Tues Jan 30

2.4 Euler's method for solving first order differential equations

Announcements:
- last page of HW due tomorrow is corrected HW score] uncorrected HW score
- 2.4-2.6 HW is not supposed to be bold & underline (HW due tomorrow is through §2.3)
- On Thursday lab, bring at least a calculator, or smartphone w Wolfram alpha. best: bring laptop with access to Matlab.

Warm-up Exercise:

Consider the IVP $\begin{cases} y'(x) = f(x,y) \\ y(1) = 2 \end{cases}$ rate of change function ///

If all you know is the value of the slope function $f(x,y)$ at the initial point $(1,2)$, say $y'(1) = f(1,2) = 3$, what's your best approximation for $y(1.1)$? for $y(.9)$?

Ans ≈ 2.3 ≈ 1.7

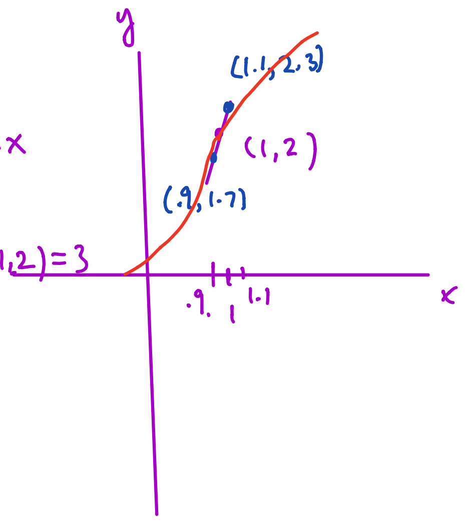
'til 10:48

$y(1.1) \approx y(1) + dy$
$= y(1) + y'(1)\Delta x$
$= 2 + 3(.1)$
$= 2.3.$

$\Delta y \approx y'(x)\Delta x$

$\Delta x = .1$

$y'(1) = f(1,2) = 3$



$y(.9) \approx y(1) + dy$
$= 2 + y'(1)\Delta x$
$= 2 + 3(-.1)$
$= 1.7$

$y' = f(x,y)$

@ (x_0, y_0)

$\Delta y \approx f(x_0,y_0)\Delta x$

2.4 Euler's method.

In these notes we will study numerical methods for approximating solutions to first order differential equations. Later in the course we will see how higher order differential equations can be converted into first order systems of differential equations. It turns out that there is a natural way to generalize what we do now in the context of a single first order differential equations, to systems of first order differential equations. So understanding this material will be an important step in understanding numerical solutions to higher order differential equations and to systems of differential equations later on. I wrote these notes using the software package Maple. Your lab questions on Thursday will let you do analogous computations in Matlab. Today we'll focus on Euler's method. Tomorrow we'll study "improved Euler" (section 2.5) and Runge Kutta (section 2.6).

Euler's Method:

The most basic method of approximating solutions to differential equations is called Euler's method, after the 1700's mathematician who first formulated it. Consider the initial value problem

$$\text{IVP}\begin{cases} \frac{dy}{dx} = f(x,y) \\ y(x_0) = y_0. \end{cases}$$

We make repeated use of the familiar (tangent line) approximation

$$y(x+\Delta x) \approx y(x) + y'(x)\Delta x \quad \bullet$$

where we substitute in the slope function $f(x,y)$, for $y'(x)$.

As we iterate this procedure we and the text will write "h" <u>fixed step size, $\Delta x := h$.</u> As we increment the inputs x and outputs y we are approximating the graph of the solution to the IVP. We begin at the initial point (x_0, y_0). Then the next horizontal value on the discrete graph will be

$$x_1 = x_0 + h = x_0 + \Delta x$$

And the next vertical value y_1 (approximating the exact value $y(x_1)$) is

$$y_1 = y_0 + f(x_0, y_0)h. \qquad = y_0 + y'(x_0)\Delta x$$

In general, if we've approximated (x_j, y_j) we set

$$x_{j+1} = x_j + h \quad \bullet$$
$$y_{j+1} = y_j + f(x_j, y_j)h.$$

We could also approximate in the $-x$ direction from the initial point (x_0, y_0), by defining e.g.

$$x_{-1} = x_0 - h$$
$$y_{-1} = y_0 - hf(x_0, y_0)$$

and more generally via

$$x_{-j-1} = x_{-j} - h$$
$$y_{-j-1} = y_{-j} - hf(x_{-j}, y_{-j})$$

...etc.

Below is a graphical representation of the Euler method, illustrated for the IVP

$$\begin{cases} y'(x) = 1 - 3x + y = f(x,y) \\ y(0) = 0 \end{cases}$$

with step size $h = 0.2$.

Exercise 1 Find the numerical values for several of the labeled points.

$x_0 = 0 \quad y_0 = 0 \quad f(x_0, y_0) = f(0,0) = 1$

$x_1 = .2 \quad \Delta y \approx f(0,0) \cdot h = 1(.2) = .2$

$y_1 = y_0 + \Delta y$

$= 0 + .2 = .2$

$x_1 = .2, \; y_1 = .2 \quad f(.2,.2) = 1 - .6 + .2$

$= .6$

$\Delta y \approx (.6)(.2)$

$= .12$

$y_2 = y_1 + \Delta y = .2 + .12 = .32$

$x_2 = .4, \; y_2 = .32$

$x_{-1} = -.2,$ from $(x_0, y_0) = (0,0)$.

$f(0,0) = 1$

$\Delta y = f(0,0)(\Delta x)$

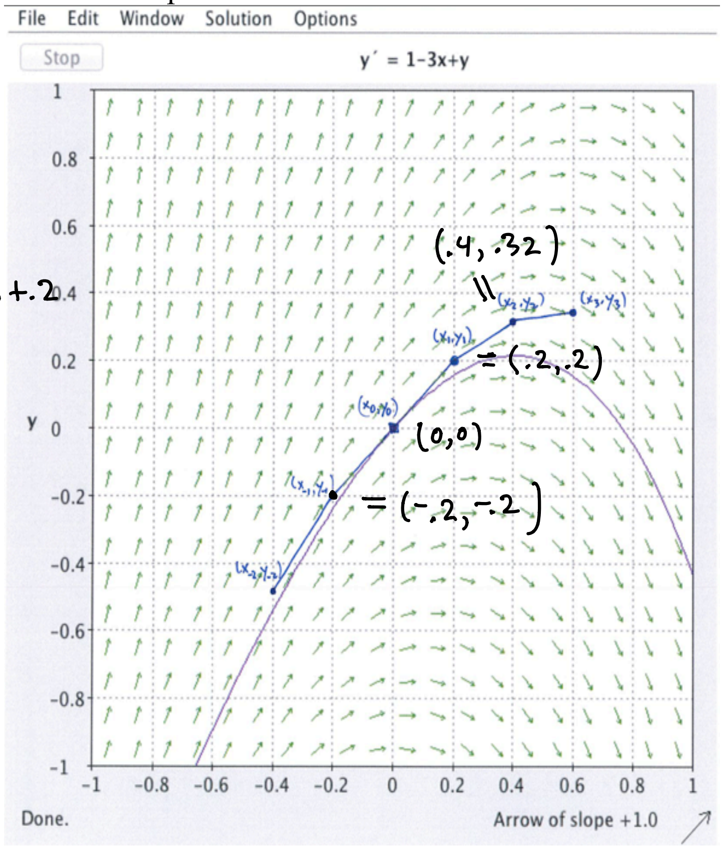
$= 1(-.2)$

$= -.2$

$y_{-1} = y_0 + \Delta y$

$= 0 - .2 = -.2$

$x_{-1} = -.2, \; y_{-1} = -.2$



As a running example in the remainder of these notes we will use one of our favorite IVP's from the time of Calculus, namely the initial value problem for $y(x)$,

$$y'(x) = y = f(x,y)$$
$$y(0) = 1.$$

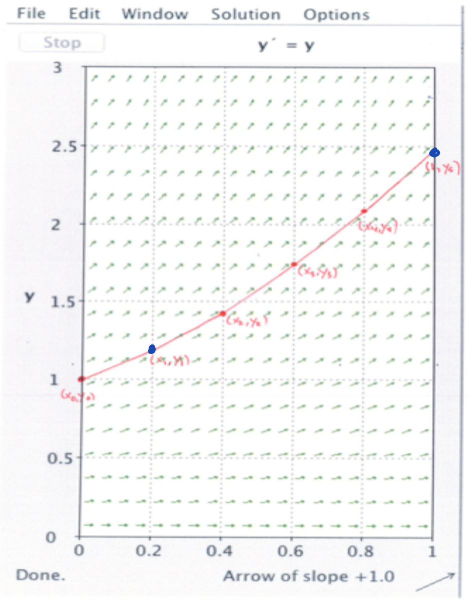
We know that $\underline{y = e^x}$ is the solution, so we'll be able to compare approximate and exact solution values.

Exercise 2 Work out by hand the approximate solution to the IVP above on the interval $[0, 1]$, with $n = 5$ subdivisions and $h = 0.2$, using Euler's method. This table might help organize the arithmetic. In this differential equation the slope function is $f(x, y) = y$ so is especially easy to compute.

step i	x_i	y_i	slope $f(x_i, y_i) = y_i$	Δx	$\Delta y = f(x_i, y_i)\Delta x$	x_{i+1}	$y_i + \Delta y$
0	0	1	1	0.2	$0.2 = 1(.2)$	0.2	$1.2 = 1 + (.2)$
1	0.2	1.2	1.2	0.2	$(1.2)(.2) = .24$	.4	$1.2 + .24 = 1.44$ | $1.2 + (1.2)(.2) = (1.2)(1+.2) = (1.2)^2$
2	.4	1.44	$1.44 = (1.2)^2$	0.2	$.288$ or $(1.2)^2(.2)$	.6	$1.44 + .288 = 1.728$ | $(1.2)^2 + (.2)(1.2)^2 = (1.2)^2(1+.2) = (1.2)^3$
3	.6	$(1.2)^3$	$(1.2)^3$	.2	$(1.2)^3(.2)$	.8	$(1.2)^3 + .2(1.2)^3 = (1.2)^4$
4	.8	$(1.2)^4$	$(1.2)^4$	.2	$(1.2)^4(.2)$	1	$(1.2)^4 + (1.2)^4(.2) = (1.2)^5 = 2.488$
5	1	$2.488 = (1.2)^5$					

Euler Table

Your work should be consistent with the picture below.



Here is the automated computation for the previous exercise, and a graph comparing the approximate solution points to the actual solution graph:

Initialize:

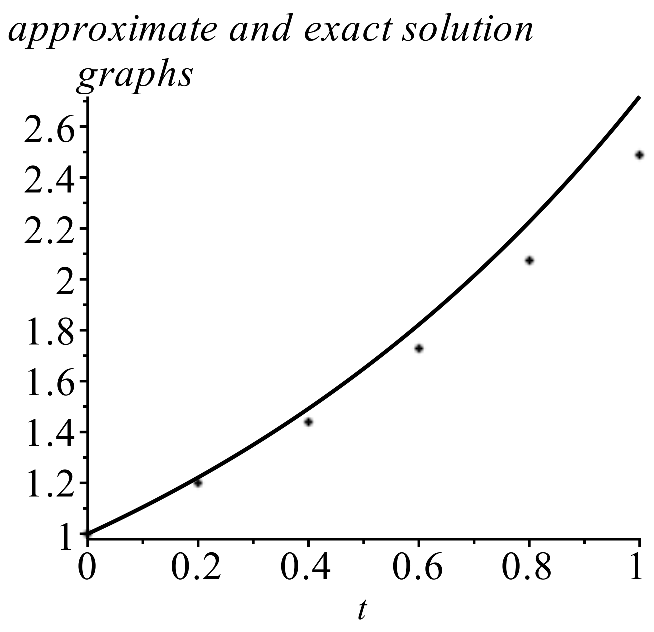
```
> restart : #clear all memory, if you wish
> Digits := 6 : #use floating point arithmetic with 6 digits.  could use more if desired
> unassign(`x`, `y`); # in case you used the letters elsewhere
> f := (x, y)→y; # slope field function for the DE  y'(x)=y
                          # change for different DE's!!!
                                   f := (x, y)→y                                    (5)
> x[0] := 0; y[0] := 1; #initial point
  h := 0.2; n := 5;   #step size and number of steps
                                      x_0 := 0
                                      y_0 := 1
                                      h := 0.2
                                      n := 5                                       (6)
> exactsol := x→e^x; #exact solution for the IVP  y'=y, y(0)=1
                             #change (or omit) for different IVP's!!!
                                exactsol := x→e^x                                  (7)
>
```

Euler Loop

```
> for i from 0 to n do  #this is an iteration loop, with index "i" running from 0 to n
     print(i, x[i], y[i], exactsol(x[i]));
          #print iteration step, current x,y, values, and exact solution value
    { k := f(x[i], y[i]); #current slope function value
    { x[i + 1] := x[i] + h;
    { y[i + 1] := y[i] + h·k;
  end do: #how to end a do loop in Maple
                                  0, 0, 1, 1
                              1, 0.2, 1.2, 1.22140
                             2, 0.4, 1.44, 1.49182
                            3, 0.6, 1.728, 1.82212
                           4, 0.8, 2.0736, 2.22554
                          5, 1.0, 2.48832, 2.71828                                 (8)
```

Plot results:

```
> with(plots) :
  Eulerapprox := pointplot({seq([x[i], y[i]], i = 0 ..n)}) : #approximate soln points
   exactsolgraph := plot(exactsol(t), t = 0 ..1, `color` = `black`) :  #exact soln graph
       #used t because x has been defined to be something else
   display({Eulerapprox, exactsolgraph}, title = `approximate and exact solution graphs`);
```



Exercise 3 Why are our approximations too small in this case, compared to the exact solution?

It should be that as your step size $h = \Delta x$ gets smaller, your approximations to the actual solution get better. This is true if your computer can do exact math (which it can't), but in practice you don't want to make the computer do too many computations because of problems with round-off error and computation time, so for example, choosing $h = .0000001$ would not be practical. But, trying $h = 0.01$ in our previous initial value problem should be instructive.

If we change the n-value to 100 and keep the other data the same we can rerun our experiment, copying, pasting, modifying previous work.

Initialize

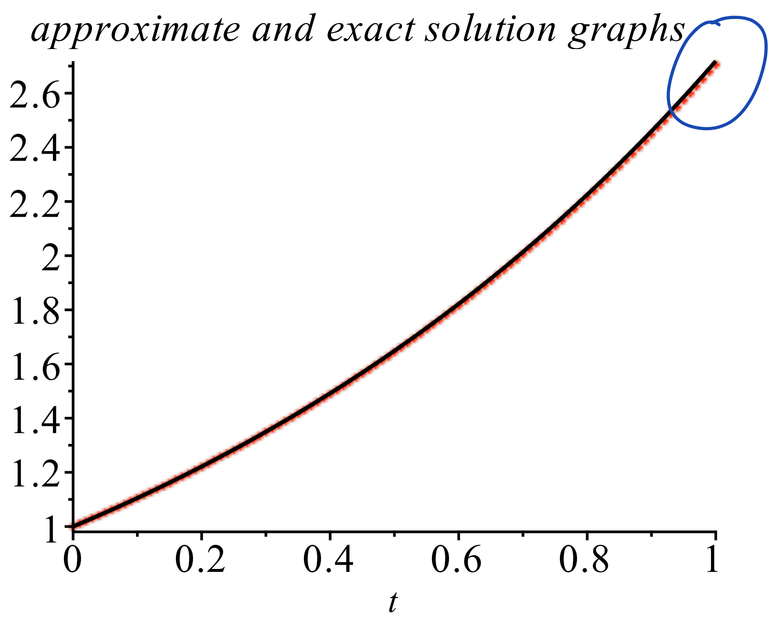
```
> restart : #clear all memory, if you wish
> unassign(`x`, `y`); # in case you used the letters elsewhere
> Digits := 6 :
> f := (x, y)→y : # slope field function for the DE  y′(x)=y
                    # change for different DE's!!!
> x[0] := 0 : y[0] := 1 : #initial point
  h := 0.01 : n := 100 :  #step size and number of steps
> exactsol := x→e^x : #exact solution for the IVP  y'=y, y(0)=1
                       #change (or omit) for different IVP's!!!
>
```

Euler Loop (modified using an "if then" conditional clause to only print every 10 steps)

```
> for i from 0 to n do  #this is an iteration loop, with index "i" running from 0 to n
      if frac(i/10) = 0
          then print(i, x[i], y[i], exactsol(x[i]));
      end if: #only print every tenth time
    k := f(x[i], y[i]); #current slope function value
    x[i + 1] := x[i] + h;
    y[i + 1] := y[i] + h·k;
  end do: #how to end a for loop in Maple
                        0, 0, 1, 1
              10, 0.10, 1.10463, 1.10517
              20, 0.20, 1.22020, 1.22140
              30, 0.30, 1.34784, 1.34986
              40, 0.40, 1.48887, 1.49182
              50, 0.50, 1.64463, 1.64872
              60, 0.60, 1.81670, 1.82212
              70, 0.70, 2.00676, 2.01375
              80, 0.80, 2.21672, 2.22554
              90, 0.90, 2.44864, 2.45960
              100, 1.00, 2.70483, 2.71828                              (9)
```

plot results:

```
> with(plots) :
  Eulerapprox ≔ pointplot({seq([x[i], y[i]], i = 0 ..n)}, color = red) :
   exactsolgraph ≔ plot(exactsol(t), t = 0 ..1, `color` = `black`) :
      #used t because x was already used has been defined to be something else
   display({Eulerapprox, exactsolgraph}, title = `approximate and exact solution graphs`);
```


approximate and exact solution graphs
t