SECTION II

NUMERICAL TECHNIQUES

ALGEBRAIC EQUATIONS 3

The purpose of this chapter is to introduce methods to solve systems of algebraic equations. After studying this module, the student should be able to:

- Solve systems of linear algebraic equations.
- Solve nonlinear functions of one variable graphically and numerically.
- Use the MATLAB function `fzero` to solve a single algebraic equation.
- Discuss the stability of iterative techniques.
- Use the MATLAB function `fsolve` to solve sets of nonlinear algebraic equations.

The major sections in this chapter are:

3.1 INTRODUCTION

In Chapter 2 we discussed how to develop a model that consists of a set of ordinary differential equations. To solve these problems we need to know the initial conditions and how the inputs and parameters change with time. Often the initial conditions will be the steady-state values of the process variables. To obtain a steady-state solution of a system of differential equations requires the solution of a set of algebraic equations. The purpose of this chapter is to review techniques to solve algebraic equations.

Consider a set of n equations in n unknowns. The representation is

$$\begin{aligned} f_1(x_1, x_2, \ldots x_n) &= 0 \\ f_2(x_1, x_2, \ldots x_n) &= 0 \\ &\cdot \\ &\cdot \\ f_n(x_1, x_2, \ldots x_n) &= 0 \end{aligned} \tag{3.1}$$

The objective is to solve for the set of variables, x_i, that force all of the functions, f_i, to zero. A solution is called a *fixed point* or an *equilibrium point.*

Vector notation is used for a compact representation

$$\mathbf{f(x)} = \mathbf{0} \tag{3.2}$$

where $\mathbf{f(x)}$ is a vector valued function. Notice that these can be the same functional relationships that were developed as a set of differential equations, with $\dot{\mathbf{x}} = \mathbf{f(x)} = \mathbf{0}$. The solution $\mathbf{x}$ is then a steady-state solution to the system of differential equations.

Before we cover techniques for systems of nonlinear equations, it is instructive to review systems of linear equations.

3.2 GENERAL FORM FOR A LINEAR SYSTEM OF EQUATIONS

Consider a linear system with n equations and n unknowns. The first equation is

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1$$

where the a's and b's are known constant parameters, and the x's are the unknowns. The second equation is:

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2$$

while the nth equation is:

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_3$$

The coefficient, a_{ij}, relates the jth dependent variable to the ith equation.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & a_{1n} \\ a_{21} & a_{22} & a_{23} & . & . & a_{2n} \\ . & . & . & . & . & . \\ a_{n1} & a_{n2} & a_{n3} & . & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ . \\ . \\ b_n \end{bmatrix} \tag{3.3}$$

Or, using compact matrix notation:

$$\mathbf{Ax} = \mathbf{b} \tag{3.3a}$$

The goal is to solve for the unknowns, $\mathbf{x}$. Notice that (3.3a) is the same form as (3.2), with

$$\mathbf{f(x)} = \mathbf{Ax} - \mathbf{b}$$

Premultiplying each side of (3a) by $\mathbf{A}^{-1}$ we find:

$$\mathbf{A}^{-1}\mathbf{A}\,\mathbf{x} = \mathbf{b}\mathbf{u}$$

and since,

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \tag{3.4}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

provided that the inverse of $\mathbf{A}$ exists. If the rank of $\mathbf{A}$ is less than n, then $\mathbf{A}$ is singular and the matrix inverse does not exist. If the condition number of $\mathbf{A}$ is very high, then the solution may be sensitive to model error. The concepts of rank and condition number are reviewed in Module 2 in Section V.

Equation (3.4) is used for conceptual purposes to represent the solution of the set of linear equations. In practice the solution is not obtained by finding the matrix inverse. Rather, equation (3.3a) is directly solved using a numerical technique such as Gaussian elimination or LU decomposition. Since the codes to implement these techniques are readily available in any numerical library, we do not review them here.

The next example illustrates how MATLAB can be used to solve a system of linear equations.

EXAMPLE 3.1 Linear Absorption Model, Solved Using MATLAB

Consider a 5-stage absorption column (presented in Module 6 in Section V) that has a model of the following form ($\mathbf{x}$ is a vector of stage liquid-phase compositions and $\mathbf{u}$ is a vector of column feed compositions):

$$0 = \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u}$$

or

$$\mathbf{A}\,\mathbf{x} = -\mathbf{B}\,\mathbf{u}$$

the solution for $\mathbf{x}$ is $\mathbf{x} = -\mathbf{A}^{-1}\mathbf{B}\,\mathbf{u}$

The values of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{u}$ are:

```
a =
   -0.3250    0.1250         0         0         0
    0.2000   -0.3250    0.1250         0         0
         0    0.2000   -0.3250    0.1250         0
         0         0    0.2000   -0.3250    0.1250
         0         0         0    0.2000   -0.3250
b =
    0.2000         0
         0         0
         0         0
         0         0
         0    0.2500
u =
         0
    0.1000
```

The following MATLAB command can be used to solve for **x**

```
» x = -inv(a)*b*u

x =
    0.0076
    0.0198
    0.0392
    0.0704
    0.1202
```

Use of the MATLAB left-division operator (\) yields the same result more efficiently (faster computation time), using the LU decomposition technique:

```
» x = -a\(b*u)
```

3.3 NONLINEAR FUNCTIONS OF A SINGLE VARIABLE

Functions of a single variable can be solved graphically by plotting $f(x)$ for many values of x and finding the values of x where $f(x) = 0$. This approach is shown in Figure 3.1. An interesting and challenging characteristic of nonlinear algebraic equations is the potential for multiple solutions, as shown in Figure 3.1b. In fact, for a single nonlinear algebraic equation it is often not possible to even know (without a detailed analysis) the number of solutions that exist. The situation is easier for polynomials because we know that an nth order polynomial has n solutions. Fortunately, many chemical process problems have a single solution that makes physical sense.

Numerical techniques for solving nonlinear algebraic equations are covered in the next section. A graphical representation will be used to provide physical insight for the numerical techniques.

Numerical methods to solve nonlinear algebraic equations are also known as *iterative techniques*. A sequence of guesses to the solution are made until we are "close enough" to the actual solution. To understand the concept of "closeness" we must use the notion of convergence tolerance.

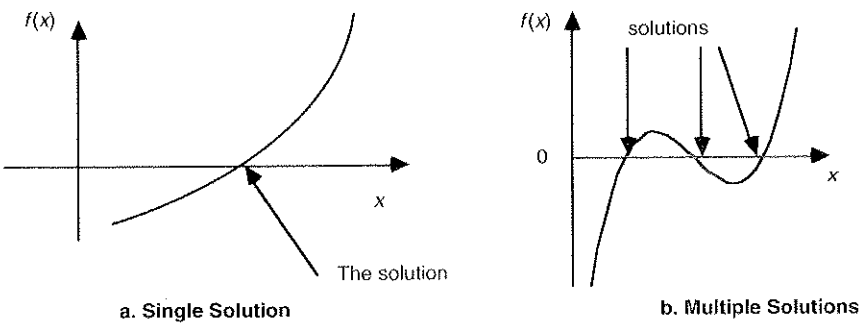


FIGURE 3.1 Graphical solution to $f(x) = 0$.

3.3.1 Convergence Tolerance

A solution to a problem $f(x) = 0$ is considered converged at iteration k if:

$$|f(x_k)| \leq \varepsilon$$

where ε is a *tolerance* that has been specified, $|f(x_k)|$ is used to denote the absolute value of function $f(x)$ evaluated at iteration k, and x_k is the variable value at iteration k. At times it is useful to base convergence on the variable rather than the function; for example, a solution can be considered converged at iteration k if the change in the variable is less than a certain *absolute tolerance*, ε_a:

$$|x_k - x_{k-1}| \leq \varepsilon_a$$

The absolute tolerance is dependent on the scaling of the variable, so a *relative tolerance* specification, ε_r, is often used:

$$\frac{|x_k - x_{k-1}|}{|x_{k-1}|} < \varepsilon_r$$

The relative tolerance specification is not useful if x is converging to 0. A combination of the relative and absolute tolerance specifications is often used, and can be expressed as

$$|x_k - x_{k-1}| < |x_{k-1}|\,\varepsilon_r + \varepsilon_a$$

The iterative methods that we present for solving single algebraic equations are: (i) direct substitution, (ii) interval halving, (iii) false position, and (iv) Newton's method.

3.3.2 Direct Substitution

Perhaps the simplest algorithm for a single variable nonlinear algebraic equation is known as direct substitution. We have been writing the relationship for a single equation in a single unknown as

$$f(x) = 0 \tag{3.5}$$

Using the direct substitution technique, we rewrite (3.5) in the form

$$x = g(x) \tag{3.6}$$

this means that our "guess" for x at iteration $k+1$ is based on the evaluation of $g(x)$ at iteration k (subscripts are used to denote the iteration)

$$x_{k+1} = g(x_k) \tag{3.7}$$

If formulated properly, (3.7) converges to a solution (within a desired tolerance)

$$x^* = g(x^*) \tag{3.8}$$

If not formulated properly, (3.7) may diverge or converge to physically unrealistic solutions, as shown by the following example.

EXAMPLE 3.2 A Reactor with Second-Order Kinetics

The dynamic model for an isothermal, constant volume, chemical reactor with a single second-order reaction is:

$$\frac{dC_A}{dt} = \frac{F}{V} C_{Af} - \frac{F}{V} C_A - kC_A^2$$

Find the steady-state concentration for the following inputs and parameters:

$$F/V = 1 \text{ min}^{-1},\ C_{Af} = 1 \text{ gmol/liter},\ k = 1 \text{ liter/(gmol min)}$$

At steady-state, $dC_A/dt = 0$, and substituting the parameter and input values, we find

$$1 - C_{As} - C_{As}^2 = 0$$

where the subscript s is used to denote the steady-state solution. For notational convenience, let $x = C_{As}$, and write the algebraic equation as

$$f(x) = -x^2 - x + 1 = 0$$

We can directly solve this equation using the quadratic formula to find $x = -1.618$ and 0.618 to be the solutions. Obviously a concentration cannot be negative, so the only physically meaningful solution is $x = 0.618$. Although we know the answer using the quadratic formula, our objective is to illustrate the behavior of the direct substitution method.

To use the direct substitution method, we can rewrite the function in two different ways: (i) $x^2 = -x + 1$ and (ii) $x = -x^2 + 1$. We will analyze (i) and leave (ii) as an exercise for the reader (see student exercise 4).

(i) Here we rewrite $f(x)$ to find the following direct substitution arrangement

$$x^2 = -x + 1$$

$$x = \sqrt{-x + 1} = g(x)$$

Or, using subscript k to indicate the kth iteration

$$x_{k+1} = \sqrt{-x_k + 1} = g(x_k)$$

For a first guess of $x_0 = 0.5$, we find the following sequence

$$x_1 = \sqrt{-0.5 + 1} = 0.7071$$

$$x_2 = \sqrt{-0.7071 + 1} = 0.5412$$

$$x_3 = \sqrt{-0.5412 + 1} = 0.6774$$

$$x_4 = \sqrt{-0.6774 + 1} = 0.5680$$

This sequence slowly converges to 0.618, as shown in Figure 3.2.

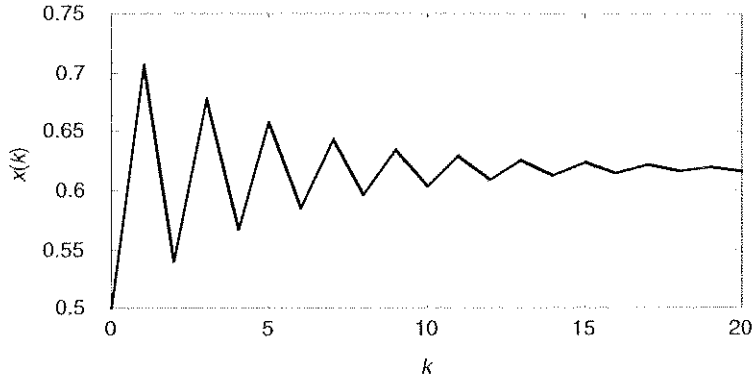


FIGURE 3.2 The iteration $x_{k+1} = \sqrt{-x_{k+1} + 1}$ with $x_0 = 0.5$. This sequence converges to 0.6180.

Notice that an initial guess of $x_0 = 0$ or 1 oscillates between 0 and 1, never converging or diverging, as shown in Figure 3.3.

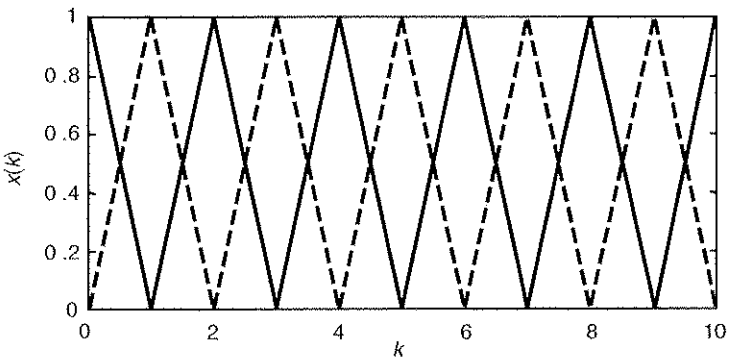


FIGURE 3.3 The iteration $x_{k+1} = \sqrt{-x + 1}$ with $x_0 = 0$ (dashed) or 1 (solid). This sequence oscillates between 0 and 1.

As noted earlier, this problem has two solutions ($x^* = -1.618$ and $x^* = 0.618$), since it is a second-order polynomial. This can be verified by plotting x versus $f(x)$ as shown in

Figure 3.4. From physical reasoning, we accept only the positive solution, since a concentration cannot be negative.

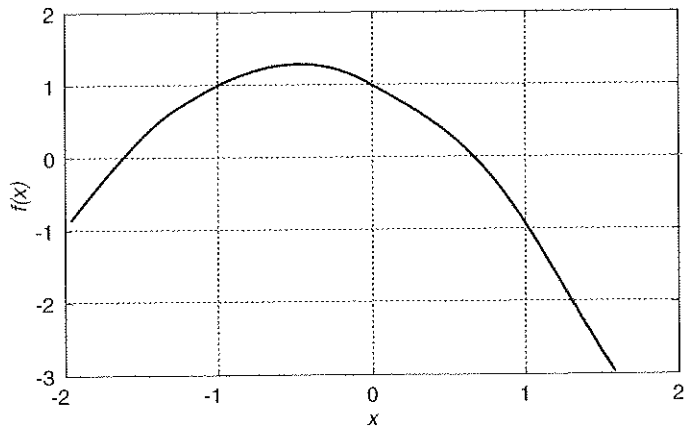


FIGURE 3.4 Plot of $f(x)$ versus x to find where $f(x) = 0$.

Example 3.2 illustrates that certain initial guesses may oscillate and never yield a solution, while other guesses may converge to a solution. It turns out that the way that a direct substitution problem is formulated may eliminate valid solutions from being reached (see student exercise 4).

These problems exist, to a certain extent, with any numerical solution technique. The potential problems appear to be worse with direct substitution; direct substitution is not generally recommended unless experience with a particular problem indicates that results are satisfactory. Direct substitution is often the easiest numerical technique to formulate for the solution of a single nonlinear algebraic equation.

If a numerical technique does not converge to a solution when the initial guess is close to the solution, we refer to the solution as unstable. The stability of iterative methods is discussed in the appendix.

3.3.3 Interval Halving (Bisection)

The interval halving technique only requires that the sign of the function value is known. The following steps are used in the interval halving technique:

1. Bracket the solution by finding two values of x, one where $f(x)$ is less than zero and another where $f(x)$ is greater than zero.
2. Evaluate the function, $f(x)$, at the midpoint of the bracket.
3. Replace the bracket limit that has the same sign as the function value at the midpoint, with the midpoint value. Check for convergence. If not converged, go back to step 2.

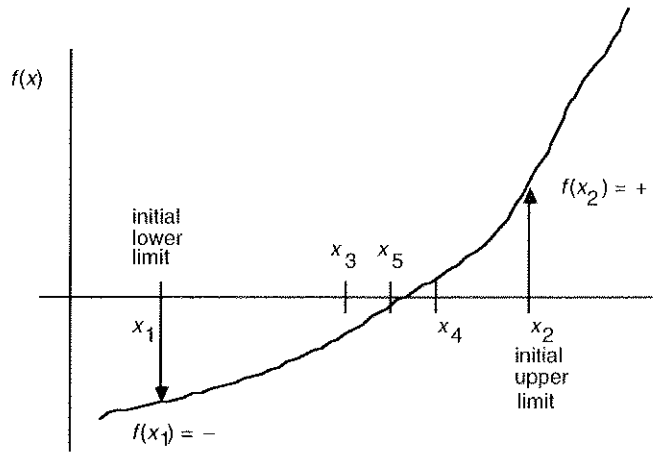


FIGURE 3.5 Illustration of the bisection technique.

An example of the interval halving approach is shown in Figure 3.5.

Notice that the solution was bracketed by

1. Finding x_1, where $f(x_1)$ is negative and x_2 where $f(x_2)$ is positive.
2. The midpoint between x_1 and x_2 was selected (x_3). The function value at x_3, $f(x_3)$, was negative, so
3. x_1 was thrown out and x_3 became the lower bracket point.
4. The midpoint between x_3 and x_2 was selected (x_4). The function value at x_4, $f(x_4)$, was positive, so
5. x_2 was thrown out and x_4 became the upper bracket point.
6. The midpoint between x_3 and x_4 was selected (x_5). The function value at x_5, $f(x_5)$, was negative, so
7. x_3 was thrown out and x_5 became the lower bracket point.

You can see that the midpoint between x_5 and x_4 will yield a positive value for $f(x_6)$, so that the x_4 point will be thrown out. You can also see that this process could go on for a very long time, depending on how close to zero you desire the solution. Engineering judgement must be used when making a convergence tolerance specification.

The advantage to interval bisection is that it is easy to understand. A disadvantage is that it is not easily extended to multivariable systems. Also, it can take a long time to reach the solution since it only uses information about the sign of the function values. The next technique is similar to interval bisection but uses the function values to determine the variable value for the next iteration.

3.3.4 False Position (Reguli Falsi)

The false position or reguli falsi technique uses the function values at two previous iterations to determine the value for the next iteration. The technique of false position consists of the following steps:

1. Select variable values x_k and x_{k+1} to bracket the solution.
2. Draw line between $f(x_k)$ and $f(x_{k+1})$ and find x_{k+2}.
3. Evaluate $f(x_{k+2})$. Replace the bracket limit that has the same sign for its function as the sign of $f(x_{k+2})$.

An example of the false position approach is shown in Figure 3.6.

The next step would be to draw a line from $f(x_3)$ to $f(x_2)$ and find x_4. Continue until a certain tolerance is met.

The false position method generally converges much more rapidly since it uses known function values to determine the next "guess" for the variable. We have shown graphically how each technique is used. You will have an opportunity in the student exercises to write an algorithm to implement the two techniques.

3.3.5 Newton's Method (or Newton-Raphson)

The most common method for solving nonlinear algebraic equations is known as Newton's method (or Newton-Raphson). Newton's method can be derived by performing a Taylor series expansion of $f(x)$:

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{f''(x)}{2}(\Delta x)^2 + \frac{f'''(x)}{6}(\Delta x)^3 + \ldots = 0 \qquad (3.9)$$

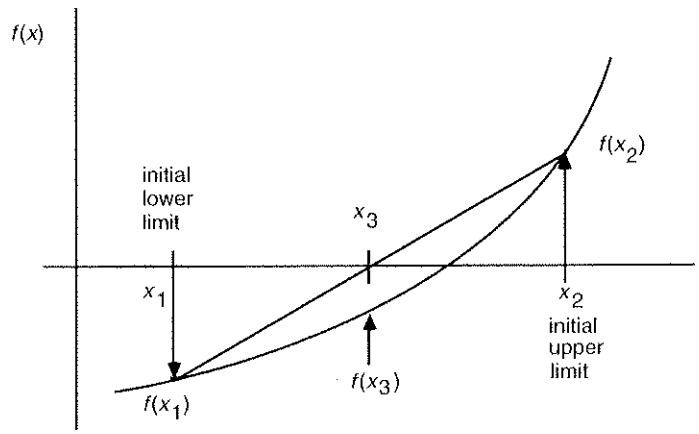


FIGURE 3.6 Illustration of the false position technique.

where

$$f'(x) = \frac{\partial f(x)}{\partial x}, f''(x) = \frac{\partial^2 f(x)}{\partial x^2}, \quad \text{and so on.}$$

Neglecting the second-order and higher derivative terms and solving for $f(x+\Delta x) = 0$, we obtain

$$\Delta x = \frac{-f(x)}{f'(x)} \tag{3.10}$$

Since this is an iterative procedure, calculate the guess for x at iteration $k+1$ as a function of the value at iteration k:

$$\text{defining} \qquad \Delta x_{k+1} = x_{k+1} - x_k \tag{3.11}$$

$$\text{from (3.10)} \qquad \Delta x_{k+1} = \frac{-f(x_k)}{f'(x_k)} \tag{3.12}$$

$$\text{from (3.11)} \qquad x_{k+1} - x_k = \frac{-f(x_k)}{f'(x_k)} \tag{3.13}$$

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}} \tag{3.14}$$

Equation (3.14) is known as Newton's method for a single-variable problem.

Notice that we can obtain the following graphical representation for Newton's method (Figure 3.7).

Starting from the initial guess of x_1, we find that x_2 is the intersection of $f'(x_1)$ with the x-axis. Evaluate $f(x_2)$ and draw a line with slope $f'(x_2)$ to the x-axis to find x_3. This procedure is continued until convergence.

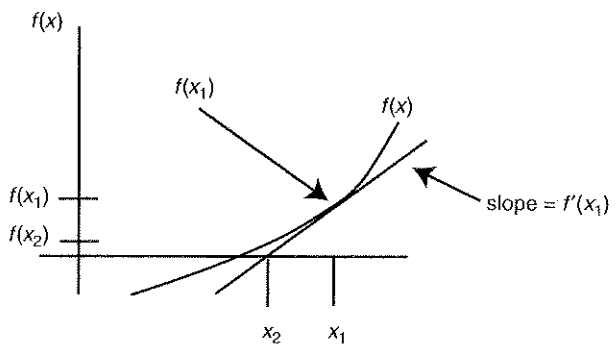


FIGURE 3.7 Illustration of Newton's method.

Advantages to Newton's method include quadratic convergence (when close to the solution) and that the method is easily extended to multivariable problems. Disadvantages include the fact that a derivative of the function is required, and that the method may not converge to a solution or may not converge to the nearest solution.

For an example of nonconvergence, consider the function shown in Figure 3.8. Here the initial guess is at a point where the derivative of function is equal to zero ($f'(x_0) = 0$), therefore there is no intersection with the x-axis to determine the next guess. We also see from (3.14) that there is no finite value for the next guess for x.

Another problem is that the solution could continuously oscillate between two values. Consider $f(x) = x^3 - x$. A plot of Newton's method for this function, with an initial guess of $x_0 = -1/\sqrt{5}$, is shown in Figure 3.9.

Notwithstanding the problems (possible division by zero or continuous oscillation) that we have shown with Newton's method, it (or some variant of Newton's) is still the most commonly used solution technique for nonlinear algebraic equations. Notice that Newton's method essentially linearizes the nonlinear model at each iteration and therefore results in successive solutions of linear models.

Notice that increasing amounts of information were needed to use the previous techniques. Interval halving required the sign of the function, reguli falsi required the value of the function, and Newton's method required the value and the derivative of the function. We also found that not all solutions to a nonlinear equation are stable when direct substitution is used. Next, we show that all solutions are stable using Newton's method.

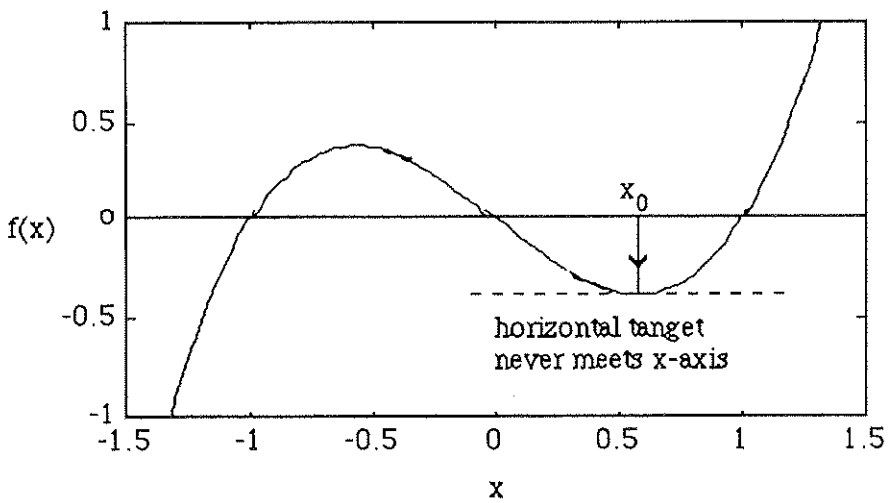


FIGURE 3.8 Problem with Newton's method when $f'(x) = 0$.

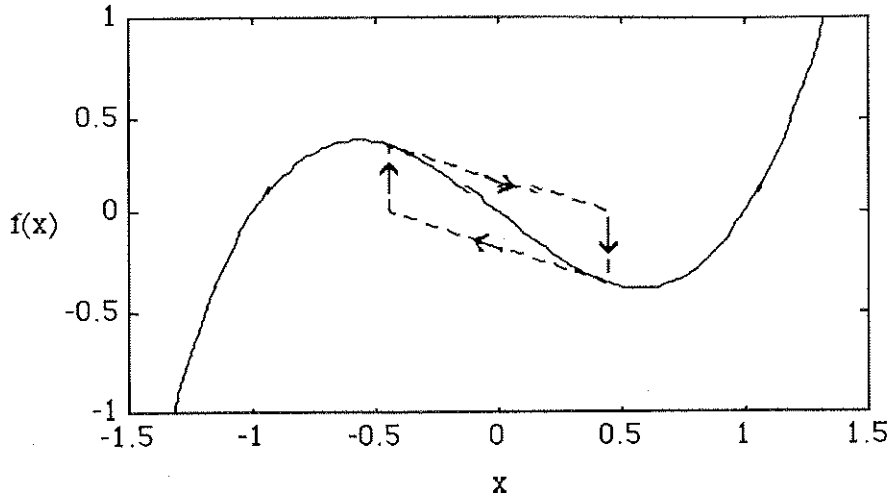


FIGURE 3.9 Oscillation of solution between two values.

3.4 MATLAB ROUTINES FOR SOLVING FUNCTIONS OF A SINGLE VARIABLE

MATLAB has two routines that can solve for the zeros of a function of a single variable. FZERO is used for a general nonlinear equation, while ROOTS can be used if the nonlinear equation is a polynomial.

3.4.1 FZERO

The first routine that we use for illustration purposes is `fzero`. `fzero` uses a combination of interval halving and false position.

In order to use `fzero`, you must first write a MATLAB m-file to generate the function that is being evaluated. Consider the function $f(x) = x^2 - 2x - 3 = 0$.

The following MATLAB m-file evaluates this function (the m-file is named `fcn1.m`):

```
function y = fcn1(x)
y = x^2 - 2*x - 3;
```

After generating the m-file `fcn1.m`, the user must provide a guess for the solution to the `fzero` routine. The following command gives an initial guess of $x = 0$.

```
y = fzero('fcn1',0)
```

MATLAB returns the answer:

```
y = -1
```

For an initial guess of $x = 2$, the user enters

```
z = fzero('fcn1',2)
```

and MATLAB returns the answer

```
z = 3
```

These results are consistent with those of Example 3.2, where we found that there were two solutions to a similar problem (we could use the quadratic formula to find them). Again, the solution obtained depends on the initial guess.

A third argument allows the user to select a relative tolerance (the default is the machine precision, `eps`). A fourth argument triggers a printing of the iterations.

3.4.2 ROOTS

Since the equation that we were solving was a polynomial equation, we could also use the MATLAB routine `roots` to find the zeros of the polynomial. Consider the polynomial function:

$$x^2 - 2x - 3 = 0$$

The user must create a vector of the coefficients of the polynomial, in descending order.

```
c = [1 -2 -3]'
```

Then the user can type the following command

```
roots(c)
```

and MATLAB returns

```
ans =
     3
    -1
```

Again, these are the two solutions that we expect.

3.5 MULTIVARIABLE SYSTEMS

In the previous sections we discussed the solution of a single algebraic equation with a single unknown variable. We covered direct substitution, bisection, reguli falsi, and Newton's method. In this section, we will discuss the reduction of a multivariable problem to a single-variable problem, as well as the multivariable Newton's method.

Consider a system of n nonlinear equations in n unknowns

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

There are some special cases where $n - 1$ variables can be solved in terms of one variable—then a single variable solution technique can be used. This approach is shown in the following example.

EXAMPLE 3.3 Reducing a two-variable problem to a single-variable problem

Solve the following system of nonlinear equations.

$$f_1(x_1,x_2) = x_1 - 4x_1^2 - x_1x_2 = 0 \tag{3.15}$$

$$f_2(x_1,x_2) = 2x_2 - x_2^2 + 3x_1x_2 = 0 \tag{3.16}$$

From (3.15) we can solve for x_2 in terms of x_1 to find:

$$x_2 = 1 - 4x_1 \tag{3.17}$$

Substituting (3.17) into (3.16), we find:

$$1 + 3x_1 - 28x_1^2 = 0 \tag{3.18}$$

which has the two solutions for x_1 (from the quadratic formula)

$$x_1 = 0.25 \quad \text{or} \quad x_1 = -0.1429$$

The corresponding values of x_2 (from (3.17)) are

$$x_2 = 0.0 \quad \text{and} \quad x_2 = 1.5714$$

Or, writing these solutions in vector form:

$$\textbf{solution 1 is } \mathbf{x} = \begin{bmatrix} 0.25 \\ 0 \end{bmatrix}, \text{ while } \textbf{solution 2 is } \mathbf{x} = \begin{bmatrix} -0.1429 \\ 1.5714 \end{bmatrix}$$

Question: Are we certain that there are only two solutions?

Observation: We can also see by inspection that the origin (sometimes called the trivial solution), $\mathbf{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, is also a solution. Another solution that is slightly less obvious is $\mathbf{x} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, which we can see by inspection satisfies (3.15) and (3.16).

Question: How did we miss the other two solutions?

Observation: Perhaps we will find the other solutions if we solve (3.16) for x_1 in terms of x_2, then substitute this result into (3.15) to solve for x_2. When this is done, we obtain the result that

$$x_2^2 - 4x_2 + 4 = 0$$

and using the quadratic formula $x_2 = \dfrac{4}{2} \pm \sqrt{\dfrac{16 - 16}{2}} = 2$

which gives us the solution $\mathbf{x} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$. Notice that we are still missing the trivial solution, $\mathbf{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

The mistake that we made was back at the first step, when we solved (3.15) for x_2 in terms of x_1 to find (3.17). We must recognize that (3.15) is quadratic in x_1, therefore there are two solutions for x_1 in terms of x_2. This is more clear if we write (3.15) as

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

and solve for x_1 to find $x_1 = 0$ or $x_1 = 1/4\,(1 - x_2)$. The reader should show that substituting these values into (3.16) will lead to the four solutions:

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 0.25 \\ 0 \end{bmatrix}, \text{and} \begin{bmatrix} -0.1429 \\ 1.5714 \end{bmatrix}.$$

The previous example illustrates the care that must be taken when using reduction techniques to solve several nonlinear algebraic equations.

If a problem cannot be reduced to a single variable, then a general multivariable strategy (such as that discussed in the next section) must be used.

3.5.1 Newton's Method for Multivariable Problems

Recall that we are solving the general set of equations

$$\mathbf{f}(\mathbf{x}) = 0 \tag{3.19}$$

That is, a set of n equations in n unknowns

$$\begin{bmatrix} f_1(x_1, x_2, \ldots x_n) \\ f_2(x_1, x_2, \ldots x_n) \\ \cdot \\ \cdot \\ f_n(x_1, x_2, \ldots x_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \tag{3.20}$$

The objective is to solve for the set of variables, x_i, that forces all of the functions, f_i, to zero.

We can use a Taylor series expansion for each f_i:

$$f_i(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \Delta x_j + \text{higher order terms} \tag{3.21}$$

Neglecting the higher order terms and writing in matrix form

$$\mathbf{f}(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \mathbf{J}\,\boldsymbol{\Delta}\mathbf{x} \tag{3.22}$$

where $\mathbf{J}$ is known as the *Jacobian*

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdot & \cdot & \dfrac{\partial f_1}{\partial x_n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdot & \cdot & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{3.23}$$

Now, since we wish to solve for $\Delta\mathbf{x}$ such that $\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}) = 0$, then (from 3.22)

$$\mathbf{f}(\mathbf{x}) + \mathbf{J}\,\Delta\mathbf{x} = 0 \tag{3.24}$$

Solving for $\Delta\mathbf{x}$ at iteration k

$$\Delta\mathbf{x}_k = -\mathbf{J}_k^{-1}\,\mathbf{f}(\mathbf{x}_k) \tag{3.25}$$

but $\Delta\mathbf{x}$ is simply the change in the $\mathbf{x}$ vector from the previous iteration

$$\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \tag{3.26}$$

Substituting (3.26) into (3.25)

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}_k^{-1}\,\mathbf{f}(\mathbf{x}) \tag{3.27}$$

Remember that $\mathbf{x}_k$ is a vector of values at iteration k. Notice that for a single equation (3.27) is:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

which is the result that we obtained in Section 3.3.4.

Comment: In practice the inverse of the Jacobian is not actually used in the solution of (3.25). Actually, (3.25) is solved as a set of linear algebraic equations, using Gaussian elimination or LU decomposition.

$$\mathbf{J}_k\,\Delta\mathbf{x}_k = -\mathbf{f}(\mathbf{x}_k)$$

where $\mathbf{J}_k$ and $\mathbf{f}(\mathbf{x}_k)$ are known at iteration k.

EXAMPLE 3.3 Revisited. Newton's method

$$f_1(x_1,x_2) = x_1 - 4x_1^2 - x_1x_2 = 0$$
$$f_2(x_1,x_2) = 2x_2 - x_2^2 + 3x_1x_2 = 0$$

or

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_1 - 4x_1^2 - x_1x_2 \\ 2x_2 - x_2^2 + 3x_1x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The Jacobian elements are

$$\mathbf{J}_{11} = \frac{\partial f_1}{\partial x_1} = 1 - 8x_1 - x_2 \qquad \mathbf{J}_{12} = \frac{\partial f_1}{\partial x_2} = -x_1$$

$$\mathbf{J}_{21} = \frac{\partial f_2}{\partial x_1} = 3x_2 \qquad \mathbf{J}_{22} = \frac{\partial f_2}{\partial x_2} = 2 - 2x_2 + 3x_1$$

so the Jacobian is written

$$\mathbf{J} = \begin{bmatrix} 1 - 8x_1 - x_2 & -x_1 \\ 3x_2 & 2 - 2x_2 + 3x_1 \end{bmatrix}$$

Consider an initial guess of $x_1 = -1$ and $x_2 = -1$. Let $\mathbf{x}(0)$ represent the vector for this initial guess

$$\mathbf{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

The value of the Jacobian at this initial guess is

$$\mathbf{J}(\mathbf{x}(0)) = \begin{bmatrix} 10 & 1 \\ -3 & 1 \end{bmatrix}$$

The inverse of the Jacobian is

$$\mathbf{J}^{-1}(\mathbf{x}(0)) = \begin{bmatrix} \frac{1}{13} & \frac{-1}{13} \\ \frac{3}{13} & \frac{10}{13} \end{bmatrix}$$

The value of the function vector for the initial guess is

$$\mathbf{f}(\mathbf{x}(0)) = \begin{bmatrix} -1 & -4 & -1 \\ 2(-1) & -1 & +3 \end{bmatrix} = \begin{bmatrix} -6 \\ 0 \end{bmatrix}$$

and the guess for $\mathbf{x}(1)$, where $\mathbf{x}(1)$ represents the vector at iteration 1, is

$$\mathbf{x}(1) = \mathbf{x}(0) - \mathbf{J}^{-1}(\mathbf{x}(0))\,\mathbf{f}(\mathbf{x}(0))$$

$$\mathbf{x}(1) = \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} \frac{1}{13} & \frac{-1}{13} \\ \frac{3}{13} & \frac{10}{13} \end{bmatrix} \begin{bmatrix} -6 \\ 0 \end{bmatrix}$$

$$\mathbf{x}(1) = \begin{bmatrix} -0.5385 \\ 0.3846 \end{bmatrix}$$

Continuing with iterations 2 through 7 we find the following results

Iteration	$\mathbf{x}_1$	$\mathbf{x}_2$
0	−1	−1
1	−0.5385	0.3846
2	−0.3104	1.0688
3	−0.2016	1.3952
4	−0.1561	1.5317
5	−0.1439	1.5683
6	−0.1429	1.5714
7	−0.1429	1.5714

The sequence of iterations for an initial guess of $x_1 = 1$ and $x_2 = 1$ is

Iteration	x_1	x_2
0	1	1
1	0.6190	0.0476
2	0.3893	0.0081
3	0.2870	0.0008
4	0.2542	0.0000
5	0.2501	0.0000
6	0.2500	0.0000

Notice that a guess of `x = [-1 -1]'` converged to `x = [-0.1429 1.5714]'` after six iterations, and a guess of `x = [ 1 1]'` converged to `x = [ 0.2500 0.0000]'` after six iterations. Other initial guesses may lead to the other two known solutions that were determined analytically.

The previous example illustrates that, for systems that have multiple solutions, the solution obtained depends on the initial guess.

3.5.2 Quasi-Newton Methods

Most computer codes actually implement some variant of Newton's method; these are referred to as quasi-Newton methods. Remember that Newton's method is guaranteed to converge only if the system is nonsingular and we are "close" to the solution.

DAMPING FACTOR

Often it is desirable to "dampen" the change in the guess for $\mathbf{x}_{k+1}$, to make Newton's method more stable. Applying a damping factor, α, to (3.27), we write

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\, \mathbf{J}_k^{-1}\, \mathbf{f}(\mathbf{x}_k) \tag{3.28}$$

where α is chosen so that $\| \mathbf{f}(\mathbf{x}_{k+1}) \| < \| \mathbf{f}(\mathbf{x}_k) \|$ and $0 < \alpha \leq 1$ (we use the $\| \mathbf{f}(\mathbf{x}_k) \|$ notation to represent the norm of the vector $\mathbf{f}(\mathbf{x}_k)$). Often α is selected to minimize $\| \mathbf{f}(\mathbf{x}_{k+1}) \|$ using a search technique, that is, α is adjusted until $\| \mathbf{f}(\mathbf{x}_{k+1}) \|$ is minimized.

It should be noted that the single-variable equivalent to (3.28) is

$$x_{k+1} = x_k - \alpha \frac{f(x_k)}{f'(x_k)} \tag{3.29}$$

HANDLING SINGULAR (OR ILL-CONDITIONED) JACOBIAN MATRICES

Notice that the Newton method with or without the damping factor requires the inverse of the Jacobian matrix (or the solution of a set of linear algebraic equations) to determine the

value for the next iteration. If the Jacobian is singular, it cannot be inverted. One method that avoids this problem is known as the Levenberg-Marquardt method:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - (\mathbf{J_k}^T\mathbf{J_k} + \beta\mathbf{I})^{-1}\,\mathbf{J_k}^T\,\mathbf{f}(\mathbf{x_k}) \tag{3.30}$$

where T represents the matrix transpose and β is an adjustable parameter used to avoid a singularity. The single variable equivalent to (3.30) is

$$x_{k+1} = x_k - \frac{f'(x_k)f(x_k)}{(f'(x_k))^2 + \beta} \tag{3.31}$$

Notice that if $\beta = 0$, the standard Newton algorithm results.

WHEN ANALYTICAL JACOBIAN MATRICES ARE NOT AVAILABLE

If an analytical Jacobian is not available, a numerical approximation to the Jacobian must be used by the quasi-Newton technique. A backward differences approximation for the Jacobian is

$$\mathbf{J_{ij}}(\mathbf{x}(k)) = \frac{\partial f_i(\mathbf{x}(k))}{\partial x_j(k)} \approx \frac{f_i(\mathbf{x}(k) + \delta x_j(k)) - f_i(\mathbf{x}(k))}{\delta x_j(k)} \tag{3.32}$$

where $\delta x_j(k)$ is a small perturbation in variable x_j at iteration k. A problem with this approach is that an n-variable problem requires $n + 1$ evaluations of the function vector at each iteration. There are other techniques that rely on infrequent function evaluations to update the Jacobian matrix.

3.6 MATLAB ROUTINES FOR SYSTEMS OF NONLINEAR ALGEBRAIC EQUATIONS

The MATLAB routine `fsolve` is used to solve sets of nonlinear algebraic equations, using a quasi-Newton method. The user must supply a routine to evaluate the function vector. It is optional to write a routine to evaluate the gradient of the function vector. As another option, the user can select the Levenburg-Marquardt method.

EXAMPLE 3.3 Reconsidered. Using MATLAB

The m-file used to implement Example 3.3 using `fsolve` is:

```
function f = nle(x)
f(1)= x(1)-4*x(1)*x(1)-x(1)*x(2);
f(2)= 2*x(2)-x(2)*x(2)+3*x(1)*x(2);
```

which is placed in an m-file called `nle.m`

The initial guess is entered

```
x0 = [1  1]';
```

and we obtain the solution by entering

```
x = fsolve('nle',x0)
```

which gives us the expected results

```
x = [0.2500    0.0000]'
```

Computationally faster results will be obtained if the analytical Jacobian is used.

$$\mathbf{J} = \begin{bmatrix} 1 - 8x_1 - x_2 & -x_1 \\ 3x_2 & 2 - 2x_2 + 3x_1 \end{bmatrix}$$

The following function file generates the analytical Jacobian for this problem.

```
function gf = gradnle(x)
gf(1,1)=1-8*x(1)-x(2);
gf(1,2)=-x(1);
gf(2,1)=3*x(2);
gf(2,2)=2-2*x(2)+3*x(1);
```

which we place in an m-file called `gradnle.m`. We can then solve this problem by entering

```
x0 = [1   1]';
options(5)=0;
x = fsolve('nle',x0,options,'gradnle')
```

The options vector can be used to select the Levenberg-Marquardt method by setting

```
options(5)=1;
```

SUMMARY

In this chapter we have presented a number of techniques to solve nonlinear algebraic equations. Each technique had a number of advantages and disadvantages—the approach that you use may depend on the problem at hand. If you have the option, it is a good idea to plot the function, $f(x)$, to see if the results agree with your numerical solution. This option may not be available with multivariable problems.

Notice that if a particular problem has multiple solutions, the actual solution obtained will depend on the initial guess. There are many actual chemical processes that have "multiple solutions," that is, there are several possible "steady-states" that the process may operate at. In practice, the steady-state obtained will depend on dynamic considerations, such as the way that the process is started up. These issues will be addressed when we discuss differential equation-based models.

Much research is being done in applied mathematics to develop numerical techniques that yield every one of the multiple solutions, without having to make many initial guesses. Among these is "homotopy continuation." These techniques are not well-developed and are beyond the scope of this text. For now, the student must be willing to

try a number of techniques, with a number of initial guesses, to find all of the solutions to a problem. It is recommended that you generally use commerically available routines for solving these problems.

The numerical techniques covered were

- Direct substitution
- Interval bisection
- Reguli falsi
- Newton's method

Newton's method (and some variants) is the most commonly used multivariable technique. The reader should be able to understand the notions of

- Jacobian
- Fixed or equilibrium points
- Convergence and stability
- Tolerance
- Iteration

The MATLAB routines that were introduced in this chapter are

`fsolve:`	Solves a system of nonlinear algebraic equations, using quasi-Newton and Levenberg-Marquardt based algorithms
`fzero:`	Solves for a single equation
`roots:`	Solves for the roots of a polynomial equation

A number of other numerical routines are availiable through the MATLAB NAG Toolbox.

FURTHER READING

More detailed treatments of numerical methods given in the textbooks by Davis, Finlayson, and Riggs:

Davis, M.E. (1984). *Numerical Methods and Modeling for Chemical Engineers.* New York: Wiley.

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

Riggs, J.B. (1994). *Numerical Techniques for Chemical Engineers,* 2nd ed. Lubbock: Texas Tech University Press.

Example numerical techniques are also presented by Felder and Rousseau:

Felder, R.M., & R.W. Rousseau. (1986). *Elementary Principles of Chemical Processes,* 2nd ed. New York: Wiley.

The following book is more of an advanced undergraduate/first-year graduate student text on numerical methods to solve chemical engineering problems. The emphasis is on FORTRAN subroutines to be used with the IMSL (FORTRAN-based) package.

Rameriz, W.F. (1989). *Computational Methods for Process Simulation.* Boston: Butterworths.

STUDENT EXERCISES

Single Variable Methods

1. Real gases do not normally behave as ideal gases except at low pressure or high temperature. A number of equations of state have been developed to account for the non-idealities (van der Waal's, Redlich-Kwong, Peng-Robinson, etc.). Consider the van der Waal's PV̂ T relationship

$$\left(P + \frac{a}{\hat{V}^2}\right)(\hat{V} - b) = RT$$

where P is pressure (absolute units), R is the ideal gas constant, T is temperature (absolute units), $\hat{V}$ is the molar volume and a and b are van der Waal's constants. The van der Waal's constants are often calculated from the critical conditions for a particular gas.

Assume that a, b, and R are given. We find that if P and T are given, there is an iterative solution required for $\hat{V}$.

a. How many solutions for $\hat{V}$ are there? Why? (Hint: Expand the PVT relationship to form a polynomial.)
b. Recall that direct substitution has the form $\hat{V}_{k+1} = g\hat{V}_k$). Write three different direct substitution formulations for this problem (call these I, II, and III).
c. What would be your first guess for $\hat{V}$ in this problem? Why?
d. Write the MATLAB m-files to solve for $\hat{V}$ using direct substitution, for each of the three formulations developed in **b.**

   Consider the following system: air at 50 atm and −100°C. The van der Waal's constants are (Felder and Rousseau, p. 201) $a = 1.33$ atm liter^2/gmol^2, $b = 0.0366$ liter/gmol, and $R = 0.08206$ liter atm/gmol K. Solve the following problems numerically.
e. Plot $\hat{V}$ as a function of iteration number for each of the direct substitution methods in **b.** Use 10 to 20 iterations. Also use the first guess that you calculated in **c.** Discuss the stability of each solution (think about the stability theorem).

f. Rearrange (1) to the form of $f(\hat{V}) = 0$ and plot $f(\hat{V})$ as a function of $\hat{V}$. How many solutions are there? Does this agree with your solution for **a**? Why or why not?

g. Use the MATLAB function `roots` to solve for the roots of the polynomial developed in **a**.

h. Write and use an m-file to solve for the equation in a using Newton's method. The numerical solution is considered converged when $\left|\dfrac{\hat{V}_k - \hat{V}_{k-1}}{\hat{V}_{k-1}}\right| < \varepsilon$. Let $\varepsilon = 0.0001$.

2. Consider the van der Waals relationship for a gas without the volume correction term ($b = 0$)

$$\left(P + \frac{a}{\hat{V}2}\right)\hat{V} = RT$$

a. How many solutions for $\hat{V}$ are there? Show the solutions analytically.

b. Which solution is correct?

3. A process furnace is heating 150 lbmol/hr of vapor-phase ammonia. The rate of heat addition to the furnace is 1.0×10^6 Btu/hr. The ammonia feedstream temperature is 550°R. Use Newton's method to find the temperature of the ammonia leaving the furnace. Assume ideal gas and use the following equation for heat capacity at constant pressure:

$$C_p = a + bT + cT^{-2}$$

$$a = 7.11\ \frac{\text{Btu}}{\text{lbmol °R}} \quad b = 3.33 \times 10^{-3}\ \frac{\text{Btu}}{\text{lbmol °R}^2} \quad c = -1.20 \times 10^5\ \frac{\text{Btu °R}}{\text{lbmol}}$$

and remember that $Q = \dot{n}\int_{T_{in}}^{T_{out}} C_p\, dT$

where $\dot{n}$ is the molar flowrate of gas and Q is the rate of heat addition to the gas per unit time. How did you determine a good first guess to use?

4. Consider Example 3.2 , $f(x) = -x^2 - x + 1 = 0$, with the direct substitution method formulated as $x = -x^2 + 1 = g(x)$, so that the iteration sequence is

$$x_{k+1} = g(x_k) = -x_k^2 + 1$$

Try several different initial conditions and show whether these converge, diverge, or oscillate between values. Discuss the stability of the two solutions $x^* = 0.618$ and $x^* = -1.618$, based on an analysis of $g'(x^*)$.

5. Show why the graphical Newton's method is equivalent to $x_{k+1} = x_k - f(x_k)/f'(x_k)$.

6. Develop an algorithm (sequence of steps) to solve an algebraic equation using interval halving (bisection).

7. Develop an algorithm (sequence of steps) to solve an algebraic equation using reguli falsi (false position). Compare and contrast this algorithm with Newton's method.

8. A component material balance around a chemical reactor yields the following steady-state equation

$$0 = \frac{F}{V} C_{in} - \frac{F}{V} C - kC^3$$

where $\frac{F}{V} = 0.1 \text{ min}^{-1}$, $C_{in} = 1.0 \frac{\text{lbmol}}{\text{ft}^3}$ and $k = 0.05 \frac{\text{ft}^6}{\text{lbmol}^2 \text{ min}}$

   a. How many steady-state solutions are there?
   b. Write two different direct substitution methods and assess the convergence of each.
   c. Perform two iterations of Newton's method using an initial guess of $C = 1.0$.

9. Consider the following direct substitution problem, which results from an energy balance problem

$$T_{k+1} = \left[\frac{15.04}{0.716 - 4.257 \times 10^{-6} T_k}\right]^2$$

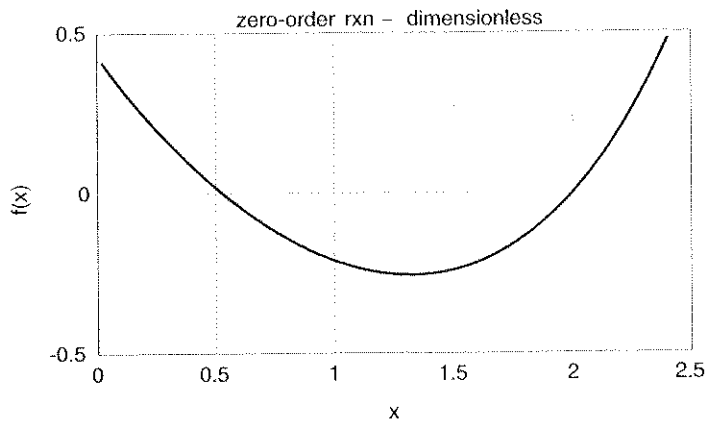
Will this method converge to the solution of $T = 443.571$? Why or why not?

10. Consider the following steady-state model for an exothermic zero-order reaction in a continuous stirred-tank reactor. The variable x is the dimensionless reactor temperature.

$$f(x) = 0.42204 \exp \frac{x}{1 + \frac{x}{20}} - 1.3x = 0$$

There are two solutions to this equation, as illustrated in the figure below. The solutions are $x = 0.55946$ and 2.00000.

   a. Formulate a direct substitution solution to this problem. Will your direct substitution technique converge to 0.55946? Will your direct substitution technique converge to $x = 2$? Use a rigorous mathematical argument in each case.
   b. For an initial guess of $x = 1.35$, would you expect Newton's method to have any trouble with this problem? Explain.
   c. What is the next guess from the reguli falsi technique if your first guess was $x = 1$ and your second guess was $x = 2.5$ (show this mathematically)?
   d. Write a function routine to solve this problem using the MATLAB function `fzero`. Also show the commands that you would give in the MATLAB command window to run `fzero`.



Plot of x versus $f(x)$ for problem 10.

11. A component material balance around a chemical reactor yields the following steady-state equation

$$0 = \frac{F}{V} C_{in} - \frac{F}{V} C - kC^{2.5}$$

where $\frac{F}{V} = 0.1\ \text{min}^{-1}$, $C_{in} = 1.0\ \frac{\text{lbmol}}{\text{ft}^3}$ and $k = 0.05\ \frac{\text{ft}^{4.5}}{\text{lbmol}^{1.5}\ \text{min}}$

a. Write two different direct substitution methods and assess the convergence of each to the steady-state concentration of 0.75354.

b. Perform two iterations of Newton's method using an initial guess of $C = 1.0$.

12. Consider the dimensionless equations for an exothermic CSTR (continuous stirred tank reactor) shown in a module in the final section of the textbook.

$$f_1(x_1,x_2) = -\phi x_1 \kappa(x_2) + (1 - x_1) \quad = 0$$

$$f_2(x_1,x_2) = \beta\phi x_1 \kappa(x_2) + (1 + \delta)x_2 = 0$$

where $\kappa(x_2) = \exp\left(\frac{x_2}{1 + x_2/\gamma}\right)$

Use $f_1(x_1,x_2)$ to solve for x_1 in terms of x_2 and substitute into $f_2(x_1,x_2)$ to obtain the single equation

$$f(x_2) = \beta\phi \frac{1}{(\phi\kappa(x_2) + 1)} \kappa(x_2) - (1 + \delta)x_2 = 0$$

Consider the parameter values $\beta = 8$, $\phi = 0.072$, $\gamma = 20$, $\delta = 0.3$. Determine the number of solutions to this problem (graphically). Find the x_1 and x_2 values for each solution, using the single variable Newton's method.

Multivariable Methods

13. Use the multivariable Newton's method to solve the following problem. Solve this as a two-variable problem. Do not reduce it to a single-variable problem via substitution.

$$f_1(\mathbf{x}) = 2x_1 - x_2 - 5 = 0$$

$$f_2(\mathbf{x}) = -x_1 - x_2 + 4 = 0$$

How many iterations does it take to converge to the solution? Explain this result conceptually.

14. A simple bioreactor model (assuming steady-state operation) is

$$0 = \left(\frac{\mu_{\max} x_2}{k_m + x_2 + k_1 x_2^2} - D\right)x_1$$

$$0 = (s_f - x_2)\,D - \left(\frac{x_1}{c}\right)\left(\frac{\mu_{\max} x_2}{k_m + x_2 + k_1 x_2^2}\right)$$

where $\mu_{\max} = 0.53$
$k_m = 0.12$
$k_1 = 0.4545$
$c = 0.4$
$s_f = 4.0$

x_1 is the biomass concentration (mass of cells) and x_2 is the substrate concentration (food source for the cells).

Find the steady-state values for x_1 and x_2 if $D = 0.3$ (There are three solutions).

a. Use `fsolve` and several initial guesses for the solution vector.
b. Perform a detailed analysis by hand (hint: the trivial solution $x_1 = 0$ and $x_2 = s_f$ should be easy to show.)

15. For the dimensionless CSTR problem (module 9 in Section V), use the MATLAB routine `fsolve` to find the solutions. Show the initial guesses and the solutions that `fsolve` converges to. Show your function routine as well as the calls to MATLAB.

$$f_1(x_1,x_2) = -\phi x_1 \kappa(x_2) + (1 - x_1) = 0$$

$$f_2(x_1,x_2) = \beta\phi x_1 \kappa(x_2) - (1 + \delta)x_2 = 0$$

where $\kappa(x_2) = \exp\left(\frac{x_2}{1 + x_2/\gamma}\right)$

and the following parameters are used

$$\beta = 8 \quad \phi = 0.072$$

$$\gamma = 20 \quad \delta = 0.3$$

APPENDIX

Stability of Numerial Solutions—Single Equations

If the iterates (x_k) from a numerical algorithm converge to a solution, we refer to that solution as being stable.

Definition 3.1

Let x^* represent the solution (fixed point) of $x^* = g(x^*)$, or $g(x^*) - x^* = 0$.

Theorem 3.1

x^* is a stable solution of $x^* = g(x^*)$, if $\left|\frac{\partial g}{\partial x}\right| < 1$ evaluated at x^*. x^* is an unstable solution of $x^* = g(x^*)$, if $\left|\frac{\partial g}{\partial x}\right| > 1$ evaluated at x^*.

If $\left|\frac{\partial g}{\partial x}\right| = 1$, then no conclusions can be drawn. For simplicity in notation, we generally use g' to represent $\left|\frac{\partial g}{\partial x}\right|$.

We continue with Example 3.2 to illustrate the numerical stability of direct substitution.

EXAMPLE 3.2 Continued. Stability of direct substitution

Consider the **Case 1** formulation,

$$x = g(x) = \sqrt{-x + 1}$$

which has the derivative

$$\frac{\partial g}{\partial x} = g' = \frac{-1}{2\sqrt{-x + 1}}.$$

If $\left|\frac{-1}{2\sqrt{-x^* + 1}}\right| < 1$, then x^* is a stable solution.

1. For $x^* = 0.618$, we find that $g'(x^*) = \frac{-1}{2\sqrt{-0.618 + 1}} = 0.8090 < 1$. So $x^* = 0.618$ is a stable solution, that is, an initial "guess" for x_0 close to 0.618 will converge to $x^* = 0.618$.

2. For $x^* = -1.618$, we find $|g'(x^*)| = \frac{-1}{2\sqrt{1.618 + 1}} = 0.3090 < 1$. So $x^* = -1.618$ should be a stable solution, that is, an initial "guess" for x_0 close to -1.618 should converge to $x^* = -1.618$.

Question: Why did we find previously that an initial guess close to −1.618 did not converge?

Observation: We must realize that the square root of a positive number has two values. For example, the square root of 1 can either be +1 or −1 (after all $(-1)^2 = 1$!).

You may be questioning the utility of a stability test for the direct substitution method, which requires that the solution be known to apply the test. Our purpose is mainly to show that the direct substitution method can be unstable. Newton's method guarantees stable solutions, if the "guess" is close to the solution.

Stability of Newton's Method for Single Equations

Here we use Theorem 3.1 to show that Newton's method is stable. We see that Newton's method can be written in the form of

$$x_{k+1} = g(x_k)$$

where

$$g(x_k) = x_k - \frac{f(x_k)}{f'(x_k)}$$

Then we can find that the derivative, $g'(x_k)$ is

$$g'(x_k) = 1 - \frac{f'(x_k)}{f'(x_k)} + \frac{f(x_k)\, f''(x_k)}{[f'(x_k)]'^2}$$

or

$$g'(x_k) = \frac{f(x_k)\, f''(x_k)}{[f'(x_k)]^2}$$

At the solution, x^*, we see that

$$g'(x^*) = \frac{f(x^*)\, f''(x^*)}{[f'(x^*)]^2}$$

And since $f(x^*) = 0$, we find that the stability constraint is satisfied

$$g'(x^*) = 0$$

as long as $f'(x^*) \neq 0$. This shows that Newton's method will converge to the solution, provided an initial guess close to the solution.

NUMERICAL INTEGRATION 4

Most chemical process models are nonlinear and rarely have analytical solutions. This chapter introduces numerical solution techniques for the integration of initial value ordinary differential equations. After studying this material, the student should be able to:

- Understand the difference between explicit and implicit Euler integration.
- Write MATLAB code to implement fixed step size Euler and Runge-Kutta techniques.
- Use the MATLAB `ode45` integration routine.

The major sections of this chapter are:

4.1 BACKGROUND

Thus far we have developed modeling equations (Chapter 2) and solved for the steady states (Chapter 3). One purpose of developing dynamic models is to be able to perform "what if" types of studies. For example, you may wish to determine how long a gas storage tank will take to reach a certain pressure if the outlet valve is closed. This requires in-

tegrating the differential equations from given initial conditions. If the dynamic equations are linear, then we can generally obtain analytical solutions; these techniques will be presented in Chapters 6 and 8 through 10. Even when systems are linear, we may wish to use numerical methods rather than analytical solutions.

At this point it is worth reviewing the difference between linear and nonlinear differential equations. An example of a linear ordinary differential equation is:

$$\frac{dx}{dt} = -x$$

since the rate of change of the dependent variable is a linear function of the dependent variable. An example of a nonlinear differential equation is:

$$\frac{dx}{dt} = -x^2$$

since the rate of change of the dependent variable is a nonlinear function of the dependent variable. Although this particulary nonlinear equation has an analytical solution, this will not normally be the case, particularly for sets of nonlinear equations. Notice that the following equation is linear:

$$\frac{dx}{dt} = -e^{-t}x$$

since the only nonlinearity is in the independent variable (t).

The purpose of this chapter is to introduce you to numerical techniques for integrating initial value ordinary differential equations. The first numerical integration technique that we will present is the *Euler* integration. In the next section we discuss two algorithms, the *explicit* and the *implicit* Euler methods.

4.2 EULER INTEGRATION

Consider a single variable ODE with the form

$$\frac{dx}{dt} = \dot{x} = f(x) \tag{4.1}$$

We consider two different approximations to the derivative. In Section 4.2.1 we consider a forward difference approximation, which leads to the explict Euler method. In Section 4.2.2 we consider a backwards differences approximation, which leads to the implicit Euler method.

4.2.1 Explicit Euler

If we use a forward difference approximation for the time derivative of (4.1), we find

$$\frac{dx}{dt} \approx \frac{x(k+1) - x(k)}{t(k+1) - t(k)} \tag{4.2}$$

where k represents the kth discrete time step of the integration. Now, assume that $f(x)$ is evaluated at $x(k)$. We will refer to this function as $f(x(k))$, and can write (4.1) and (4.2) as

$$\frac{x(k+1)-x(k)}{t(k+1)-t(k)} = f(x(k)) \tag{4.3}$$

Normally we will use a fixed increment of time, that is, $t(k+1) - t(k) = \Delta t$, where Δt is the *integration step size*. Then we write (4.3) as

$$\frac{x(k+1)-x(k)}{\Delta t} = f(x(k)) \tag{4.3}$$

Solving for $x(k+1)$

$$x(k+1) = x(k) + \Delta t\, f(x(k)) \quad \textit{Explicit Euler} \tag{4.4}$$

We can view (4.4) as a prediction of x at $k+1$ based on the value of x at k and the slope at k, as shown in Figure 4.1.

Equation (4.4) is the expression for the *explicit Euler* method for a single variable. The general statement for a multivariable problem is

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \Delta t\, \mathbf{f}(\mathbf{x}(k)) \tag{4.5}$$

Where $\mathbf{x}(k)$ is a vector of state variable values at time step k and $\mathbf{f}(\mathbf{x}(k))$ is a vector of functions evaluated at step k. Equations (4.4) and (4.5) are *explicit* because the state variable value at time step $k+1$ is only a function of the variable values at step k. This method is straightforward and easy to program on either a handheld calculator or a computer. A major disadvantage is that a small step size must be used for accuracy. However, if too small of a step size is used, then numerical truncation problems may result. Explict Euler is not often used in practice, but is covered here for illustrative purposes.

4.2.2 Implicit Euler

This method uses a backwards difference approximation for the derivative in (4.1). The function (or vector of functions) is evaluated at time step $k+1$ rather than time step k:

$$\frac{x(k+1)-x(k)}{\Delta t} = f(x(k+1)) \tag{4.6}$$

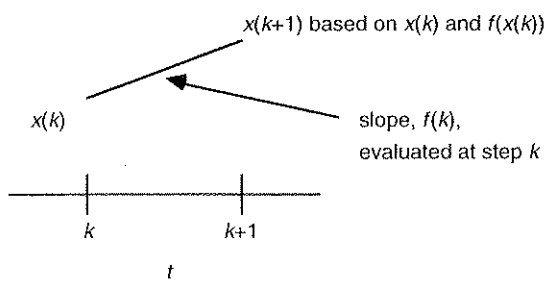


FIGURE 4.1 Pictorial representation of the explicit Euler method.

which can be written

$$x(k+1) = x(k) + \Delta t f(x(k+1)) \quad \textit{Implicit Euler} \tag{4.7}$$

Equation (4.7) is *implicit* because the value $x(k+1)$ must be known in order to solve for $x(k+1)$. What this generally requires is a nonlinear algebraic solution technique, such as Newton's method. For a linear system, equation (4.7) can be explicitly solved to obtain the form

$$x(k+1) = g(x(k))$$

The following section compares the explicit and implicit methods for a single linear ordinary differential equation. In particular, we compare how the integration step size (Δt) affects the stability of each method.

4.2.3 Numerical Stability of Explicit and Implict Euler Methods

Consider the tank height problem covered in Example 2.1. There was no inflow, and the outlet flowrate was assumed to be linearly related to height, which gave us the following equation:

$$\frac{dx}{dt} = -\frac{1}{\tau}x = \lambda x = f(x) \tag{4.8}$$

where $\tau = A/\beta$, $\lambda = -1/\tau$ and the state variable x is the tank height. Since the variables are separable, the reader should show that the *analytical solution* is:

$$x(t) = x(0)e^{-t/\tau} = x(0)e^{\lambda t} \tag{4.9}$$

Next, we compare this analytical solution with the explicit and implicit Euler solutions.

EXPLICIT EULER

The function value at step k is:

$$f(x(k)) = -\frac{1}{\tau}x(k)$$

and the state variable value at the next time step is:

$$x(k+1) = x(k) + -\frac{\Delta t}{\tau}x(k) = \left(1 - \frac{\Delta t}{\tau}\right)x(k) \tag{4.10}$$

IMPLICIT EULER

The implicit Euler method evaluates the function at $k+1$ rather than k:

$$f(x(k+1)) = -\frac{1}{\tau}x(k+1)$$

and the state variable at step $k+1$ is:

$$x(k+1) = x(k) + -\frac{\Delta t}{\tau} x(k+1) \tag{4.11}$$

Notice that, since this is a linear problem, a nonlinear algebraic equation solver is not needed for (4.11). We can rewrite (4.11) as

$$x(k+1) = \frac{1}{1+\frac{\Delta t}{\tau}} x(k) \tag{4.12}$$

In the next section we compare the numerical stability of the explicit and implicit Euler methods.

NUMERICAL STABILITY

The *explicit* Euler solution, written in terms of the initial condition, is (from (4.10)):

$$x(k+1) = \left(1 - \frac{\Delta t}{\tau}\right)^{k+1} x(0)$$

which will be stable if $|1 - \Delta t/\tau| < 1$. This is the same result if we use the representation:

$$x(k+1) = g(x(k)) = \left(1 - \frac{\Delta t}{\tau}\right) x(k)$$

and the stability requirement that $|g'| < 1$. The explicit Euler method is then stable if:

$$-1 < 1 - \frac{\Delta t}{\tau} < 1$$

and will oscillate for:

$$-1 < 1 - \frac{\Delta t}{\tau} < 0$$

These criteria lead to the explicit Euler stability condition of:

$$0 < \Delta t < 2\tau$$

while the solution will have a stable, oscillatory solution for:

$$\tau < \Delta t < 2\tau$$

and a stable, monotonic solution for:

$$0 < \Delta t < \tau$$

The *implicit* Euler solution, written in terms of the initial condition, is (from (4.12)):

$$x(k+1) = \left(\frac{1}{1+\frac{\Delta t}{\tau}}\right)^{k+1} x(0)$$

which will be stable if $|1/1+\Delta t,\tau| < 1$. This is the same result if we use the representation

$$x(k+1) = g(x(k)) = \left(\frac{1}{1 + \frac{\Delta t}{\tau}} \right) x(k)$$

and the stability requirement that $|g' \neq < 1$. Notice that the implicit Euler method is stable for any value of Δt (as long as the sign of Δt is correct) and will not oscillate.

EXAMPLE 4.1 Numerical Comparison of Explicit and Implicit Euler

Let $x(0) = 4$, $\tau = 5$, and $\Delta t = 1$. Table 4.1 compares the exact solution (4.9) with the explicit Euler (4.10) and implicit Euler (4.12) methods.

TABLE 4.1 Linear First Order Example ($\tau = 5$, $\Delta t = 1$)

t	x, exact	x, Explicit Euler	error	x, Implicit Euler	error
0	4.0000	4.0000		4.0000	
1	3.2749	3.2000	−2.3%	3.3333	1.8%
2	2.6813	2.5600	−4.5%	2.7778	3.6%
3	2.1952	2.0480	−6.7%	2.3148	5.4%
4	1.7979	1.6384	−8.9%	1.9290	7.3%
5	1.4715	1.3107	−10.9%	1.6075	9.2%

The results shown in Table 4.1 are illustrated graphically by the curves in Figure 4.2.

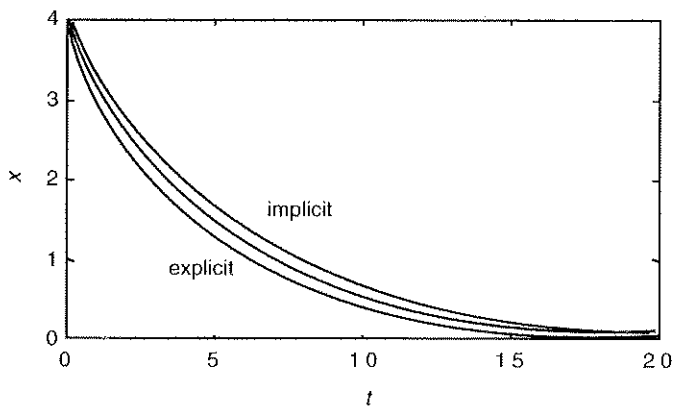


FIGURE 4.2 Comparison of the exact solution with the explicit and implicit Euler for $\Delta t = 1$.

Larger Integration Step Size. We have seen the well-behaved response for $\Delta t = 1$ (which is $\tau/5$). Consider now a larger Δt. We can see from (4.10) that the explicit Euler method predicts $x = 0$ for all time after time 0, if $\Delta t = \tau$. Indeed, the explicit Euler solution is oscillatory for $\Delta t > \tau$, for this process. For example, let $\Delta t = 6$ for this problem. The results are shown in Table 4.2. These results are illustrated more graphically by the curves plotted in Figure 4.3. The implicit Euler technique has monotonic behavior and more closely approximates the exact solution. We see that the implicit Euler method can tolerate a larger integration step size than the explicit Euler technique.

TABLE 4.2 Linear First Order Example ($\tau = 5$, $\Delta t = 6$)

t	x, exact	x, Explicit Euler	x, Implicit Euler
0	4.0000	4.0000	4.0000
6	1.2048	−0.8000	1.8182
12	0.3629	0.1600	0.8264
18	0.1093	−0.0320	0.3757
24	0.0329	0.0064	0.1708

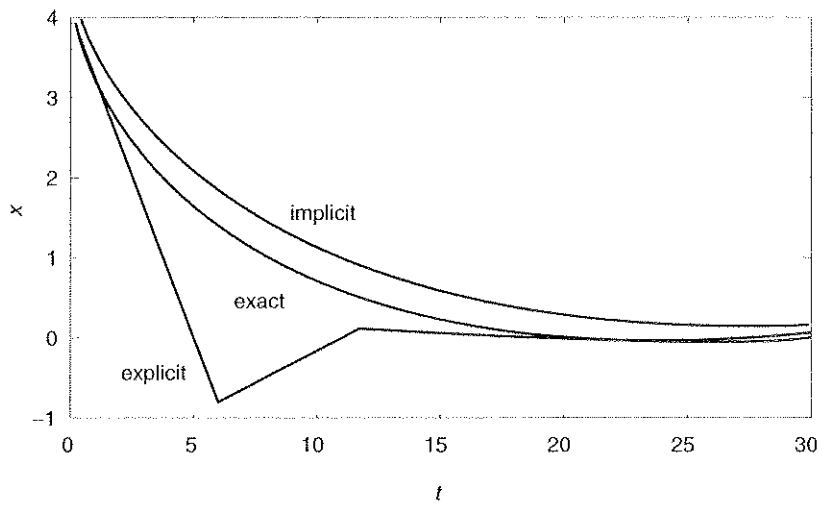


FIGURE 4.3 Comparison of exact solution with implicit and explicit Euler for $\Delta t = 6$.

We have seen that there is a limit to how large a step size can be tolerated by the explicit Euler method before it goes unstable, while the Implicit Euler method remains stable for any step size. This is true for a simple linear ordinary differential equation. The following example illustrates an important issue when solving nonlinear equations. That is, the implicit Euler method requires an iterative solution at each time step.

EXAMPLE 4.2 Explicit and Implicit Euler—Nonlinear System

Consider a surge tank with an outlet flowrate that depends on the square root of the height of liquid in the tank. When there is no inlet flow, the model has the following form

$$\frac{dx}{dt} = -a\sqrt{x} \tag{4.13a}$$

where x is the tank height and $a = \beta/A$ (= flow coefficient/cross-sectional area).

Analytical solution (exact). The analytical solution is

$$x(t) = [\sqrt{x(0)} - at/2]^2 \tag{4.13b}$$

Next, we compare this solution with the explicit and implicit Euler solutions.

Explicit Euler. The explicit Euler method yields the following equation

$$x(k+1) = x(k) - \Delta t\, a\sqrt{x(k)} \tag{4.14}$$

For the numerical example of $a = 0.8$, curves for 3 different Δts are shown in Figure 4.4.

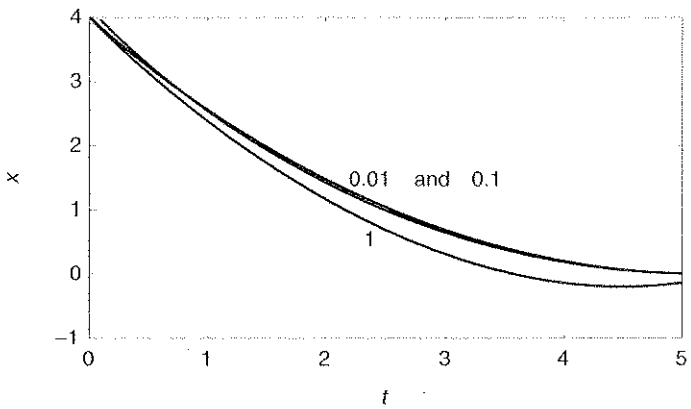


FIGURE 4.4 Explicit Euler solution. $\Delta t = 0.01$, 0.1, and 1.0. The 0.01 and 0.1 step sizes yield virtually identical results, while there is a significant error in a step size of 1.0.

Implicit Euler. The implicit Euler method yields the following equation

$$x(k+1) = x(k) - \Delta t\, a\sqrt{x(k+1)} \tag{4.15}$$

Recall that when we were dealing with a linear equation we were able to rewrite the implicit Euler equation to yield an explicit calculation of the state variable at the next time step. Here we see that this is impossible for a nonlinear equation. Rewriting (4.15),

$$x(k+1) + \Delta t\, a\sqrt{x(k+1)} - x(k) = 0 \tag{4.16}$$

we see that (4.16) requires an iterative solution. That is, at time step $k+1$ we must use a numerical method that will solve a nonlinear algebraic equation. We know from Chapter 3 that a num-

ber of techniques, including Newton's method, can be used. To illustrate clearly one approach that we can take, let y represent the value of x at step $k + 1$ for which we desire to find the solution. We can rewrite (4.16) as

$$y + b\sqrt{y} - c = 0 \tag{4.17}$$

where $y = x(k + 1)$, $b = \Delta t\, a$, and $c = x(k)$. If Newton's method is used, we can write:

$$g(y(i)) = y(i) + b\sqrt{y(i)} - c \tag{4.18}$$

where (i) is an index to indicate the ith iteration of Newton's method.

$$y(i + 1) = y(i) - \frac{g(y(i))}{g'(y(i))} \tag{4.19}$$

where

$$g'(y(i)) = 1 + \frac{b}{2\sqrt{y(i)}} \tag{4.20}$$

We can write (4.19) as (from (4.18), (4.19), and (4.20))

$$y(i + 1) = y(i) - \left[\frac{y(i)^{1.5} + b\, y(i) - c\sqrt{y(i)}}{\sqrt{y(i)} + \dfrac{b}{2}} \right] \tag{4.21}$$

Equation (4.21) is then iterated to convergence.

Summarizing, at step k+1 of an integration, we must iteratively solve for the value of x at k+1. That is, (4.21) is iteratively solved to convergence, in order to find $x(k + 1)$ in (4.16).

COMMENT ON IMPLICIT INTEGRATION TECHNIQUES

We have seen that the implicit Euler method is more accurate and stable than the explicit Euler method. We also noted that, for nonlinear systems, a nonlinear equation must be iteratively solved at each time step. The implicit Euler method allows a much larger time step, but some of the computational savings much be sacrificed in the interative solution of the nonlinear algebraic equation at each time step. There are a number of more advanced implicit methods that are used in a number of commercially available integration codes. In this text we emphasize *explicit* techniques, which are used by the MATLAB routines `ode23` and `ode45`. SIMULINK has choices of some implicit integration routines.

An explicit technique that is more accurate than the explicit Euler technique is known as the Runge-Kutta method and is shown in the next section.

4.3 RUNGE-KUTTA INTEGRATION

This technique is an extension of the Euler method. In the Euler method, the derivative at time step k was used to predict the solution at step k+1. *Runge-Kutta* methods use the Euler technique to predict the x value at intermediate steps, then use averages of the slopes at intemediate steps for the full prediction from the beginning of the time step.

4.3.1 Second-Order Runge-Kutta

The first Runge-Kutta approach that we discuss is the second-order Runge-Kutta method, which is also known as the midpoint Euler method, for reasons that will become clear. The Euler technique is first used to predict the state value at $\Delta t/2$. The derivative is evaluated at this midpoint, and used to predict the value of x at the end of the step, Δt, as shown in Figure 4.5.

Let m_1 represent the slope at the initial point and m_2 represent the slope (dx/dt) at the midpoint:

$$m_1 = f(t(k), x()) \tag{4.22}$$

$$m_2 = f\left(t(k) + \frac{\Delta t}{2}, x(k) + \frac{\Delta t}{2} m_1\right) \tag{4.23}$$

$$x(k+1) = x(k) + m_2 \Delta t \tag{4.24}$$

For autonomous systems, the derivative functions are not explicitly functions of time, so (4.22) and (4.23) can be written:

$$m_1 = f(x(k)) \tag{4.22a}$$

$$m_2 = f\left(x(k) + \frac{\Delta t}{2} m_1\right) \tag{4.23a}$$

or

$$m_2 = f\left(x(k) + \frac{\Delta t}{2} f(x(k))\right) \tag{4.23b}$$

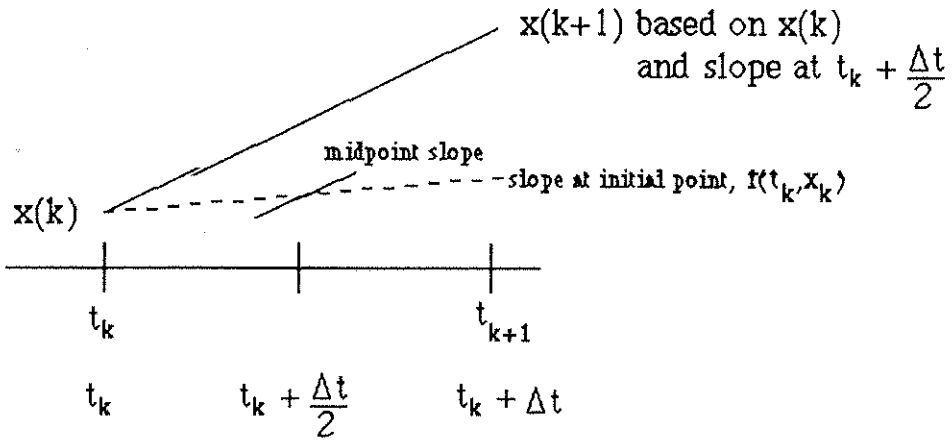


FIGURE 4.5 Pictorial representation of second-order Runge-Kutta (midpoint Euler) technique.

Equation (4.24) can now be written:

$$x(k+1) = x(k) + \Delta t\, f\left(f\left(x(k) + \frac{\Delta t}{2} f(x(k))\right)\right) \tag{4.24a}$$

which is of the form:

$$x(k+1) = g(x(k))$$

It should be noted that the second-order Runge-Kutta is accurate to the order of Δt^2, while the explicit Euler is accurate to the order of Δt.

EXAMPLE 4.3 Second-order Runge-Kutta (Midpoint Euler) Technique

Consider again the first-order process:

$$\frac{dx}{dt} = f(x) = -\frac{1}{\tau} x \tag{4.8}$$

$$m_1 = f(x(k)) = \left(-\frac{1}{\tau} x(k)\right) = \frac{-1}{\tau} x(k) \tag{4.25}$$

$$m_2 = f\left(x(k) + \frac{1}{2} m_1 \Delta t\right) = \frac{-1}{\tau}\left[\left(x(k) - \frac{\Delta t}{2\tau} x(k)\right)\right] \tag{4.26}$$

For $\Delta t = 1$, $\tau = 5$, and $x(0) = 4.0$:

From (4.25) $$m_1 = \frac{-1}{5} x(0) = \frac{-4}{5}$$

From (4.26) $$m_2 = \frac{-1}{5}\left(1 - \frac{1}{10}\right) x(0) = \frac{-1}{5}(1 - 0.1)(4) = -\frac{3.6}{5}$$

From (4.24) $$x(1) = x(0) + m_2 \Delta t$$

$$x(1) = 4.0 - 0.72 = 3.2800$$

Compare this with the analytical result of $4.0\, e^{-1/5} = 3.2749$

Notice that the error is 0.16%. Contrast this with an error of −2.3% from Table 4.1 for the Euler method. For the same step size, Runge-Kutta techniques are more accurate than the standard Explicit Euler technique.

Thus far, we have used single variable examples. The next example is for a two-state variable system.

EXAMPLE 4.4 Two-state Variable System, Second-order Runge-Kutta Method

Consider two interacting tanks in series, shown in Figure 4.6, with outlet flowrates that are a function of the square root of tank height. Notice that the flow from tank 1 is a function of $\sqrt{h_1 - h_2}$, while the flowrate out of tank 2 is a function of $\sqrt{h_2}$.

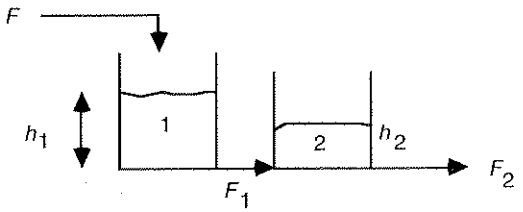


FIGURE 4.6 Interacting tanks.

The following modeling equations describe this system

$$\begin{bmatrix} \frac{dh_1}{dt} \\ \frac{dh_2}{dt} \end{bmatrix} = \begin{bmatrix} f_1(h_1,h_2,F) \\ f_2(h_1,h_2,F) \end{bmatrix} = \begin{bmatrix} \frac{F}{A_1} - \frac{\beta_1}{A_1}\sqrt{h_1 - h_2} \\ \frac{\beta_1}{A_2}\sqrt{h_1 - h_2} - \frac{\beta_2}{A_2}\sqrt{h_2} \end{bmatrix} \tag{4.27}$$

For the following parameter values:

$$\beta_1 = 2.5\,\frac{\text{ft}^{2.5}}{\text{min}} \quad \beta_2 = \frac{5}{\sqrt{6}}\,\frac{\text{ft}^{2.5}}{\text{min}} \quad A_1 = 5\ \text{ft}^2 \quad A_2 = 10\ \text{ft}^2$$

and the input: $F = 5$ ft^3/min
the steady-state height values are:

$$h_{1s} = 10 \quad h_{2s} = 6$$

Numerically, we can write (4.27) as:

$$\begin{bmatrix} \frac{dh_1}{dt} \\ \frac{dh_2}{dt} \end{bmatrix} = \begin{bmatrix} f_1(h_1,h_2) \\ f_2(h_1,h_2) \end{bmatrix} = \begin{bmatrix} 1 - 0.5\sqrt{h_1 - h_2} \\ 0.25\sqrt{h_1 - h_2} - \frac{1}{2\sqrt{6}}\sqrt{h_2} \end{bmatrix} \tag{4.28}$$

Since this system is autonomous (no explicit dependence on time), we can leave t out of the arguments:

$$m_1 = f(h(k)) = \begin{bmatrix} f_1(h_1(k), h_2(k)) \\ f_2(h_1(k), h_2(k)) \end{bmatrix}$$

$$m_2 = \begin{bmatrix} f_1(h_1(k) + \frac{\Delta t}{2} m_{11}, h_2(k) + \frac{\Delta t}{2} m_{21}) \\ f_2(h_1(k) + \frac{\Delta t}{2} m_{11}, h_2(k) + \frac{\Delta t}{2} m_{21}) \end{bmatrix}$$

$$h(k+1) = \begin{bmatrix} h_1(k+1) \\ h_2(k+1) \end{bmatrix} = \begin{bmatrix} h_1(k) \\ h_2(k) \end{bmatrix} + m_2 \Delta t$$

Let the initial conditions be $h_1(0) = 12$ ft and $h_2(0) = 7$ ft. Also, let $\Delta t = 0.2$ minutes. For $k = 0$, we find

$$m_1 = \begin{bmatrix} 1 - 0.5\sqrt{12-7} \\ 0.25\sqrt{12-7} - \dfrac{1}{2\sqrt{6}}\sqrt{7} \end{bmatrix} = \begin{bmatrix} -0.118034 \\ 0.018955 \end{bmatrix}$$

so

$$\begin{bmatrix} h_1(0) \\ h_2(0) \end{bmatrix} + \frac{\Delta t}{2} m_1 = \begin{bmatrix} 12 + \dfrac{0.2}{2}(-0.118034) \\ 7 + \dfrac{0.2}{2}(0.018955) \end{bmatrix} = \begin{bmatrix} 11.988197 \\ 7.001896 \end{bmatrix}$$

$$h_1(0) + \frac{\Delta t}{2} m_{11} = 12 + \frac{0.2}{2}(-0.118034) = 11.988197 \text{ ft}$$

$$h_2(0) + \frac{\Delta t}{2} m_{21} = 7 + \frac{0.2}{2}(0.018955) = 7.001896 \text{ ft}$$

$$m_{12} = f_1\left(h_1(0) + \frac{\Delta t}{2} m_{11}, h_2(0) + \frac{\Delta t}{2} m_{12}\right) = f_1(11.988197, 7.001896)$$

$$= 1 - 0.5\sqrt{11.988197 - 7.001896} = -0.116501$$

$$m_{22} = f_2\left(h_1(0) + \frac{\Delta t}{2} m_{11}, h_2(0) + \frac{\Delta t}{2} m_{12}\right) = f_2(11.988197, 7.001896)$$

$$= 0.25\sqrt{11.988197 - 7.001896} - \frac{1}{2\sqrt{6}}\sqrt{7.001896} = 0.018116$$

$$h_1(1) = h_1(0) + m_{21}\Delta t = 12 + -0.116501\,(0.2) = 11.976700 \text{ ft}$$

$$h_2(1) = h_2(0) + m_{22}\Delta t = 7 + 0.018116\,(0.2) = 7.003623 \text{ ft}$$

and we can continue for the next time step, $k = 1$. A plot of the response of this system is shown in Figure 4.7. The response of h_2 actually increases slightly before decreasing—this is missed because of the scaling.

Notice that when h_1 is greater than h_2, the flow is from tank 1 to tank 2; while when h_1 is less than h_2, the flow is from tank 2 to tank 1 (although this cannot occur at steady-state). Since we have assigned a positive value to F_1 when the flow is from tank 1 to tank 2, then a negative value of F_1 indicates the opposite flow. Care must be taken when solving this problem numerically, so that the square root of a negative number is not taken. For this purpose, the sign function is used

$$F_1 = \beta_1 \text{ sign}(h_1 - h_2) \sqrt{|h_1 - h_2|}$$

where sign $(h_1 - h_2) = 1$ if $h_1 > h_2$ and sign $(h_1 - h_2) = -1$ if $h_2 > h_1$.

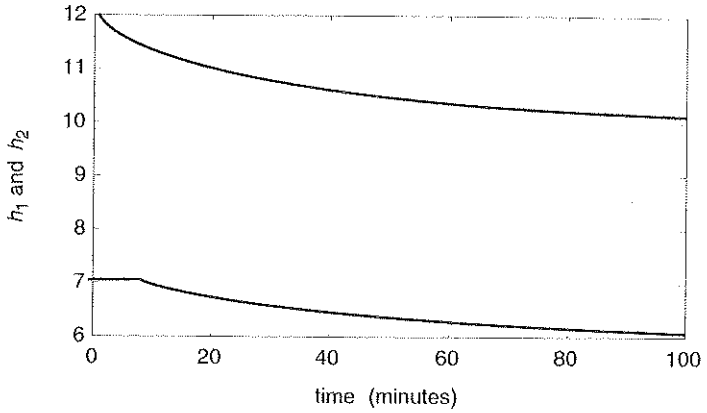


FIGURE 4.7 Transient response of the interacting tank example, using second-order Runge-Kutta.

The idea behind the second-order Runge-Kutta can be extended to higher-orders. The most commonly used method is the *fourth-order Runge-Kutta* method as outlined in the next section.

4.3.2 Fourth-Order Runge-Kutta

Using this method, the approximations are more accurate than explicit Euler or second-order Runge-Kutta. The idea is to use the initial slope (m_1) to generate a first guess for the state variable at the midpoint of the integration interval. This first guess is then used to find the slope at the midpoint (m_2). A "corrected" midpoint slope (m_3) is then found by using m_2. A final slope (m_4) is found at the end of a step using m_3. A weighted average of these slopes is then used for the integration. The algorithm is

$$m_1 = f(t(k), x(k)) \tag{4.29}$$

$$m_2 = f\left(t(k) + \frac{1}{2}\Delta t, x(k) + \frac{1}{2} m_1 \Delta t\right) \tag{4.30}$$

$$m_3 = f\left(t(k) + \frac{1}{2}\Delta t, x(k) + \frac{1}{2} m_2 \Delta t\right) \tag{4.31}$$

$$m_4 = f(t(k) + \Delta t, x(k) + m_3 \Delta t) \tag{4.32}$$

$$x(k+1) = x(k) + \left[\frac{m_1}{6} + \frac{m_2}{3} + \frac{m_3}{3} + \frac{m_4}{6}\right]\Delta t \tag{4.33}$$

To become more familiar with integration techniques, you should solve some of your initial problems using the explicit Euler method. Make certain that your step size is small enough so that the errors do not build up too rapidly. As you find a need for more accuracy, you should then use the fourth-order Runge-Kutta method. In Section 4.4 we introduce the MATLAB routines that are available for numerical integration.

SELECTION OF INTEGRATION STEP SIZE

Generally, integration step size must be "small" for Euler, can be larger for second-order Runge-Kutta (as far as accuracy is concerned), and can be still larger for fourth-order Runge-Kutta. Particularly for Euler, step sizes that are too large can be unstable or inaccurate. Step sizes that are too small may waste computer time or have numerical truncation errors since the state variables may not change much from step to step. If the student uses a fixed step size, then it is generally a good idea to try larger and smaller step sizes to see if the results change significantly. Generally, you will want to use as large a step size as possible. A particular challenge is from "stiff" systems (time constants that span a wide range), where a commerical code specifically for stiff systems should be used. One well-known implicit method for stiff systems is Gear's method, which is available in SIMULINK. Implicit methods will only work well for systems that are continous. If discontinuous systems are simulated (for example, step changes at certain times), then Runge-Kutta methods should be used.

Most commerical integration packages use a variable step size. The integration step size is automatically chosen and varied from step-to-step to assure accuracy while minimizing computation time. The integration routines in MATLAB use a variable integration step size.

4.4 MATLAB INTEGRATION ROUTINES

The primary purpose of the previous sections in this chapter was to review simple numerical techniques for integrating initial value ordinary differential equations. We have illustrated the techniques with some simple numerical examples, implemented as m-files in MATLAB. In practice, we do not recommend that you write your own integration routines. You will spend much time debugging these routines and they will generally not be as powerful as existing academic or commerical integration routines. Your goal should be to provide the correct formulation of the model, specifying the correct initial conditions and parameters. You should generally use a well-documented, commerical or public domain integration code to implement your simulation.

MATLAB has several routines for numerical integration; two are `ode23` and `ode45`. `ode23` uses second-order and `ode45` uses fourth-order Runge-Kutta integration. Both routines use a *variable integration step size* (Δt is not constant). The integration

step size is adjusted by the routine to provide the necessary accuracy, without taking too much computation time.

4.4.1 ode23 and ode45

To use ode23 or ode45, the reader must first generate an m-file to evaluate the state variable derivatives. Then the student gives the command:

```
[t,x]= ode45('xprime',[t0,tf]x0)
```

where

xprime	is a string variable containing the name of the m-file for the derivatives
t0	is the initial time
tf	is the final time
x0	is the initial condition vector for the state variables (usually a column vector)

The arrays that are returned are

t	a (column) vector of time
x	an array of state variables as a function of time (column 1 is state 1, etc.)

For example, if the time vector has 50 elements, and there are three state variables, then the state variable vector has the 50 rows and three columns. After the integration is performed, if the student wishes to plot all three variables as a function of time, she/he simply types

```
plot(t,x)
```

If you only want to plot the second state variable, then the command `plot(t,x(:,2))` is given.

EXAMPLE 4.4 Revisited Solution Using MATLAB Routine ode45

First, the following file titled twotnk.m was generated:

```
function hdot = twotnk(t,h);

const=(1/(2*sqrt(6)));

hdot(1) = 1-0.5*sqrt(h(1)-h(2));

  hdot(2) = 0.25*sqrt(h(1)-h(2))-const*sqrt(h(2));
```

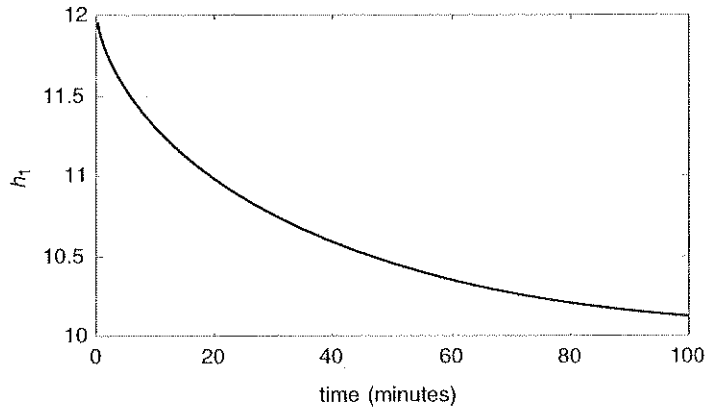
Then, the following command is entered in the MATLAB command window:

```
[t,h]=ode45('twotnk',[0100],[12 7]');
```

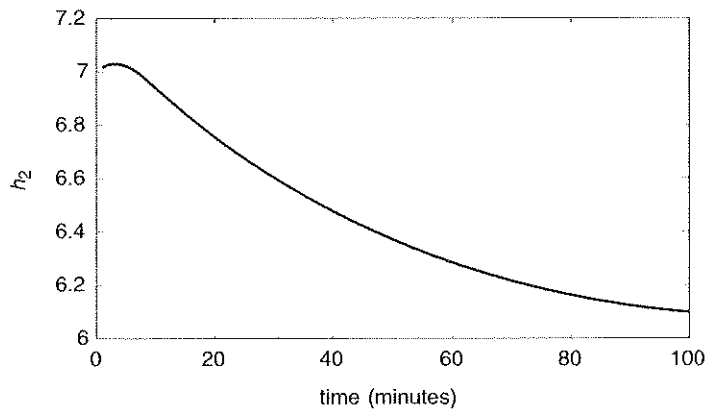
Notice that we are generating two arrays, t and h, and using `ode45`. The function file is named `twotnk.m`. The initial time is $t0 = 0$ and the final time is $tf = 100$. The initial condition is $h0 = [12\ 7]'$. At the MATLAB prompt (») the following commands were given:

```
plot(t,h(:,1))
plot(t,h(:,2))
```

The transient responses are shown in Figure 4.8.



a. Height of Tank 1



a. Height of Tank 2

FIGURE 4.8 Transient response curves for interacting tank example.

Notice the tremendous reduction in effort when compared with generating your own Runge-Kutta code.

Often it is desirable to know the state variable values at a particular time or at fixed time steps. A variable step size algorithm yields variable values that are not at a fixed step sizes. One has two options. If the variable step size is smaller than that of the variable step, then we could reduce the step size. The major disadvantage is that computation time will increase.

The best option (and that recommended by MATLAB) is to use a spline fit to interpolate or extrapolate the values to desired points. The routine used is `interp1`.

SUMMARY

It is important for the student to understand the Euler, as well as the second and fourth order Runge-Kutta integration techniques. When using your own fixed step size integration code, be careful with the selection of Δt. In practice, it is preferable to use a commercial integration code, which automatically selects the integration step size.

The MATLAB routines used were

`ODE23`: Variable step size, second-order Runge-Kutta
`ODE45`: Variable step size, fourth-order Runge-Kutta

FURTHER READING

A nice treatment of numerical integration is provided by:

Parker, T.S., & L.O. Chua. (1989). *Practical Numerical Algorithms for Chaotic Systems.* New York: Springer-Verlag.

A treatment of integration techniques with chemical engineering applications is presented by Davis.

Davis, M. E. (1984). *Numerical Methods and Modeling for Chemical Engineers.* New York: Wiley.

The following book is more of an advanced undergraduate/first-year graduate student text on numerical methods to solve chemical engineering problems. The emphasis is on FORTRAN subroutines to be used with the IMSL (FORTRAN-based) package.

Rameriz, W.F. (1989). *Computational Methods for Process Simulation.* Boston: Butterworths.

STUDENT EXERCISES

1. Consider the scaled predator-prey equations.

$$\frac{dy_1}{dt} = \alpha(1 - y_2)\, y_1$$

$$\frac{dy_2}{dt} = -\beta(1 - y_1)y_2$$

The parameters are $\alpha = \beta = 1.0$ and the initial conditions are $y_1(0) = 1.5$ and $y_2(0) = 0.75$. The time unit is days.

a. Solve these equtions using explicit Euler integration. Compare various integration step sizes. What Δt do you recommend? In addition to transient responses (t versus y_1 and y_2), also plot "phase-plane" plots (y_1 versus y_2).

b. Solve these equations using the MATLAB integration routine `ode45`. Compare the transient response curves with the Euler results.

c. How do the initial conditions effect the response of y_1 and y_2? Please elaborate.

2. Consider a CSTR with a second-order reaction. Assume that the rate of reaction (per unit volume) is proportional to the square of the concentration of the reacting component. Assuming constant volume and constant density, show that the modeling equation is:

$$\frac{dC}{dt} = \frac{F}{V} C_i - \frac{F}{V} C - k_2 C^2$$

Use the following parameters:

$$\frac{V}{F} = 5 \text{ min} \quad k_2 = 0.32 \text{ ft}^3 \text{ lbmol}^{-1} \text{ min}^{-1}$$

and a steady-state inlet concentration of

$$C_{is} = 1.25 \text{ lbmol ft}^{-3}$$

Calculate the steady-state concentration of $C_s = 0.625$ lbmol ft^{-3}.

Assume that a step change in the inlet concentration occurs at $t = 0$. That is, C_i changes from 1.25 lbmol ft^{-3} to 1.75 lbmol ft^{-3} at $t = 0$ minutes. Use `ode45` to simulate how the outlet concentration changes as a function of time.

3. Analyze the stability of the fourth order Runge-Kutta method for the classical first-order process.

$$\frac{dx}{dt} = -x$$

What is the largest integration step size before the numerical solution becomes unstable?

4. A gas surge drum has two components (hydrogen and methane) in the feedstream. Let y_i and y represent the mole fraction of methane in the feedstream and drum, respectively. Find *dP/dt* and *dy/dt* if the inlet and outlet flowrates can vary. Also assume that the inlet concentration can vary. Assume the ideal gas law for the effect of pressure and composition on density.

   Assume that the gas drum volume is 100 liters. The temperature of the drum is 31.5 deg C (304.65 K).

   At steady-state the drum pressure is 5 atm, the molar flowrate in and out is 2 gmol/min and the concentration is 25% methane, 75% hydrogen.

   Use Euler integration and `ode45` to solve the following problems. Discuss the effect of integration step size when using Euler integration. In all cases, you are initially at steady-state.

   a. Assume that the molar flowrates remain constant, but the inlet methane concentration is changed to 50%. Find how pressure and composition change with time.

   b. Assume that the molar flowrate out of the drum is proportional to the difference in pressure between the drum and the outlet header, which is at 2 atm pressure. Perform a step change in inlet concentration to 50% methane, simultaneously with a step change in inlet flowrate to 3 gmol/min.

   c. Assume that the MASS flowrate out of the drum is proportional to the square root of the difference in pressure between the drum and the outlet header (which is at 2 atm pressure). Again, perform a step change in inlet concentration to 50% methane, simultaneously with a step change in inlet flowrate to 3 gmol/min.

   d. Assume that the MASS flowrates in and out are proportional to the square root of the pressure drops. Assume that the steady-state inlet gas header is at 5 atm. Perform a step change in inlet concentration to 50% methane, simultaneously with a step change in inlet pressure to 6 atm.

5. Pharmacokinetics is the study of how drugs infused to the body are distributed to other parts of the body. The concept of a compartmental model is often used, where it is assumed that the drug is injected into compartment 1. Some of the drug is eliminated (reacted) in compartment 1, and some of it diffuses into compartment 2 (the rest accumulates in compartment 1). Similarly, some of the drug that diffuses into compartment 2 diffuses back into compartment 1, while some is eliminated by reaction and the rest accumulates in compartment 2. The rates of diffusion and reaction are directly proportional to the concentration of drug in the compartment of interest. The following balance equations describe the rate of change of drug concentration in each compartment.

$$\frac{dx_1}{dt} = -(k_{10} + k_{12})\,x_1 + k_{21}\,x_2 + u$$

$$\frac{dx_2}{dt} = k_{12}\,x_1 - (k_{20} + k_{21})\,x_2$$

where x_1 and x_2 = drug concentrations in compartments 1 and 2 (μg/kg patient weight), and u = rate of drug input to compartment 1 (scaled by the patient weight, μg/kg min).

Experimental studies (of the response of the compartment 1 concentration to various drug infusions) have led to the following parameter values:

$$(k_{10} + k_{12}) = 0.26 \text{ min}^{-1}$$
$$(k_{20} + k_{21}) = 0.094 \text{ min}^{-1}$$
$$k_{12}k_{21} = 0.015 \text{ min}^{-1}$$

for the drug atracurium, which is a muscle relaxant. Notice that the parameters have not been independently determined. Show (through numerical simulation) that all of the following values lead to the same results for the behavior of x_1, while the results for x_2 are different. Let the initial concentration be 0 for each compartment, and assume a constant drug infusion rate of 5.2 μg/kg min.

a. $k_{12} = k_{21}$
b. $k_{12} = 2\,k_{21}$
c. $k_{12} = 0.5\,k_{21}$

Discuss how the concentration of compartment 2 (if measurable) could be used to determine the actual values of k_{12} and k_{21}.

Use the MATLAB function `ode45` for your simulations.

6. A stream contains a waste chemical, W, with a concentration of 1 mol/liter. To meet EPA and state standards, at least 90% of the chemical must be removed by reaction. The chemical decomposes by a second-order reaction with a rate constant of 1.5 liter/(mol hr). The stream flowrate is 100 liter/hr and two available reactors (400 and 2000 liters) have been placed in series (the smaller reactor is placed before the larger one).

a. Write the modeling equations for the concentration of the waste chemical. Assume constant volume and constant density. Let

C_{w1} = concentration in reactor 1, mol/liter
C_{w2} = concentration in reactor 2, mol/liter
F = volumetric flowrate, liter/hr
V_1 = liquid volume in reactor 1, liters
V_2 = liquid volume in reactor 2, liters
k = second-order rate constant, liter/(mol hr)

b. Show that the steady-state concentrations are 0.33333 mol/liter (reactor 1) and 0.09005 mol/liter (reactor 2), so the specification is met.

(*Hint:* You need to solve quadratic equations to obtain the concentrations.)

c. The system is not initially at steady-state. Write a function file and use `ode45` for the following:

(i). If $C_{w1}(0) = 0.3833$ and $C_{w2}(0) = 0.09005$, find how the concentrations change with time.

(ii). If $C_{w1}(0) = 0.3333$ and $C_{w2}(0) = 0.14005$, find how the concentrations change with time.

7. Consider a batch reactor with a series reaction where component A reacts to form the desired component B *reversibly*. Component B can also react to form the undesired component C. The process objective is to maximize the yield of component B. A mathematical model is used to predict the time required to achieve the maximum yield of B.

The reaction scheme can be characterized by

$$A \underset{k_{1r}}{\overset{k_{1f}}{<\!\!=\!\!>}} B \xrightarrow{k_2} C$$

Here k_{1f} and k_{1r} represent the kinetic rate constants for the forward and reverse reactions for the conversion of A to B, while k_2 represents the rate constant for the conversion of B to C.

Assuming that each of the reactions is first-order, the reactor operates at constant volume, and there are no feed or product streams, the modeling equations are:

$$\frac{dC_A}{dt} = -k_{1f} C_A + k_{1r} C_B$$

$$\frac{dC_B}{dt} = k_{1f} C_A - k_{1r} C_B - k_2 C_B$$

$$\frac{dC_C}{dt} = k_2 C_B$$

where C_A, C_B, and C_C represent the concentrations (mol/volume) of components A, B, and C, respectively.

a. For $k_{1f} = 2$, $k_{1r} = 1$, and $k_2 = 1.25$ hr^{-1}, use `ode45` to solve for the concentrations as a function of time. Assume an initial concentration of A of $C_{A0} = 1$ mol/liter. Then plot the concentrations as a function of time. For what time is the concentration of B maximized?

b. Usually there is some uncertainty in the rate constants. If the real value of k_2 is 1.5 hr^{-1} find how the concentrations vary with time and compare with part **a**.