Application 4.3A
The Runge-Kutta Method for 2-Dimensional Systems

Figure 4.3.11 in the text lists TI-85 and BASIC versions of the program **RK2DIM** that implements the Runge-Kutta iteration

$$
\begin{aligned}
k_1 &= f(t_n, x_n, y_n)\\
l_1 &= g(t_n, x_n, y_n)\\
k_2 &= f(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}h\,k_1, y_n + \tfrac{1}{2}h\,l_1)\\
l_2 &= g(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}h\,k_1, y_n + \tfrac{1}{2}h\,l_1)\\
k_3 &= f(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}h\,k_2, y_n + \tfrac{1}{2}h\,l_2)\\
l_3 &= g(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}h\,k_2, y_n + \tfrac{1}{2}h\,l_2)\\
k_4 &= f(t_n + h, x_n + h\,k_3, y_n + h\,l_3)\\
l_4 &= f(t_n + h, x_n + h\,k_3, y_n + h\,l_3)\\
k &= \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right),\\
l &= \tfrac{1}{6}\left(l_1 + 2l_2 + 2l_3 + l_4\right)\\
x_{n+1} &= x_n + h\,k\\
y_{n+1} &= y_n + h\,l\\
t_{n+1} &= t_n + h
\end{aligned}
\qquad (1)
$$

for the two dimensional system

$$
\begin{aligned}
x' &= f(t, x, y),\\
y' &= g(t, x, y)
\end{aligned}
\qquad (2)
$$

You should note that **RK2DIM** closely parallels the one-dimensional Runge-Kutta program listed in Fig. 2.6.11, with a single line to calculate one value or slope there replaced (where appropriate) with two lines here to calculate a *pair* of *x*- and *y*-values or slopes. Note also that the notation used is essentially that of Equations. (13) and (14) in Section 4.3.

The sections below illustrate the use of *Maple*, *Mathematica*, and MATLAB to implement the Runge-Kutta iteration in (1), and apply it to the initial value problem

$$
\begin{aligned}
x' &= -\pi\, y, & x(0) &= 1\\
y' &= \pi\, x, & y(0) &= 0
\end{aligned}
\qquad (3)
$$

whose exact solution is given by

$$x(t) = \cos(\pi t), \qquad y(t) = \sin(\pi t).$$

This exercise should prepare you for the following investigations.

Investigation A
Let a be the largest and b the smallest nonzero digit of your student ID number. Then use the Runge-Kutta method to approximate the solution of the initial value problem defined by

$$x' = -a\ y, \qquad y' = b\ x$$

with $x(0)$ being the next smallest digit and $y(0)$ the next largest digit of your ID. Also use your system's "dsolve" function to find the exact symbolic solution. Finally, compare your approximate solution points with the corresponding exact solution points.

Investigation B
Suppose that *you* jump from an airplane at an initial altitude of 10,000 ft, and that your *downward* position $x(t)$ and velocity $y(t)$ (in ft/sec after t seconds) satisfy the initial value problem

$$\begin{aligned} x' &= y, & x(0) &= 0 \\ (W/g)\,y' &= W - (0.0015)(100y + y^2), & y(0) &= 0 \end{aligned}$$

where W denotes *your* weight in pounds and $g = 32$ ft/sec^2. Use the Runge-Kutta method to solve numerically for $x(t)$ and $y(t)$ during your first 15 to 20 seconds of free fall, with successive step sizes h and $h/2$ small enough to get results consistent to 2 decimal places. How far have you free-fallen after 10 seconds? After 20 seconds? Use your results (for the first 20 seconds) to determine the total time required for your descent to the ground.

Investigation C
Your spacecraft is traveling at constant velocity *V*, approaching a distant earth-like planet with mass *M* and radius *R*. When activated, your deceleration system provides a constant thrust *T* until impact with the planet's surface. During the deceleration period, your distance $x(t)$ from the planet's center satisfies the differential equation

$$\frac{d^2x}{dt^2} = T - \frac{GM}{x^2}$$

where $G \approx 6.6726 \times 10^{-11}\ \text{N} \cdot (\text{m/kg})^2$ is the usual gravitational constant. Your question is — At what altitude above the planet's surface should your deceleration system be activated in order to achieve a soft touchdown? For a reasonable problem, you can take

$M = 5.97 \times 10^{24}$ kg
$R = 6.38 \times 10^{6}$ m
$V = p \times 10^{4}$ km/hr
$T = g + q$ m/s^2,

where $g = GM/R^2$ is the planet's surface gravitational acceleration, while p is the smallest nonzero digit and q the next-smallest digit in your student ID number. Find the "ignition altitude" accurate to the nearest meter, and the resulting "descent time" accurate to the nearest tenth of a second.

Using *Maple*

To apply the Runge-Kutta method to the initial value problem in (3), we let

```
Digits := 6:
pi := evalf(Pi)  # numerical value of Pi
```

and define the right-hand side functions in our two differential equations:

```
f := (t,x,y) -> -pi*y:
g := (t,x,y) ->  pi*x:
```

Suppose that we want to approximate the solution functions $x = \cos(\pi t)$ and $y = \sin(\pi t)$ on the interval [0, 1/2] corresponding to angular arguments between 0 and $\pi/2$. To approximate the solution with initial values $x(t_0) = x_0$, $y(t_0) = y_0$ on the interval $[t_0, t_f]$, we enter first the initial values (and final t-value)

```
t0 := 0:
x0 := 1:     y0 := 0:
tf := 0.5:
```

and then the desired number n of steps, the interval at which we want to print results, and the resulting step size h.

```
n := 12:                  # number of subintervals
m := 2:                   # to print every mth step
h := evalf((tf - t0)/n): # step size
```

After we initialize the values of t, x, and y,

```
t := t0:    x := x0:    y := y0:
```

the Runge-Kutta method is implemented by the **for** loop below, which carries out the iteration in (1) n times in succesStion to take n steps across the interval from $t = t_0$ to $t = t_f$. It simplies the printing to define in advance the "formatting string"

```
fmt := `%10.0f %10.4f %10.4f \n`:
```

where the notation **%w.df** specifies printing the corresponding value in a "field" w spaces wide and with d decimal places. Then Runge-Kutta loop is then

```
for i from 1 to n do
  k1 := f(t,x,y):
  l1 := g(t,x,y):                          # left-hand slopes
  k2 := f(t+h/2,x+h*k1/2,y+h*l1/2):
  l2 := g(t+h/2,x+h*k1/2,y+h*l1/2):  # 1st midpt slopes
  k3 := f(t+h/2,x+h*k2/2,y+h*l2/2):
  l3 := g(t+h/2,x+h*k2/2,y+h*l2/2):  # 2nd midpt slopes
  k4 := f(t+h,x+h*k3,y+h*l3):
  l4 := g(t+h,x+h*k3,y+h*l3):              # right-hand slopes
  k := (k1+2*k2+2*k3+k4)/6:                # average x-slope
  l := (l1+2*l2+2*l3+l4)/6:                # average y-slope
  x := x + h*k:                            # update x
  y := y + h*l:                            # update y
  t := t + h:                              # update t
  if trunc(i/m) =i/m then                  # display each
       printf(fmt,180*t,x,y) fi;           #  mth value
  od:
```

```
      15        .9659      .2588
      30        .8660      .5000
      45        .7071      .7071
      60        .5000      .8660
      75        .2588      .9659
      90        .0000     1.0000
```

Note that we have arranged to print angles in *degrees* in the first column of output. The second and third columns give the corresponding approximate values of the cosine and sine functions. Noting the familiar values $\cos 60° = \sin 30° = \frac{1}{2}$, $\cos 30° = \sin 60° = \frac{1}{2}\sqrt{3} \approx 0.8660$, and $\cos 45° = \sin 45° = \frac{1}{2}\sqrt{2} \approx 0.7071$, we see that the Runge-Kutta method with only $n = 12$ subintervals has provided 4 decimal places of accuracy on the whole range from 0º to 90º.

If only the final endpoint result is wanted explicitly, then the print command can be removed from the loop and executed immediately following it (just as we did with the Euler loop in Project 2.4). For a different initial value problem, we need only enter the appropriate functions $f(x, y)$ and $g(x, y)$ for our new differential equations and the

desired initial and final values in the initial commands above, then re-execute the subsequent ones.

Using *Mathematica*

To apply the Runge-Kutta method to the initial value problem in (3), we let

```
pi = N[Pi];          (* numerical value of Pi *)
```

and define the right-hand side functions in our two differential equations:

```
f[t_,x_,y_] := -pi*y
g[t_,x_,y_] :=  pi*x
```

Suppose that we want to approximate the solution functions $x = \cos(\pi t)$ and $y = \sin(\pi t)$ on the interval [0, 1/2] corresponding to angular arguments between 0 and $\pi/2$. To approximate the solution with initial values $x(t_0) = x_0$, $y(t_0) = y_0$ on the interval $[t_0, t_f]$, we enter first the initial values (and final *t*-value)

```
t0 = 0;   x0 = 1;   y0 = 0;   tf = 0.5;
```

and then the desired number n of steps, the interval at which we want to print results, and the resulting step size h.

```
n = 12;              (* number of subintervals  *)
m = 2;               (* to print every mth step *)
h = (tf - t0)/n;     (* step size               *)
```

After we initialize the values of t, x, and y,

```
t = t0;    x = x0;    y = y0;
```

the Runge-Kutta method itself is implemented by the **Do** loop below, which carries out the iteration in (1) n times in succesStion to take n steps across the interval from $t = t_0$ to $t = t_f$.

```
Do[
   k1 = f[t,x,y];
   l1 = g[t,x,y];                    (* left-hand slopes *)
   k2 = f[t+h/2,x+h*k1/2,y+h*l1/2];
   l2 = g[t+h/2,x+h*k1/2,y+h*l1/2];(* 1st midpt slopes *)
   k3 = f[t+h/2,x+h*k2/2,y+h*l2/2];
   l3 = g[t+h/2,x+h*k2/2,y+h*l2/2];(* 2nd midpt slopes *)
   k4 = f[t+h,x+h*k3,y+h*l3];
   l4 = g[t+h,x+h*k3,y+h*l3];     (* right-hand slopes *)
```

```
k  = (k1+2*k2+2*k3+k4)/6;       (* average x-slope  *)
l  = (l1+2*l2+2*l3+l4)/6;       (* average y-slope  *)
x  = x + h*k;                   (* update x  *)
y  = y + h*l;                   (* update y  *)
t  = t + h;                     (* update t  *)
If[ Floor[i/m] == i/m,
    Print[180*t,PaddedForm[x,{10,4}],
    PaddedForm[y,{10,4}]]],
                    (* display each mth value *)
{i,1,n} ]
```

```
15.      0.9659      0.2588
30.      0.8660      0.5000
45.      0.7071      0.7071
60.      0.5000      0.8660
75.      0.2588      0.9659
90.      0.0000      1.0000
```

Note that we have arranged to print angles in *degrees* in the first column of output. The second and third columns give the corresponding approximate values of the cosine and sine functions. The **PaddedForm[x,{10,4}]** specification formats a result by printing it in a "field" 10 spaces wide and with 4 decimal places. Noting the familiar values $\cos 60° = \sin 30° = \frac{1}{2}$, $\cos 30° = \sin 60° = \frac{1}{2}\sqrt{3} \approx 0.8660$, and $\cos 45° = \sin 45° = \frac{1}{2}\sqrt{2} \approx 0.7071$, we see that the Runge-Kutta method with only $n = 12$ subintervals has provided 4 decimal places of accuracy on the whole range from 0º to 90º.

If only the final endpoint result is wanted explicitly, then the print command can be removed from the loop and executed immediately following it (just as we did with the Euler loop in Project 2.4). For a different initial value problem, we need only enter the appropriate functions $f(x, y)$ and $g(x, y)$ for our new differential equations and the desired initial and final values in the initial commands above, then re-execute the subsequent ones.

Using MATLAB

To apply the Runge-Kutta method to the initial value problem in (3), we begin by defining the right-hand side functions $f(t, x, y)$ and $g(t, x, y)$ in our two differential equations:

```
f = inline('-pi*y','t','x','y');
g = inline('pi*x','t','x','y');
```

Suppose that we want to approximate the solution functions $x = \cos(\pi t)$ and $y = \sin(\pi t)$ on the interval [0, 1/2] corresponding to angular arguments between 0 and $\pi/2$. To approximate the solution with initial values $x(t_0) = x_0$, $y(t_0) = y_0$ on the interval $[t_0, t_f]$, we enter first the initial values (and final *t*-value)

```
t0 = 0;
x0 = 1;
y0 = 0;
tf = 0.5;
```

and then the desired number n of steps, the interval at which we want to print results, and the resulting step size h.

```
n = 12;              % number of subintervals
m =  2;              % to print every mth step
h = (tf - t0)/n;     % step size
```

After we initialize the values of t, x, and y,

```
t = t0;
x = x0;
y = y0;
```

and the first line of our desired table of values,

```
result = [t0,x0,y0];
result =
     0     1     0
```

the Runge-Kutta method itself is implemented by the **for** loop below, which carries out the iteration in (1) n times in succesStion to take n steps across the interval from $t = t_0$ to $t = t_f$.

```
for i = 1:n,
   k1 = f(t,x,y);
   l1 = g(t,x,y);                    % left-hand slopes
   k2 = f(t+h/2,x+h*k1/2,y+h*l1/2);
   l2 = g(t+h/2,x+h*k1/2,y+h*l1/2); % 1st midpt slopes
   k3 = f(t+h/2,x+h*k2/2,y+h*l2/2);
   l3 = g(t+h/2,x+h*k2/2,y+h*l2/2); % 2nd midpt slopes
   k4 = f(t+h,x+h*k3,y+h*l3);
   l4 = g(t+h,x+h*k3,y+h*l3);       % right-hand slopes
   k = (k1+2*k2+2*k3+k4)/6;         % average x-slope
   l = (l1+2*l2+2*l3+l4)/6;         % average y-slope
   x = x + h*k;                  % Euler step to update x
   y = y + h*l;                  % Euler step to update y
```

```
   t = t + h;                    % update t
   if floor(i/m) == i/m,
      result = [result;[180*t,x,y]];
      end                        % adjoin new row of data
   end
```

Then the command

```
result
result =
         0    1.0000         0
   15.0000    0.9659    0.2588
   30.0000    0.8660    0.5000
   45.0000    0.7071    0.7071
   60.0000    0.5000    0.8660
   75.0000    0.2588    0.9659
   90.0000    0.0000    1.0000
```

displays our table of Runge-Kutta results.

Note that we have arranged to print angles in *degrees* in the first column of output. The second and third columns give the corresponding approximate values of the cosine and sine functions. Noting the familiar values $\cos 60° = \sin 30° = \frac{1}{2}$, $\cos 30° = \sin 60° = \frac{1}{2}\sqrt{3} \approx 0.8660$, and $\cos 45° = \sin 45° = \frac{1}{2}\sqrt{2} \approx 0.7071$, we see that the Runge-Kutta method with only $n = 12$ subintervals has provided 4 decimal places of accuracy on the whole range from 0^o to 90^o.

For a different initial value problem, we need only re-define the functions $f(x, y)$ and $g(x, y)$ corresponding to our new differential equations and the desired initial and final values in the initial commands above, then re-execute the subsequent ones.