Euler's Method

euler.mws

Douglas B. Meade (meade@math.sc.edu)

1 Overview

The direction fields introduced in the previous section provide a natural way to construct numerical solutions to an initial value problem (IVP). This method, which is commonly called *Euler's Method*, is implemented as an optional argument to the version of dsolve (and the DEtools package) found in Maple V, Release 4 (see ?dsolve,classical and ?DEtools,DEplot), but it is more instructive to create our own implementation. This implementation will evolve as we see the ideas behind Euler's method we will be able to construct our own implementation of Euler's Method. This procedure will then be used in a variety of problems.

2 Objectives

- to motivate and understand Euler's method
- to develop a Maple implementation, Euler, of Euler's method
- to use Euler to investigate new problems

3 New Maple Commands

- -> the arrow operator (used for defining functions in Maple)
- for .. do .. od the main Maple loop structure
- display combines several plots into a composite plot (from the plots package)
- proc define a Mape procedure (or function)

- seq construct a sequence of values

Note that while the default numerical method used in DEplot is a fourth-order Runge–Kutta method, a number of other "classical" Euler-type methods are also available: Euler's method ('euler'), modified Euler's method ('impoly'), and improved Euler's method ('heunform'). These can be specified using the method=; please see the on-line help for dsolve,classical for a full listing of available methods and their usage..

4 Introduction

The basic idea behind Euler's method is to use the slope at the current point – which is KNOWN (why?) – to determine the direction in which the solution should move. The next point on the approximate solution is found by taking a step, with a specified stepsize (h) in this direction. The process now repeats with the new current point.

5 Example 1 (p. 25)

Use Euler's Method with stepsize $h = .1$ to approximate the solution to the initial value problem $y' = x\sqrt{y}$, y(1) = 4: at the points x=1.1, 1.2, 1.3, 1.4, and 1.5.

5.1 Solution

```
> restart;
> with( plots ):
> with( DEtools ):
```

Following the algorithm developed in the text, we have must identify the following four pieces of information for this problem:

5.1.1 the function, f, that is the right-hand side of the differential equation

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

$$f := (x,\, y) \to x\,\sqrt{y}$$

5.1.2 the stepsize

```
> h := 0.1;
```

$$h := .1$$

5.1.3 the number of steps to be taken

```
> N := 5;
```

$$N := 5$$

5.1.4 and the initial condition

```
> X[0] := 1;
> Y[0] := 4;
```

$$X_0 := 1$$

$$Y_0 := 4$$

5.1.5 Comment on Maple usage – functions

Note that the function f is entered as a function, not an expression. The main reason for this is that the value of the function can be obtained using the standard notation, *e.g.*, f(1,4) and not the more cumbersome eval(subs(x=1, y=4, f)). Think about the simplicity that this implementation brings as you evaluate the next set of commands.

The slope at the initial point ($x = 1$, $y = 4$) is

```
> f( X[0], Y[0] );
```

$$2$$

so the approximate value of the solution at the first point ($x = 1.1$) is approximated to be

```
> X[1] := X[0] + h;
> Y[1] := Y[0] + h*f( X[0], Y[0] );
```

$$X_1 := 1.1$$

$$Y_1 := 4.2$$

Repeating the same steps four more times yields:

```
> X[2] := X[1] + h;
> Y[2] := Y[1] + h*f( X[1], Y[1] );
```

$$X_2 := 1.2$$

$$Y_2 := 4.425432917$$

```
> X[3] := X[2] + h;
> Y[3] := Y[2] + h*f( X[2], Y[2] );
```

$$X_3 := 1.3$$

$$Y_3 := 4.677873472$$

```
> X[4] := X[3] + h;
> Y[4] := Y[3] + h*f( X[3], Y[3] );
```

$$X_4 := 1.4$$

$$Y_4 := 4.959042570$$

```
> X[5] := X[4] + h;
> Y[5] := Y[4] + h*f( X[4], Y[4] );
```

$$X_5 := 1.5$$

$$Y_5 := 5.270807280$$

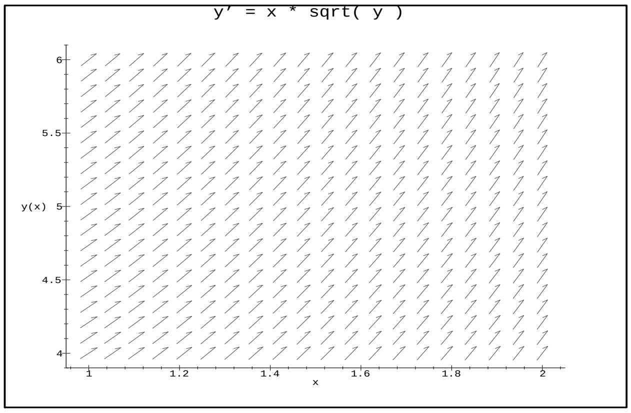
5.2 Graphical Interpretation of Solution

To understand what has just been computed, let's look at some plots. First, create the direction field for this problem

```
> ODE := diff( y(x), x ) = x*sqrt(y(x));
```

$$ODE := \frac{\partial}{\partial x}\,\mathrm{y}(x) = x\,\sqrt{\mathrm{y}(x)}$$

```
> dir_plot := DEplot( ODE, y, x=1..2, y=4..6 ):
> display( dir_plot, title=`y' = x * sqrt( y )` );
```



The points computed in Euler's Method can also be plotted

```
> euler_pts := [X[0], Y[0]], [X[1], Y[1]], [X[2], Y[2]],
> [X[3], Y[3]], [X[4], Y[4]], [X[5], Y[5]];
```

euler_pts := [1, 4], [1.1, 4.2], [1.2, 4.425432917], [1.3, 4.677873472], [1.4, 4.959042570], [1.5, 5.270807280]

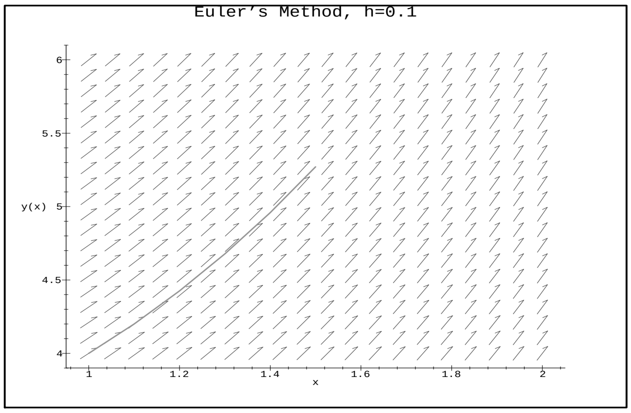
5.2.1 Comment on Maple usage – seq

Since there are only six (6) points, it's not difficult to assemble the points in the necessary form. This would not be convenient if there were considerably more points. A more general way to use Maple to assemble the data is to use the seq command (see ?seq).

```
> euler_pts := seq( [X[i],Y[i]], i=0..5 );
```

euler_pts := [1, 4], [1.1, 4.2], [1.2, 4.425432917], [1.3, 4.677873472], [1.4, 4.959042570], [1.5, 5.270807280]

```
> euler_plot := plot( [ euler_pts ], color=GREEN, thickness=3 ):
> display( { dir_plot, euler_plot }, title=`Euler's Method, h=0.1`
);
```



Observe how the computed solution closely follows the arrows in the direction field.

5.2.2 Comment on Maple usage – display

Note that the two plot commands (DEplot and plot) are both terminated with a colon (:); this prevents the displaying of the plot structure. The plots are displayed using the display command from the plots package (see ?plots and ?plots,display for more information).

6 First Implementation of Euler's Method – loops

Euler's Method is very repetitive. In practice it would not be uncommon to use several dozen, if not hundreds, of steps. This seems natural for writing as a loop. The basic looping command in Maple is the for .. do .. od command. Here is one example of how this command can be used to create a Maple implementation of Euler's Method.

```
> euler_pts := [ X[0], Y[0] ]:
> for n from 0 to N-1 do
> X[n+1] := X[n] + h;
> Y[n+1] := Y[n] + h * f( X[n], Y[n] );
> new_pt := [ X[n+1], Y[n+1] ];
> euler_pts := euler_pts, new_pt;
> od:
> euler_pts;
```

[1, 4], [1.1, 4.2], [1.2, 4.425432917], [1.3, 4.677873472], [1.4, 4.959042570], [1.5, 5.270807280]

6.1 Obtain a new approximation for the solution at $x = 1.5$ using

6.1.1 twice as many steps as the original problem

```
> h := 0.05;
> N := 10;
```

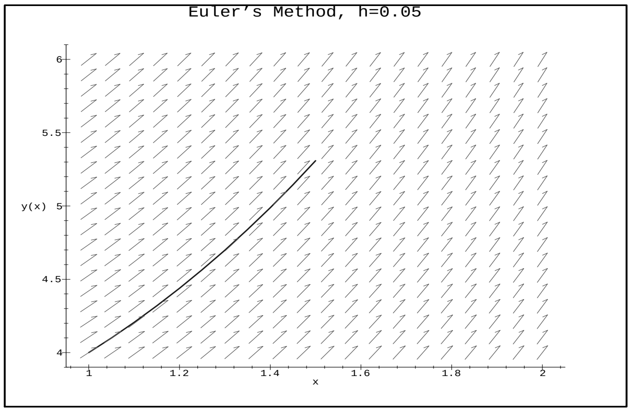
$$h := .05$$

$$N := 10$$

```
> euler_pts := [ X[0], Y[0] ]:
> for n from 0 to N-1 do
> X[n+1] := X[n] + h;
> Y[n+1] := Y[n] + h * f( X[n], Y[n] );
> new_pt := [ X[n+1], Y[n+1] ];
> euler_pts := euler_pts, new_pt;
> od:
> euler_pts;
```

[1, 4], [1.05, 4.10], [1.10, 4.206304398], [1.15, 4.319105421], [1.20, 4.438604552], [1.25, 4.565012528], [1.30, 4.698549340], [1.35, 4.839444233], [1.40, 4.987935707], [1.45, 5.144271515], [1.50, 5.308708668]

```
> euler_plot2 := plot( [ euler_pts ], color=BLUE, thickness=3 ):
> display( { dir_plot, euler_plot2 }, title=`Euler's Method, h=0.05`
);
```



6.1.2 ten times as many steps as the original problem

fill in the missing information (indicated by ??) to obtain this information

```
> h := ??;
> N := ??;
> euler_pts := [ X[0], Y[0] ]:
> for n from 0 to N-1 do
```

```
> X[n+1] := X[n] + h;
> Y[n+1] := Y[n] + h * f( X[n], Y[n] );
> new_pt := [ X[n+1], Y[n+1] ];
> euler_pts := euler_pts, new_pt;
> od:
> euler_pts;
> euler_plot3 := plot( [ euler_pts ], color=YELLOW, thickness=3
):
> display( { dir_plot, euler_plot3 }, title='Euler's Method, h=?¿
);
```

7 Final Implementation of Euler's Method – procedure

It quickly becomes tiresome to copy the commands for Euler's Method every time they are needed. It's also a little restrictive to insist that the independent variable be x and the dependent variable y. These problems can be avoided by using the proc command to create a Maple procedure for Euler's Method. Here is how that would look:

```
> Euler := proc( f, ic, h, N )
> local n, euler_pts, new_pt, X, Y;
> X[0] := ic[1];
> Y[0] := ic[2];
> euler_pts := [ X[0], Y[0] ]:
> for n from 0 to N-1 do
> X[n+1] := X[n] + h;
> Y[n+1] := Y[n] + h * f( X[n], Y[n] );
> new_pt := [ X[n+1], Y[n+1] ];
> euler_pts := euler_pts, new_pt;
> od:
> RETURN( euler_pts );
> end;
```

$$
\begin{array}{l}
\mathit{Euler} := \textbf{proc}(f,\, ic,\, h,\, N) \\
\quad \textbf{local} n,\, \mathit{euler\_pts},\, \mathit{new\_pt},\, X,\, Y; \\
\qquad X_0 := ic_1\,; \\
\qquad Y_0 := ic_2\,; \\
\qquad \mathit{euler\_pts} := [X_0,\, Y_0]\,; \\
\qquad \textbf{for}\, n\, \textbf{from}\, 0\, \textbf{to}\, N-1\, \textbf{do} \\
\qquad\quad X_{n+1} := X_n + h\,; \\
\qquad\quad Y_{n+1} := Y_n + h \times f(X_n,\, Y_n)\,; \\
\qquad\quad \mathit{new\_pt} := [X_{n+1},\, Y_{n+1}]\,; \\
\qquad\quad \mathit{euler\_pts} := \mathit{euler\_pts},\, \mathit{new\_pt} \\
\qquad \textbf{od}; \\
\qquad \mathrm{RETURN}(\mathit{euler\_pts}) \\
\quad \textbf{end}
\end{array}
$$

To illustrate the use of Euler, let's recompute the solution to the original problem.

```
> Euler( f, [1,4], 0.1, 5 );
```

[1, 4], [1.1, 4.2], [1.2, 4.425432917], [1.3, 4.677873472], [1.4, 4.959042570], [1.5, 5.270807280]

8 Problem # 16 (p. 29)

The examples considered so far have been rather simple. This problem provides a somewhat better illustration of the use of Euler's Method (and Euler). The differential equation is

```
> ODE := diff( T(t), t ) = K*(M^4-T^4);
```

$$ODE := \frac{\partial}{\partial t}\,\mathrm{T}(t) = K\,(M^4 - T^4)$$

```
> K := 40^(-4);
> M := 70;
```

$$K := \frac{1}{2560000}$$

$$M := 70$$

8.1 Solution

This problem, which is an application of Stefan's Law of Radiation, can be solved by creating a function that evaluates to the right-hand side of the differential equation (with appropriate values for the constants) and a single call to Euler.

```
> f := (t,T) -> K*(M^4-T^4);
```

$$f := (t, T) \to K\,(M^4 - T^4)$$

```
> temp_pts := Euler( f, [0,100], 0.1, 20 );
```

temp_pts := [0, 100], [.1, 97.03164063], [.2, 94.50684189], [.3, 92.32862067], [.4, 90.42790328], [.5, 88.75381371], [.6, 87.26784100], [.7, 85.94016436], [.8, 84.74724485], [.9, 83.67019423], [1.0, 82.69363876], [1.1, 81.80490839], [1.2, 80.99344589], [1.3, 80.25036815], [1.4, 79.56813510], [1.5, 78.94029616], [1.6, 78.36129346], [1.7, 77.82630729], [1.8, 77.33113343], [1.9, 76.87208475], [2.0, 76.44591161]

8.2 Problem Extensions

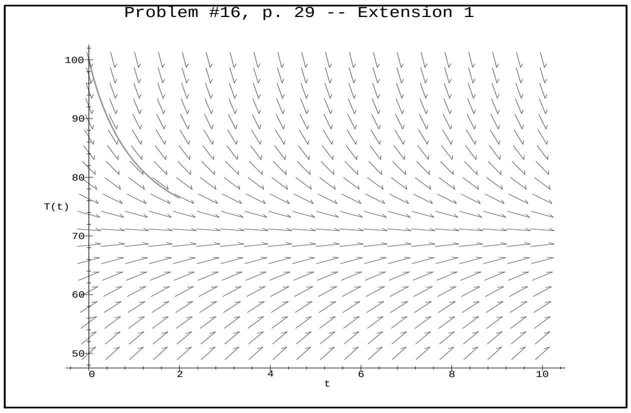
8.2.1 1.

Display the approximate solution computed by Euler's method on top of the direction field for this problem.

8.2.2 Solution

These plots can be prepared exactly as in the first example in this worksheet.

```
> dir_plot := DEplot( ODE, T, t=0..10, T=50..100 ):
> temp_plot := plot( [ temp_pts ], color=GREEN, thickness=3 ):
> display( { dir_plot, temp_plot },
> title=`Problem #16, p.  29 -- Extension 1` );
```



8.2.3 2.

Repeat the Problem to determine the temperature of the body at the end of each of the first 10 minutes using a stepsize of *a*) $h = .1$ and *b*) $h = .5$. Plot both sets of points on top of the direction field. Which solution is better? How can you tell?

8.2.4 Solution

```
> temp_pts1 := Euler( ??, ??, ??, ??  );
> temp_plot1 := plot( [ temp_pts1 ], color=??, thickness=3 ):
> temp_pts2 := Euler( ??, ??, ??, ??  );
> temp_plot2 := plot( [ temp_pts2 ], color=??, thickness=3 ):
> display( { dir_plot, temp_plot1, temp_plot2 },
> title=`Problem #16, p.  29 -- Extension 2` );
```

8.2.5 3.

What can you say about the temperature in the body as t continues to increase?

8.2.6 Solution

Sorry, you don't expect me to provide all of the answers do you?

9 Copyright

Department of Mathematics, Univ. of South Carolina, Columbia, SC 29208
(803) 777-6183
meade@math.sc.edu