Tuesday start here.

Recall: $mv' = -mg - kv$

$\div m \quad v' = -g - \rho v = -\rho\left(v + \frac{g}{\rho}\right)$; $v_\tau = -\frac{g}{\rho}$

Exercise 4: Now consider the linear drag model for the same deadbolt, with the same initial velocity of $5\,g = 49\ \frac{m}{s}$. We'll assume that our deadbolt has a measured terminal velocity of $v_\tau = -245\ \frac{m}{s} = -25\,g$, so $|v_\tau| = 25\,g = \frac{g}{\rho} \Rightarrow \rho = .04$ (convenient). So, from our earlier work:

$|v_\tau| = \frac{g}{\rho}$

$25g = \frac{g}{\rho}$

$\rho = \frac{1}{25}$

$$* \quad v = v_\tau + (v_0 - v_\tau)\, e^{-\rho t}$$

$$y = y_0 + t\,v_\tau + \frac{(v_0 - v_\tau)\,(1 - e^{-\rho t})}{\rho}$$

So,

$$* \quad v = -\frac{g}{\rho} + \left(v_0 + \frac{g}{\rho}\right) e^{-\rho t} = -245 + 294\, e^{-.04\,t}.$$

$$** \quad y = 0 - 245\,t + \frac{294}{.04}\left(1 - e^{-.04\,t}\right).$$

set $v(t)=0$ solve for t

$y(t_1)$ where $v(t_1)=0$.

When does the object reach its maximum height, what is this height, and how long does the object fall? Compare to the no-drag case with the same initial velocity, in Exercise 3.

Maple check, and then work:

$-245 + 294\, e^{-.04t} = 0$ ...

solve $y(t)=0$ for t, say $t=t_2$

object fell for $t_2 - t_1$ sec.

```
> with(DEtools):
> g := 9.8; ρ := .04; v0 := 49;
```

$$g := 9.8,\quad \rho := 0.04,\quad v0 := 49 \qquad \textbf{(8)}$$

```
> dsolve({v'(t) = -g - ρ·v(t), v(0) = v0}, v(t));
```

$$v(t) = -245 + 294\, e^{-\frac{1}{25}t} \qquad \textbf{(9)}$$ *

```
> v2 := t→-245.0 + 294 e^(-1/25 t);
```

$$v2 := t \to -245.0 + 294\, e^{-\frac{1}{25}t} \qquad \textbf{(10)}$$

```
> solve(v2(t) = 0, t);
```

$$4.5580 \qquad \textbf{(11)}$$

how long deadbolt is going up → w/o drag it went up for 5 sec. makes sense

```
> y2 := t→y0 + int(-g/ρ + e^(-ρ s)·(v0 + g/ρ), s = 0..t);
```

$$y2 := t \to y0 + \int_0^t \left(-\frac{g}{\rho} + e^{-\rho s}\left(v0 + \frac{g}{\rho}\right)\right) ds \qquad \textbf{(12)}$$

```
> v2(t);
  294/.04;
  y0 := 0;
  y2(t);
```

$$-245.0 + 294\, e^{-\frac{1}{25} t}$$

$$7350.0$$

$$y0 := 0$$

$$7350. - 245.\, t - 7350.\, e^{-0.040000\, t} \qquad (13)$$

t is in 2 very different places!

** y(t) = 0

```
> solve(v2(t) = 0, t);
  solve(y2(t) = 0, t);
  y2(4.558);
```

$t_1 =$ 4.5580

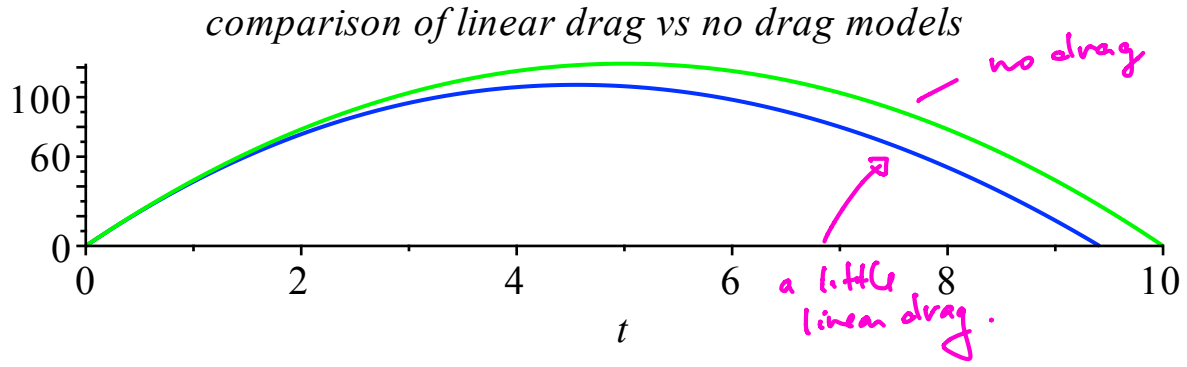
$t_2 =$ 9.4110, 0. ← when deadbolt lands (vs. 10 sec w/o drag)

108.28 ← max height (compared to 122.5 m w/o drag ... makes sense) (14)

falling for $t_2 - t_1 = 9.41 - 4.56 = 4.85$ sec. longer to fall down

picture:

```
> with(plots):
  plot1 := plot(y1(t), t = 0..10, color = green):
  plot2 := plot(y2(t), t = 0..9.4110, color = blue):
  display({plot1, plot2}, title = `comparison of linear drag vs no drag models`);
```




```
>
```

Math 2250-004
Week 4 notes, January 30-February 3. Sections 2.3, 2.4-2.6, 3.1

Tuesday: finish §2.3 example. Then

Mon Jan 30: Use notes from last Friday to discuss improved velocity models, section 2.3

Tues Jan 31 and Wed Feb 1: Sections 2.4-2.6 numerical methods for differential equations.

You may wish to download this file in Maple from our class homework or lecture page, or by directly opening the URL

http://www.math.utah.edu/~korevaar/2250spring17/week4.mw

from Maple. It contains discussion and Maple commands which may help you with your Maple/Matlab/Mathematica work later this week, in addition to our in-class discussions.

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.

.....

If you decide to use these notes in the Math Department Computer lab (and for printing out class notes for free), Math Department login for students is as follows:

login name: Suppose your name is Public, John Q. Then your login name is

c-pcjq

(If several UU math students have the same initials as you, then you may actually be signed up as one of c-pcjq1, c-pcjq2, c-pcjq3 or c-pcjq4.)

password: If your UID is u***4986 then your password is

pcjq4986

(unless you've changed it). Even if you had to add a number to your login name, your password will only have the four letters part of your login name.

To open this document from Maple, go to our Math 2280 lecture page, then to the link above. Use the File/Open URL option inside Maple, and copy and paste the URL into the dialog box.

save a copy of this file to your home directory (using whatever name seems appropriate).

.....

In this handout 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.

We will be working through material from sections 2.4-2.6 of the text.

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

$$\frac{dy}{dx} = f(x,y)$$
$$y(x_0) = y_0.$$

slope function.

We make repeated use of the familiar approximation

$$\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}$$

(x_0, y_0) initial point

$\Delta x = h$

x_{-2} x_{-1} x_0 x_1 x_2 x_3

with fixed "step size" $\Delta x := h$ at our discretion (the smaller the better, probably, if we want to be accurate).

We consider approximating the graph of the solution to the IVP. Begin at the initial point (x_0, y_0). Let

$$x_1 = x_0 + h$$

To get the Euler approximation y_1 to the exact value $y(x_1)$ use the rate of change $f(x_0, y_0)$ at your initial point (x_0, y_0), i.e. $\Delta y \approx f(x_0, y_0)\Delta x$, so

$\frac{\Delta y}{\Delta x} \approx \frac{dy}{dx} = f(x,y)$

$$y_1 = y_0 + f(x_0, y_0)h.$$

(x_0, y_0)

(x_1, y_1)

$x_0 + h \quad y_0 + \Delta y$

$y_1 = y_0 + f(x_0, y_0)\frac{\Delta x}{h}$

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

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

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

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

with step size $h = 0.2$.

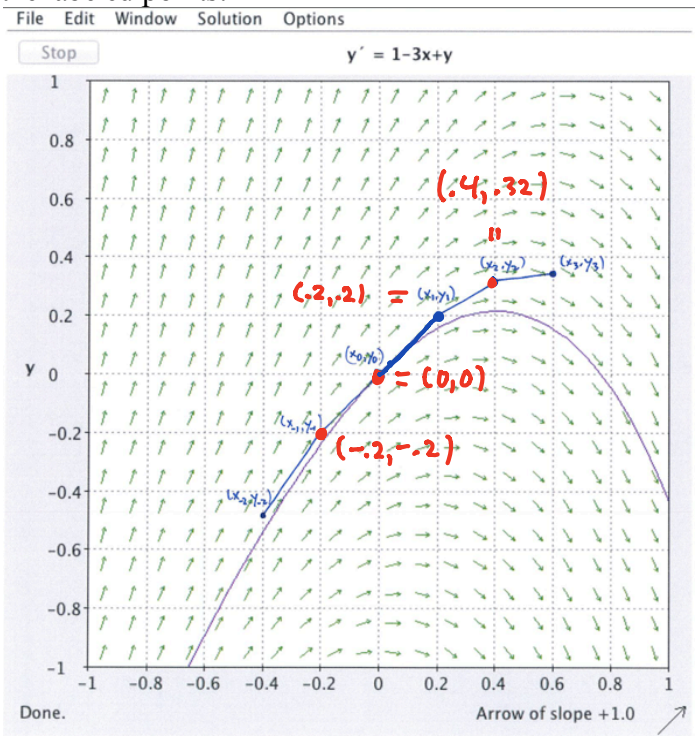
Exercise 1 Find the numerical values for several of the labeled points.

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

$x_1 = .2$ $\quad y_1 = y_0 + 1(.2) = 0 + .2 = .2$ $\quad f(.2, .2) = 1 - .6 + .2 = .6$

$x_2 = .4$ $\quad y_2 = y_1 + f(x_1, y_1) \cdot (.2) = .2 + (.6)(.2) = .32$

$x_{-1} = -.2$ $\quad y_{-1} = y_0 - .2\,(f(x_0, y_0)) = 0 - .2\, f(0,0) = -.2$



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 $y(x)$,

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

$y(x) = e^x$

$(x_0, y_0) = (0, 1)$

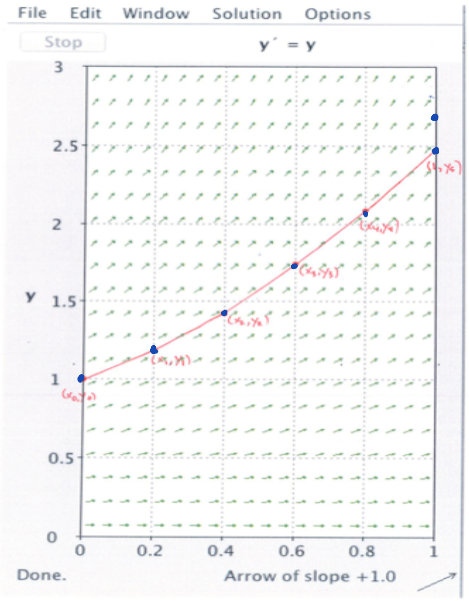
We know that $y = e^x$ is the solution.

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	$k = f(x_i, y_i)$ $= y_i$	$\Delta x = h$	$\Delta y = h \cdot k$ $.2 \cdot 1$	$x_i + \Delta x$ $0 + .2$	$y_i + \Delta y$ $1 + .2$
0	0	1	1	0.2	0.2	0.2	1.2
1	0.2	1.2	1.2	.2	$(1.2)(.2) = .24$	.4	$1.2 + (.2)(1.2) = (1.2)(1+.2) = (1.2)^2$ 1.44
2	.4	1.44	$(1.2)^2$ 1.44	.2	$(1.2)(.2)$ $.288$	.6	$(1.2)^2 + (1.2).2 = (1.2)^2(1.2)$ $1.728 = (1.2)^3$ $1.44 + .288 = 1.728$
3						.8	$(1.2)^4$
4						1	$(1.2)^5 = 2.488$
5	1	$(1.2)^5$ $= 2.488$					

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:

Exercise 3 Enter the following commands (by putting your cursor in the command fields and hitting enter). Discuss what each command is doing, and ask for any needed clarification. The first collection of commands is initializing the data. The second set is running the Euler loop. If you forget to initialize the data first, you might run into trouble trying to run the second set. No matter what is written in the document, commands are not executed until the cursor is put into a command field, and the <enter> or <return> key is pressed.

```
>
```

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!!!
```

$$f := (x, y) \rightarrow y \qquad \textbf{(1)}$$

```
> x[0] := 0; y[0] := 1; #initial point
  h := 0.2; n := 5;   #step size and number of steps
```

$$x_0 := 0$$

$$y_0 := 1$$

$$h := 0.2$$

$$n := 5 \qquad \textbf{(2)}$$

```
> exactsol := x→e^x; #exact solution for the IVP  y'=y, y(0)=1
                     #change (or omit) for different IVP's!!!
```

$$exactsol := x \rightarrow e^x \qquad \textbf{(3)}$$

```
>
```

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
```

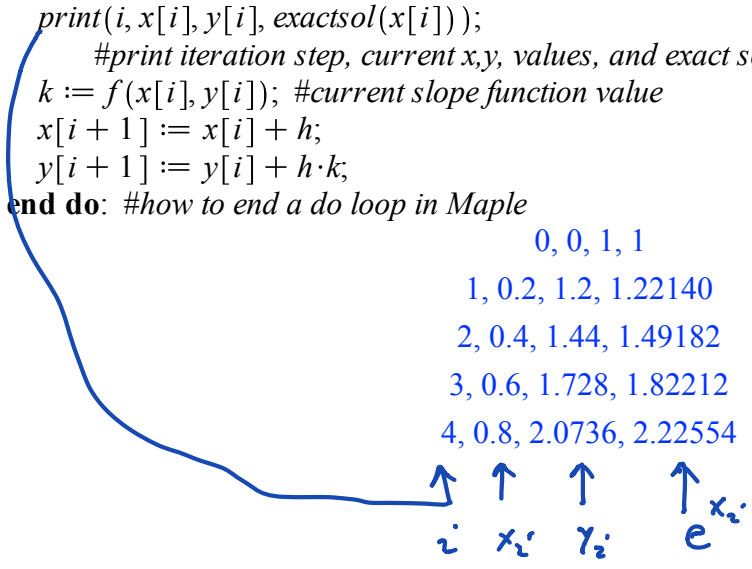
0, 0, 1, 1

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

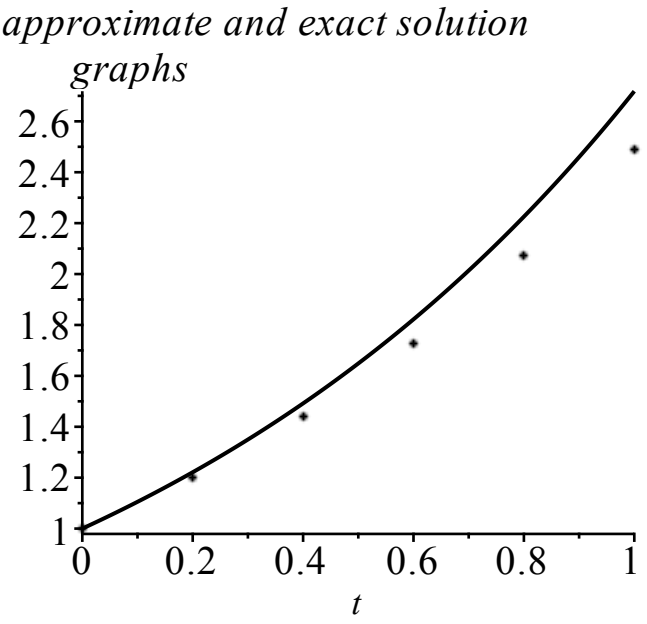


(4)

5, 1.0, 2.48832, 2.71828 what we got by hand! **(4)**

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 4: Why are your approximations too small in this case, compared to the exact solution?

because we used constant slope on each subinterval (from left endpoints, but slope function "y" gets larger as y gets larger, so we were underestimating actual slopes

It should be that as your step size h 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!!!
```

$$f := (x, y) \to y \tag{5}$$

```
> x[0] := 0; y[0] := 1; #initial point
  h := 0.01; n := 100;   #step size and number of steps
```

$$x_0 := 0$$

$$y_0 := 1$$

$$h := 0.01$$

$$n := 100 \tag{6}$$

```
> exactsol := x→e^x; #exact solution for the IVP  y'=y, y(0)=1
                     #change (or omit) for different IVP's!!!
```

$$exactsol := x \to e^{x} \tag{7}$$

```
>
```

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 ← $e^1 = e$

better estimate, but took a lot of computation

i x_i y_i e^{x_i}

(8)

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`);
```