J,JT,JTJ)
C...
C...
C...
C... Left hand side coupling matrix (1-L)*J*J + L*I in eq. (1))
      CALL CMAT(N,JTJ,L,ID,JTJPLI)
C...
C...
C...
C... Right hand side vector (-J*f in eq. (1))
      CALL MULT1(N,N,1,JT,F,JTF)
C...
C... Solve for the vector of derivatives, dx/dt, in eq. (1) (in
C... effect, eq. (1) is premultiplied by the inverse of the LHS
C... coupling matrix). DECOMP performs an LU decomposition and
C... computes the condition number of JTJPLI
      CALL DECOMP(N,N,JTJPLI,COND,IPVT,WORK)
C...
C... Compute derivative vector dx/dt from JTJPLI with RHS JTF,
C... i.e., solve eq. (1) for dx/dt. The solution is returned
C... in JTF
      CALL SOLVE(N,N,JTJPLI,JTF,IPVT)
C...
C... Transfer array JTF to array DXDT so the derivative vector,
C... dx/dt, is available in COMMON/F/ for the ODE integrator

```

```

        DO 2 I=1,N
            DXDT(I)=JTF(I,1)
2       CONTINUE
        RETURN
        END

```

The programming of the function vector is

```

        SUBROUTINE FUNC(N,X,F)
C...
C... Subroutine FUNC computes the vector of functions,
C... -f, in eq. (1) (note the minus sign, as required
C... in the RHS of eq. (1))
C...
C... Arguments
C...
C...     N     order of the NLE system (number of NLE)
C...           (input)
C...
C...     X     solution vector of size N (input)
C...
C...     F     function vector of size N (output)
C...
        IMPLICIT DOUBLE PRECISION(A-H,O-Z)
        DIMENSION X(N), F(N)
C...
C... COMMON area for the problem parameters
        COMMON/P/ VR, Q, CA0, RK1, RK2, RK3, RK4
C...
C... Problem variables
        CA=X(1)
        CB=X(2)
        CC=X(3)
        CD=X(4)
C...
C... Function vector
        F(1)=-CA+VR*(-RK1*CA-RK2*CA**1.5D0+RK3*CC**2)/Q+CA0

```

```

F(2)=-CB+VR*(2.0D0*RK1*CA-RK4*CB**2)/Q
F(3)=-CC+VR*(RK2*CA**1.5D0-RK3*CC**2+RK4*CB**2)/Q
F(4)=-CD+VR*(RK4*CB**2)/Q
C...
C... Reverse the sign of the function vector in accordance
C... with the RHS of eq. (1)
      DO 1 I=1,N
          F(I)=-F(I)
1      CONTINUE
      RETURN
      END

```

The programming of the Jacobian matrix is:

```

      SUBROUTINE JAC(N,X,J)
C...
C... Subroutine JAC computes the Jacobian matrix
C... of the vector of functions, f, in eq. (1)
C... (note the elements of J are computed for f,
C... not -f)
C...
C... Arguments
C...
C...      N      order of the NLE system (number of NLE)
C...             (input)
C...
C...      X      solution vector of size N (input)
C...
C...      J      Jacobian matrix of size N x N (output)
C...
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DOUBLE PRECISION J
      DIMENSION X(N), J(N,N)
C...
C... COMMON area for the problem parameters
      COMMON/P/ VR, Q, CA0, RK1, RK2, RK3, RK4
C...

```

```

C... Problem variables
CA=X(1)
CB=X(2)
CC=X(3)
CD=X(4)

C...
C... Jacobian matrix
J(1,1)=-1.0D0+VR*(-RK1-1.5D0*RK2*CA**0.5)/Q
J(1,2)=0.0D0
J(1,3)=VR*(2.0D0*RK3*CC)/Q
J(1,4)=0.0D0
J(2,1)=VR*(2.0D0*RK1)/Q
J(2,2)=-1.0D0+VR*(-2.0D0*RK4*CB)/Q
J(2,3)=0.0D0
J(2,4)=0.0D0
J(3,1)=VR*(1.5D0*RK2*CA**0.5D0)/Q
J(3,2)=VR*(2.0D0*RK4*CB)/Q
J(3,3)=-1.0D0+VR*(-2.0D0*RK3*CC)/Q
J(3,4)=0.0D0
J(4,1)=0.0D0
J(4,2)=VR*(2.0D0*RK4*CB)/Q
J(4,3)=0.0D0
J(4,4)=-1.0D0
RETURN
END

```

The output from the complete program is:

```

RUN NO. - 1 Four component, steady state CSTR
          equations

INITIAL T - 0.000D+00

FINAL T - 0.200D+03

PRINT T - 0.500D+02

```

NUMBER OF DIFFERENTIAL EQUATIONS - 4

INTEGRATION ALGORITHM - LSODES

MAXIMUM INTEGRATION ERROR - 0.100D-06

1

t = 0.0
Ca = 0.50000 dCa/dt = -0.416D+01 res f1 = 0.616D+00
Cb = 0.50000 dCb/dt = -0.261D+01 res f2 = -0.130D+01
Cc = 0.50000 dCc/dt = -0.176D+01 res f3 = 0.184D+00
Cd = 0.50000 dCd/dt = -0.122D+01 res f4 = 0.300D+00

t = 50.0
Ca = 0.31814 dCa/dt = 0.120D-03 res f1 = -0.200D-02
Cb = 0.78160 dCb/dt = 0.249D-03 res f2 = -0.222D-02
Cc = 0.53095 dCc/dt = 0.405D-03 res f3 = -0.136D-02
Cd = 0.48780 dCd/dt = 0.375D-03 res f4 = -0.926D-03

t = 100.0
Ca = 0.31886 dCa/dt = -0.899D-04 res f1 = -0.992D-05
Cb = 0.78387 dCb/dt = -0.648D-04 res f2 = -0.211D-04
Cc = 0.53495 dCc/dt = -0.444D-04 res f3 = -0.128D-04
Cd = 0.49155 dCd/dt = -0.410D-04 res f4 = -0.812D-05

t = 150.0
Ca = 0.31887 dCa/dt = -0.118D-05 res f1 = 0.857D-06
Cb = 0.78388 dCb/dt = -0.118D-05 res f2 = 0.887D-06
Cc = 0.53498 dCc/dt = -0.419D-06 res f3 = 0.257D-06
Cd = 0.49158 dCd/dt = -0.874D-06 res f4 = -0.303D-07

t = 200.0
Ca = 0.31887 dCa/dt = 0.111D-06 res f1 = 0.135D-06
Cb = 0.78388 dCb/dt = -0.213D-07 res f2 = 0.209D-06
Cc = 0.53498 dCc/dt = 0.166D-06 res f3 = 0.779D-07
Cd = 0.49158 dCd/dt = -0.165D-07 res f4 = 0.121D-07

COMPUTATIONAL STATISTICS

LAST STEP SIZE	0.658D+02
LAST ORDER OF THE METHOD	1
TOTAL NUMBER OF STEPS TAKEN	147
NUMBER OF FUNCTION EVALUATIONS	232
NUMBER OF JACOBIAN EVALUATIONS	4

The four residuals follow the semi-log variation with t discussed previously (eq. (30)).

References and Software

NAG routines pertaining to nonlinear equations include:

Chapter C02 - Zeros of Polynomials

C02AFF - All zeros of complex polynomial,
modified Laguerre method

C02AGF - All zeros of real polynomial,
modified Laguerre method

C02AHF - All zeros of complex quadratic

C02AJF - All zeros of real quadratic

Chapter C05 - Roots of One or More Transcendental Equations

C05ADF - Zero of continuous function in given
interval, Bus and Dekker algorithm

C05AGF - Zero of continuous function, Bus and
Dekker algorithm, from given starting
value, binary search for interval

C05AJF - Zero of continuous function, continuation
method, from a given starting value

C05AVF - Binary search for interval containing
zero of continuous function (reverse
communication)

C05AXF - Zero of continuous function by continuation
method, from given starting value (reverse
communication)

- C05AZF - Zero in given interval of continuous function by Bus and Dekker algorithm (reverse communication)
- C05NBF - Solution of system of nonlinear equations using function values only (easy-to-use)
- C05NCF - Solution of system of nonlinear equations using function values only (comprehensive)
- C05NDF - Solution of systems of nonlinear equations using function values only (reverse communication)
- C05PBF - Solution of system of nonlinear equations using 1st derivatives (easy-to-use)
- C05PCF - Solution of system of nonlinear equations using 1st derivatives (comprehensive)
- C05PDF - Solution of systems of nonlinear equations using 1st derivatives (reverse communication)
- C05ZAF - Check user's routine for calculating 1st derivatives

A Web page with additional information about the solution of nonlinear equations:

<http://www-fp.mcs.anl.gov/otc/Guide/OptWeb>

[/continuous/unconstrained/nonlineareq/index.html](http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/continuous/unconstrained/nonlineareq/index.html)

Byrne, G. D., and C. A. Hall (1973), Numerical Solution of Systems of Non-linear Algebraic Equations, Academic Press, New York.

Dennis, J.E. and R.B. Schnabel (1986), Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, NJ.

Finlayson, B.A. (1980), Applied Nonlinear Analysis in Chemical Engineering, McGraw-Hill, New York.

Riggs, J. B. (1994), An Introduction to Numerical Methods for Chemical Engineers, 2nd ed., Texas Tech University Press, Lubbock, Texas.

Schiesser, W. E. (1994), Computational Mathematics in Engineering and Applied Science: ODEs, DAEs and PDEs, CRC Press, Boca Raton, FL.