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

A Maple Worksheet to Accompany Section 1.3 of Nagel and Saff

DIRECTION FIELDS and EULER'S METHOD

Math 242 -- Spring 1995

Doug Meade (meade)
25 January 1995

WWW URL: http://www.math.scarolina.edu/~meade/math242/worksheets/sec1-3.ms

---

Objectives:

- illustrate the use of Maple's DEplot command
- implement Euler's method for approximating the solution to a first-order IVP

New Maple Commands:

```
DEplot      -- produces a plot of a direction field for an ODE
display     -- for creating a single plot from several plots
                  (also used for animation)
for .. do   -- loop structure; basic syntax is as follows (see also ?for)
                    for n from a to b do
                        <do something>
                    od;
print       -- forces the printing of a quantity inside a loop
->          -- the arrow operator
                 (used for defining a function, so you can say f(2) instead of subs(x=2,f))
```

DIRECTION FIELDS

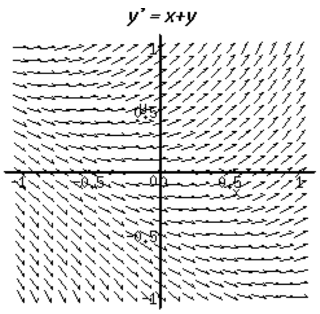
Direction fields are a simple, pictorial, technique for undestanding how solutions to a differential equation will behave, WITHOUT COMPUTING THE SOLUTION. Direction fields are particularly useful when you first look at a differential equation. This is particularly true when working with numerical solutions. The direction field provides a simple comparison for any computed solution. (If tthe arrows in the direction field are not parallel with the computed solution, then the computed solution is in error.)

The DEplot command can produce different types of plots for solutions of differential equations. Our most common use will be to create a plot of the direction field for an ODE. Here is how that might look:

> ODE := diff(y(x), x) = x+y(x);

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

> DEplot(ODE, [x,y], x=-1..1, y=-5..7, arrows=THIN, title='y' = x+y');

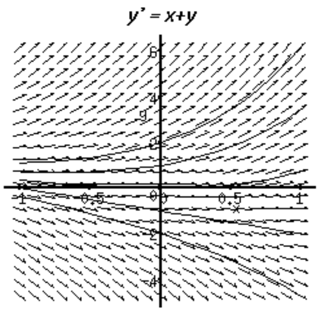


>

Note how it is important to be able to identify the dependent and independent variables in the problem.

Our other main use of DEplot will be to draw approximate solutions for specified initial conditions.

> DEplot(ODE, [x,y], x=-1..1, { [0,-2], [0,-1], [0,0], [0,1], [0,2] }, arrows=THIN, title='y' = x+y');



>

The ONLY change is that the range of the dependent variable is replaced with a set of initial conditions.

Here's an example for you to work on. First, plot the direction field for the differential equation

p' = -p*(1-p)*(5-2*p)

NOTE: You may assume t>0 and N>0.

```
> ODE := ? = ? ;
> DEplot( ODE, [ ?, ? ], ?, ?, arrows=THIN, title='?' );
```

Now, find examples of initial conditions with the property that the solution approaches i) N=0, ii) N=1, and iii) N=2.5 as t->infinity. For your final plot, include the solution curves with each of these three properties.

```
>
>
>
>
```

NOTE: This example is typical of models that arise in the study of population dynamics.

Prior to starting with Euler's Method, we need to learn a new Maple skill. To date our Maple work has been done with expressions, not functions. When working by hand, the distinction is not too critical, but we have to be more careful with Maple. To illustrate the difference, look at these commands:

```
> F := x^2;
```

$$F := x^2$$

```
> F(2);
```

$$\mathrm{x}(2)^2$$

Since F is just an expression, the request for F(2) makes no sense to Maple. In order to evaluate the expression F with x=2, it is necessary to use substitute:

```
> subs( x=2, F );
```

$$4$$

For most of our work, expressions (and subs) are what I recommend. But, there are times when we really do want to be able to write F(2) and have Maple understand our request. To achieve this it is necessary to define F as a function using the arrow operator (->). Here's how this might look:

```
> F := (x) -> x^2;
```

$$F := x \rightarrow x^2$$

```
> F(2);
```

$$4$$

This will be used in our computations with Euler's Method.

EXAMPLE 2 (p. 17)

The goal is to use Euler's method with step size h=0.1 to approximate the solution to the IVP

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

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

```
> IC := y(1)=4;
```

$$IC := \mathrm{y}(1) = 4$$

The first step should always be to look at the direction field

```
> DEplot( ODE, [x,y], x=0..2, y=0..5, arrows=THIN, title='y' = x * sqrt(y)' );
```

Five pieces of information are needed to program Euler's method for the solution of the IVP

```
> restart;
  with( plots ): with( DEtools ):
> ODE := diff( y(x), x ) = f(x,y(x));
  IC := y(X[0])=Y[0];
```

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

$$IC := \mathrm{y}\left(X_0\right) = Y_0$$

1. The slope of the FUNCTION at any point (x,y) -- remember that f specifies the slope of the solution at the point (x,y)

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

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

2. The location, X[0], where the ''initial condition'' is specified

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

$$X_0 := 1$$

3. The value, Y[0], of the function at X

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

$$Y_0 := 4$$

4. The step size, h.

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

$$h := .1$$

5. The number of steps to be taken, N.

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

$$N := 5$$

Now that the problem is specified, let's compute approximate values for the solution

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

$$X_1 := 1.1$$

$$Y_1 := 4.2$$

```
> 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]);
```

$$X_5 := 1.5$$

$$Y_5 := 5.270807280$$

These results are very close to the ones given in the book.

Let's see how these numbers look relative to the direction field. Here's how I can superimpose two plots to form a new plot.

```
> 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]];
```

$$euler\_pts := [1, 4], [1.1, 4.2], [1.2, 4.425432917], [1.3, 4.677873472], [1.4, 4.959042570], [1.5, 5.270807280]$$

```
> euler_plot := plot( [ euler_pts ], color=GREEN ):        # NOTE THE  COLON (:)
> dfield_plot := DEplot( ODE, [x,y], x=0..2, y=3..5, arrows=THIN ):
> display( {dfield_plot, euler_plot }, title=‘Example 2 (h=0.1), p. 17‘ );
>
```

Observe how the computed solution is always moving in the direction in which the arrows point.

The steps in Euler's method can be continued as long as you have the patience to type in the different lines. Since the steps in each iteration are the same, it seems as though this would be an ideal place to use a loop.

```
> restart;
  with( plots ): with( DEtools ):
> ODE := diff( y(x), x ) = f(x,y(x)):
  IC := y(X[0])=Y[0]:
```

Enter the data for the problem here

```
> f := (x,y) -> x*sqrt(y);
  X[0] := 0;
  Y[0] := 4;
  h := 0.1;
  N := 15;
```

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

$$X_0 := 0$$

$$Y_0 := 4$$

$$h := .1$$

$$N := 15$$

Let's check that the problem has been correctly entered

```
> ODE;
  IC;
```

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

$$\mathrm{y}(0) = 4$$

Maple Implementation of Euler’s Method
(Do not change the next 8 lines.)

```
> 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;
    print( new_pt );
  od:
```

$$[.1, 4]$$

$$[.2, 4.02]$$

$$[.3, 4.060099875]$$

$$[.4, 4.120548944]$$

$$[.5, 4.201745485]$$

$$[.6, 4.304236283]$$

$$[.7, 4.428716204]$$

$$[.8, 4.576027811]$$

$$[.9, 4.747161028]$$

$$[1.0, 4.943252854]$$

$$[1.1, 5.165587126]$$

$$[1.2, 5.415594334]$$

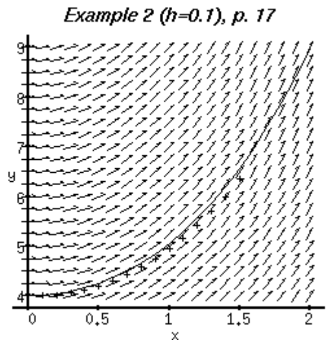
$$[1.3, 5.694851489]$$

$$[1.4, 6.005082033]$$

$$[1.5, 6.348155798]$$

This time when we draw the plot, let’s include the solution curve that satisfies the initial condition.

```
> euler_plot := plot( [ euler_pts ], style=POINT, color=GREEN ):
  dfield_plot := DEplot( ODE, [x,y], x=0..2, { [X[0],Y[0]] }, arrows=THIN ):
  display( {dfield_plot, euler_plot }, title=‘Example 2 (h=0.1), p. 17‘ );
```




```
>
```

Observe that the plot includes the both the approximate solution from Euler’s Method and the solution curve that satisfies the initial condition. Notice how the two curves separate as we move away from the initial condition.

Now it’s your turn to play with this example.

1.
For a numerical method to be effective it must have the property that solutions become more accurate as smaller steps are taken. Of course this means that it will be necessary to take more steps, but overall the solution will be better.

To see what I am talking about, change the step size to h=0.01 and recompute the approximate solution on the interval [1,1.5]. Compare the new solution with the values printed in the text (see Table 1.1 on p. 17).

2.
Suppose the initial condition is changed to y(0)=4 and the solution is to be determined on the interval [0.1.5]. Make the necessary changes in the above commands to compute this solution with h=0.1 (and with h=0.01).

ANOTHER EXAMPLE

Consider the first-order linear IVP with RHS f(x,y) = -y*sin(x) + cos(x) and initial condition y(0)=3.

Use Euler's Method to approximate the solution at t=1, t=2, t=4, t=8, and t=16 for step sizes h = 1, h=0.5, and h=0.25.

NOTE: WORK WITH TWO PARTNERS ON THIS. EACH OF YOU COMPUTE THE SOLUTION WITH ONE STEP SIZE, THEN COMPARE THE SOLUTIONS. HOW WELL DO THEY APPROXIMATE THE TRUE SOLUTION?

```
> f := (x,y) -> ? :
  X[0] := ?:
  Y[0] := ?:
  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;
    print( new_pt );
  od:
```

```
> euler_plot := plot( [ euler_pts ], style=POINT, color=GREEN ):
  dirfield := DEplot( ODE, [x,y], x=0..?, { [X[0],Y[0]] }, arrows=THIN, stepsize=0.25):
  display( { euler_plot, dirfield }, title=‘y’=-y*sin(x)+cos(x), y(0)=0‘ );
>
```

APPENDIX A. Additional Examples

Here are some additional examples that you might like to consider. For each problem, you should carry out the following steps:

a) Draw the direction field on a suitable rectangle.

b) From the plot of the direction field form a conclusion about how the solutions behave (particularly as t becomes large). Look for evidence that solutions behave differently for different initial conditions.

c) Plot a few solution curves to confirm your conclusions from b).

```
> diff( y(t), t ) = sin(t) + 1/2*y(t);
>
> diff( y(t), t ) = sin(t) - 1/2*y(t);
>
> diff( y(t), t ) = 2-t-t*y(t);
>
> diff( y(t), t ) = exp( -t ) -  2*y(t);
>
> diff( y(t), t ) = 1 + exp(t) - y(t);
>
> diff( y(t), t ) = 3*t^2/(3*y(t)^2-4);
>
> diff( y(t), t ) = - sin(2*t)/cos(3*y(t));
>
> diff( y(t), t ) = y(t)^2 - t;
>
```

APPENDIX B: Verifying that a function is a solution to an ODE..

Example from class on Wednesday, January 25, 1995.

Verify that y=x^3 is a solution to

```
> ODE := x*diff( y(x),x ) = 3*y(x);
```

```
> subs( y(x)=x^3, ODE );
```

We'd like to have Maple evaluate the derivative. This sounds like a job for simpllfy!

```
> simplify( " );
```

Are there other solutions to this ODE?

When is y=c*x^3 a solution? (We know it is for c=1, but what about other choices of c?)

```
> simplify( subs( y(x)=c*x^3, ODE ) );
```

This shows that FOR ANY REAL NUMBER c, y=c*x^3 is a solution to the ODE.

Are all functions solutions to this ODE?

```
> simplify( subs( y(x)=x^2, ODE ) );
```

NOPE, this function isn't a solution.

```
> simplify( subs( y(x)=sin(x), ODE ) );
>
```

Neither is this function.

We will shortly learn that the ONLY solutions to this ODE are the functions y=c*x^3.

END OF WORKSHEET

```
>
```