Chapter 3

Solving Nonlinear Equations

Core Topics

(i) Estimation of Errors in Numerical Solutions (3.2)

(ii) Bisection Method (3.3)

(iii) Regula Falsi Method (3.4)

(iv) Newton's Method (3.5)

(v) Secant Method (3.6)

(vi) Fixed Point Iteration Method (3.7)

(vii) Use of MATLAB built in Functions for Solving Nonlinear Equations (3.8)

(viii) Equations with Multiple Roots (3.9)

(ix) Systems of Nonlinear Equations (3.10)

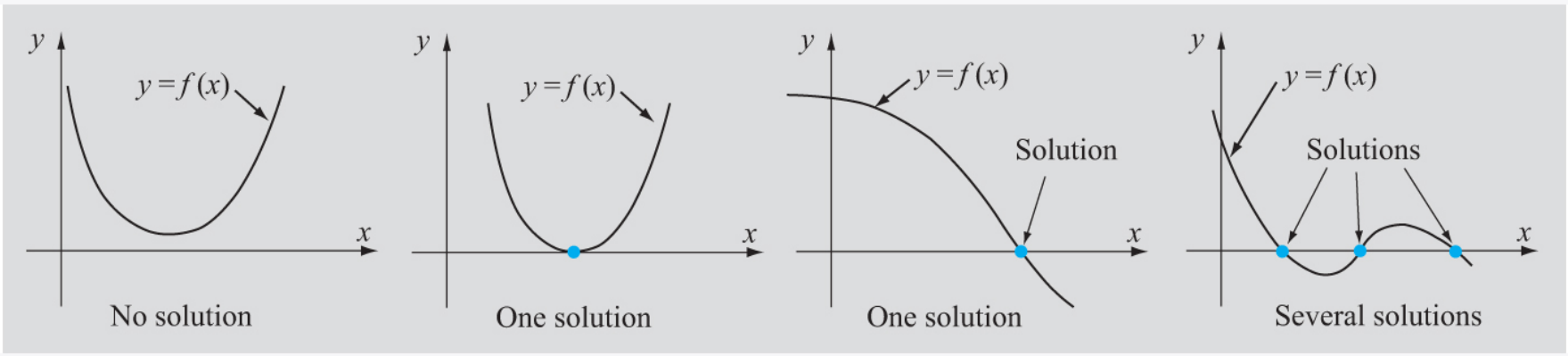
Background:

An equation of one variable can be written in the form:

$$f(x) = 0 \qquad (3.1)$$

A solution to the equation (sometimes called a root of the equation) is a numerical value of x that satisfies the equation.

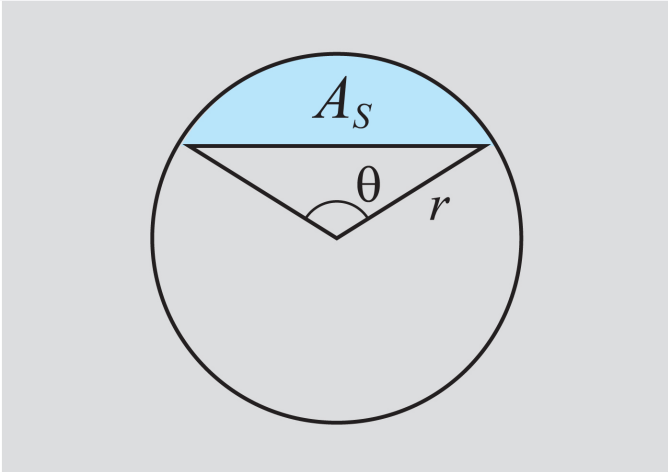
Graphically this is the point (or points) where the function $f(x)$ intersects the x –axis.



Background:

Sometimes the solution(s) can be obtained analytically (such as using the formula for solving a quadratic equation) but sometimes there is no analytic solution and the equation must be solved numerically. The following example should illustrate this. Consider the area of a segment of a circle of radius r:

$$A_S = \frac{1}{2}r^2(\theta - \sin\theta) \qquad (3.2)$$



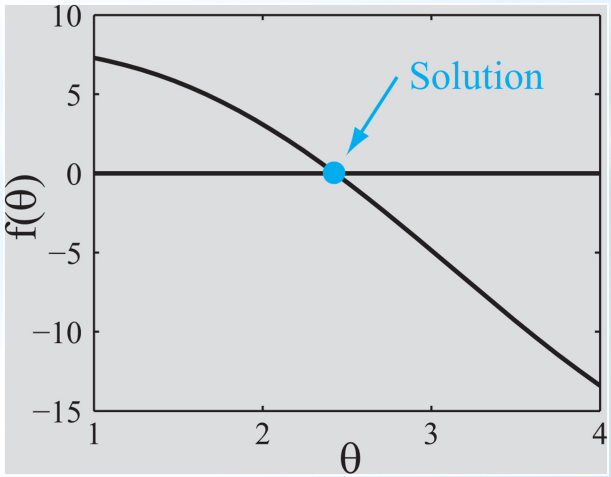
Background:

$$A_S = \frac{1}{2}r^2(\theta - \sin\theta) \qquad (3.2)$$

The problem is to determine θ where A_S and r are given. This equation is non-analytic in θ therefore we must use a numerical method to solve. That is to say that we use an algorithm that gives us a close but approximate solution to the problem. Lets say we have $A_S = 8m^2$ and $r = 3m$ then:

$$f(\theta) = 8 - 4.5(\theta - \sin\theta) = 0 \qquad (3.3)$$

The plot shows that the solution lies between 2 and 3.



Background:

Substituting $\theta = 2.4rad$ gives $f(\theta) = 0.2396$. $\theta = 2.43rad$ gives $f(\theta) = 0.003683$ etc.

By numerically approximating the solution we obtain an approximation to the solution of $f(\theta)$ but we don't (unless we are very lucky) get an exact solution.

This means that there is always an error in our approximation. This error must be quantified as it tells us the range in which the exact solution lies.

Estimation of Errors in Numerical Solutions:

Let x_{TS} be the true (exact) solution such that $f(x_{TS}) = 0$ and let x_{NS} be the numerical solution such that $f(x_{NS}) = \varepsilon$ where ε is a small number.

True Error (in the solution, x)

$$TrueError = x_{TS} - x_{NS} \qquad (3.4)$$

but in general x_{TS} is unknown so we have to use other measures.

Tolerance (in $f(x)$)

$$ToleranceInf = |f(x_{TS}) - f(x_{NS})| = |0 - \varepsilon| = |\varepsilon \qquad (3.5)$$

Estimation of Errors in Numerical Solutions:

Tolerance in the Solution (x)

If it is known that the solution is in the domain $[a, b]$ then

$$x_{NS} = \frac{a+b}{2} \qquad (3.6)$$

plus or minus a tolerance of:

$$Tolerance = \left|\frac{b-a}{2}\right| \qquad (3.7)$$

Estimation of Errors in Numerical Solutions:

True Relative Error

$$TrueRelativeError = \left| \frac{x_{TS} - x_{NS}}{x_{TS}} \right| \qquad (3.8)$$

but x_{TS} may be unknown so we use:

Estimated Relative Error

$$EstimatedRelativeError = \left| \frac{x_{NS}^{(n)} - x_{NS}^{(n-1)}}{x_{NS}^{(n-1)}} \right| \qquad (3.9)$$

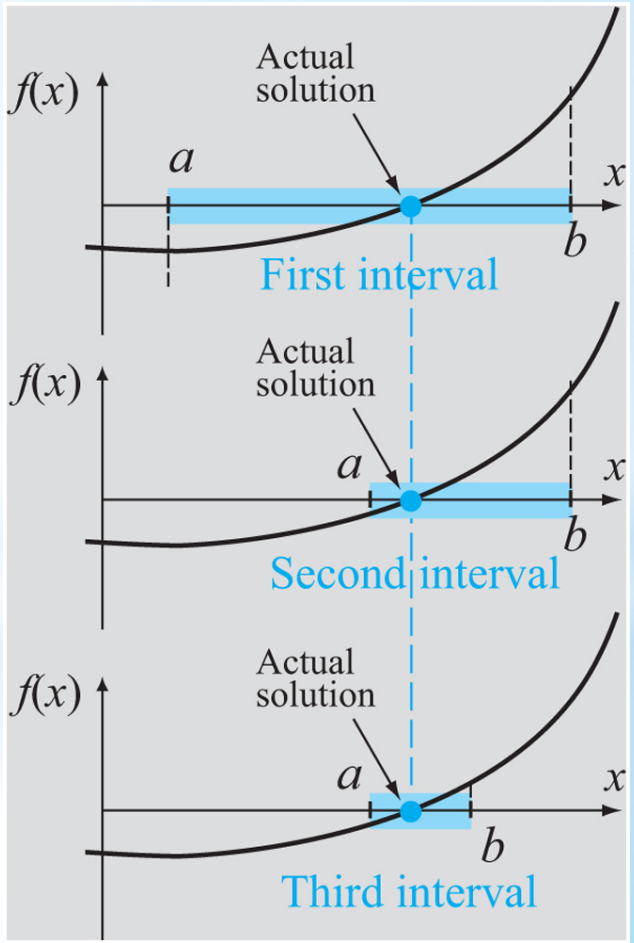
When the numerical solution is close to the true solution then $x_{NS}^{(n)} - x_{NS}^{(n-1)} << x_{NS}^{(n)}$ then the Estimated Relative Error is approximately the same as the True relative Error.

Background:

The methods for solving nonlinear equations are divided into two groups: Bracketing Methods and Open Methods.

In bracketing methods the interval (on the x −axis) containing the solution is identified (for example by algorithmically determining by successive computations in what range the function goes from positive to negative).

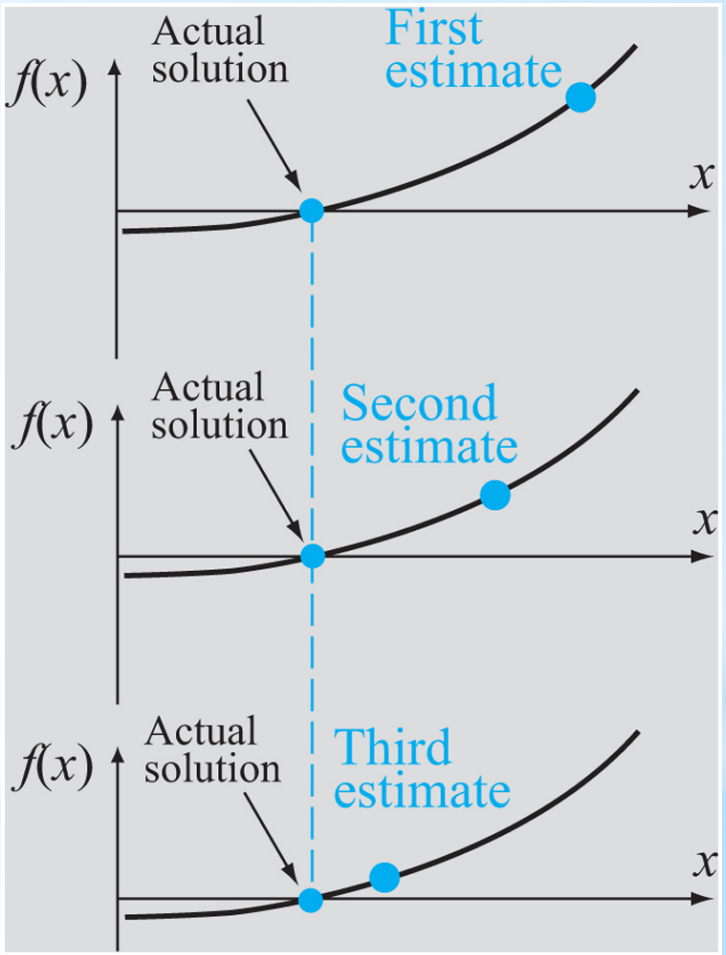
The size of the interval is successively reduced until the distance between the endpoints is less than the desired accuracy of the solution.



Background:

Open methods achieve this goal by making an initial guess at the solution and narrow down the search through further guesswork discarding solutions that have a higher error than that currently in hand and directing the search in that which gives a smaller error.

Open methods are more efficient but may not yield a solution. Bracketing methods on the other hand always converge to a solution



Overview:

Two bracketing methods are covered in this course:

(i) The Bisection Method
(ii) The Regula Falsi Method

Three open methods are covered:

(i) Newton's Method
(ii) The Secant Method
(iii) Fixed Point Iteration

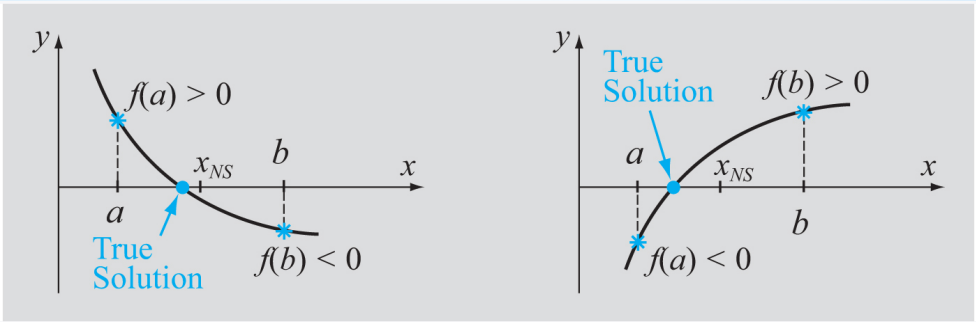
All these methods assume that $f(x)$ is continuous and differentiable over the appropriate interval and that the equation has a solution.

The Bisection Method:

The Bisection Method is a method for finding the solution of $f(x) = 0$ when it is known that within a given interval $[a, b]$ *(*this is why it is called a bracketing method).

Algorithm for the Bisection Method

1. Choose the first interval by finding points a and b such that a solution exists in this domain. This means that $f(a)$ and $f(b)$ have different signs so that $f(a)f(b) < 0$.



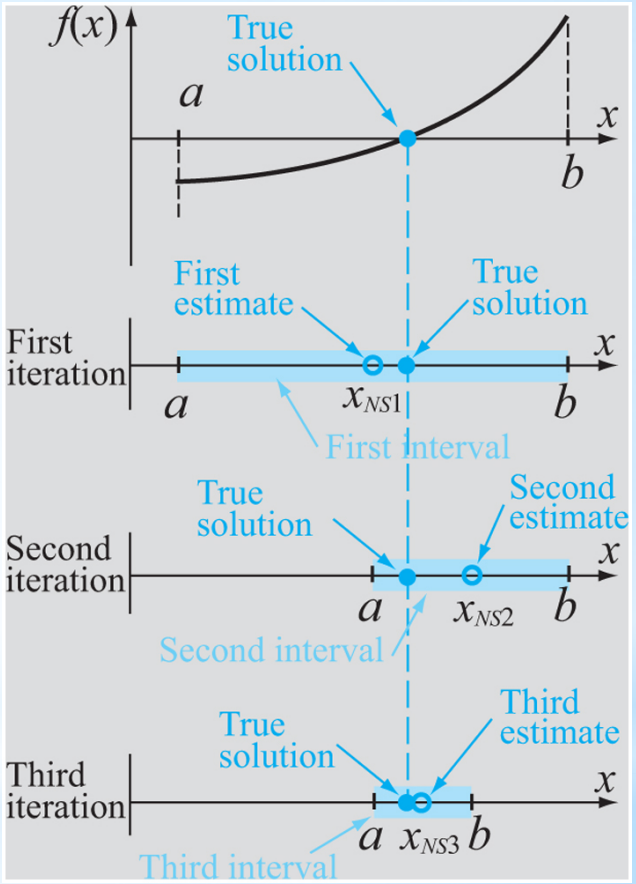
Algorithm for the Bisection Method:

2. The first estimate of the numerical solution is then:

$$x_{NS1} = \frac{a+b}{2}$$

3. Determine whether the true solution is between a and x_{NS1} or between x_{NS1} and b.
This is done by checking the sign of the product $f(a)f(x_{NS1})$.

4. Select the subinterval that contains the true solution and go back to step 2.



Steps 2. through 4. are repeated until a specified tolerance or error bound is achieved.

Example:

Example 3-1: Solution of a nonlinear equation using the bisection method.

Write a MATLAB program, in a script file, that determines the solution of the equation $8 - 4.5(x - \sin x) = 0$ by using the bisection method. The solution should have a tolerance of less than 0.001 rad. Create a table that displays the values of a, b, x_{NS}, $f(x_{NS})$, and the tolerance for each iteration of the bisection process.

SOLUTION

To find the approximate location of the solution, a plot of the function $f(x) = 8 - 4.5(x - \sin x)$ is made by using the `fplot` command of MATLAB. The plot (Fig. 3-8), shows that the solution is between $x = 2$ and $x = 3$. The initial interval is chosen as $a = 2$ and $b = 3$.

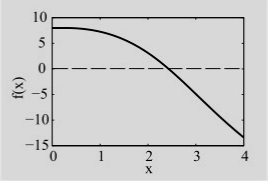


Figure 3-8: A plot of the function $f(x) = 8 - 4.5(x - \sin x)$.

A MATLAB program that solves the problem is as follows.

Program 3-1: Script file. Bisection method.

```
clear all
F = @ (x) 8-4.5*(x-sin(x));
a = 2; b = 3; imax = 20; tol = 0.001;
Fa = F(a); Fb = F(b);
if  Fa*Fb > 0
  disp('Error:The function has the same sign at points a and b.')
else
  disp('iteration   a       b    (xNS) Solution  f(xNS)  Tolerance')
  for i = 1:imax
      xNS = (a + b)/2;
      toli = (b - a)/2;
      FxNS = F(xNS);
      fprintf('%3i  %11.6f %11.6f %11.6f  %11.6f %11.6f\n', i, a, b, xNS, FxNS, toli)
      if FxNS = = 0
         fprintf('An exact solution x =%11.6f was found',xNS)
          break
      end
      if toli < tol
          break
      end
      if i = = imax
          fprintf('Solution was not obtained in %i iterations',imax)
          break
      end
      if F(a)*FxNS < 0
          b = xNS;
      else
          a = xNS;
      end
  end
end
```

Define $f(x)$ as an anonymous function.

Assign initial values to a and b, define max number of iterations and tolerance.

Stop the program if the function has the same sign at points a and b.

Calculate the numerical solution of the iteration, *xNS*.

Calculate the current tolerance.

Calculate the value of *f*(*xNS*) of the iteration.

Stop the program if the true solution, $f(x) = 0$, is found.

Stop the iterations if the tolerance of the iteration is smaller than the desired tolerance.

Stop the iterations if the solution was not obtained and the number of the iteration reaches `imax`.

Determine whether the true solution is between a and *xNS*, or between *xNS* and b, and select a and b for the next iteration.

When the program is executed, the display in the Command Window is:

```
iteration     a           b       (xNS) Solution   f(xNS)      Tolerance
  1        2.000000    3.000000    2.500000    -0.556875    0.500000
  2        2.000000    2.500000    2.250000     1.376329    0.250000
  3        2.250000    2.500000    2.375000     0.434083    0.125000
  4        2.375000    2.500000    2.437500    -0.055709    0.062500
  5        2.375000    2.437500    2.406250     0.190661    0.031250
  6        2.406250    2.437500    2.421875     0.067838    0.015625
  7        2.421875    2.437500    2.429688     0.006154    0.007813
  8        2.429688    2.437500    2.433594    -0.024755    0.003906
  9        2.429688    2.433594    2.431641    -0.009295    0.001953
 10        2.429688    2.431641    2.430664    -0.001569    0.000977
```

The numerical solution.

The value of the function at the numerical solution.

The last tolerance (satisfies the prescribed tolerance).

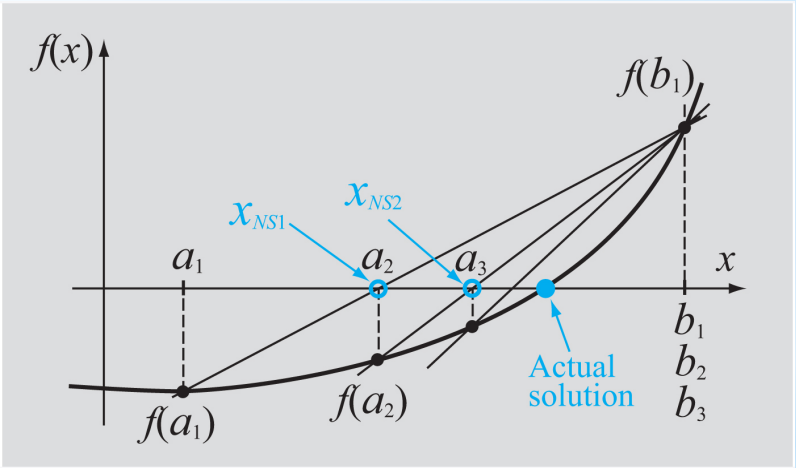
The output shows that the solution with the desired tolerance is obtained in the 10^{th} iteration.

The Regula Falsi Method:

Approximates the solution by calculating the intercept of the line containing $f(a)$ and $f(b)$ with the interval $[a, b]$ where the solution is known to exist.

The equation of the line is:

$$y = \frac{f(b) - f(a)}{b - a}(x - b) + f(b)$$



Its intercept with the x –axis is:

$$x_{NS} = \frac{af(b) - bf(a)}{f(b) - f(a)} \qquad (3.11)$$

Algorithm for the Regula Falsi Method:

1. Choose an interval $[a, b]$ such that a solution exists between them. This means that $f(a)$ and $f(b)$ have opposite signs.

2. Calculate the first estimate of the numerical solution x_{NS1} using Eq. (3.11) above.

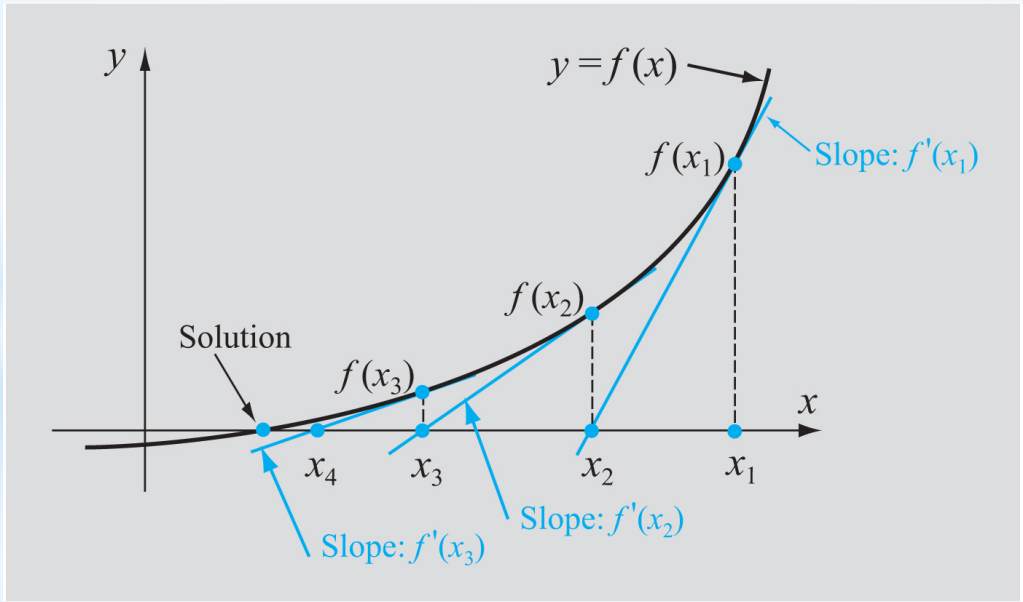
3. Determine whether the actual solution is between a and x_{NS1} or between x_{NS1} and b.

4. Select the appropriate subinterval and go back to Step 2.

5. Repeat until desired accuracy is reached.

Newton's Method:

Approximates the solution initially as the intercept of the tangent to the function, at an initial guess-point, with the x –axis. Subsequent iterations approximate the solution as the intercept of the tangent to the function, at the point defined by the previous estimate, with the x –axis. Note that it does not necessarily converge.



Algorithm for Newton's Method:

1. Choose $(x_1, f(x_1))$ as a starting point near the solution

2. Using the identity for slope of the tangent at $(x_1, f(x_1))$:

$$f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2} \qquad (3.12)$$

solve for x_2 using:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \qquad (3.13)$$

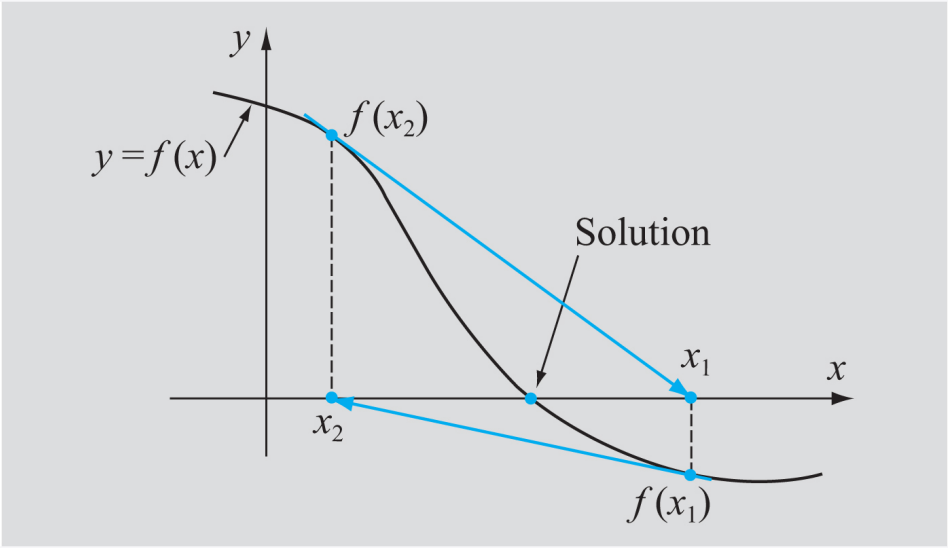
3. Repeat Step 2. iteratively using the formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (3.14)$$

until a desired accuracy in x_i is reached.

Notes on Newton's Method:

Newton's method may not converge to the correct solution in particular if $f(x) \approx 0$ in the vicinity of the solution.. It can be shown that if $f'(x), f''(x)$ are all continuous, $f'(x) \neq 0$ at the solution and if the starting value x_1 is near the solution then the method converges.



Example:

Example 3-2: Solution of equation using Newton's method.

Find the solution of the equation $8 - 4.5(x - \sin x) = 0$ (the same equation as in Example 3-1) by using Newton's method in the following two ways:

(*a*) Using a nonprogrammable calculator, calculate the first two iterations on paper using six significant figures.

(*b*) Use MATLAB with the function `NewtonRoot` that is listed in Fig. 3-11. Use 0.0001 for the maximum relative error and 10 for the maximum number of iterations.

In both parts, use $x = 2$ as the initial guess of the solution.

SOLUTION

In the present problem, $f(x) = 8 - 4.5(x - \sin x)$ and $f'(x) = -4.5(1 - \cos x)$.

(*a*) To start the iterations, $f(x)$ and $f'(x)$ are substituted in Eq. (3.14):

$$x_{i+1} = x_i - \frac{8 - 4.5(x_i - \sin x_i)}{-4.5(1 - \cos x_i)} \tag{3.20}$$

In the first iteration, $i = 1$ and $x_1 = 2$, and Eq. (3.20) gives:

$$x_2 = 2 - \frac{8 - 4.5(2 - \sin(2))}{-4.5(1 - \cos(2))} = 2.48517 \tag{3.21}$$

For the second iteration, $i = 2$ and $x_2 = 2.48517$, and Eq. (3.20) gives:

$$x_3 = 2.48517 - \frac{8 - 4.5(2.48517 - \sin(2.48517))}{-4.5(1 - \cos(2.48517))} = 2.43099 \tag{3.22}$$

(*b*) To solve the equation with MATLAB using the function `NewtonRoot`, the user must create user-defined functions for $f(x)$ and $f'(x)$. The two functions, called `FunExample2` and `FunDerExample2`, are:

```
function y = FunExample2(x)
y = 8 - 4.5*(x - sin(x));
```

and

```
function y = FunDerExample2(x)
y = -4.5 + 4.5*cos(x);
```

Once the functions are created and saved, the `NewtonRoot` function can be used in the Command Window:

The user-defined functions are entered as function handles.

```
>> format long
>> xSolution = NewtonRoot(@FunExample2,@FunDerExample2,2,0.0001,10)
xSolution =
   2.430465741723630
```

A comparison of the results from parts *a* and *b* shows that the first four digits of the solution (2.430) are obtained in the second iteration. (In part *b*, the solution process stops in the fourth iteration; see Problem 3.19.) This shows, as was mentioned before, that Newton's method usually converges fast. In Example 3-1 (bisection method), the first four digits are obtained only after 10 bisections.

Example:

Example 3-3: Convergence of Newton's method.

Find the solution of the equation $\frac{1}{x} - 2 = 0$ by using Newton's method. For the starting point (initial estimate of the solution) use:
(*a*) $x = 1.4$, (*b*) $x = 1$, and (*c*) $x = 0.4$

SOLUTION

The equation can easily be solved analytically, and the exact solution is $x = 0.5$.

For a numerical solution with Newton's method the function, $f(x) = \frac{1}{x} - 2$, and its derivative, $f'(x) = -\frac{1}{x^2}$, are substituted in Eq. (3.14):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{\frac{1}{x_i} - 2}{-\frac{1}{x_i^2}} = 2(x_i - x_i^2) \tag{3.23}$$

(*a*) When the starting point for the iterations is $x_1 = 1.4$, the next two iterations, using Eq. (3.23), are:

$$x_2 = 2(x_1 - x_1^2) = 2(1.4 - 1.4^2) = -1.12 \quad \text{and} \quad x_3 = 2(x_2 - x_2^2) = 2[(-1.12) - (-1.12)^2] = -4.7488$$

These results indicate that Newton's method diverges. This case is illustrated in Fig. 3-13*a*.

(*b*) When the starting point for the iterations is $x = 1$, the next two iterations, using Eq. (3.23), are:

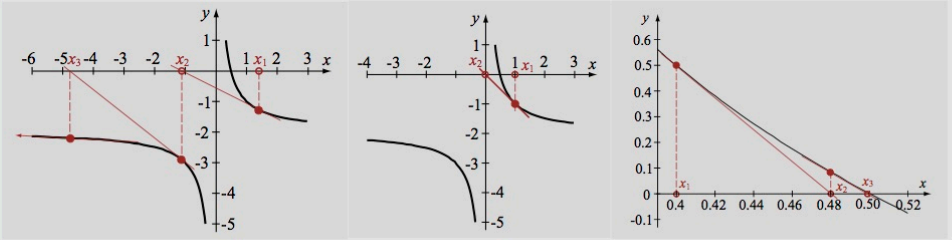
$$x_2 = 2(x_1 - x_1^2) = 2(1 - 1^2) = 0 \quad \text{and} \quad x_3 = 2(x_2 - x_2^2) = 2(0 - 0^2) = 0$$

From these results it looks like the solution converges to $x = 0$, which is not a solution. At $x = 0$, the function is actually not defined (it is a singular point). A solution is obtained from Eq. (3.23) because the equation was simplified. This case is illustrated in Fig. 3-13*b*.

(*c*) When the starting point for the iterations is $x = 0.4$, the next two iterations, using Eq. (3.23), are:

$$x_2 = 2(x_1 - x_1^2) = 2(0.4 - 0.4^2) = 0.48 \quad \text{and} \quad x_3 = 2(x_2 - x_2^2) = 2(0.48 - 0.48^2) = 0.4992$$

In this case, Newton's method converges to the correct solution. This case is illustrated in Fig. 3-13*c*. This example also shows that if the starting point is close enough to the true solution, Newton's method converges.

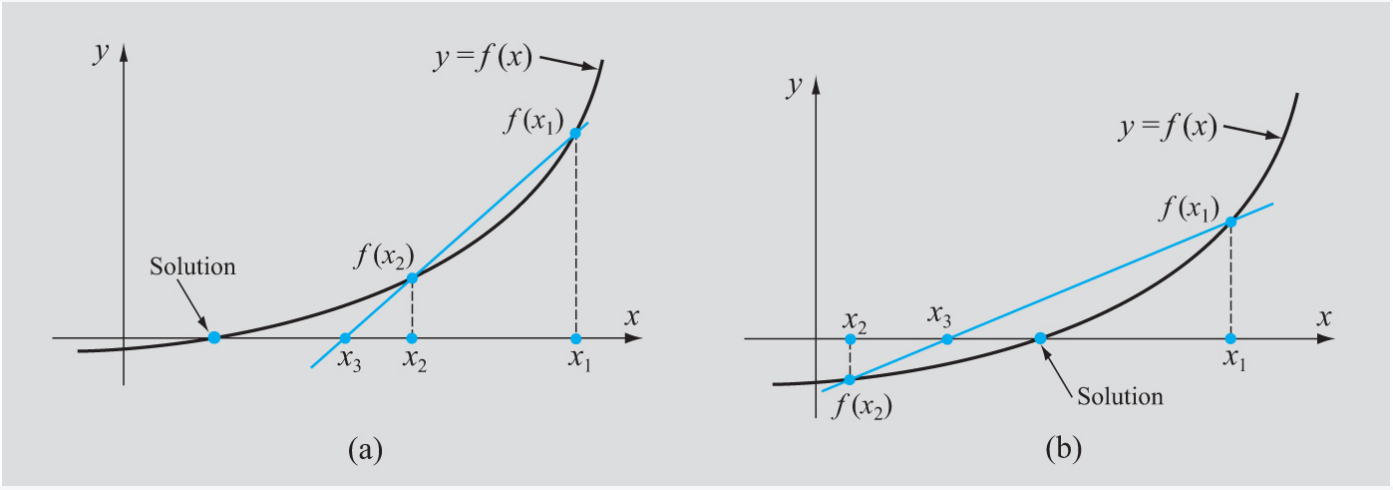


(*a*) (*b*) (*c*)

Figure 3-13: Solution with Newton's method using different starting points.

The Secant Method:

Same idea as Newton's Method but instead we use a secant to the function instead of a tangent. This means that we initially need two points in the neighbourhood of the solution to begin with.



Algorithm for the Secant Method:

1. Choose $(x_1, f(x_1))$, $(x_2, f(x_2))$ as starting points near the solution

2. Using the identity (for the secant containing $(x_1, f(x_1))$, $(x_2, f(x_2))$):

$$\frac{f(x_1) - f(x_2)}{x_1 - x_2} = \frac{f(x_2) - 0}{x_2 - x_3} \qquad (3.24)$$

   Solve for x_3:

$$x_3 = x_2 - \frac{f(x_2)(x_1 - x_2)}{f(x_1) - f(x_2)} \qquad (3.25)$$

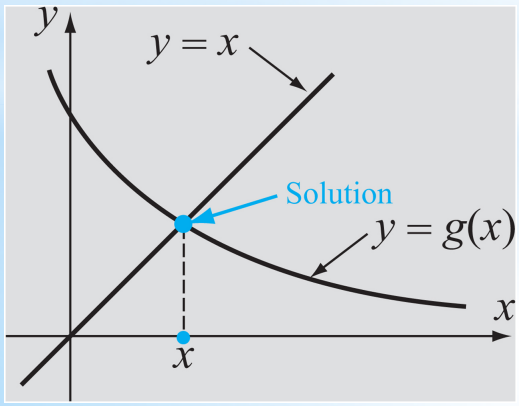
3. Now we use $(x_2, f(x_2))$, $(x_3, f(x_3))$ as points defining a new secant and repeat Step 2. iteratively using the formula:

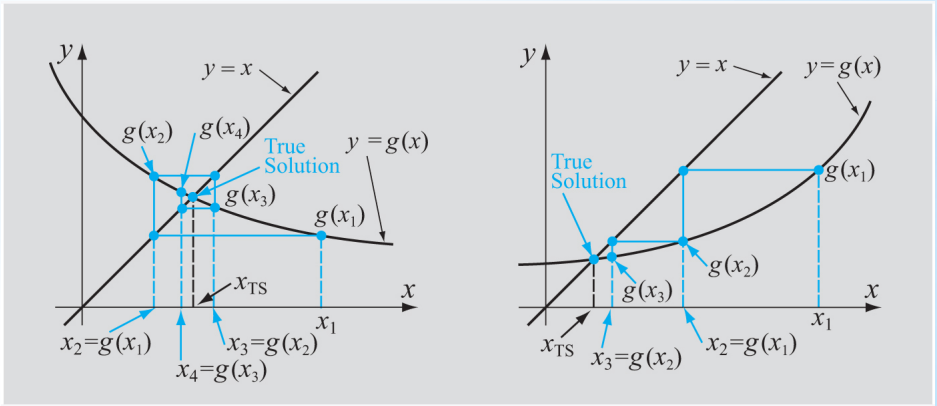
$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \qquad (3.26)$$

   until a desired accuracy is reached

The Fixed-Point Iteration Method:

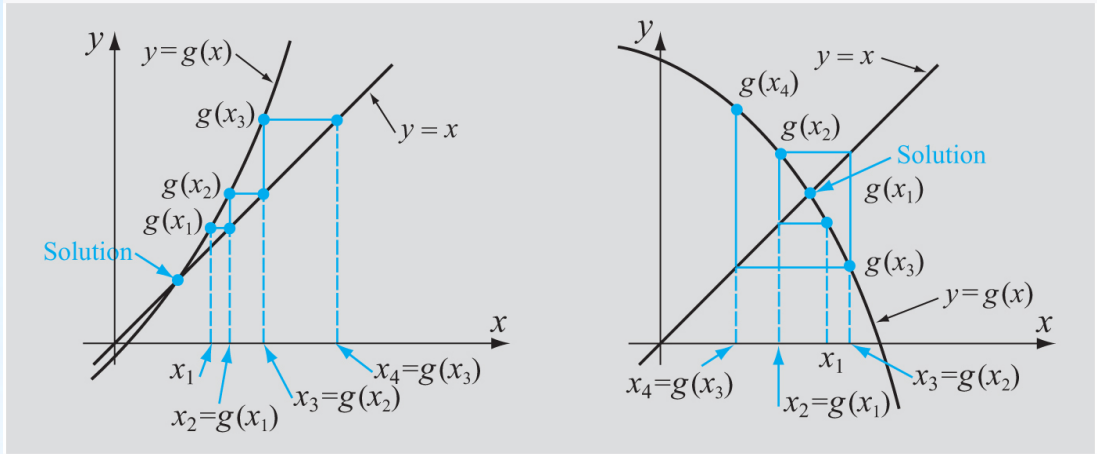
This method is carried out by writing the function $f(x) = 0$ in the form $x = g(x)$. This equation holds true at the intersection of the line $y = x$ and the function $y = g(x)$. The problem then boils down to obtaining the x-ordinate of this point of intersection. We then take a value of x near the solution (the point of intersection). This is the first estimate of the solution. We then find $g(x)$. This gives us a location on the function $y = g(x)$. We use this y-ordinate to obtain an x-ordinate using the line $y = x$ - this is our second estimate of the solution. The process is repeated iteratively until a desired accuracy is reached.





The Fixed-Point Iteration Method:

The Fixed-Point Iteration Method is prone to diverge depending on the manner in which $x = g(x)$ is expressed.



If the function is slowly varying such that in the neighbourhood of the solution:

$$|g'(x)| < 1 \qquad (3.30)$$

then the process will converge to the correct solution.

The Fixed-Point Iteration Method:

Consider:

$$xe^{0.5x} + 1.2x - 5 = 0 \qquad (3.31)$$

We can write $x = g(x)$ in a number of different ways:

$$x = \frac{5 - xe^{0.5x}}{1.2}$$

or

$$x = \frac{5}{e^{0.5x} + 1.2}$$

or

$$x = \frac{5 - 1.2x}{e^{0.5x}}$$

The Fixed-Point Iteration Method:

Only when expressed as the second option do we get

$$|g'(x)| < 1; x = 1, 2$$

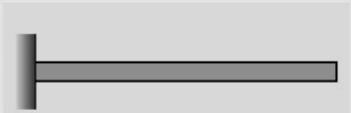
Hence, when expressed in this form the method converges:

$$x = \frac{5}{e^{0.5x} + 1.2}$$

Example:

Example 3-4: Solution of equation with multiple roots.

The natural frequencies, ω_n, of free vibration of a cantilever beam are determined from the roots of the equation:



$$f(k_nL) = \cos(k_nL)\cosh(k_nL) + 1 = 0 \qquad (3.37)$$

where L is the length of the beam, and the frequency ω_n is given by:

$$\omega_n = (k_nL)^2\sqrt{\frac{EI}{\rho AL^4}}$$

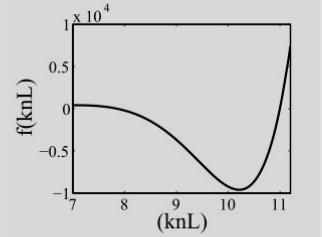
in which E is the elastic modulus, I is the moment of inertia, A is the cross-sectional area, and ρ is the density per unit length.

(a) Determine the value of the first root by defining smaller intervals over which the roots exist and using the secant method.

(b) Write a MATLAB program in a script file that determines the value of k_nL for the first four roots.

SOLUTION

Equation (3.37) is identical to Eq. (3.36), and a plot that shows the location of the first two roots is presented in Fig. 3-21. The location of the next two roots is shown in the figure on the right where the function is plotted over the interval $[7, 11.2]$. It shows that the third root is around 8 and the fourth root is near 11.



(a) The value of the first root is determined by using the `SecantRoot` user-defined function that is listed in Fig. 3-16. First, however, a user-defined function for the function $f(k_nL)$ in Eq. (3.37) is written (the function name is `FunExample3`):

```
function y = FunExample3(x)
y = cos(x)*cosh(x) + 1;
```

The first root is between 1 and 2. To find its solution numerically, the user-defined function `SecantRoot` (listed in Fig. 3.16) is used with $a = 1$, $b = 2$, $Err = 0.0001$, and $imax = 10$:

```
>> FirstSolution = SecantRoot(@FunExample3,1,2,0.0001,10)
FirstSolution =
   1.875104064602412
```

`format long` is used in MATLAB.

(b) Next, a MATLAB program that automatically finds the four roots is written. The program evaluates the function over a set of successive intervals and looks for sign changes. It starts at $k_nL = 0$ and uses an increment of 0.2 up to a value of $k_nL = 11.2$. If a change in sign is detected, the root within that interval is determined by using MATLAB's built-in `fzero` function.

```
clear all
F = @ (x) cos(x)*cosh(x)+1;
Inc = 0.2;
i = 1;
KnLa = 0;                          % Define the left point of the first increment.
KnLb = KnLa + Inc;                 % Define the right point of the first increment.
while KnLb <= 11.2
  if F(KnLa)*F(KnLb) < 0           % Check for a sign change in the value of the function.
      Roots(i) = fzero(F,[KnLa,KnLb]);   % Determine the root within the interval if a sign change was detected.
      i = i + 1;
  end
  KnLa = KnLb;                     % Define the left point of the next increment.
  KnLb = KnLa + Inc;               % Define the right point of the next increment.
end
Roots
```

When the program is executed, the display in the Command Window is:

```
Roots =
 1.875104068711961  4.694091132974175  7.854757438237613 10.995540734875467
```

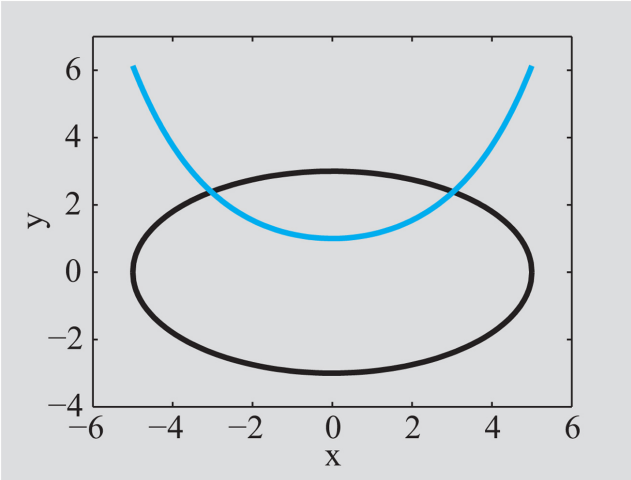
These are the values of the first four roots.

Systems of Nonlinear Equations:

A system of nonlinear equations consists of two or more nonlinear equations that have to be solved simultaneously. E.g. to find the intersection of a catenary and an ellipse:

$$f_1(x, y) = y - \frac{1}{2}(e^{x/2} + e^{(-x)/2}) = 0 \qquad (3.38)$$

$$f_2(x, y) = 9x^2 + 25y^2 - 225 = 0 \qquad (3.39)$$



Systems of Nonlinear Equations:

We will examine two methods for solving systems of nonlinear equations, namely:

1. Newton's Method
2. The Fixed Point Iteration Method.

Newton's Method (for Nonlinear Equations):

Consider:

$$f_1(x, y) = 0$$
$$f_2(x, y) = 0 \qquad (3.40)$$

Let x_2, y_2 be the true solution and x_1, y_1 be the first estimate of the solution. Then the Taylor Series expansion for the functions f_1 and f_2 about (x_2, y_2) is:

$$f_1(x_2, y_2) = f_1(x_1, y_1) + (x_2 - x_1)\left.\frac{\partial f_1}{\partial x}\right|_{x_1, y_1} + (y_2 - y_1)\left.\frac{\partial f_1}{\partial y}\right|_{x_1, y_1} + \ldots (3.41)$$

$$f_2(x_2, y_2) = f_2(x_1, y_1) + (x_2 - x_1)\left.\frac{\partial f_2}{\partial x}\right|_{x_1, y_1} + (y_2 - y_1)\left.\frac{\partial f_2}{\partial y}\right|_{x_1, y_1} + \cdots (3.42)$$

Newton's Method (for Nonlinear Equations):

Since x_2, y_2 are close to x_1, y_1 then we can neglect the higher order terms giving:

$$\left.\frac{\partial f_1}{\partial x}\right|_{x_1, y_1} \Delta x + \left.\frac{\partial f_1}{\partial y}\right|_{x_1, y_1} \Delta y = -f_1(x_1, y_1) \qquad (3.43)$$

$$\left.\frac{\partial f_2}{\partial x}\right|_{x_1, y_1} \Delta x + \left.\frac{\partial f_2}{\partial y}\right|_{x_1, y_1} \Delta y = -f_2(x_1, y_1) \qquad (3.44)$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Cramer's Rule and Solution to a System of Simultaneous Equations:

Cramer's Rule states that the solution (for $[x]$), if it exists, is given by:

$$x_j = \frac{\det(a'_j)}{\det(a)} \text{ for } j = 1, 2, \ldots, n \qquad (2.44)$$

where a'_j is the matrix formed by replacing the jth column of the matrix $[a]$ with the column vector $[b]$.

Note: $[a]^{-1}$ exists iff $\det(a) \neq 0$

$\det(a) = 0$ if one or more columns or rows of $[a]$ are not linearly independent.

The Jacobian:

The Jacobian is a quantity that arises when solving systems of non-linear equations. Given $f_1(x, y) = a$ and $f_2(x, y) = b$ (where a and b are constants) then the Jacobian matrix (given here for a 2×2 matrix) is:

$$[J] = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} \qquad (2.71)$$

The Jacobian determinant or simply the Jacobian is:

$$f_1, f_2) = \det\left(\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}\right) = \left(\frac{\partial f_1}{\partial x}\right)\left(\frac{\partial f_2}{\partial y}\right) - \left(\frac{\partial f_1}{\partial y}\right)\left(\frac{\partial}{\partial}\right. \qquad (2.72)$$

Newton's Method (for Nonlinear Equations):

This system of equations can then be solved using Cramer's Rule:

$$\Delta x = \frac{-f_1(x_1, y_1)\left.\frac{\partial f_2}{\partial y}\right|_{x_1, y_1} + f_2(x_1, y_1)\left.\frac{\partial f_1}{\partial y}\right|_{x_1, y_1}}{J(f_1(x_1, y_1), f_2(x_1, y_1))} \qquad (3.45)$$

$$\Delta y = \frac{-f_2(x_1, y_1)\left.\frac{\partial f_1(x_1, y_1)}{\partial x}\right|_{x_1, y_1} + f_1(x_1, y_1)\left.\frac{\partial f_2}{\partial x}\right|_{x_1, y}}{J(f_1(x_1, y_1), f_2(x_1, y_1))} \qquad (3.46)$$

where

$$J(f_1, f_2) = det\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} \qquad (3.47)$$

Newton's Method (for Nonlinear Equations):

Then:

$$x_2 = x_1 + \Delta x$$
$$y_2 = y_1 + \Delta y \qquad (3.48)$$

Note that x_2, y_2 obtained here are not exactly the true solutions since we truncated the Taylor Series. A new estimate can be obtained by repeating the process using x_2, y_2 as the new starting point and continuing the process until a desired accuracy is reached.

Newton's Method (for Nonlinear Equations):

Newton's Method can be generalised for a system of n nonlinear equations which have the form:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\ldots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \tag{3.54}$$

which gives:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} -f_1 \\ -f_2 \\ \cdots \\ -f_n \end{bmatrix} \tag{3.55}$$

Newton's Method (for Nonlinear Equations):

Algorithm:

1. Estimate an initial solution. $x_{1,i}, x_{2,i}, x_{3,i}$ …
2. Solve Equation (3.55) for $\Delta x_1, \Delta x_2, \Delta x_3$ …
3. Calculate a new estimate of the solution using:

$$\begin{aligned} x_{1,\,i+1} &= x_{1,\,i} + \Delta x_1 \\ x_{2,\,i+1} &= x_{2,\,i} + \Delta x_2 \\ &\cdots \\ x_{n,\,i+1} &= x_{n,\,i} + \Delta x_n \end{aligned} \qquad (3.56)$$

4. Repeat until the desired accuracy is reached

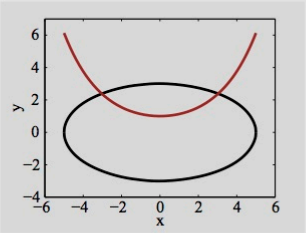
Newton's Method (for Nonlinear Equations):

Newton's Method does not necessarily converge but when it does it does so quickly. The method will likely converge if the functions and their derivatives are continuous and bounded near the solution. Also the Jacobian must be non-zero and the initial estimates must be close to the solution.

Example:

Example 3-5: Solution of a system of nonlinear equations using Newton's method.

The equations of the catenary curve and the ellipse, which are shown in the figure, are given by:



$$f_1(x, y) = y - \frac{1}{2}(e^{x/2} + e^{(-x)/2}) = 0 \qquad (3.49)$$

$$f_2(x, y) = 9x^2 + 25y^2 - 225 = 0 \qquad (3.50)$$

Use Newton's method to determine the point of intersection of the curves that resides in the first quadrant of the coordinate system.

SOLUTION

Equations (3.49) and (3.50) are a system of two nonlinear equations. The points of intersection are given by the solution of the system. The solution with Newton's method is obtained by using Eqs. (3.43) and (3.44). In the present problem, the partial derivatives in the equations are given by:

$$\frac{\partial f_1}{\partial x} = -\frac{1}{4}(e^{x/2} - e^{(-x)/2}) \quad \text{and} \quad \frac{\partial f_1}{\partial y} = 1 \qquad (3.51)$$

$$\frac{\partial f_2}{\partial x} = 18x \quad \text{and} \quad \frac{\partial f_2}{\partial y} = 50y \qquad (3.52)$$

The Jacobian is given by:

$$J(f_1, f_2) = det\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \det\begin{bmatrix} -\frac{1}{4}(e^{x/2} - e^{(-x)/2}) & 1 \\ 18x & 50y \end{bmatrix} = -\frac{1}{4}(e^{x/2} - e^{(-x)/2})50y - 18x \qquad (3.53)$$

Substituting Eqs. (3.51)–(3.53) in Eqs. (3.45) and (3.46) gives the solution for Δx and Δy.

The problem is solved in the MATLAB program that is listed below. The order of operations in the program is:

- The solution is initiated by the initial guess, $x_i = 2.5$, $y_i = 2.0$.
- The iterations start. Δy and Δx are determined by substituting x_i and y_i in Eqs. (3.45) and (3.46).
- $x_{i+1} = x_i + \Delta x$, and $y_{i+1} = y_i + \Delta y$ are determined.
- If the estimated relative error (Eq. (3.9)) for both variables is smaller than 0.001, the iterations stop. Otherwise, the values of x_{i+1} and y_{i+1} are assigned to x_i and y_i, respectively, and the next iteration starts.

The program also displays the solution and the error at each iteration.

```
%   Solution of Chapter 3 Example 5
F1 = @ (x,y) y - 0.5*(exp(x/2) + exp(-x/2));
F2 = @ (x,y) 9*x^2 + 25*y^2 - 225;
F1x = @ (x) -(exp(x/2) - exp(-x/2))/4;
F2x = @ (x) 18*x;
F2y = @ (y) 50*y;
Jacob = @ (x,y) -(exp(x/2) - exp(-x/2))/4*50*y - 18*x;
xi = 2.5; yi = 2; Err = 0.001;
for i = 1:5
  Jac = Jacob(xi,yi);
  Delx = (-F1(xi,yi)*F2y(yi) + F2(xi,yi))/Jac;
  Dely = (-F2(xi,yi)*F1x(xi) + F1(xi,yi)*F2x(xi))/Jac;
  xip1 = xi + Delx;
  yip1 = yi + Dely;
  Errx = abs((xip1 - xi)/xi);
  Erry = abs((yip1 - yi)/yi);
  fprintf('i =%2.0f x = %7.4f y = %7.4f Error in x = %7.4f Error in y = %-
7.4f\n',i,xip1,yip1,Errx,Erry)
  if Errx < Err & Erry < Err
      break
  else
      xi = xip1; yi = yip1;
  end
end
```

Assign the initial estimate of the solution.

Start the iterations.

Calculate Δx and Δy with Eqs. (3.45) and (3.46).

Calculate x_{i+1} and y_{i+1}.

If the error is not small enough, assign x_{i+1} to x_i, and y_{i+1} to y_i.

When the program is executed, the display in the Command Window is:

```
i = 1 x = 3.1388  y = 2.4001  Error in x = 0.25551 Error in y = 0.20003
i = 2 x = 3.0339  y = 2.3855  Error in x = 0.03340 Error in y = 0.00607
i = 3 x = 3.0312  y = 2.3859  Error in x = 0.00091 Error in y = 0.00016
```

These results show that the values converge quickly to the solution.

The Fixed Point Iteration Method:

Algorithm:

1. We begin with:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\ldots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned} \qquad (3.57)$$

2. We write:

$$\begin{aligned} x_1 &= g_1(x_1, x_2, \ldots, x_n) \\ x_2 &= g_2(x_1, x_2, \ldots, x_n) \\ &\ldots \\ x_n &= g_n(x_1, x_2, \ldots, x_n) \end{aligned} \qquad (3.58)$$

The Fixed Point Iteration Method:

3. We then choose an initial estimate for the solution: $x_{1,1}$, $x_{2,1}$, $x_{3,1}$, ... which is substituted into the r.h.s. of Equation (3.58). The l.h.s. is our new second estimate of the solution.

4. We continue this process until a desired accuracy is reached.

The Fixed Point Iteration Method:

The Fixed Point Iteration Method will converge under the following sufficient but not necessary conditions:

1. If the functions, g, and their derivatives are continuous in the neighbourhood of the solution.

2. If $\sum_i \left|\frac{\partial g_n}{\partial x_i}\right| \leq 1$

3. The initial estimate is close to the solution.

Recommended Problems:

Problems to be solved by hand (do at least 2):

3.2, 3.8, 3.14

Problems to be programmed in MATLAB:

3.26

Problems in math, science and engineering:

3.27 (you may use only (b))

You should pick out problems that you find interesting/challenging and do these too.