Section 7 – Ordinary Differential Equations

Differential equations play an important role in engineering since the descriptions of many physical phenomena are best formulated in terms of their rates of change. Differential equations that involve one independent variable are called *ordinary differential equations*; those that involve more than one are called *partial differential equations*. Differential equations are classified with respect to their order. Second-order equations, for example, include second derivatives. For example, the position of a mass-spring-damper system is given by the second-order equation

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

Higher-order differential equations can be reduced to a system of first-order differential equations. For example, let $v = dx/dt$ and $dv/dt = d^2x/dt^2$, so the second-order equation above can be replaced with an equivalent system of 2 first-order differential equations

$$\frac{dv}{dt} = \frac{-cv - kx}{m}$$
$$\frac{dx}{dt} = v$$

Therefore, we will first focus on the solution of first-order differential equations, and later discuss the solutions of systems of them.

Consider the first-order ordinary differential equation

$$\frac{dy}{dx} = f(x, y)$$

One way to calculate the value of y at a new position x would be

$$y_{i+1} = y_i + (slope)\Delta x$$

This concept will be the basis of the first class of solution techniques to examine.

7.1 Euler's Method

The first derivative gives a direct estimate of the slope:

$$y_{i+1} = y_i + f(x, y)h$$

Where $h = \Delta x$ and $f(x,y)$ is evaluated at the current point (x_i,y_i). This is Euler's (or the Euler-Cauchy or point-slope) method .

Two main sources of error take place when solving ordinary differential equations, truncation and round-off error. Truncation error results from the approximation of y. This in turn has two parts, a *local truncation error* from each step and a *propagated truncation error* that accumulates and is carried through the calculation. The sum of the two is the *global truncation error*.

The local truncation error in Euler's method is $O(h^2)$. This can be decreased by taking smaller steps, and the calculation will be error-free if the function $f(x,y)$ is linear. Thus, Euler's method is a *first-order technique*, and the global error is $O(h)$.

Pseudocode – General ODE, Euler's Method

a) Main or Driver program

```
SUB ODE_Driver(y, xi, xf, dx, xout)
' y = initial value, dependent variable
' xi = initial value, independent variable
' xf = final value, independent variable
' dx = calculation step
' xout = output interval
      x = xi
      m = 0
      xpm = x
      ypm = y
      DO
            xend = x + xout
            IF (xend > xf) THEN xend = xf
            h = dx
            CALL Integrator(x, y, h, xend)
            m = m + 1
            xpm = x
            ypm = y
            DISPLAY x, y RESULTS
            IF (x >= xf) EXIT
      END DO
END SUB
```

b) Integrator - to take one output step

```
SUB Integrator(x, y, h, xend)
      DO
            IF (xend - x < h) THEN h = xend - x
            CALL Euler(x, y, h, ynew)
            y = ynew
            IF (x >= xend) EXIT
      END DO
END SUB
```

c) Euler's Method for a Single ODE

```
SUB Euler(x, y, h, ynew)
      ynew = y + dydx(x, y) * h
      x = x + h
END SUB
```

d) Routine to evaluate the derivative

```
FUNCTION dydx(x, y)
      dydx = f(x, y)
END FUNCTION
```

7.1.1 Improvements to Euler's Method

The fundamental problem with Euler's method is that the derivative at the beginning of the interval is used to estimate the slope across the entire interval. Improvements to Euler's method focus on obtaining a more accurate representation of the average slope across the interval.

Huen's Method

Use the slope at the beginning of the interval $f(x_i,y_i)$ to find an initial estimate for y at the end of the interval (the *predictor*)

$$y_{i+1}^{o} = y_i + f(x_i, y_i)h$$

Use this result to calculate the slope at the end of the interval

$$y_{i+1}^{'} = y_i + f(x_{i+1}, y_{i+1}^{0})h$$

Average these two estimates to yield a better estimate of the average slope across the interval (the *corrector*)

$$y_{i+1} = y_i + \frac{h}{2}\left[f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{o})\right]$$

The corrector can be iterated (although it does not have to be) to obtain an even more refined estimate for y_{i+1}. With small step sizes this iteration can quickly converge and yield better results.

If the derivative is a function of only the independent variable, $dy/dx = f(x)$, then the predictor and corrector steps can be combined to yield

$$y_{i+1} = y_i + \frac{f(x_i) + f(x_{i+1})}{2} h$$

This is equivalent to the trapezoidal rule of numerical integration. The local error is therefore $O(h^3)$ with global error $O(h^2)$, a second-order method.

Midpoint or Improved Polygon Method

Euler's method can be used to predict the value of y at the midpoint of the interval

$$y_{i+1/2} = y_i + f(x_i, y_i)h/2$$

The slope at the midpoint of the interval is then

$$y'_{i+1/2} = f(x_{i+1/2}, y_{i+1/2})$$

And can be used to represent the average slope across the interval

$$y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2})h$$

Note that this method does not involve a predictor/corrector and thus cannot be iterated to achieve better accuracy. This method is equivalent to the open 1-point Newton-Cotes integration formula, and had local error of $O(h^3)$ with global error $O(h^2)$.

7.2 Runge-Kutta Methods

These are a class of techniques that achieve high accuracy without the use of high order derivatives. All are of the general form

$$y_{i+1} = y_i + \phi(x_i, y_i, h)h$$

Where the *increment function* ϕ is of the form

$$\begin{aligned}
\phi &= a_1 k_1 + a_2 k_2 + \ldots + a_n k_n \\
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + p_1 h, y_i + q_{11} k_1 h) \\
k_3 &= f(x_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h) \\
&\vdots \\
k_n &= f(x_i + p_n h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \ldots + q_{n-1,n-1} k_{n-1} h)
\end{aligned}$$

The coefficients a, p and q are determined by setting y_{i+1} equal to terms in a Taylor expansion. The first-order (n=1) Runge-Kutta method reduces immediately to Euler's method.

7.2.1 Second-Order Runge-Kutta Methods

Using the Taylor expansion and n=2, the second-order Runge-Kutta methods have the following form:

$$\begin{aligned}
y_{i+1} &= y_i + (a_1 k_1 + a_2 k_2)h \\
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + p_1 h, y_i + q_{11} k_1 h)
\end{aligned}$$

So that $a_1 + a_2 = 1$, $a_2 p_1 = ½$ and $a_2 q_{11} = ½$. All of the second-order Runge-Kutta methods have local error $O(h^3)$ and global error $O(h^2)$.

Heun's Method with a single corrector ($a_2 = ½$)

$$\begin{aligned}
y_{i+1} &= y_i + (k_1/2 + k_2/2)h \\
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + h, y_i + k_1 h)
\end{aligned}$$

Midpoint Method ($a_2 = 1$)

$$y_{i+1} = y_i + k_2 h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h/2, y_i + k_1 h/2)$$

Ralston's Method ($a_2 = 2/3$, minimum bound on the truncation error)

$$y_{i+1} = y_i + (k_1/3 + 2k_2/3)h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + 3h/4, y_i + 3k_1 h/4)$$

7.2.2 Third-Order Runge-Kutta Method

The third-order Runge-Kutta method has local error $O(h^4)$ and global error $O(h^3)$:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 4k_2 + k_3)h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$
$$k_3 = f(x_i + h, y_i - k_1 h + 2k_2 h)$$

If the slope is a function of only the independent variable, $f(x,y) = f(x)$, this method reduces to the Simpon's 1/3 Rule.

7.2.3 Fourth-Order Runge-Kutta Method

The fourth-order Runge-Kutta has local error $O(h^5)$ and global error $O(h^4)$ and is by far the most popular of the R-K methods:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$
$$k_3 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h)$$
$$k_4 = f(x_i + h, y_i + k_3 h)$$

This method also reduces to the Simpson's 1/3 Rule if the slope is a function of only the independent variable, $f(x,y) = f(x)$. Higher-order Runge-Kutta methods exist, but due to programming complexity and accuracy requirements they are seldom used.

Pseudocode – 4th Order Runge-Kutta, Single ODE

```
SUB RK4(x, y, h, ynew)
      k1 = dydx(x, y)
      k2 = dydx(x + h/2, y + k1 * h/2)
      k3 = dydx(x + h/2, y + k2 * h/2)
      k4 = dydx(x + h, y + k3 * h)
      ynew = y + (k1 + 2 * k2 + 2 * k3 + k4) * h/6
      x = x + h
END SUB
```

7.3 Systems of Simultaneous Ordinary Differential Equations

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \ldots, y_n)$$
$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \ldots, y_n)$$
$$\vdots$$
$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \ldots, y_n)$$

Note that we will need n initial conditions to start the calculations. These systems of equations are straight forward to implement, care must be taken to correctly calculate and apply the slopes to the variables.

7.4 Adaptive Runge-Kutta Methods

Some ordinary differential equations have functions that change gradually over part of the domain, allowing the use of large step sizes, and regions of rapid change which requires the use of smaller, more precise, steps. Algorithms that adjust the step size as necessary are called *adaptive*, and require an estimate of the local error at each step to apply adaptive step-size control.

Two primary approaches exist. The first estimates the error as the difference in two predictions using different step sizes. The second estimates the error as the differences using two different orders of the Runge-Kutta method.

7.4.1 Adaptive Runge-Kutta, or Step-Halving, Method

This method involves taking each time step twice, once a single full step (y_1) and again as two half steps (y_2). The difference between the two estimates is itself a measure of the local truncation error:

$$\Delta = y_2 - y_1$$

Not only can this calculation be used for adaptive control of the step size, it can also be used to correct the more precise estimate:

$$y_2 \leftarrow y_2 + \frac{\Delta}{15}$$

And provide an improved local estimate of $O(h^5)$.

7.4.2 Runge-Kutta Fehlberg

This approach uses the difference between the fifth-order R-K and the fourth-order R-K methods to estimate the error at each time step. These two methods are used because they happen to share some calculations, and only six total function evaluations are required at each time step to complete both methods. See the text for details.

7.4.3 Step Size Control

Once the local error has been estimated, one can decide to increase the step size if the local error is small or decrease the step size if the local error exceeds a specified tolerance. One strategy is given as

$$h_{new} = h_{old}\left|\frac{\Delta_{new}}{\Delta_{present}}\right|^{\alpha}$$

Where $\alpha = 0.2$ if the step size is increased ($\Delta_{present} \le \Delta_{old}$) and $\alpha = 0.25$ if the step size is decreased ($\Delta_{present} > \Delta_{old}$). Δ_{new} is usually related to a relative error level, or $\Delta_{new} = \varepsilon y_{scale}$, where ε is an overall tolerance level and $y_{scale} = y$ to give fractional relative errors. Another reliable way to do this is to set

$$y_{scale} = |y| + \left|h\frac{dy}{dx}\right|$$

A far simpler and more useful technique simply increases the step size as $h_{new} = ch_{old}$ when the relative error is low and decreases the step size $h_{new} = h_{old} / c$ when the error is large. Useful values of c range generally between 1 (no step size control) and 4/3. It should be noted that when the error at a given step is determined to be too large, that step should be recalculated with a smaller step h until the error falls below the specified tolerance and the solution can proceed.

Pseudocode – Step halving + adaptive step size control

```
eps = 1e-6                              ' acceptable error in step
CALL RK4(x, y, h, ynew)                 ' call method with one full step
x = x - h                               ' step back for second evaluation
CALL RK4(x, y, h/2, ynew1)              ' call method with two half steps
CALL RK4(x, ynew1, h/2, ynew2)
delta = ABS(ynew2 - ynew)               ' estimate relative error
IF delta > eps AND h > eps /100         ' error too large, redo
      x = x - h
      h = h * 0.8
ELSE                                    ' error is acceptable
      y = ynew2 + (ynew2 - ynew)/15     ' apply correction
      if delta < (eps/10) THEN          ' if error small enough, increase h
            h = h / 0.75                ' but not too much
            IF h > (xout / 10) THEN h = xout / 10
      END IF
END IF
```

7.5 Stiffness

"Stiff" systems are those that superimpose rapidly changing components and slowly varying components at every time step. Most often, the rapidly changing components are transients that die out quickly. The difficulty is that the rapidly varying parts require small time steps, and since they are always present, adaptive step size control doesn't work.

Instead of the explicit methods examined thus far, implicit methods sometimes work well. Consider the *backward or implicit Euler's method*, which evaluates the derivative at a future time:

$$y_{i+1} = y_i + \frac{dy_{i+1}}{dt} h$$

If the homogeneous part of the slope is $dy/dt = -ay$, then

$$y_{i+1} = y_i - a y_{i+1} h \quad \text{or} \quad y_{i+1} = \frac{y_i}{1 + ah}$$

Which is unconditionally stable, $|y_i| \to 0$ as $i \to \infty$. This method is 1st-order accurate.

However, implicit formulations grow in complexity as the order increases, and even more for nonlinear ODEs. Gear devised a set of implicit formulations that have large stability limits based on backward difference formulations. These are the most widely used methods to solve stiff systems.

7.6 Multistep Methods

Multistep methods do not use information at a single point (x_i, y_i) to predict the dependent variable at a future point y_{i+1}. Instead, they use several previous points to determine a likely trajectory for the next point.

7.6.1 The Non-Self-Starting Heun Method

Recall that Heun's method uses Euler's method as a predictor (a forward difference)

$$y^o_{i+1} = y_i + f(x_i, y_i) h$$

And the trapezoidal rule as a corrector

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y^o_{i+1})}{2} h$$

The predictor is $O(h^2)$ while the corrector is $O(h^3)$, thus the predictor is the weak link in the process, especially since the iterative corrector is dependent on the accuracy of this initial prediction. To find a predictor that is $O(h^3)$, use the previous point y_{i-1} (a central difference)

$$y_{i+1}^{o} = y_{i-1} + f(x_i, y_i)2h$$

This is now $O(h^3)$ but uses a step size that is twice as large. Note that y_{i-1} is not available at the beginning of the calculation, so this method is not self-starting.

In general

$$\text{predictor}: \quad y_{i+1}^{o} = y_{i-1}^{m} + f(x_i, y_i^{m})2h$$

$$\text{corrector}: \quad y_{i+1}^{j} = y_i^{m} + \frac{f(x_i, y_i^{m}) + f(x_i, y_{i+1}^{j-1})}{2}h \qquad j = 1,2,\ldots,m$$

The corrector is iterated m times, usually not enough to actually converge but only to improve the prediction. Typically, m=2.

The truncation error per step can be estimated as

$$E_c = -\frac{y_{i+1}^{o} - y_{i+1}^{m}}{5}$$

Which can be used to develop modifiers for the predictor and corrector:

$$\text{corrector modifier}: \quad y_{i+1}^{m} \leftarrow y_{i+1}^{m} - \frac{y_{i+1}^{m} - y_{i+1}^{o}}{5}$$

$$\text{predictor modifier}: \quad y_{i+1}^{o} \leftarrow y_{i+1}^{o} + \frac{4}{5}(y_i^{m} - y_i^{o})$$

Note that the predictor modifier uses values from the previous step. The use of these modifiers is optional but can speed up convergence.

Step Size Control

Constant step size – Easy to implement but choice must be made small enough for convergence within m=2 iterations. Re-run the problem, halving the step size each time, until convergence is reached.

Variable step size – Monitor the number of iterations for convergence of corrector and adjust h so that m=2. Alternatively, double or halve the step size, maintaining the number of iterations to converge the corrector.

7.6.2 Higher-Order Multistep Methods

Higher-order multistep methods are based on open and closed Newton-Cotes or Adams integration formulas. Adams formulas use several points to estimate the integral only in the last segment of the interval, leading to slightly more accuracy.

Milne's Method

This method uses a 3-point Newton-Cotes open formula as a predictor

$$y_{i+1}^{o} = y_{i-3}^{m} + \frac{4h}{3}(2f_i^m - f_{i-1}^m + 2f_{i-2}^m)$$

And a 3-point Newton-Cotes closed formula as a corrector

$$y_{i+1}^{j} = y_{i-1}^{m} + \frac{h}{3}(f_{i-1}^m + 4f_i^m + f_{i+1}^{j-1})$$

There are, however, stability problems with this corrector. Using a more stable corrector leads to *Hamming's method*:

$$y_{i+1}^{j} = \frac{1}{8}\left[9y_i^m - y_{i-2}^m + 3h(f_{i+1}^{j-1} + 2y_i^m - f_{i-1}^m)\right]$$

With correctors

$$E_p = \frac{9}{121}(y_i^m - y_i^o) \;;\; E_c = -\frac{112}{121}(y_{i+1}^m - y_{i+1}^o)$$

Fourth-Order Adams Method

This method uses a 4th-order Adams-Bashford (open) formula as a predictor

$$y_{i+1}^{j} = y_i^m + h(\frac{55}{24}f_i^m - \frac{59}{24}f_{i-1}^m + \frac{37}{24}f_{i-2}^m - \frac{9}{24}f_{i-3}^m)$$

and a 4th-order Adams-Moulton (closed) formula as a corrector

$$y_{i+1}^{j} = y_i^m + h(\frac{9}{24}f_{i+1}^{j-1} + \frac{19}{24}f_i^m - \frac{5}{24}f_{i-1}^m + \frac{1}{24}f_{i-2}^m)$$

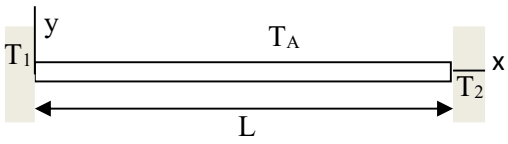
with modifiers

$$E_p = \frac{251}{270}(y_i^m - y_i^o) \;;\; E_c = -\frac{19}{270}(y_{i+1}^m - y_{i+1}^o)$$

7.7 Boundary Value Problems

Up to now we have dealt with *initial-value problems*, where initial conditions specified at one point are sufficient to determine the constants of integration and complete the solution. Problems that specify conditions at extreme points or boundaries of the system are called *boundary-value problems*. As an example, consider the heat balance in a long, thin rod

$$\frac{d^2T}{dx^2} + h'(T_a - T) = 0$$



Where *h'* is the heat transfer coefficient that describes the rate of heat dissipation to the surrounding air and T_a is the temperature of the surrounding air. To solve this problem, two boundary conditions must be specified, for example the temperatures at the ends of the rod

$$T(x = 0) = T_1 \qquad T(x = L) = T_2$$

For a 10-m rod with T_a = 20 C, T_1 = 40 C, T_2 = 200 C and h' = 0.01/m^2, the solution is

$$T(x) = 73.4523e^{0.1x} - 53.4523e^{-0.1x} + 20C$$

7.7.1 The Shooting Method

This method treats a boundary-value problem as an equivalent initial-value problem. Specify all the boundary conditions at x = 0 (T_1 = 40 C and make an initial guess at $dT/dx = z$), then solve the problem to find the temperature at $x = L$. If the solution does not match the boundary condition at $x = L$, adjust the boundary condition at x = 0 and re-solve the problem.

If the problem is linear (as in this example), obtaining two solutions and interpolating between them will supply the necessary boundary conditions at x = 0. If the problem is non-linear, for example if the following, better, approximation for the heat transfer from the bar is used

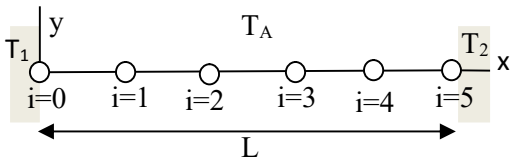
$$\frac{d^2T}{dx^2} + h''(T_a - T)^4 = 0$$

the shooting method can be cast as a general root-finding problem to determine the appropriate starting values in order to arrive at the other boundary value(s). The shooting method is straightforward but because of the need to repeatedly solve the problem it is not particularly efficient.

7.7.2 Finite Difference Methods

Finite difference methods discretize the domain of the solution and transform the linear differential equation into a set of simultaneous linear equations. For the heat transfer example above, divide the rod into equal-length segments, 5 in the example here. The divided-difference approximation is

$$\frac{d^2T}{dx^2} \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$$



The differential equation then becomes

$$\frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} + h'(T_a - T_i) = 0$$

or

$$-T_{i-1} + (2 + h'\Delta x^2)T_i - T_{i+1} = h'\Delta x^2 T_a$$

Appling this equation to the interior nodes of the rod (assuming that the temperature is known at the ends x=0 and x=L) yields

$$\begin{aligned}
(2 + h'\Delta x^2)T_1 - T_2 &= h'\Delta x^2 T_a + T_0 \\
-T_1 + (2 + h'\Delta x^2)T_2 - T_3 &= h'\Delta x^2 T_a \\
-T_2 + (2 + h'\Delta x^2)T_3 - T_4 &= h'\Delta x^2 T_a \\
-T_3 + (2 + h'\Delta x^2)T_4 &= h'\Delta x^2 T_a + T_5
\end{aligned}$$

The most common boundary condition is where the dependent variable is specified on the boundary, as in the example above. To apply these *fixed,* or *Dirichlet*, boundary conditions, replace the variables with their known values and eliminate the equation at that node.

One can also specify the derivative of the dependent variable at the boundary nodes. For example, the heat flux *dT/dx* can be specified at $x = 0$ or $x = L$ above. These *gradient*, or *Neumann*, boundary conditions are also approximated with finite differences. For example, let the boundary condition *dT/dx* be specified at $i = 0$ above. Taking a central difference approximation at $i = 0$ gives

$$\left.\frac{dT}{dx}\right|_0 = \frac{T_1 - T_{-1}}{2\Delta x}$$

Substituting this approximation into the finite difference equation for node $i = 0$ eliminates the fictitious node at $i = -1$ and gives

$$(2 + h'\Delta x^2)T_0 - 2T_1 = h'\Delta x^2 T_a - 2\Delta x \left.\frac{dT}{dx}\right|_0$$

The system of equations for the nodal temperatures then becomes

$$\begin{aligned}
(2 + h'\Delta x^2)T_0 - 2T_1 &= h'\Delta x^2 T_a - 2\Delta x \left.\frac{dT}{dx}\right|_0 \\
-T_0 + (2 + h'\Delta x^2)T_1 - T_2 &= h'\Delta x^2 T_a \\
-T_1 + (2 + h'\Delta x^2)T_2 - T_3 &= h'\Delta x^2 T_a \\
-T_2 + (2 + h'\Delta x^2)T_3 - T_4 &= h'\Delta x^2 T_a \\
-T_3 + (2 + h'\Delta x^2)T_4 &= h'\Delta x^2 T_a + T_5
\end{aligned}$$

These systems of equations can be solved for the unknown temperatures by any of the methods discussed previously.

7.7.3 Eigenvalue Problems

Eigenvalue, or *characteristic-value*, problems are a class of boundary value problems common in vibrations, elasticity and areas that deal with oscillating systems. These problems have the general form

$$[A - \lambda I]\{X\} = 0$$

Where λ are the eigenvalues and X are the associated eigenvectors.

Polynomial Method – Develop the set of equations $[A-\lambda I]\{X\}=0$. Expand the determinant of $A-\lambda I$, which will be a polynomial whose roots are λ. Solve for the roots with either Müller's or Bairstow's method, deflating in order to find all of the eigenvalues/eigenvectors (there will be one for each row of *A*).

Power Method – Write the system as $AX=\lambda X$. For an initial guess, assume that $X=\{1,1,1...1\}^T$, substitute and solve for a new set of *X*. Normalize with respect to the largest value of *X*. Iterate until convergence. Upon convergence, the normalization factor will be the largest eigenvalue, with eigenvector equal to *X*. If matrix *A* is symmetric, it can then be deflated using Hotelling's method, $A_2=A_1-\lambda_1 X_1 X_1^T$, where A_1 is the original matrix and λ_1, X_1 are the largest eigenvalue/eigenvector pair. Proceed in this manner to find the largest several eigenvalues. This method cannot usually be used to find all of the eigenvalues due to the accumulation of significant round-off errors.

To find the smallest eigenvalue/eigenvector pairs, perform the power method on the inverse of *A*, deflating in order to eliminate those already found.

Many more advanced techniques exist for finding eigenvalues.