Application 2.5
Improved Euler Implementation

Figure 2.5.10 in the text lists TI-85 and BASIC programs implementing the improved Euler method to approximate the solution of the initial value problem

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

considered in Example 2 of Section 2.5 in the text. The comments provided in the final column should render these programs intelligible even if you have little familiarity with the BASIC and TI programming languages.

To apply the improved Euler 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 improved Euler method in systems like *Maple*, *Mathematica*, and MATLAB. To begin this project, you should implement the improved Euler method on your calculator or in a programming language of your choice. Test your program by application first to the initial value problem in (1), and then to some of the problems for Section 2.5 in the text. Then carry out one or more of the following investigations.

Famous Numbers, Again

The problems below describe the numbers $e \approx 2.7182818$, $\ln 2 \approx 0.6931472$, and $\pi \approx 3.1415927$ 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 to 5 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 $y' = 4/(1 + x^2), \quad y(0) = 0$.

Logistic Population Investigation

Apply your improved Euler program to the initial value problem

$$\frac{dy}{dt} = \frac{1}{3}y(8-y), \; y(0) = 1$$

of Example 3 in Section 2.5 in the text. In particular, verify (as claimed) that the approximate solution with step size $h = 1$ levels off at $y \approx 4.3542$ rather than at the limiting value $y = 8$ of the exact solution. Perhaps a table of values for $0 \le x \le 100$ will make this apparent.

For you own logistic population to investigate, you might consider the initial value problem

$$\frac{dy}{dt} = \frac{1}{n}y(m-y), \qquad y(0) = 1$$

where m and n are (for instance) the largest and smallest digits in your student ID number. Does the improved Euler approximation with step size $h = 1$ level off at the "correct" limiting value of the exact solution? If not, find a smaller value of h such that it does.

With Periodic Harvesting and Restocking

The differential equation

$$\frac{dy}{dt} = k\,y(M-y) - h\sin\left(\frac{2\pi t}{P}\right)$$

models a logistic population that is periodically harvested and restocked with period P and maximal harvesting/restocking rate h. A numerical approximation program was used to plot the typical solution curves for the case $k = M = h = P = 1$ that are shown in Fig. 2.5.12 in the text. This figure suggests — though it does not suffice to prove — the existence of a threshold initial population such that

- Starting with an initial population above this threshold, the population oscillates (perhaps with period P?) about the (unharvested) stable limiting population $y(t) \equiv M$, while

- The population dies out if it starts with an initial population below this threshold.

Use an appropriate plotting utility to investigate your own logistic population with periodic harvesting and restocking (selecting typical values of the parameters k, M, h, and P). Do the observations indicated above appear to hold for your population?

Using *Maple*

To apply the improved Euler method to the initial value problem in (1), we first define the right-hand 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 $y(x_0) = y_0$ 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);
```

$$h := .10000$$

After we initialize the values of x and y,

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

the improved Euler method itself is implemented by the following **`for`** loop, which carries out the iteration

$$k_1 = f(x_n, y_n), \qquad k_2 = f(x_n + h, y_n + h\,k_1),$$
$$k = \tfrac{1}{2}(k_1 + k_2),$$
$$y_{n+1} = y_n + h\,k, \qquad x_{n+1} = x_n + h$$

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,y+h*k1):     # the right-hand slope
         k := (k1 + k2)/2:        # the average slope
         y := y + h*k:            # Euler step to update y
         x := x + h:              # update x
```

```
print(x,y);                # display current values
od:
```

```
                  .100000,  1.11000
                  .200000,  1.24205
                  .300000,  1.39847
                  .400000,  1.58181
                  .500000,  1.79490
                  .600000,  2.04087
                  .700000,  2.32316
                  .800000,  2.64559
                  .900000,  3.01238
                  1.00000,  3.42818
```

Note that x is updated after y in order that the computation $k_1 = f(x, y)$ 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 the improved Euler method with $n = 10$ steps gives $y(1) \approx 3.42818$ 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 $y(1) = 2e - 2 \approx 3.43656$. Thus our improved Euler approximation underestimates the actual value by about 0.24% (as compared with the 7.25% error observed in the Euler approximation of the Section 2.4 project).

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 the Section 2.4 project). 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 the improved Euler method to the initial value problem in (1), we first define the right-hand function $f(x, y) = x + y$ in the differential equation.

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

To approximate the solution with initial value $y(x_0) = y_0$ 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
0.1
```

After we initialize the values of x and y,

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

the improved Euler method itself is implemented by the following `Do` loop, which carries out the iteration

$$k_1 = f(x_n, y_n), \qquad k_2 = f(x_n + h, y_n + h\,k_1),$$
$$k = \tfrac{1}{2}(k_1 + k_2),$$
$$y_{n+1} = y_n + h\,k, \qquad x_{n+1} = x_n + h$$

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

```
Do[  k1 = f[x,y];           (* left-hand slope     *)
     k2 = f[x+h, y+h*k1];   (* right-hand slope    *)
      k  = (k1 + k2)/2;     (* average slope       *)
      y  = y + h*k;         (* improved Euler step *)
      x = x + h;            (* update x            *)
      Print[x,"     ",y],   (* display x and y     *)
     {i,1,n} ]

0.1      1.11
0.2      1.24205
0.3      1.39847
0.4      1.5818
0.5      1.79489
0.6      2.04086
0.7      2.32315
0.8      2.64558
0.9      3.01236
1.       3.42816
```

Note that x is updated after y in order that the computation $k_1 = f(x, y)$ 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 the improved Euler method with $n = 10$ steps gives $y(1) \approx 3.42816$ 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 $y(1) = 2e - 2 \approx 3.43656$. Thus our improved Euler approximation underestimates the actual value by about 0.24% (as compared with the 7.25% error observed in the Euler approximation of the Section 2.4 project).

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 the Section 2.4 project). 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 the improved Euler 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 $y(x_0) = y_0$ 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
h =
    0.1000
```

After we initialize the values of x and y,

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

and the column vectors $\mathbf{X}$ and $\mathbf{Y}$ of approximate values

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

the improved Euler method itself is implemented by the following **`for`** loop, which carries out the iteration

$$k_1 = f(x_n, y_n), \qquad k_2 = f(x_n + h, y_n + h\,k_1),$$
$$k = \tfrac{1}{2}(k_1 + k_2),$$
$$y_{n+1} = y_n + h\,k, \qquad x_{n+1} = x_n + h$$

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,y+h*k1);         % right-hand slope
     k = (k1 + k2)/2;            % average 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    1.1100
    0.2000    1.2421
    0.3000    1.3985
    0.4000    1.5818
    0.5000    1.7949
    0.6000    2.0409
    0.7000    2.3231
    0.8000    2.6456
    0.9000    3.0124
    1.0000    3.4282
```

In particular, we see that $y(1) \approx 3.4282$ 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    3.4282
```

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 improved Euler approximation underestimates the actual value by about 0.24% (as compared with the 7.25% error observed in the Euler approximation of the Section 2.4 project).

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 Improved Euler Method

The **for** loop above is a bit long for ready entry in MATLAB's command mode. The following function **impeuler** was defined by simple editing of the function **euler1** of Project 2.4.

```
function  [X,Y] = impeuler(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,y+h*k1);        % right-hand slope
    k = (k1 + k2)/2;           % average slope
    y = y + h*k;               % improved Euler 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 **impeuler.m**, we need only assume also that the function $f(x, y)$ has been defined and saved in the file **f.m**.

The function **impeuler** applies the improved Euler 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] = impeuler(0,1, 1, 10);
```

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].