Application 2.6
Runge-Kutta Implementation

Figure 2.6.11 in the text lists TI-85 and BASIC programs implementing the Runge-Kutta method to approximate the solution of the initial value problem

$$\frac{dy}{dx} = x + y, \quad y(0) = 1 \tag{1}$$

considered in Example 1 of Section 2.6. Even if you have little familiarity with the BASIC and TI programming languages, the comments provided in the final column should make it clear how each is implementing the Runge-Kutta iteration

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}h\,k_1), \\ k_3 &= f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}h\,k_2) \\ k_4 &= f(x_n + h, y_n + h\,k_3) \\ k &= \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_2 + k_4\right), \\ y_{n+1} &= y_n + h\,k, \\ x_{n+1} &= x_n + h \end{aligned} \tag{2}$$

to make the step from (x_n, y_n) to (x_{n+1}, y_{n+1}).

To apply the Runge-Kutta method to a differential equation $dy/dx = f(x, y)$, one need only change the initial line of the program, in which the function f is defined. To increase the number of steps (and thereby decrease the step size) one need only change the value of N specified in the second line of the program.

We illustrate below the implementation of the Runge-Kutta method in systems like *Maple*, *Mathematica*, and MATLAB. To begin this project, you should implement the Runge-Kutta method on your calculator or in a programming language of your choice. First test your program by carrying through its application to the initial value problem in (1), and then apply it to solve some of the problems for Section 2.6 in the text. Then carry out the following "famous numbers" and "skydiver's descent" investigations.

A. Famous Numbers, One Last Time

The problems below describe the numbers $e \approx 2.71828182846$, $\ln 2 \approx 0.69314718056$, and $\pi \approx 3.14159265359$ as specific values of certain initial value problem solutions. In each case, apply the improved Euler method with $n = 10,\ 20,\ 40,\ \cdots$ subintervals

(doubling n each time). How many subintervals are needed to obtain — twice in succession — the correct value of the target number rounded off accurate to 9 decimal places?

1. The number $e = y(1)$ where $y(x)$ is the solution of the initial value problem $y' = y, \quad y(0) = 1$.

2. The number $\ln 2 = y(2)$ where $y(x)$ is the solution of the initial value problem $y' = 1/x, \quad y(1) = 0$.

3. The number $\pi = y(1)$ where $y(x)$ is the solution of the initial value problem $y' = 4/(1+x^2), \quad y(0) = 0$.

B. The Skydiver's Descent

Recall the 60-kg skydiver of Example 3 in Section 2.6 of the text. She falls vertically with initial velocity zero from an initial height of 5 kilometers, and experiences an upward force F_R of air resistance given in terms of her velocity v (in m/s) by $F_R = 0.0096\left(100v + 10v^2 + v^3\right)$. Then her downward velocity function $v(t) > 0$ satisfies the $ma = F$ equation

$$60\frac{dv}{dt} = 60 \times 9.8 - 0.0096(100v + 10v^2 + v^3) \tag{3}$$

(with $m = 60$ and $g = 9.8$ in mks units), and hence the initial value problem

$$\frac{dv}{dt} = 9.8 - 0.00016(100v + 10v^2 + v^3), \qquad v(0) = 0. \tag{4}$$

The skydiver reaches her terminal velocity when the forces of gravity and air resistance balance, so $dv/dt = 0$. We saw in Example 3 that this terminal velocity is approximately 35.578 m/s.

First apply the Runge-Kutta method with step size $h = 0.1$ to generate the table of velocities shown in Fig. 2.6.8 of the text, where we see that the skydiver has achieved her terminal velocity after 20 seconds of free fall. Then approximate the skydiver's successive (downward) positions $y_1, y_2, y_3, \ldots$ by beginning with the initial position $y(0) = y_0 = 0$ and calculating

$$y_{n+1} = y_n + v_n h + \tfrac{1}{2} a_n h^2 \tag{5}$$

$(n = 1, 2, 3, \ldots)$ where $a_n = v'(t_n)$ is the particle's approximate acceleration at time t_n.

Formula (5) would give the correct increment (from y_n to y_{n+1}) *if* the acceleration a_n remained constant during the time interval $[t_n, t_{n+1}]$. You should obtain the position data shown in Fig. 2.6.13 in the text, where it appears that the skydiver falls 629.866 meters during her first 20 seconds of descent. She then free falls the remaining 4370.134 meters to the ground at her terminal speed of 35.578 meters per second. Hence her total time of descent is 20 + 4370.134/35.578 = 142.833 seconds, or about 2 min 23 sec.

For an individual problem to solve using whatever computational system is available to you, analyze your own skydive (perhaps from a different height), using your own mass m (in kilograms) and a plausible air resistance force of the form $F_R = a\,v + b\,v^2 + c\,v^3$. For instance, for coefficients of the same magnitude used above, you could take $a = 0.01\,p^2$, $b = 0.01q$, and $c = 0.01$ where p and q are the two largest digits in your student ID number.

Using *Maple*

To apply the Runge-Kutta method to the initial value problem in (1), we first define as usual the right-hand function in the differential equation.

```
f := (x,y) -> x + y:
```

To approximate the solution with initial value $y(x_0) = y_0$ on the interval $[x_0, x_f]$, we enter first the initial values

```
x0 := 0:    y0 := 1:
xf := 1:
```

and then the desired number n of steps and the resulting step size h.

```
n := 10:
h := evalf((xf - x0)/n):
```

After we initialize the values of x and y,

```
x := x0:    y := y0:
```

the Runge-Kutta method itself is implemented by the **for** loop below, which carries out the iteration in (2) n times in succession to take n steps across the interval from $x = x_0$ to $x = x_f$.

```
for i from 1 to n do
   k1 := f(x,y):              # the left-hand slope
   k2 := f(x+h/2,y+h*k1/2):   # 1st midpoint slope
   k3 := f(x+h/2,y+h*k2/2):   # 2nd midpoint slope
```

```
k4 := f(x+h,y+h*k3):        # the right-hand slope
k := (k1+2*k2+2*k3+k4)/6:   # the average slope
y := y + h*k:               # R-K step to update y
x := x + h:                 # update x
print(x,y);                 # display current values
od:
```

```
             .1000000,   1.110342
             .2000000,   1.242806
             .3000000,   1.399718
             .4000000,   1.583650
             .5000000,   1.797443
             .6000000,   2.044238
             .7000000,   2.327506
             .8000000,   2.651082
             .9000000,   3.019206
            1.000000,   3.436563
```

Thus the Runge-Kutta method with $n = 10$ steps gives $y(1) \approx 3.436563$ for the initial value problem in (1). The actual value of the exact solution $y(x) = 2e^x - x - 1$, at $x = 1$ is $y(1) = 2e - 2 \approx 3.436564$, so with only 10 steps the Runge-Kutta gives nearly 6-decimal place accuracy!

Automating the Runge-Kutta Method

The following *Maple* procedure makes n steps from x_0 to x_f to approximate a solution of the initial value problem $y' = f(x,y), \quad y(x_0) = y_0$.

```
rk := proc(x0,xf,y0,n)
      local x,y,h,i,k1,k2,k3,k4,k,R;
      x := x0;
      y := y0;
      R := x,y;
      h := evalf((xf-x0)/n);
      for i from 1 by 1 to n do
         k1 := f(x,y);
           k2 := f(x+h/2,y+h*k1/2);
           k3 := f(x+h/2,y+h*k2/2);
           k4 := f(x+h,y+h*k3);
          k := (k1+2*k2+2*k3+k4)/6;
          y := y + h*k;
          x := x + h;
          R := R,x,y
          od
      end:
```

The output of **rk** is the list **R** of successive values $x_0, y_0, x_1, y_1, x_2, y_2, \ldots, x_n, y_n$ which we can easily print as a table:

```
R := rk(0,1,1,10):
     for i from 1 by 1 to 11 do
     print(R[2*i-1],R[2*i])  od;
```

```
                0,   1
          .1000000000    1.110341667
          .2000000000,   1.242805142
          .3000000000,   1.399716994
          .4000000000,   1.583648480
          .5000000000,   1.797441277
          .6000000000,   2.044235924
          .7000000000,   2.327503253
          .8000000000,   2.651079126
          .9000000000,   3.019202827
         1.000000000,   3.436559488
```

The Skydiver Problem

We now redefine f as the skydiver's acceleration function.

```
f := (t,v) -> 9.8 - 0.00016*(100*v + 10*v^2 + v^3):
```

The following commands take $k = 200$ steps from $t = 0$ to $t = 20$, and assemble the lists **T** of successive times and **V** of successive velocities.

```
k := 200:
R := rk(0,20,0,k):
t := 0:  v := 0:    # initial time and velocity
T := t:  V := v:    # initialize lists
for i from 1 by 1 to k do
T := T,R[2*i+1];
V := V,R[2*i+2]
od:
```

The print loop

```
for i from 1 by 10 to 201 do
     print(T[i],V[i])  od;
```

then displays the time-velocity data shown in Fig. 2.6.8 in the text. Finally we use (5) to calculate the corresponding list **Y** of downward positions of the skydiver.

```
Y[1] := 0:   # initial position
h := 0.1:    # step size
```

```
for n from 1 by 1 to k do
     a := f(T[n],V[n]);
     Y[n+1] := Y[n] + V[n]*h + 0.5*a*h^2
     od:
```

Then the print loop

```
for i from 1 by 20 to 201 do
     print(T[i],V[i],Y[i])  od;
```

displays the time-velocity-position data shown in Fig. 2.6.13 in the text. We see that the skydiver falls 629.866 meters during her first 20 seconds of descent.

Using *Mathematica*

To apply the Runge-Kutta method to the initial value problem in (1), we first define as usual the right-hand function in the differential equation.

```
f[x_,y_] := x + y
```

To approximate the solution with initial value $y(x_0) = y_0$ on the interval $[x_0, x_f]$, we enter first the initial values

```
x0 = 0;      y0 = 1;
xf = 1;
```

and then the desired number n of steps and the resulting step size h.

```
n = 10;
h = (xf - x0)/n  //  N;
```

After we initialize the values of x and y,

```
x = x0;   y = y0;
```

the Runge-Kutta method itself is implemented by the following **Do** loop, which carries out the iteration in (2) n times in succession to take n steps across the interval from $x = x_0$ to $x = x_f$.

```
Do[  k1 = f[x,y];              (* left-hand slope  *)
     k2 = f[x+h/2, y+h*k1/2]; (* 1st midpt slope  *)
     k3 = f[x+h/2, y+h*k2/2]; (* 2nd midpt slope  *)
     k4 = f[x + h, y + h*k3]; (* right-hand slope *)
     k  = (k1+2k2+2k3+k4)/6;  (* average slope    *)
     y = y + h*k;             (* Runge-Kutta step *)
     x = x + h;               (* update x         *)
```

```
     Print[x,"     ",y],      (* display x and y  *)
     {i,1,n} ]
```

```
0.1     1.11034
0.2     1.24281
0.3     1.39972
0.4     1.58365
0.5     1.79744
0.6     2.04424
0.7     2.3275
0.8     2.65108
0.9     3.0192
1.      3.43656
```

Thus the Runge-Kutta method with $n = 10$ steps gives $y(1) \approx 3.43656$ for the initial value problem in (1). The actual value of the exact solution $y(x) = 2e^x - x - 1$, at $x = 1$ is $y(1) = 2e - 2 \approx 3.436564$, so with only 10 steps the Runge-Kutta gives 5-decimal place accuracy!

Automating the Runge-Kutta Method

The following *Mathematica* function makes n steps from x_0 to x_f to approximate a solution of the initial value problem $y' = f(x, y), \quad y(x_0) = y_0$.

```
rk[x0_, xf_, y0_, n_] :=
     Module[{h,x,y,X,Y,i,k1,k2,k3,k4,k},
     h = (xf - x0)/n;
     x = x0; y = y0;
     X = {x};  Y = {y};
     Do[k1 = f[x, y];
          k2 = f[x+h/2, y+h*k1/2];
          k3 = f[x+h/2, y+h*k2/2];
          k4 = f[x+h, y+h*k3];
          k = (k1+2*k2+2*k3 k4)/6;
          y = y + h*k;
          x = x + h;
     X = Append[X, x]; Y = Append[Y, y],
     {i, 1, n}];
     {X, Y}]
```

The output of **rk** is the pair **{X,Y}** of lists **X** and **Y** of successive x- and y-values, which we can easily print in table format:

```
{X, Y} = rk[0., 1., 1, 10];
TableForm[Transpose[{X, Y}]]
```

```
0.      1
0.1     1.11034
0.2     1.24281
0.3     1.39972
0.4     1.58365
0.5     1.79744
0.6     2.04424
0.7     2.3275
0.8     2.65108
0.9     3.0192
1.0     3.43656
```

The Skydiver Problem

We now redefine f as the skydiver's acceleration function.

```
f[t_,v_] := 9.8 - 0.00016 (100v + 10v^2 + v^3)
```

We can use **rk** to take $k = 200$ steps from $t = 0$ to $t = 20$ and calculate the lists **T** of successive times and **V** of successive velocities of the skydiver.

```
k = 200;
{T, V} = rk[0., 20, 0, k];
```

Then

```
{T[[k+1]], V[[k+1]]}
{20., 35.5779}
```

verifies that the terminal velocity of 35.578 m/s has been achieved after 20 seconds. Finally we use (5) to calculate the corresponding list **Y** of downward positions.

```
Y = Table[0, {i, 1, k + 1}];
h = 0.1;
Do[  a = f[T[[n]], V[[n]]];
     Y[[n + 1]] = Y[[n]] + V[[n]]*h + 0.5*a*h^2,
     {n, 1, k}];
```

Then the result

```
{T[[k+1]], V[[k+1]], Y[[k+1]]}
{20., 35.5779, 629.866}
```

indicates that the skydiver falls 629.866 meters during her first 20 seconds of descent.

Using MATLAB

To apply the Runge-Kutta method to the initial value problem in (1), suppose as usual that the right-hand side function

```
function  yp = f(x,y)
yp = x + y;    % yp = y'
```

in the differential equation has been defined and saved in the text file **f.m**.

To approximate the solution with initial value $y(x_0) = y_0$ on the interval $[x_0, x_f]$, we enter first the initial values

```
x0 = 0;     y0 = 1;     xf = 1;
```

and then the desired number n of steps and the resulting step size h.

```
n = 10;
h = (xf - x0)/n;
```

After we initialize the values of x and y,

```
x = x0;    y = y0;
```

and the column vectors **X** and **Y** of approximate values

```
X = x;     Y = y;
```

the Runge-Kutta method itself is implemented by the following **for** loop, which carries out the iteration (2) n times in succession to take n steps across the interval from $x = x_0$ to $x = x_f$.

```
for i = 1 : n                  % for i = 1 to n do
    k1 = f(x,y);               % left-hand slope
    k2 = f(x+h/2,y+h*k1/2);    % 1st midpoint slope
    k3 = f(x+h/2,y+h*k2/2);    % 2nd midpoint slope
    k4 = f(x+h,y+h*k3):        % the right-hand slope
     k = (k1+2*k2+2*k3+k4)/6;  % the average slope
     y = y + h*k;              % R-K step to update y
     x = x + h;                % update x
     X = [X; x];               % adjoin new x-value
     Y = [Y; y];               % adjoin new y-value
     end
```

Note that x is updated after y in order that the computation $k = f(x, y)$ can use the left-hand values (with neither yet updated).

As output, the loop above produces the resulting column vectors **X** and **Y** of x- and y-values that can be displayed simultaneously using the command

```
format long
[X,Y]

ans =
                   0   1.00000000000000
   0.10000000000000   1.11034166666667
   0.20000000000000   1.24280514170139
   0.30000000000000   1.39971699412508
   0.40000000000000   1.58364848016137
   0.50000000000000   1.79744127719368
   0.60000000000000   2.04423592418387
   0.70000000000000   2.32750325319355
   0.80000000000000   2.65107912658463
   0.90000000000000   3.01920282756014
   1.00000000000000   3.43655948827033
```

which displays the column vectors **X** and **Y** side-by-side. Thus the Runge-Kutta method with $n = 10$ steps gives $y(1) \approx 3.43656$ for the initial value problem in (1). The actual value of the exact solution $y(x) = 2e^x - x - 1$ at $x = 1$ is $y(1) = 2e - 2 \approx 3.436564$, so with only 10 steps the Runge-Kutta gives 5-decimal place accuracy!

For a different initial value problem, we need only define the appropriate function $f(x, y)$ in the file **f.m**, then enter the desired initial and final values in the first command above and re-execute the subsequent ones.

Automating the Runge-Kutta Method

The following MATLAB function makes n steps from x_0 to x_f to approximate a solution of the initial value problem $y' = f(x, y)$, $y(x_0) = y_0$.

```
function  [X,Y] = rk(x,xf,y,n)

h = (xf - x)/n;                 % step size
X = x;                          % initial x
Y = y;                          % initial y
for i = 1 : n                   % begin loop
   k1 = f(x,y);                 % left-hand slope
   k2 = f(x+h/2,y+h*k1/2);      % 1st midpoint slope
   k3 = f(x+h/2,y+h*k2/2);      % 2nd midpoint slope
   k4 = f(x+h,y+h*k3);          % right-hand slope
   k = (k1+2*k2+2*k3+k4)/6;     % average slope
   y = y + h*k;                 % Runge-Kutta step
   x = x + h;                   % new x
```

```
X = [X;x];                    % update x-column
Y = [Y;y];                    % update y-column
end                           % end loop
```

With this function saved in the text file **rk.m**, we need only assume also that the function $f(x, y)$ has been defined and saved in the file **f.m**.

The function **rk** applies the Runge-Kutta method to take n steps from x to x_f starting with the initial value y of the solution. The output of **rk** consists of the pairs **X** and **Y** of column vectors of successive x- and y-values. For instance, with **f** as previously defined, the command

```
[X,Y] = rk(0, 1, 1, 10);  [X,Y]
```

is a one-liner that generates the table **[X,Y]** displayed above to approximate the solution of the initial value problem $y' = x + y$, $y(0) = 1$ on the x-interval [0, 1].

The Skydiver's Descent

To apply the Runge-Kutta method to the initial value problem in (3), we re-define the MATLAB function **f.m** as the skydiver's acceleration function:

```
function  vp = f(t,v)
vp = 9.8 - 0.00016*(100*v + 10*v^2 + v^3);
```

Then the commands

```
k = 200;                      % 200 subintervals
[T,V] = rk(0,20,0,k);         % Runge-Kutta approximation
```

take $k = 200$ steps from $t = 0$ to $t = 20$ and calculate the column vectors **T** of successive times and **V** of successive velocities. Hence the command

```
[t(1:10:k+1);v(1:10:k+1)]   % Display every 10th entry
```

produces the table time-velocity data shown in Fig. 2.6.8 of the text, and verifies that the skydiver has achieved her terminal velocity 35.578 m/s after 20 seconds. Finally, the commands

```
Y = zeros(k+1,1);                          % initialize Y
h = 0.1;                                   % step size
for n = 1:k                                % for n=1 to k
   a = f(T(n),V(n));                       % acceleration
   Y(n+1) = Y(n) + V(n)*h + 0.5*a*h^2; % Equation (5)
   end                                      % end loop
```

```
[T(1:20:k+1),V(1:20:k+1),Y(1:20:k+1)]  % each 20th entry
```

calculate the corresponding column vector **Y** successive downward positions of the skydiver, and then display the time-velocity-position data shown in the table of Fig. 2.6.13 in the text. We see that the skydiver falls 629.866 meters during her first 20 seconds of descent.