Application 2.4
Implementing Euler's Method

One's understanding of a numerical algorithm is sharpened by considering its implementation in the form of a calculator or computer program. Figure 2.4.13 in the text lists TI-85 and BASIC programs implementing Euler's method to approximate the solution of the initial value problem

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

as shown in the table of Fig 2.4.8. The comments provided in the final column of the table should make these programs intelligible even if you have little familiarity with the BASIC and Texas Instruments programming languages. To increase the number of steps (and thereby decrease the step size) you need only change the value of **N** specified in the first line of the program. To apply Euler's method to a different equation $dy/dx = f(x, y)$, you need only change the single line that calculates the function value **F**.

We illustrate below the implementation of Euler's method in systems like *Maple*, *Mathematica*, and MATLAB. For this project, you should implement Euler's method on your own calculator or in a programming language of your choice. First test your implementation by carrying through its application to the initial value problem in (1), and then apply it to solve some of the problems for Section 2.4 in the text. Then carry out the following investigation.

Famous Numbers Investigation

The problems below describe the numbers $e \approx 2.71828$, $\ln 2 \approx 0.69315$, and $\pi \approx 3.14159$ as specific values of certain initial value problem solutions. In each case, apply Euler's method with $n = 50, 100, 200, \cdots$ subintervals (doubling n each time). How many subintervals are needed to obtain — twice in succession — the correct value of the target number rounded off to 3 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 $y' = 4/(1+x^2), \quad y(0) = 0$.

Also, explain in each problem what the point is — why the approximate value of the indicated famous number is, indeed, the expected numerical result.

Using *Maple*

To apply Euler's method to the initial value problem in (1), we first define the right-hand side function $f(x, y) = x + y$ in the differential equation.

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

$$f := (x, y) \to x + y$$

To approximate the solution with initial value $y(x_0) = x_0$ on the interval $[x_0, x_f]$, we enter first the initial values of x and y and the final value of x.

```
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);
```

$$h := .10000$$

After we initialize the values of x and y,

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

Euler's method itself is implemented by the following **for**-loop, which carries out the iteration

$$y_{n+1} = y_n + h\, f(x_n, y_n), \qquad x_{n+1} = x_n + h$$

n times in successtion to take n steps across the interval from $x = x_0$ to $x = x_f$.

```
for i from 1 to n do
    k := f(x,y):          # the left-hand slope
    y := y + h*k:         # Euler step to update y
    x := x + h:           # update x
    print(x,y);           # display current values
    od:
```

```
.10000,  1.1000
.20000,  1.2200
.30000,  1.3620
```

.40000, 1.5282
.50000, 1.7210
.60000, 1.9431
.70000, 2.1974
.80000, 2.4871
.90000, 2.8158
1.0000, 3.1874

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).

The output consists of x- and y-columns of resulting x_i- and y_i-values. In particular, we see that Euler's method with $n = 10$ steps gives $y(1) \approx 3.1874$ for the initial value problem in (1). The exact solution is $y(x) = 2e^x - x - 1$, so the actual value at $x = 1$ is $y(1) = 2e - 2 \approx 3.4366$. Thus our Euler approximation underestimates the actual value by about 7.25%.

If only the final endpoint result is wanted explicitly, then the print command can be removed from the loop and executed following it:

```
x := x0:    y := y0:      # re-initialize
for i from 1 to n do
    k := f(x,y):          # the left-hand slope
    y := y + h*k:         # Euler step to update y
    x := x + h:           # update x
    od:

print(x,y);
```

1.0000, 3.1874

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

Using *Mathematica*

To apply Euler's method to the initial value problem in (1), we first define the right-hand side function $f(x, y) = x + y$ in the differential equation.

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

To approximate the solution with initial value $y(x_0) = x_0$ on the interval $[x_0, x_f]$, we enter first the initial values of x and y and the final value of x.

```
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
0.1
```

After we initialize the values of x and y,

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

Euler's method itself is implemented by the following **Do**-loop, which carries out the iteration

$$y_{n+1} = y_n + h\, f(x_n, y_n), \qquad x_{n+1} = x_n + h$$

n times in succesStion to take n steps across the interval from $x = x_0$ to $x = x_f$.

```
Do[  k = f[x,y];           (* the left-hand slope     *)
     y = y + h*k;          (* Euler step to update y  *)
     x = x + h;            (* update x                *)
     Print[x,"    ",y],    (* display updated values  *)
     {i,1,n} ]
```

```
0.1     1.1
0.2     1.22
0.3     1.362
0.4     1.5282
0.5     1.72102
0.6     1.94312
0.7     2.19743
0.8     2.48718
0.9     2.8159
1.      3.18748
```

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).

The output consists of x- and y-columns of resulting x_i- and y_i-values. In particular, we see that Euler's method with $n = 10$ steps gives $y(1) \approx 3.1875$ for the initial value problem in (1). The exact solution is $y(x) = 2e^x - x - 1$, so the actual value at $x = 1$ is $y(1) = 2e - 2 \approx 3.4366$. Thus our Euler approximation underestimates the actual value by about 7.25%.

If only the final endpoint result is wanted explicitly, then the print command can be removed from the loop and executed following it:

```
Do[  k = f[x,y];          (* the left-hand slope     *)
     y = y + h*k;         (* Euler step to update y  *)
     x = x + h,           (* update x                *)
     {i,1,n} ]

Print[x,"     ",y]
1.      3.18748
```

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

Using MATLAB

To apply Euler's method to the initial value problem in (1), we first define the right-hand function $f(x, y)$ in the differential equation. User-defined functions in MATLAB are defined in (ASCII) text files. To define the function $f(x, y) = x + y$ we save the MATLAB function definition

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

in the text file **f.m**.

To approximate the solution with initial value $y(x_0) = x_0$ 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
h =
    0.1000
```

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;
```

Euler's method itself is implemented by the following **for**-loop, which carries out the iteration

$$y_{n+1} = y_n + h\, f(x_n, y_n), \qquad x_{n+1} = x_n + h$$

n times in succesStion 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
    k = f(x,y);            % the left-hand slope
    y = y + h*k;           % Euler 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

```
[X,Y]
ans =
         0    1.0000
    0.1000    1.1000
    0.2000    1.2200
    0.3000    1.3620
    0.4000    1.5282
    0.5000    1.7210
    0.6000    1.9431
    0.7000    2.1974
    0.8000    2.4872
    0.9000    2.8159
    1.0000    3.1875
```

In particular, we see that $y(1) \approx 3.1875$ for the initial value problem in (1). If only this final endpoint result is wanted explicitly, then we can simply enter

```
[X(n+1), Y(n+1)]
ans =
    1.0000    3.1875
```

The index **n+1** (instead of **n**) is required because the initial values x_0 and y_0 are the initial vector elements **X(1)** and **Y(1)**, respectively.

The exact solution of the initial value problem in (1) is $y(x) = 2e^x - x - 1$, so the actual value at $x = 1$ is $y(1) = 2e - 2 \approx 3.4366$. Thus our Euler approximation underestimates the actual value by about 7.25%.

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 Euler's Method

The **for**-loop above can be automated by saving the MATLAB function definition

```
function  [X,Y] = euler1(x,xf,y,n)
h = (xf - x)/n;                % step size
X = x;                         % initial x
Y = y;                         % initial y
for i = 1 : n                  % begin loop
     y = y + h*f(x,y);         % Euler iteration
     x = x + h;                % new x
     X = [X;x];                % update x-column
     Y = [Y;y];                % update y-column
     end                       % end loop
```

in the text file **euler1.m** (we use the name **euler1** to avoid conflict with MATLAB's built-in **euler** function). This function assumes that the function $f(x, y)$ has been defined and saved in the MATLAB file **f.m**.

The function **euler1** applies Euler's method to take n steps from x to x_f starting with the initial value y of the solution. For instance, with **f** as previously defined, the command

```
[X,Y] = euler1(0,1, 1, 10);  [X,Y]
```

is a one-liner that generates 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].