MATLAB 与工程应用

ODE with boundary condition

MATLAB 与工程应用

Numerical Methods for Differential Equations

Outline

First-Order Differential Equations

Higher-Order Differential Equations

Special Methods for Linear Equations

Example Problem

- Consider an 80 kg paratrooper falling from 600 meters.
- The trooper is accelerated by gravity, but decelerated by drag on the parachute

Governing Equation

- m=paratrooper mass (kg)
- g=acceleration of gravity (m/s^2)
- V=trooper velocity (m/s)
- Initial velocity is assumed to be zero

$$m\frac{dV}{dt} = -mg - \frac{4}{15}V * |V|$$

Solving ODE's Numerically

- Euler's method is the simplest approach
- Consider most general first order ODE: dy/dt=f(t,y)
- Approximate derivative as $(y_{i+1}-y_i)/dt$
- Then:

$$\frac{dy}{dt} \approx \frac{y_{i+1} - y_i}{\Delta t} = f(t, y_i)$$

$$y_{i+1} \approx y_i + \Delta t \; f(t, y_i)$$

A Problem

- Unfortunately, Euler's method is too good to be true
- It is unstable, regardless of the time step chosen
- We must choose a better approach
- The most common is 4th order Runge-Kutta

Runge-Kutta Techniques

- Runge-Kutta uses a similar, but more complicated stepping algorithm

$$k_1 = \Delta t * f(t, y_i)$$

$$k_2 = \Delta t * f(t + \frac{\Delta t}{2}, y_i + \frac{k_1}{2})$$

$$k_3 = \Delta t * f(t + \frac{\Delta t}{2}, y_i + \frac{k_2}{2})$$

$$k_4 = \Delta t * f(t + \Delta t, y_i + k_3)$$

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

Approach

- Choose a time step
- Set the initial condition
- Run a series of steps
- Adjust time step
- Continue

First-Order Differential Equations

ORDINARY DIFFERENTIAL EQUATION (ODE)

INITIAL-VALUE PROBLEM

In this section, we introduce numerical methods for solving rst-order dif ferential equations. In Section 9.4 we show how to extend the techniques to higher-order equations.

An *ordinary differential equation* (ODE) is an equation containing ordinary derivatives of the dependent variable. An equation containing partial derivatives with respect to two or more independent variables is a *partial differential equation* (PDE). Solution methods for PDEs are an advanced topic, and we will not treat them in this text. In this chapter we limit ourselves to *initial-value problems* (IVPs). These are problems where the ODE must be solved for a given set of

values speci ed at some initial time, which is usually taken to be $t = 0$. Other types of ODE problems are discussed at the end of Section 9.6.

It will be convenient to use the following abbreviated "dot" notation for derivatives.

$$\dot{y}(t) = \frac{dy}{dt} \qquad \ddot{y}(t) = \frac{d^2y}{dt^2}$$

The *free response* of a differential equation, sometimes called the homogeneous solution or the initial response, is the solution for the case where there is no forcing function. The free response depends on the initial conditions. The *forced response* is the solution due to the forcing function when the initial conditions are zero. For linear differential equations, the complete or total response is the sum of the free and the forced responses. Nonlinear ODEs can be recognized by the fact that the dependent variable or its derivatives appear raised to a power or in a transcendental function. For example, the equations $\dot{y} = y^2$ and $\dot{y} = \cos y$ are nonlinear.

The essence of a numerical method is to convert the differential equation into a difference equation that can be programmed. Numerical algorithms differ partly as a result of the speci c procedure used to obtain the dif ference equations. It is important to understand the concept of "step size" and its effects on solution accuracy. To provide a simple introduction to these issues, we consider the simplest numerical methods, the *Euler method* and the *predictor-corrector method*.

The Euler Method

EULER METHOD

The *Euler method* is the simplest algorithm for numerical solution of a differential equation. Consider the equations

$$\frac{dy}{dt} = f(t,y) \qquad y(0) = y_0 \tag{9.3–1}$$

where $f(t, y)$ is a known function and y_0 is the initial condition, which is the given value of $y(t)$ at $t = 0$. From the de nition of the derivative,

$$\frac{dy}{dt} = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

If the time increment Δt is chosen small enough, the derivative can be replaced by the approximate expression

$$\frac{dy}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t} \tag{9.3–2}$$

Assume that the function $f(t, y)$ in Equation (9.3–1) remains constant over the time interval $(t, t + \Delta t)$, and replace Equation (9.3–1) by the following approximation:

$$\frac{y(t + \Delta t) - y(t)}{\Delta t} = f(t, y)$$

or

$$y(t + \Delta t) = y(t) + f(t, y)\Delta t \tag{9.3–3}$$

STEP SIZE

The smaller Δt is, the more accurate are our two assumptions leading to Equation (9.3–3). This technique for replacing a differential equation with a difference equation is the *Euler method*. The increment Δt is called the *step size*.

Equation (9.3–3) can be written in more convenient form as

$$y(t_{k+1}) = y(t_k) + \Delta t f[t_k, y(t_k)] \tag{9.3–4}$$

where $t_{k+1} = t_k + \Delta t$. This equation can be applied successively at the times t_k by putting it in a `for` loop. The accuracy of the Euler method can be improved sometimes by using a smaller step size. However, very small step sizes require longer run times and can result in a large accumulated error due to roundoff effects.

The Predictor-Corrector Method

The Euler method can have a serious de ciency in problems where the variables are rapidly changing, because the method assumes the variables are constant over the time interval Δt. One way of improving the method is to use a better approximation to the right-hand side of Equation (9.3–1). Suppose instead of the Euler approximation (9.3–4) we use the average of the right-hand side of Equation (9.3–1) on the interval (t_k, t_{k+1}). This gives

$$y(t_{k+1}) = y(t_k) + \frac{\Delta t}{2}(f_k + f_{k+1}) \tag{9.3–5}$$

where

$$f_k = f[t_k, y(t_k)] \tag{9.3–6}$$

with a similar de nition for f_{k+1}. Equation (9.3–5) is equivalent to integrating Equation (9.3–1) with the trapezoidal rule.

The dif culty with Equation (9.3–5) is that f_{k+1} cannot be evaluated until $y(t_{k+1})$ is known, but this is precisely the quantity being sought. A way out of this dif culty is to use the Euler formula (9.3–4) to obtain a preliminary estimate of $y(t_{k+1})$. This estimate is then used to compute f_{k+1} for use in Equation (9.3–5) to obtain the required value of $y(t_{k+1})$.

The notation can be changed to clarify the method. Let $h = \Delta t$ and $y_k = y(t_k)$, and let x_{k+1} be the estimate of $y(t_{k+1})$ obtained from the Euler formula (9.3–4). Then, by omitting the t_k notation from the other equations, we obtain the following description of the predictor-corrector process.

Euler predictor $$x_{k+1} = y_k + hf(t_k, y_k) \tag{9.3–7}$$

Trapezoidal corrector $$y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, x_{k+1})] \tag{9.3–8}$$

Runge-Kutta Methods

The Taylor series representation forms the basis of several methods of solving differential equations, including the Runge-Kutta methods. The Taylor series may be used to represent the solution $y(t + h)$ in terms of $y(t)$ and its derivatives as follows.

$$y(t + h) = y(t) + h\dot{y}(t) + \frac{1}{2}h^2\ddot{y}(t) + \cdots \tag{9.3–9}$$

The number of terms kept in the series determines its accuracy. The required derivatives are calculated from the differential equation. If these derivatives can be found, Equation (9.3–9) can be used to march forward in time. In practice, the high-order derivatives can be dif cult to calculate, and the series (9.3–9) is truncated at some term. The Runge-Kutta methods were developed because of the dif culty in computing the derivatives. These methods use several evaluations of the function $f(t, y)$ in a way that approximates the Taylor series. The number of terms in the series that is duplicated determines the order of the Runge-Kutta method. Thus, a fourth-order Runge-Kutta algorithm duplicates the Taylor series through the term involving h^4.

MATLAB ODE Solvers

In addition to the many variations of the predictor-corrector and Runge-Kutta algorithms that have been developed, there are more-advanced algorithms that use a variable step size. These "adaptive" algorithms use larger step sizes when the solution is changing more slowly. MATLAB provides several functions, called *solvers*, that implement the Runge-Kutta and other methods with variable step size. Two of these are the `ode45` and `ode15s` functions. The `ode45` function uses a combination of fourth- and fth-order Runge-Kutta methods. It is a general-purpose solver whereas `ode15s` is suitable for more-dif cult equations called "stiff" equations. These solvers are more than suf cient to solve the problems in this text. It is recommended that you try `ode45` rst. If the equation proves dif cult to solve (as indicated by a lengthy solution time or by a warning or error message), then use `ode15s`.

In this section we limit our coverage to rst-order equations. Solution of higher-order equations is covered in Section 9.4. When used to solve the equation $\dot{y} = f(t, y)$, the basic syntax is (using `ode45` as the example)

```
[t,y] = ode45(@ydot, tspan, y0)
```

where `@ydot` is the handle of the function le whose inputs must be t and y, and whose output must be a column vector representing dy/dt, that is, $f(t, y)$. The number of rows in this column vector must equal the order of the equation. The syntax for `ode15s` is identical. The function le `ydot` may also be speci ed by a character string (i.e., its name placed in single quotes), but use of the function handle is now the preferred approach.

The vector `tspan` contains the starting and ending values of the independent variable t, and optionally any intermediate values of t where the solution is

desired. For example, if no intermediate values are speci ed, `tspan is [t0, tfinal]`, where `t0` and `tfinal` are the desired starting and ending values of the independent parameter t. As another example, using `tspan = [0, 5, 10]` tells MATLAB to nd the solution at $t = 5$ and at $t = 10$. You can solve equation backward in time by specifying `t0` to be greater than `tfinal`.

The parameter `y0` is the initial value y(0). The function le must have its rst two input arguments as `t` and `y` in that order, even for equations where $f(t, y)$ is not a function of t. You need not use array operations in the function le because the ODE solvers call the le with scalar values for the ar guments. The solvers may have an additional argument, `options`, which is discussed at the end of this section.

First consider an equation whose solution is known in closed form, so that we can make sure we are using the method correctly.

EXAMPLE 9.3–1 Response of an *RC* Circuit

The model of the *RC* circuit shown in Figure 9.3–1 can be found from Kirchhoff's voltage law and conservation of charge. It is $RC\dot{y} + y = v(t)$. Suppose the value of *RC* is 0.1 s. Use a numerical method to find the free response for the case where the applied voltage is zero and the initial capacitor voltage is $y(0) = 2$ V. Compare the results with the analytical solution, which is $y(t) = 2e^{-10t}$.

Solution

The equation for the circuit becomes $0.1\dot{y} + y = 0$. First solve this for $\dot{y}$: $\dot{y} = -10y$. Next define and save the following function file. Note that the order of the input arguments must be t and y even though t does not appear on the right-hand side of the equation.

```
function ydot = RC_circuit(t,y)
% Model of an RC circuit with no applied voltage.
ydot = -10*y;
```

The initial time is $t = 0$, so set `t0` to be 0. Here we know from the analytical solution that $y(t)$ will be close to 0 for $t \geq 0.5$ s, so we choose `tfinal` to be 0.5 s. In other problems we generally do not have a good guess for `tfinal`, so we must try several increasing values of `tfinal` until we see enough of the response on the plot.

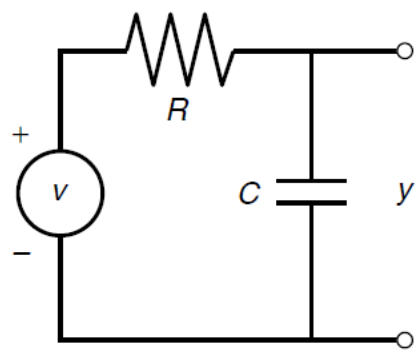


Figure 9.3–1 An *RC* circuit.

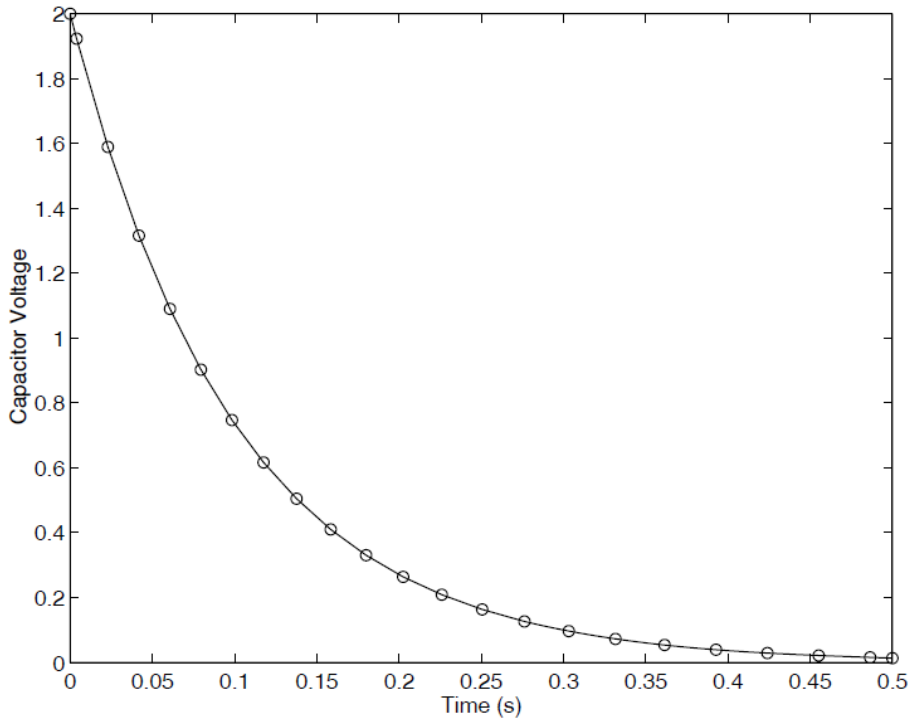


Figure 9.3–2 Free response of an *RC* circuit.

The function `ode45` is called as follows, and the solution plotted along with the analytical solution `y_true`.

```
[t, y] = ode45(@RC_circuit, [0, 0.5], 2);
y_true = 2*exp(-10*t);
plot(t,y,'o',t,y_true), xlabel('Time(s)'),...
   ylabel('Capacitor Voltage')
```

Note that we need not generate the array `t` to evaluate `y_true` because `t` is generated by the `ode45` function. The plot is shown in Figure 9.3–2. The numerical solution is marked by the circles, and the analytical solution is indicated by the solid line. Clearly the numerical solution gives an accurate answer. Note that the step size has been automatically selected by the `ode45` function.

T9.3–1 Use MATLAB to compute and plot the solution of the following equation.

$$10\frac{dy}{dt} + y = 20 + 7\sin 2t \qquad y(0) = 15$$

Practice

- The outbreak of an insect population can be modeled with the equation below.
- R=growth rate
- C=carrying capacity
- N=# of insects
- N_c=critical population
- Second term is due to bird predation

$$\frac{dN}{dt} = RN\left(1-\frac{N}{C}\right)-\frac{rN^2}{N_c^2+N^2}$$

Parameters

- 0<t<50 days
- R=0.55 /day
- N(0)=10,000
- C=10,000
- N_c=10,000
- r=10,000 /day

- What is steady state population?
- How long does it take to get there?

$$\frac{dN}{dt} = RN\left(1-\frac{N}{C}\right) - \frac{rN^2}{N_c^2 + N^2}$$

- Note: this is a first order ode

```
function insects
clear all
tr=[0 ??];
initv=??;
[t,y]=ode45(@f, tr, initv);
plot(t,y)
ylabel('Number of Insects')
xlabel('time')
%
function rk=f(t,y)
rk= ??;
```

Practice

- Let h be the depth of water in a spherical tank
- If we open a drain at the tank bottom, the pressure at the bottom will decrease as the tank empties, so the drain rate decreases with h
- Find the time to empty the tank

Parameters

- R=5 ft; Initial height=9 ft
- 1 inch hole for drain

$$\frac{dh}{dt} = -\frac{0.0334\sqrt{h}}{10h - h^2}$$

- How long does it take to drain the tank?

Rockets

- A rocket's mass decreases as it burns fuel
- Find the final velocity of a rocket if:
- T=48000 N; m_0=2200 kg
- R=0.8; g=9.81 m/s^2; b=40 s

$$m\frac{dv}{dt} = T - mg$$

$$m = m_0\left(1 - \frac{rt}{b}\right)$$

When the differential equation is nonlinear, we often have no analytical solution to use for checking our numerical results. In such cases we can use our physical insight to guard against grossly incorrect results. We can also check the equation for singularities that might affect the numerical procedure. Finally, we can sometimes use an approximation to replace the nonlinear equation with a linear one that can be solved analytically. Although the linear approximation does not give the exact answer, it can be used to see if our numerical answer is "in the ballpark." The following example illustrates this approach.

EXAMPLE 9.3–2 Liquid Height in a Spherical Tank

Figure 9.3–3 shows a spherical tank for storing water. The tank is lled through a hole in the top and drained through a hole in the bottom. If the tank's radius is r, you can use integration to show that the volume of water in the tank as a function of its height h is given by

$$V(h) = \pi r h^2 - \pi \frac{h^3}{3} \tag{9.3–10}$$

Torricelli's principle states that the liquid ow rate through the hole is proportional to the square root of the height h. Further studies in uid mechanics have identi ed the relation more precisely, and the result is that the volume ow rate through the hole is given by

$$q = C_d A \sqrt{2gh} \tag{9.3–11}$$

where A is the area of the hole, g is the acceleration due to gravity, and C_d is an experimentally determined value that depends partly on the type of liquid. For water, $C_d = 0.6$ is a common value. We can use the principle of conservation of mass to obtain a differential equation for the height h. Applied to this tank, the principle says that the rate of change of liquid volume in the tank must equal the ow rate out of the tank; that is,

$$\frac{dV}{dt} = -q \tag{9.3–12}$$

From Equation (9.3–10),

$$\frac{dV}{dt} = 2\pi r h \frac{dh}{dt} - \pi h^2 \frac{dh}{dt} = \pi h(2r - h) \frac{dh}{dt}$$

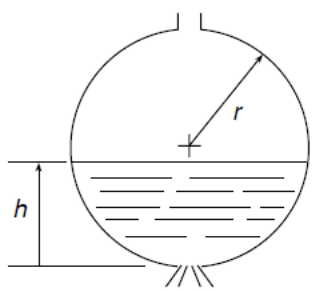


Figure 9.3–3 Draining of a spherical tank.

Substituting this and Equation (9.3–11) into Equation (9.3–12) gives the required equation for h.

$$\pi(2rh - h^2)\frac{dh}{dt} = -C_d A\sqrt{2gh} \tag{9.3–13}$$

Use MATLAB to solve this equation to determine how long it will take for the tank to empty if the initial height is 9 ft. The tank has a radius of $r = 5$ ft and has a 1-in.-diameter hole in the bottom. Use $g = 32.2$ ft/sec^2. Discuss how to check the solution.

Solution:

With $C_d = 0.6$, $r = 5$, $g = 32.2$, and $A = \pi(1/24)^2$, Equation (9.3–13) becomes

$$\frac{dh}{dt} = -\frac{0.0334\sqrt{h}}{10h - h^2} \tag{9.3–14}$$

We can first check the above expression for dh/dt for singularities. The denominator does not become zero unless $h = 0$ or $h = 10$, which correspond to a completely empty and a completely full tank. So we will avoid singularities if $0 < h < 10$.

Finally, we can use the following approximation to estimate the time to empty. Replace h on the right side of Equation (9.3–14) with its average value, namely, $(9 - 0)/2 = 4.5$ ft. This gives $dh/dt = -0.00286$, whose solution is $h(t) = h(0) - 0.00286t = 9 - 0.00286t$. According to this equation, the tank will be empty at $t = 9/0.00286 = 3147$ sec, or 52 min. We will use this value as a "reality check" on our answer.

The function file based on Equation (9.3–14) is

```
function hdot = height(t,h)
hdot = -(0.0334*sqrt(h))/(10*h-h^2);
```

The file is called as follows, using the `ode45` solver.

```
[t, h]=ode45 (@height, [0, 2475], 9);
plot(t,h),xlabel('Time (sec)'), ylabel('Height (ft)')
```

The resulting plot is shown in Figure 9.3–4. Note how the height changes more rapidly when the tank is nearly full or nearly empty. This is to be expected because of the effects of the tank's curvature. The tank empties in 2475 sec, or 41 min. This value is not grossly different from our rough estimate of 52 min, so we should feel comfortable accepting the numerical results. The value of the final time of 2475 sec was found by increasing the final time until the plot showed that the height became 0.

Higher-Order Differential Equations

To use the ODE solvers to solve an equation higher than order 1, you must rst write the equation as a set of rst-order equations. This is easily done. Consider the second-order equation

$$5\ddot{y} + 7\dot{y} + 4y = f(t) \qquad (9.4\text{–}1)$$

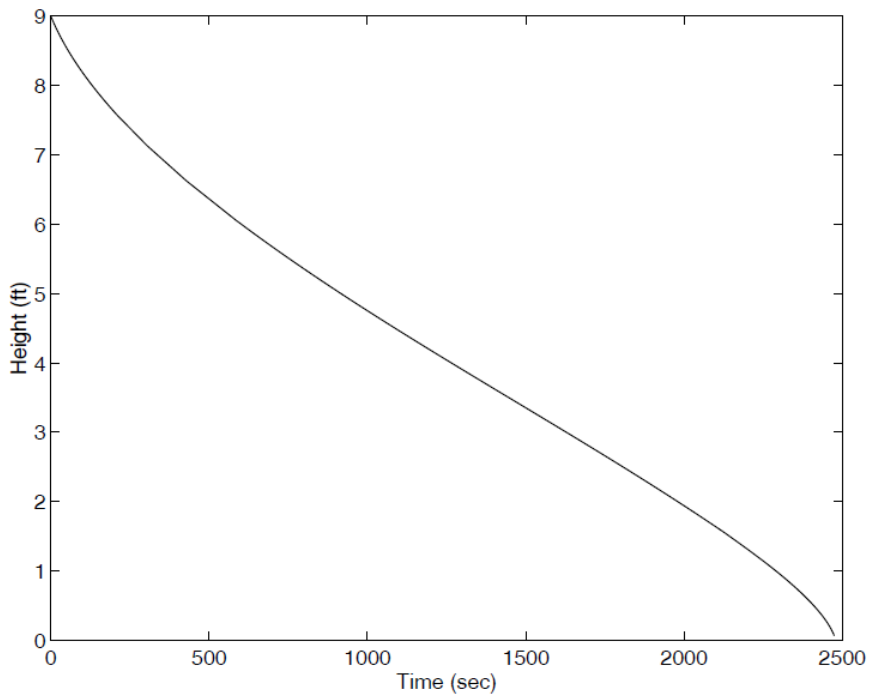


Figure 9.3–4 Plot of water height in a spherical tank.

Solve it for the highest derivative:

$$\ddot{y} = \frac{1}{5}f(t) - \frac{4}{5}y - \frac{7}{5}\dot{y} \tag{9.4–2}$$

De ne two new variables x_1 and x_2 to be y and its derivative $\dot{y}$. That is, de ne $x_1 = y$ and $x_2 = \dot{y}$. This implies that

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{5}f(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

A Nonlinear Pendulum Model

EXAMPLE 9.4–1

The pendulum shown in Figure 9.4–1 consists of a concentrated mass m attached to a rod whose mass is small compared to m. The rod's length is L. The equation of motion for this pendulum is

$$\ddot{\theta} + \frac{g}{L}\sin\theta = 0 \tag{9.4–3}$$

Suppose that $L = 1$ m and $g = 9.81$ m/s^2. Use MATLAB to solve this equation for $\theta(t)$ for two cases: $\theta(0) = 0.5$ rad and $\theta(0) = 0.8\pi$ rad. In both cases $\dot{\theta}(0) = 0$. Discuss how to check the accuracy of the results.

Solution

If we use the small-angle approximation $\sin \approx \theta$, the equation becomes

$$\ddot{\theta} + \frac{g}{L}\theta = 0 \tag{9.4–4}$$

which is linear and has the solution

$$\theta(t) = \theta(0)\cos\sqrt{\frac{g}{L}}\,t \tag{9.4–5}$$

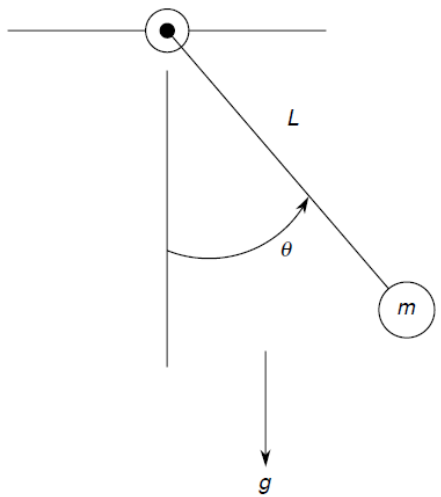


Figure 9.4–1 A pendulum.

if $\dot{\theta}(0) = 0$. Thus the amplitude of oscillation is $\theta(0)$, and the period is $P = 2\pi\sqrt{L/g} = 2.006$ s. We can use this information to select a final time and to check our numerical results.

First rewrite the pendulum equation (9.4–3) as two first-order equations. To do this, let $x_1 = \theta$ and $x_2 = \dot{\theta}$. Thus

$$\dot{x}_1 = \dot{\theta} = x_2$$
$$\dot{x}_2 = \ddot{\theta} = -\frac{g}{L}\sin x_1$$

The following function file is based on the last two equations. Remember that the output `xdot` must be a *column* vector.

```
function xdot = pendulum(t,x)
g = 9.81; L = 1;
xdot = [x(2); -(g/L)*sin(x(1))];
```

This file is called as follows. The vectors `ta` and `xa` contain the results for the case where $\theta(0) = 0.5$. In both cases, $\dot{\theta}(0) = 0$. The vectors `tb` and `xb` contain the results for $\theta(0) = 0.8\pi$.

```
[ta, xa] = ode45(@pendulum, [0,5], [0.5, 0];
[tb, xb] = ode45(@pendulum, [0,5], [0.8*pi, 0];
plot(ta, xa(:,1), tb,xb(:,1)), xlabel ('Time (s)'), . . .
   ylabel('Angle (rad)'), gtext('Case 1'), gtext('Case 2')
```

The results are shown in Figure 9.4–2. The amplitude remains constant, as predicted by the small-angle analysis, and the period for the case where $\theta(0) = 0.5$ is a little larger than 2 s, the value predicted by the small-angle analysis. So we can place some confidence in the numerical procedure. For the case where $\theta(0) = 0.8\pi$, the period of the

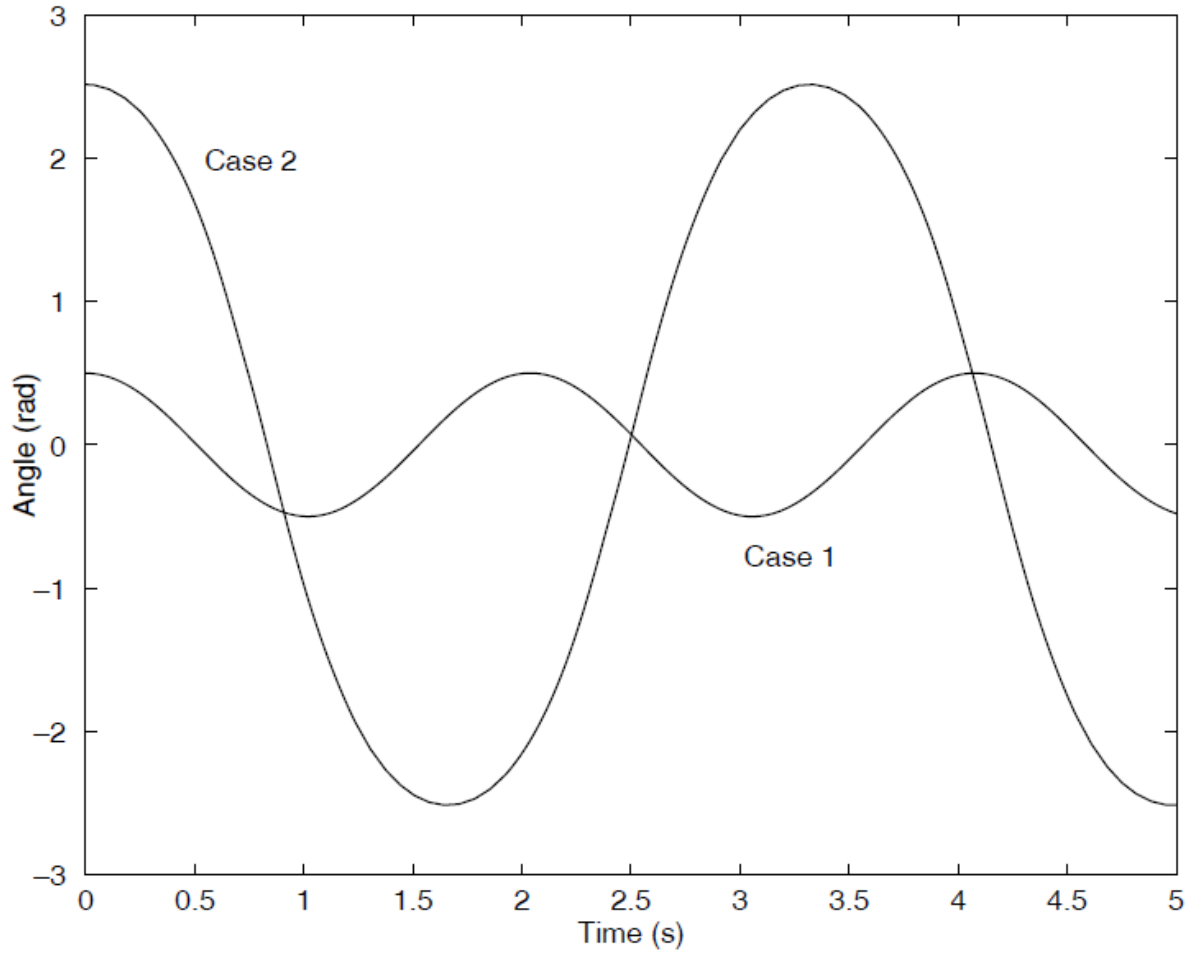


Figure 9.4–2 The pendulum angle as a function of time for two starting positions.

numerical solution is about 3.3 s. This illustrates an important property of nonlinear differential equations. The free response of a linear equation has the same period for any initial conditions; however, the form and therefore the period of the free response of a nonlinear equation often depend on the particular values of the initial conditions.

Table 9.4–1 Syntax of the ODE solver `ode45`

Command	Description
`[t, y] = ode45(@ydot, tspan, y0, options)`	Solves the vector differential equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ speci ed by the function le whose handle is `@ydot` and whose inputs must be t and $\mathbf{y}$, and whose output must be a *column* vector representing $d\mathbf{y}/dt$; that is, $\mathbf{f}(t, \mathbf{y})$. The number of rows in this column vector must equal the order of the equation. The vector `tspan` contains the starting and ending values of the independent variable t, and optionally any intermediate values of t where the solution is desired. The vector `y0` contains the initial values. The function le must have two input arguments, `t` and `y`, even for equations where $f(t, y)$ is not a function of t. The `options` argument is created with the `odeset` function. The syntax is identical for the solver `ode15s`.
`options = odeset ('name1', 'value1' 'name2', 'value2', . . .)`	Creates an integrator options structure `options` to be used with the ODE solver, in which the named properties have the speci ed values, where `name` is the name of a *property* and `value` is the value to be assigned to the property. Any unspeci ed properties have default values. Typing `odeset` with no input arguments displays all property names and their possible values.

Second Order Equations

- Consider a falling object with drag

$$\ddot{y} = -g - \frac{4}{15m}\dot{y}|\dot{y}|$$

$$y(0) = h$$

$$\dot{y}(0) = 0$$

Preparing for solution

- We must break second order equation into set of first order equations
- We do this by introducing new variable (z=dy/dt)

$$z = \dot{y}$$

$$\dot{z} = \ddot{y}$$

$$\dot{z} = -\frac{4}{15m} z|z| - g$$

$$\dot{z} = -\frac{4}{15m} z|z| - g$$

$$\dot{y} = z$$

$$y(0) = h; \quad z(0) = 0$$

Solving

- Now we have to send a set of equations and a set of initial values to the ode45 routine
- We do this via vectors
- Let w be vector of solutions: w(1)=y and w(2)=z
- Let r be vector of equations: r(1)=dy/dt and r(2)=dz/dt

Function to Define Equation

$$\frac{dy}{dt} = z = w(2)$$

$$\frac{dz}{dt} = -\frac{4}{15m} w(2) * |w(2)| - g$$

```
function r=rkfalling(t,w)
...
r=zeros(2,1);
r(1)=w(2);
r(2)= -k*w(2).*abs(w(2))-g;
```

The Routines

```
tr=[0 15]; %seconds
initv=[600 0]; %start 600 m high
[t,y]=ode45(@rkfalling, tr, initv)
plot(t,y(:,1))
ylabel('x (m)')
xlabel('time(s)')
figure
plot(t,y(:,2))
ylabel('velocity (m/s)')
xlabel('time(s)')
```

Function

```
function r=rkfalling(t,w)
mass=80;
k=4/15/mass;
g=9.81;
r=zeros(2,1);
r(1)=w(2);
r(2)= -k*w(2).*abs(w(2))-g;
```

General Second Order Equations

- We can write a general second order equation as shown:
- To solve:
  - Define f
  - Set initial conditions
  - Set time range

$$\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)$$

or

$$\frac{dy}{dt} = z$$

$$\frac{dz}{dt} = f(t, y, z)$$

The Routines

```
tr=[0 15]; %seconds
initv=[600 0]; %start 600 m high
[t,y]=ode45(@rkfalling, tr, initv)
plot(t,y(:,1))
ylabel('x (m)')
xlabel('time(s)')
figure
plot(t,y(:,2))
ylabel('velocity (m/s)')
xlabel('time(s)')
```



```
function r=rkfalling(t,w)
mass=80;
k=4/15/mass;
g=9.81;
r=zeros(2,1);
r(1)=w(2);
r(2)= -k*w(2).*abs(w(2))-g;
```

```
clear all
tr=[0 15]; %seconds
initv=[600 0]; %start 600 m high
[t,y]=ode45(@rkfalling, tr, initv);
plot(t,y(:,1))
ylabel('x (m)')
xlabel('time(s)')
figure
plot(t,y(:,2))
ylabel('velocity (m/s)')
xlabel('time(s)')

function r=rkfalling(t,w)
mass=80;
k=4/15/mass;
g=9.81;
r=zeros(2,1);
r(1)=w(2);
r(2)= k*w(2)^2-g;
```

Practice

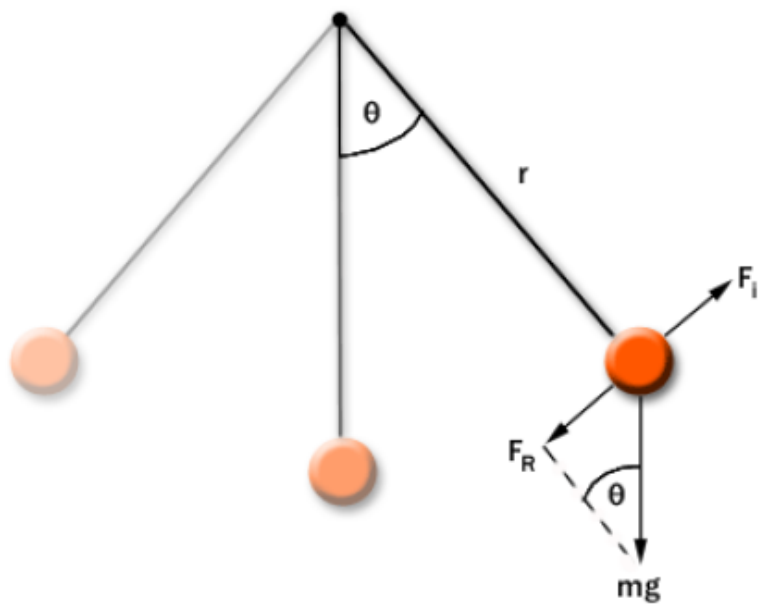
- Run to duplicate earlier results for velocity
- Change initial velocity to 10 m/s and run again

$$m\frac{d^2y}{dt^2} = -mg - \frac{4}{15}\frac{dy}{dt}\left|\frac{dy}{dt}\right|$$

Practice-nonlinear pendulum

- r=1 m; g=9.81 m/s^2
- Initial angle =π/8, π/2, π-0.1

$$\frac{d^2\theta}{dt^2} = -\frac{g}{r}\sin(\theta)$$



```
%pendulum problem
tr=[0 5]; %seconds
init=[pi/8 0]; %start at pi/8
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
ylabel('theta')
xlabel('time(s)')
hold on
init=[pi/2 0]; %start at pi/2
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
init=[pi-0.1 0]; %start at pi-0.1
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
hold off
disp('press any key to continue')
pause
```

```
function p=rkpend(t,w)
r=1;
g=9.81;
p=zeros(2,1);
p(1)=w(2);
p(2)= -g/r*sin(w(1));
```

Systems

- For systems of first order ODEs, just define both equations.

Practice

- Consider an ecosystem of rabbits r and foxes f. Rabbits are fox food.
- Start with 300 rabbits and 150 foxes
- α=0.01

$$\frac{dr}{dt} = 2r - \alpha rf$$

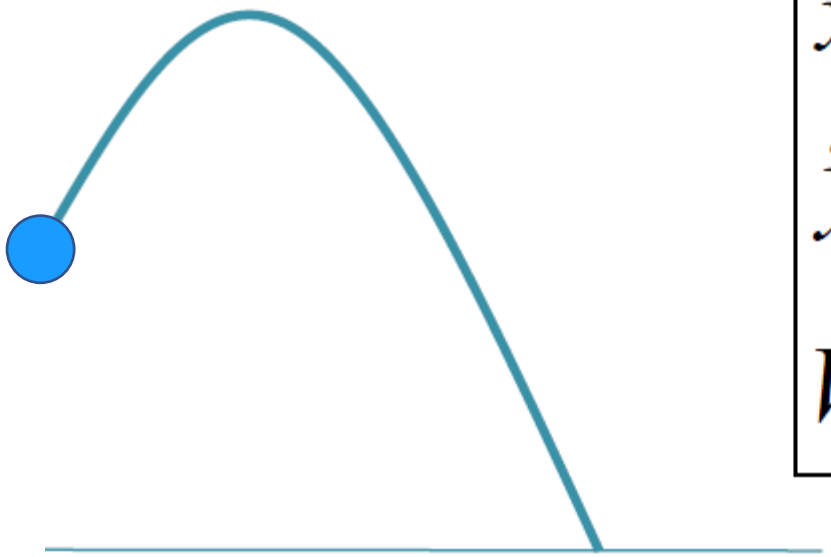
$$\frac{df}{dt} = -f + \alpha rf$$

- r=w(1)
- f=w(2)

```
function z=rkfox(t,w)
alpha=0.01;
r=zeros(2,1);
z(1)=2*w(1)-alpha*w(1)*w(2);
z(2)= -w(2)+alpha*w(1)*w(2);
```

Higher Order Equations

- Suppose we want to model a projectile



$$\ddot{x} = -k\,\dot{x}V$$

$$\ddot{y} = -k\,\dot{y}V - g$$

$$V = \sqrt{\dot{x}^2 + \dot{y}^2}$$

Now we need 4 1st order ODEs

$$\dot{x} = s$$

$$\dot{s} = -k\, s\, V$$

$$\dot{y} = z$$

$$\dot{z} = -k\, z\, V - g$$

$$V = \sqrt{s^2 + z^2}$$

The Code

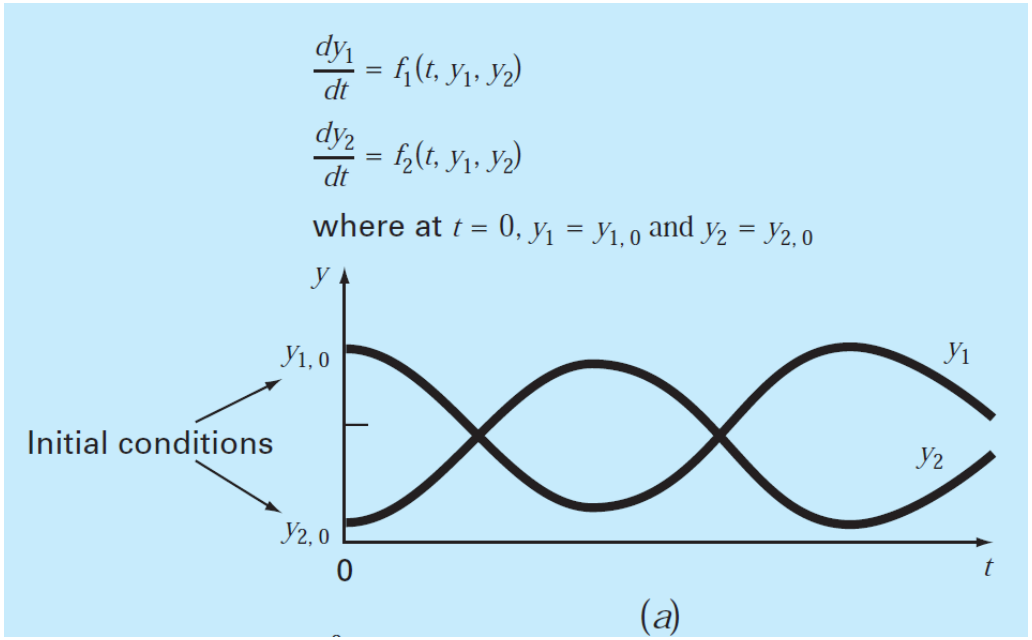
```
clear all;
tspan=[0 1.1]
wnot(1)=0; wnot(2)=10;
wnot(3)=0; wnot(4)=10;
[t,y]=ode45('rkprojectile',tspan,wnot);
plot(t,y(:,1),t,y(:,3))
figure
plot(y(:,1),y(:,3))
```

The Function

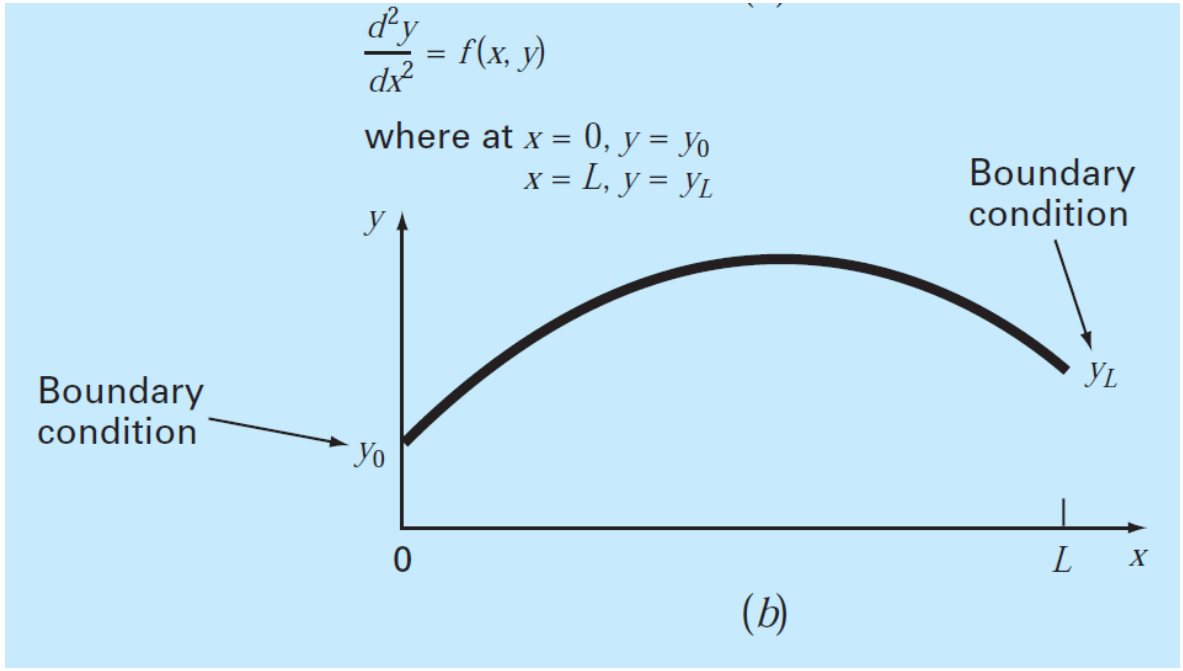
```
function r=rkprojectile(t,w)
g=9.8 l;
x=w(l); s=w(2); y=w(3); z=w(4);
vel=sqrt(s.^2+z.^2);
r=zeros(4,l);
r(l)=s;
r(2)=-s*vel;
r(3)=z;
r(4)=-z*vel-g;
```

Boundary Value Problems

Initial-value vs. boundary value problems



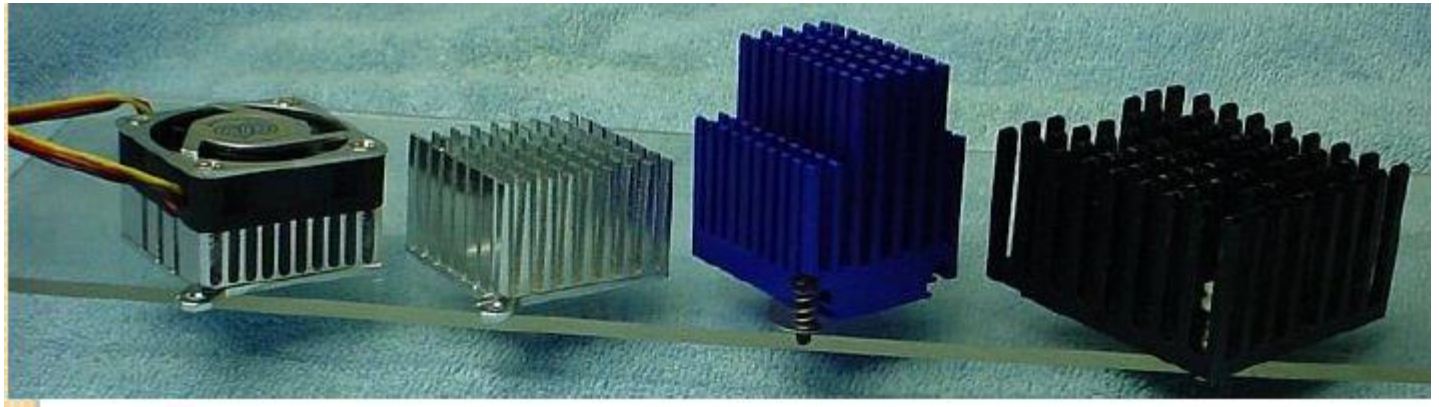
An initial-value problem where all the conditions are specified at the same value of the independent variable



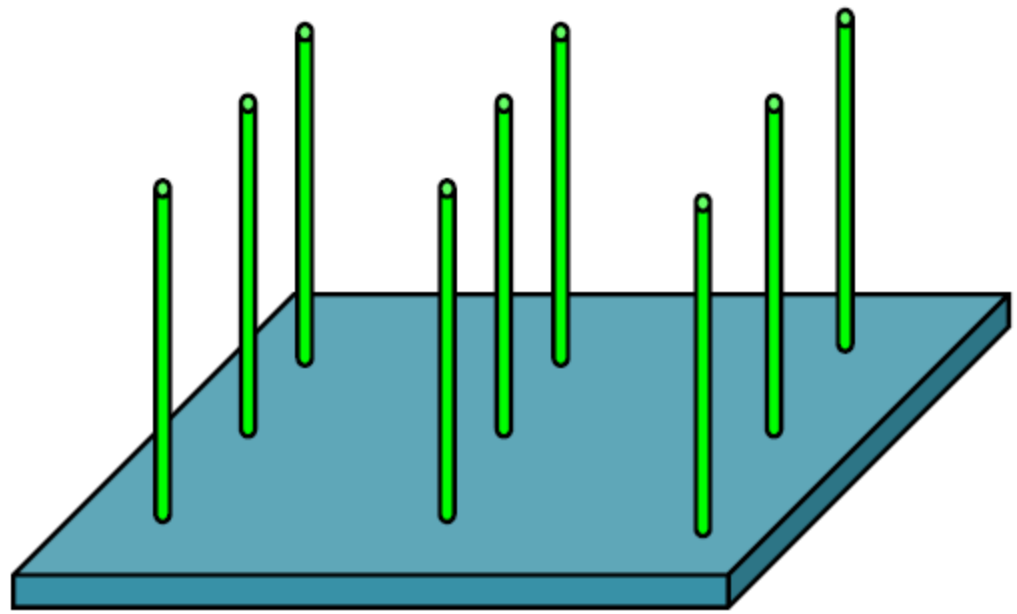
A boundary-value problem where the conditions are specified at different values of the independent variable

Case Study

- We will analyze a cooling configuration for a computer chip
- We increase cooling by adding a number of fins to the surface
- These are high conductivity (aluminum) pins which provide added surface area



The Case - Schematic



The Case - Modeling

- The temperature distribution in the pin is governed by:

$$\frac{d^2T}{dx^2} - \frac{hC}{kA}(T - T_f) = 0$$

$$T(0) = 40\,C$$

$$\left.\frac{dT}{dx}\right|_{x=L} = 0$$

Finite Difference Techniques

- Used to solve boundary value problems
- We'll look at an example

$$\frac{d^2y}{dx^2} + y = 1$$

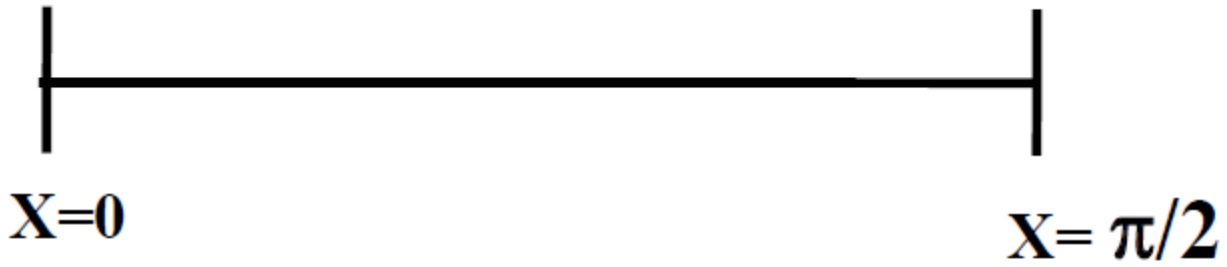
$$y(0) = 1$$

$$y(\frac{\pi}{2}) = 0$$

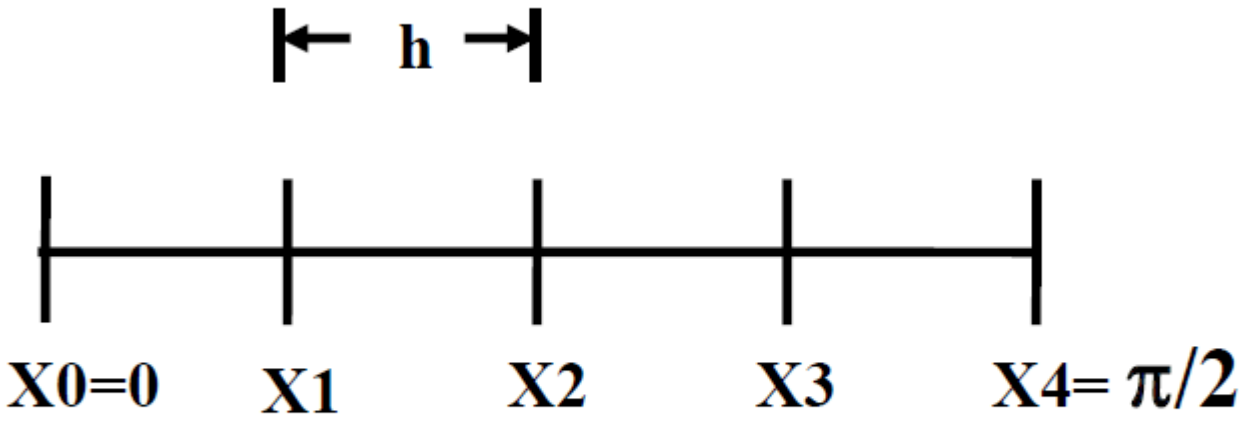
Two Steps

- Divide interval into steps
- Write differential equation in terms of values at these discrete points

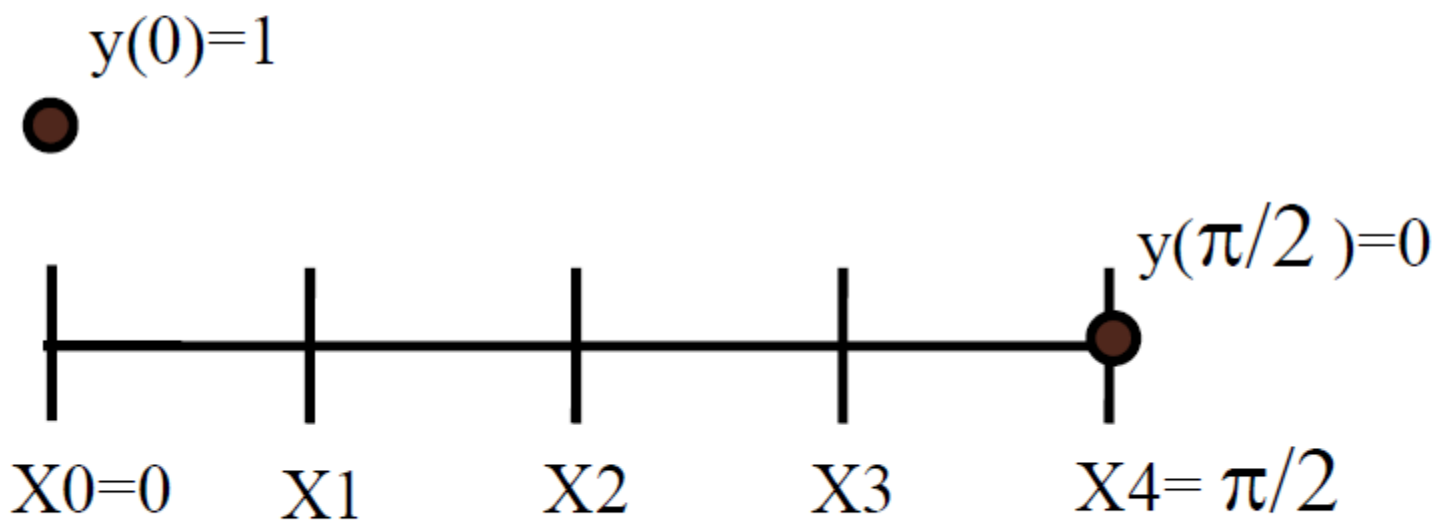
Solution is Desired from x=0 to π/2



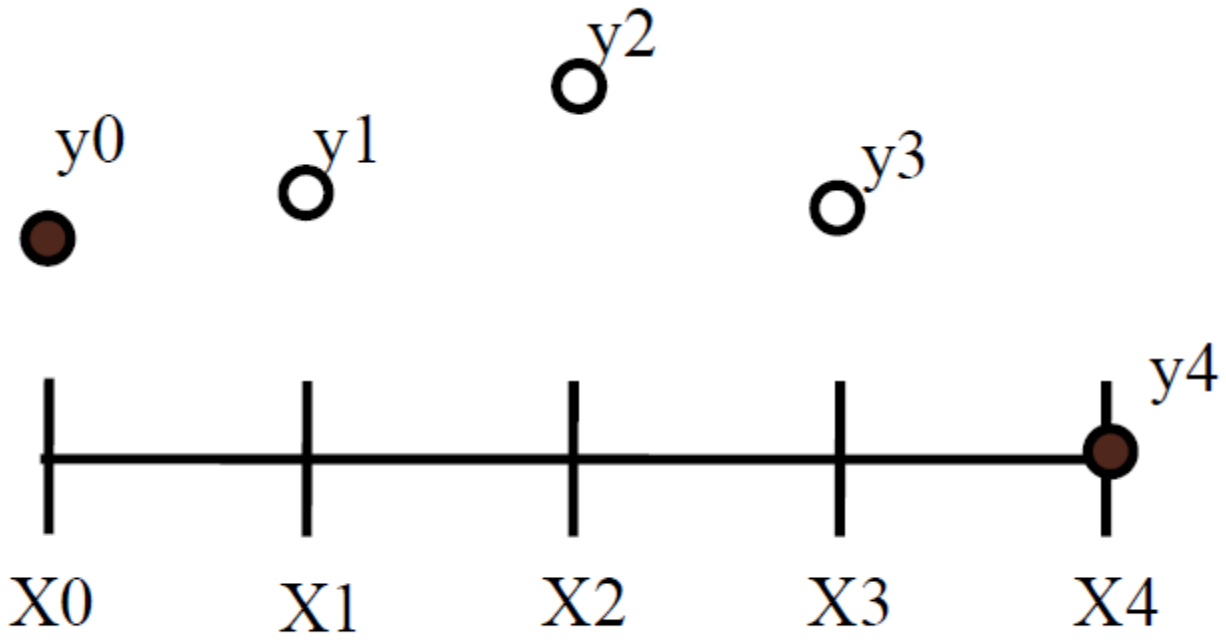
Divide Interval into Pieces



Boundary Values



Calculate Internal Values



Forward Finite Divided Difference Method

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2}h^2 + \cdots$$

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

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

$$f''(x_i) = \frac{f(x_{i+2}) - 2\,f(x_{i+1}) + f(x_i)}{h^2} + O(h)$$

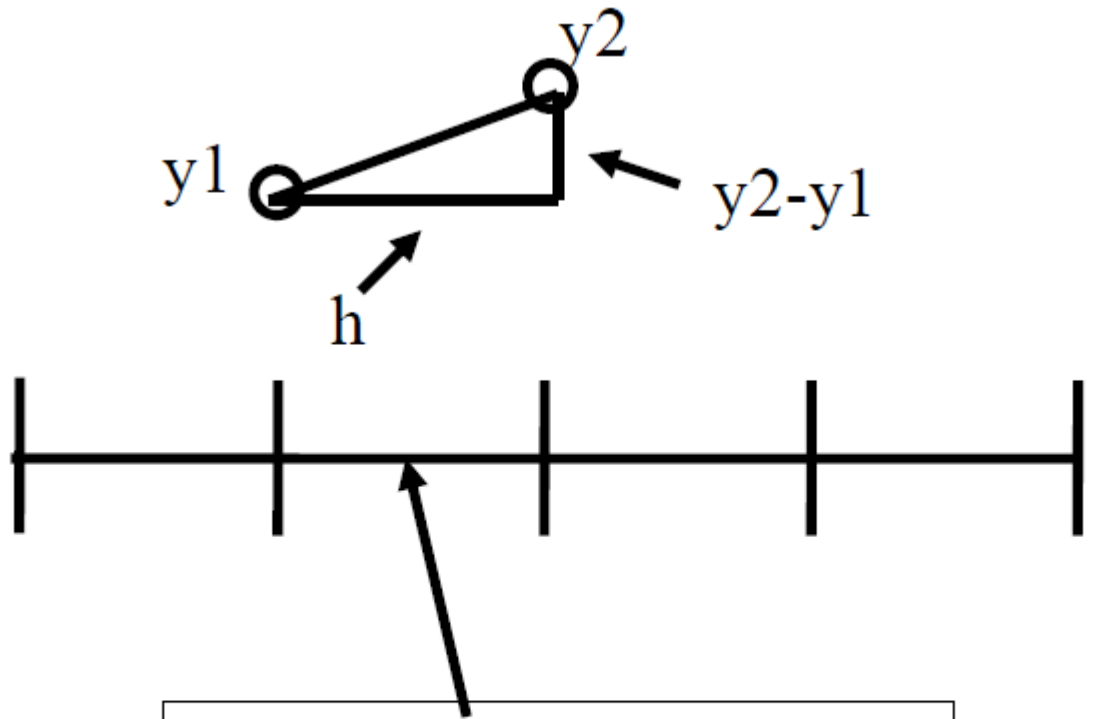
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2\,f(x_{i+1}) + f(x_i)}{2h^2}h + O(h^2)$$

Centered Finite Divided Difference Method

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \qquad O(h^2)$$

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} \qquad O(h^2)$$

Approximation
s



$$\left.\frac{dy}{dx}\right|_{i+1/2} \approx \frac{y_{i+1} - y_i}{h}$$

Second Derivative

$$\left.\frac{d^2y}{dx^2}\right|_i \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

Substitute

$$\frac{d^2y}{dx^2} + y = 1$$

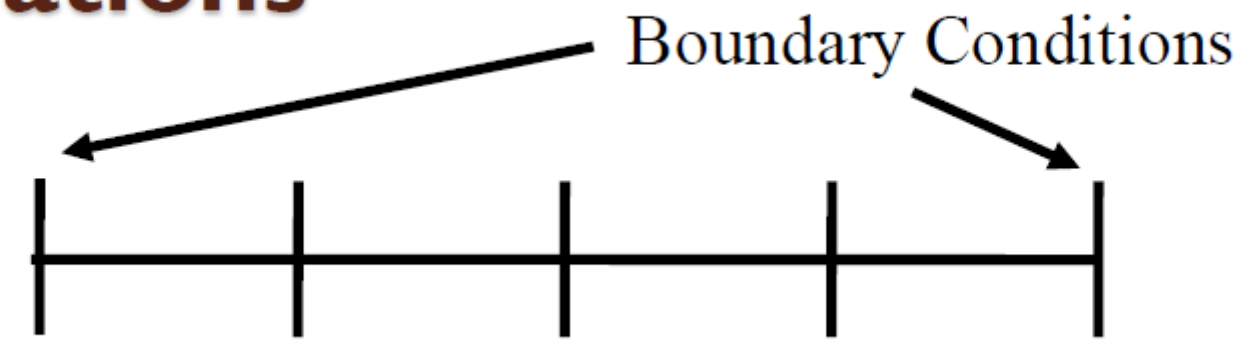
becomes

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + y_i = 1$$

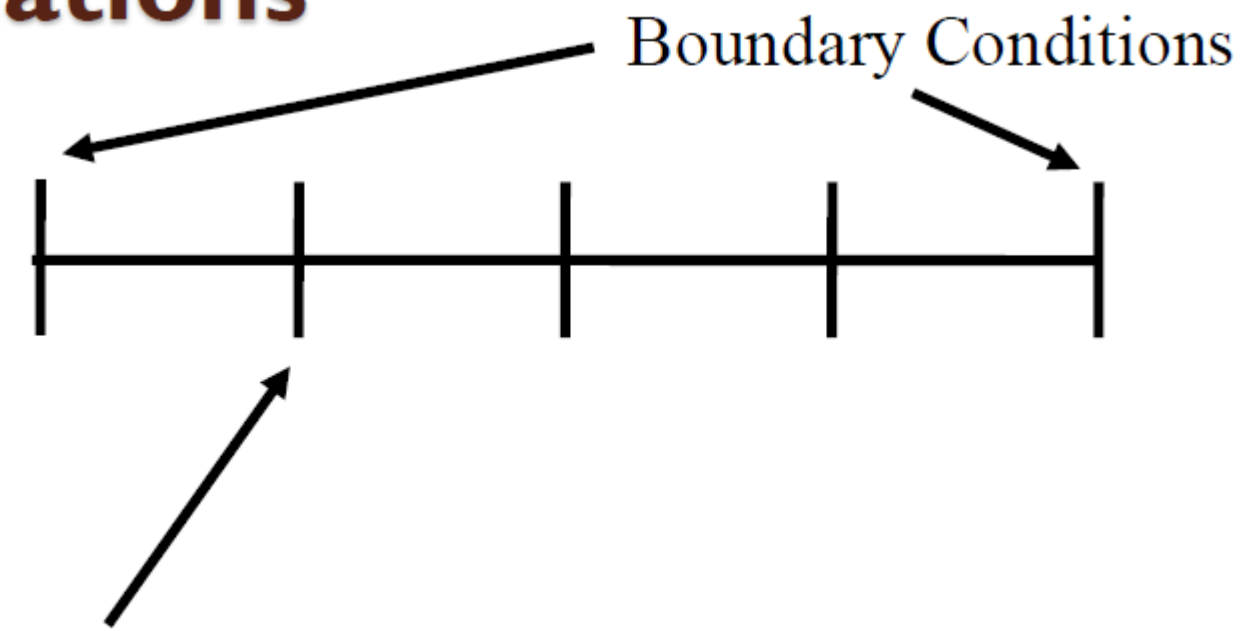
or

$$y_{i-1} - (2 - h^2)y_i + y_{i+1} = h^2$$

Equations

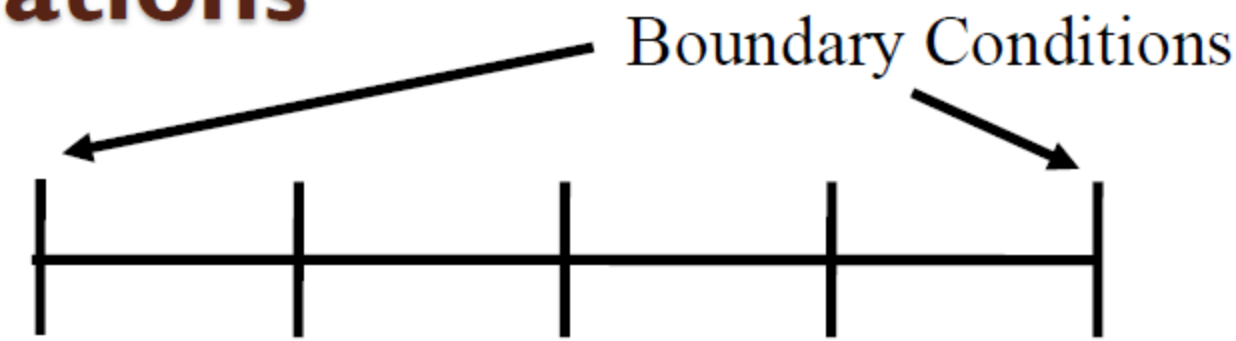


Equations



$$y_2 - 2y_1 + y_0 + h^2 y_1 = h^2$$

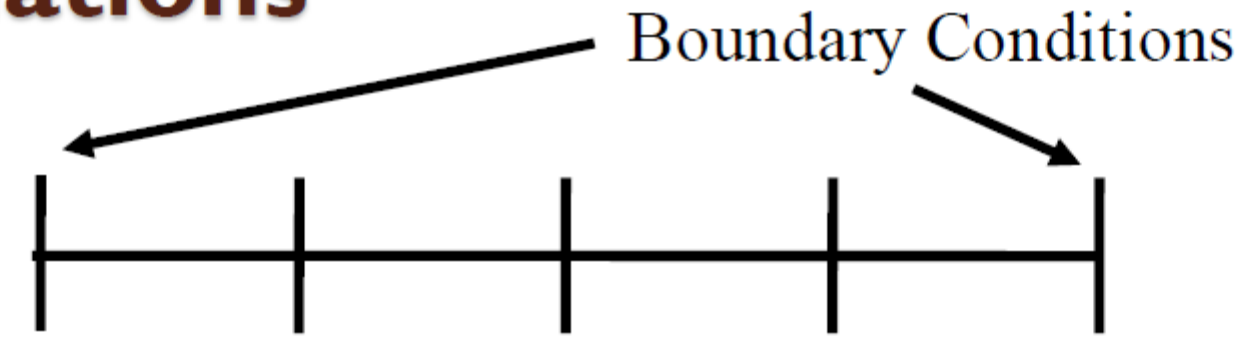
Equations



$$y_2 - 2y_1 + y_0 + h^2 y_1 = h^2$$

$$y_3 - 2y_2 + y_1 + h^2 y_2 = h^2$$

Equations



$$y_2 - 2y_1 + y_0 + h^2 y_1 = h^2$$

$$y_4 - 2y_3 + y_2 + h^2 y_3 = h^2$$

$$y_3 - 2y_2 + y_1 + h^2 y_2 = h^2$$

Using Matlab

$$
\begin{aligned}
&y_0 = 1 \\
&y_0 - (2 - h^2) y_1 + y_2 = h^2 \\
&y_1 - (2 - h^2) y_2 + y_3 = h^2 \\
&y_2 - (2 - h^2) y_3 + y_4 = h^2 \\
&y_4 = 0
\end{aligned}
$$

Convert to matrix and solve

Using Matlab

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -(2-h^2) & 1 & 0 & 0 \\ 0 & 1 & -(2-h^2) & 1 & 0 \\ 0 & 0 & 1 & -(2-h^2) & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ h^2 \\ h^2 \\ h^2 \\ 0 \end{Bmatrix}$$

The Script

```
h=pi/2/4;
A=[1 0 0 0 0; 1 -(2-h^2) 1 0 0; 0 1 -(2-h^2) 1 0;
   ...
    0 0 1 -(2-h^2) 1; 0 0 0 0 1];
b=[1; h^2; h^2; h^2; 0];
x=linspace(0,pi/2,5);
y=A\b;
plot(x,y)
```

What if we use N divisions

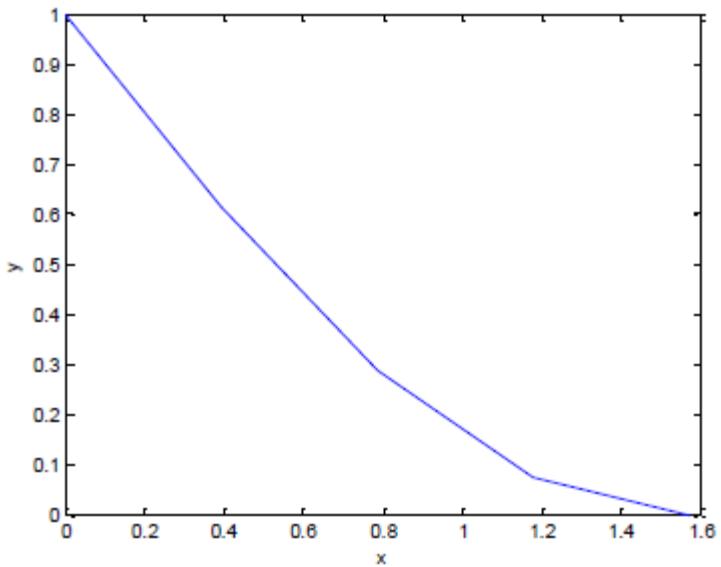
- N divisions, N+1 mesh points
- Matrix is N+1 by N+1

$$h = \frac{\pi/2}{N}$$

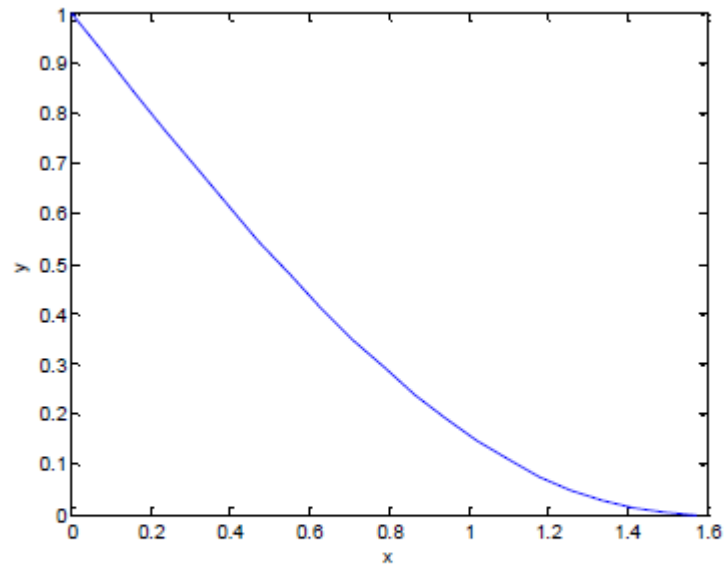
The Script

```
N=4;
h=pi/2/N;
A=-(2-h^2)*eye(N+1);
A(1,1)=1;
A(N+1,N+1)=1;
for i=1:N-1
   A(i+1,i)=1;
   A(i+1,i+2)=1;
end
b=h^2*ones(N+1,1);
b(1)=1;
b(N+1)=0;
x=linspace(0,pi/2,N+1);
y=A\b;
plot(x,y)
```

Results



N=4



N=20

Script Using Diag

- The diag command allows us to put a vector on the diagonal of a matrix
- We can use this to put in the "1's" just off the diagonal in this matrix
- Syntax: **diag(V,K)** - V is the vector, K tells which diagonal to place the vector in

New Script

```
A=-(2-h^2)*eye(N+1)+diag(v,1)+diag(v,-1);
A(1,2)=0;
A(N+1,N)=0;
A(1,1)=1;
A(N+1,N+1)=1;
```

A Built-In Routine

- Matlab includes **bvp4c**
- This carries out finite differences on systems of ODEs
- **SOL = BVP4C(ODEFUN,BCFUN,SOLINIT)**
  - odefun defines ODEs
  - bcfun defines boundary conditions
  - solinit gives mesh (location of points) and guess for solutions (guesses are constant over mesh)

Using bvp4c

- odefun is a function, much like what we used for ode45
- bcfun is a function that provides the boundary conditions at both ends
- solinit created in a call to the bvpinit function and is a vector of guesses for the initial values of the dependent variable

Preparing our Equation

- Let y be variable 1 – y(1)
- Then dy/dx (=z) is variable 2 – y(2)

$$\frac{d^2y}{dx^2} + y = 1$$

$$\frac{dy}{dx} = z = y(2)$$

$$\frac{d^2y}{dx^2} = \frac{dz}{dx} = 1 - y(1)$$

$$y(0) = 1$$

$$y(\frac{\pi}{2}) = 0$$

```
function dydx = bvp4ode(x,y)
dydx = [  y(2)    1-y(1) ];
```

Boundary Conditions

- ya(1) is y(1) at x=a
- ya(2) is y(2) at x=a
- yb(1) is y(1) at x=b
- yb(2) is y(2) at x=b
- In our case, y(1)-1=0 at x=a and y(1)=0 at x=b

```
function res = bvp4bc(ya,yb)
    res = [  ya(1)-1    yb(1) ];
```

Initialization

- How many mesh points? **10**
- Initial guesses for y(1) and y(2)
- Guess y=1, z=-1
- Guess more critical for nonlinear equations

```
xlow=0;
xhigh=pi/2;
solinit =
  bvpinit(linspace(xlow,xhigh,10),[1 -1]);
```

Postprocessing

```
xint = linspace(xlow,xhigh);
Sxint = deval(sol,xint);
plot(xint,Sxint(1,:))
```

The Script

```
function bvp4
xlow=0;  xhigh=pi/2;
solinit = bvpinit(linspace(xlow,xhigh,10),[1 -1]);
sol = bvp4c(@bvp4ode,@bvp4bc,solinit);
xint = linspace(xlow,xhigh);
Sxint = deval(sol,xint);
plot(xint,Sxint(1,:))
% --------------------------------------------
function dydx = bvp4ode(x,y)
dydx = [  y(2)        1-y(1) ];
% --------------------------------------------
function res = bvp4bc(ya,yb)
res = [  ya(1)-1        yb(1) ];
```

Things to Change for Different Problems

```
function bvp4
xlow=0;  xhigh=pi/2;
solinit = bvpinit(linspace(xlow,xhigh,10),[1 -1]);
sol = bvp4c(@bvp4ode,@bvp4bc,solinit);
xint = linspace(xlow,xhigh,20);
Sxint = deval(sol,xint);
plot(xint,Sxint(1,:))
% ---------------------------------------
function dydx = bvp4ode(x,y)
dydx = [  y(2)        1-y(1) ];
% ---------------------------------------
function res = bvp4bc(ya,yb)
res = [  ya(1)-1        yb(1) ];
```

bvpskeleton.m

```
xlow=???;
xhigh=???;
solinit = bvpinit(linspace(xlow,xhigh,20),[1 0]);
sol = bvp4c(@bvp4ode,@bvp4bc,solinit);
xint = linspace(xlow,xhigh);
Sxint = deval(sol,xint);
eps=0.1;
analyt=(exp(xint/eps)-1)/(exp(1/eps)-1);
plot(xint,Sxint(1,:),xint,analyt,'r')
% ---------------------------------------------
function dydx = bvp4ode(x,y)
eps=0.1;
dydx = [  ???? ; ???? ];
% ---------------------------------------------
function res = bvp4bc(ya,yb)
res = [  ???? ;   ???? ];
```

Practice

Modify the code
bvpskeleton. m
and solve the boundary value problem shown at the right for ε=0.1 and compare to the analytical solution

$$\varepsilon \frac{d^2y}{dx^2} - \frac{dy}{dx} = 0$$

$$y(0) = 0$$

$$y(1) = 1$$

$$y_{analytical} = \frac{e^{x/\varepsilon} - 1}{e^{1/\varepsilon} - 1}$$

What about BCs involving derivatives?

- If we prescribe a derivative at one end, we cannot just place a value in a cell.
- We'll use finite difference techniques to generate a formula
- The formulas work best when "centered", so we will use a different approximation for the first derivative.

Derivative BCs

- Consider a boundary condition of the form dy/dx=0 at x=L
- Finite difference (centered) is:

$$\frac{dy}{dx} \approx \frac{y_{i+1} - y_{i-1}}{2h} = 0$$

or

$$y_{i+1} = y_{i-1}$$

Derivative BCs

- So at a boundary point on the right we just replace y_{i+1} with y_{i-1} in the formula
- Consider:

$$\frac{d^2y}{dx^2} + y = 0$$

$$y(0) = 1$$

$$\frac{dy}{dx}(1) = 0$$

Finite Difference Equation

- We derive a new difference equation

$$\frac{d^2y}{dx^2} + y = 0$$

becomes

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + y_i = 0$$

or

$$y_{i-1} - (2 - h^2)y_i + y_{i+1} = 0$$

Derivative BCs

- **The difference equation at the last point is**

$$y_{N-1} - \left(2 - h^2\right) y_N + y_{N+1} = 0$$

but

$$y_{N-1} = y_{N+1}$$

so

$$2y_{N-1} - \left(2 - h^2\right) y_N = 0$$

Final Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -(2-h^2) & 1 & 0 & 0 \\ 0 & 1 & -(2-h^2) & 1 & 0 \\ 0 & 0 & 1 & -(2-h^2) & 1 \\ 0 & 0 & 0 & 2 & -(2-h^2) \end{bmatrix} \begin{Bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

New Code

```
h=1/4
A=[1 0 0 0 0;
    1 -(2-h^2) 1 0 0;
    0 1 -(2-h^2) 1 0;
    0 0 1 -(2-h^2) 1;
    0 0 0 2 -(2-h^2)]
B=[1; 0; 0; 0; 0]
y=A\B
```

Using bvp4c

- The boundary condition routine allows us to set the derivative of the dependent variable at the boundary

Preparing Equation

$$\frac{d^2y}{dx^2} + y = 0$$

$$\frac{dy}{dx} = z = y(2)$$

$$\frac{d^2y}{dx^2} = \frac{dz}{dx} = -y(1)$$

$$y(0) = 1$$

$$\frac{dy}{dx}(1) = 0$$

so

$$ya(1) = 1$$

$$yb(2) = 0$$

or

$$ya(1) - 1 = 0$$

$$yb(2) = 0$$

The Script

```
function bvp5
xlow=0;  xhigh=1;
solinit = bvpinit(linspace(xlow,xhigh,10),[1 -1]);
sol = bvp4c(@bvp5ode,@bvp5bc,solinit);
xint = linspace(xlow,xhigh);
Sxint = deval(sol,xint);
plot(xint,Sxint(1,:))
% ----------------------------------------------
function dydx = bvp5ode(x,y)
dydx = [  y(2)        -y(1) ];
% ----------------------------------------------
function res = bvp5bc(ya,yb)
res = [  ya(1)-1        yb(2) ];
```

Practice

- Solve the Case Study Problem
- Use L=25 mm, T_f=20 C, and hC/kA=4000 /m^2
- JPB: need a skeleton here

$$\frac{d^2T}{dx^2} - \frac{hC}{kA}(T - T_f) = 0$$

$$T(0) = 40\,C$$

$$\left.\frac{dT}{dx}\right|_{x=L} = 0$$

Practice

- A 1 W, 2 Mohm resistor which is 30 mm long has a radius of 1 mm. Determine the peak temperature if the outside surface is held at room temperature.
- Use k=0.1 W/m-K and Q=2.1 MW/m^2

$$\frac{d^2T}{dr^2}+\frac{1}{r}\frac{dT}{dr}+\frac{Q}{k}=0$$

$$T(R)=20\,C$$

$$\frac{dT}{dr}(0)=0$$

Practice

- Repeat the previous problem with convection to external environment.
- Use k=0.1 W/m-K and Q=2.1 MW/m^2
- Also,h=10 W/m^2-K and T_e=20 C

$$\frac{d^2T}{dr^2}+\frac{1}{r}\frac{dT}{dr}+\frac{Q}{k}=0$$

$$h\left(T(R)-T_e\right)=\frac{1}{2}QR$$

$$\frac{dT}{dr}(0)=0$$