Math 2280-002 Week 4 notes

We will not necessarily finish the material from a given day's notes on that day. We may also add or subtract some material as the week progresses, but these notes represent an in-depth outline of what we plan to cover. These notes include material from 2.3-2.6, with an introduction to 3.1-3.2.

Mon Jan 28

2.3 Improved acceleration models: linear and quadratic drag forces.

Announcements:

Warm-up Exercise:

2.3 Improved acceleration models: velocity-dependent drag

For 1-dimensional particle motion, with

$$\text{position } x(t) \text{ (or } y(t)\text{)},$$
$$\text{velocity } x'(t) = v(t), \text{ and}$$
$$\text{acceleration } x''(t) = v'(t) = a(t)$$

We have Newton's 2^{nd} law

$$m\, v'(t) = F$$

where F is the net force, possibly a sum consisting of several terms.

- By now we're very familiar with constant force $F = m\,\alpha$, where α is a constant:

$$v'(t) = \alpha$$
$$v(t) = \alpha\, t + v_0$$
$$x(t) = \frac{1}{2}\alpha\, t^2 + v_0\, t + x_0 .$$

Examples we've seen already in this course

- $\alpha = -g$ near the surface of the earth, if up is the positive direction, or $\alpha = g$ if down is the positive direction.
- α arising from a charged particle moving through a constant electric field (lab problem)
- boats or cars moving with constant acceleration or deceleration (homework).

New today !!! Combine a constant background force with a velocity-dependent drag force, at the same time. The text calls this a "resistance" force:

$$m\, v'(t) = m\,\alpha + F_R$$

Empirically/mathematically the resistance forces F_R depend on velocity, in such a way that their magnitude is

$$|F_R| \approx k\,|v|^p \quad , 1 \le p \le 2 .$$

- $p = 1$ (linear model, drag proportional to velocity):

$$m\, v'(t) = m\, \alpha - k\, v$$

This linear model makes sense for "slow" velocities, as a linearization of the frictional force function, assuming that the force function is differentiable with respect to velocity...recall Taylor series for how the velocity resistance force might depend on velocity:

$$F_R(v) = F_R(0) + F_R'(0)\, v + \frac{1}{2!} F_R''(0) v^2 + \ldots$$

$F_R(0) = 0$ and for small enough v the higher order terms might be negligable compared to the linear term, so

$$F_R(v) \approx F_R'(0)\, v \approx -k\, v \ .$$

We write $-k\, v$ with $k > 0$, since the frictional force opposes the direction of motion, so sign opposite of the velocity's.

http://en.wikipedia.org/wiki/Drag_(physics)#Very_low_Reynolds_numbers:_Stokes.27_drag

Exercise 1: Let's rewrite the linear drag model

$$m\, v'(t) = m\, \alpha - k\, v$$

as

$$v'(t) = \alpha - \rho\, v$$

where the $\rho = \frac{k}{m}$. Now construct the phase diagram for v . (Hint: there is one critical value for v.)

The value of the constant velocity solution is called the *terminal velocity,* which makes good sense when you think about the underlying physics and phase diagram.

• $p = 2$, for the power in the resistance force. This can be an appropriate model for velocities which are not "near" zero....described in terms of "Reynolds number" Accounting for the fact that the resistance opposes direction of motion we get

$$m\,v'(t) = m\,\alpha - k\,v^2 \quad \text{if } v > 0$$

$$m\,v'(t) = m\,\alpha + k\,v^2 \quad \text{if } v < 0.$$

Do you understandt the sign of the drag terms in these two cases?

http://en.wikipedia.org/wiki/Drag_(physics)#Drag_at_high_velocity

Once again letting $\rho = \frac{k}{m}$ we can rewrite the DE's as

$$v'(t) = \alpha - \rho\,v^2 \quad \text{if } v > 0$$
$$v'(t) = \alpha + \rho\,v^2 \quad \text{if } v < 0.$$

Exercise 2) Consider the case in which $\alpha = -g$, so we are considering vertical motion, with up being the positive direction.

$$v'(t) = -g - \rho\,v^2 \quad \text{if } v > 0$$
$$v'(t) = -g + \rho\,v^2 \quad \text{if } v < 0.$$

Draw the phase diagrams. Note that each diagram contains a half line of v-values. Make conclusions about velocity behavior in case $v_0 > 0$ and $v_0 \leq 0$. Is there a terminal velocity?

How would you set up and get started on finding the solutions to these two differential equations? A couple of your homework problems are related to this quadratic drag model.

Exercise 3a Returning to the linear drag model and with. Solve the IVP

$$v'(t) = \alpha - \rho\, v$$
$$v(0) = v_0$$

and verify that your solutions are consistent with the phase diagram analysis two pages back. (This is, once again, our friend the first order constant coefficient linear differential equation.)

3b integrate the velocity function above to find a formula for the position function $y(t)$. Write $y(0) = y_0$.

Comparison of Calc 1 constant acceleration vs. linear drag acceleration model:

We consider the bow and deadbolt example from the text, page 102-104. It's shot vertically into the air (watch out below!), with an initial velocity of 49 $\frac{m}{s}$. (That initial velocity is chosen because its numerical value is 5 times the numerical value of $g = 9.8 \ \frac{m}{s^2}$, which simplifies some of the computations.) In the no-drag case, this could just be the vertical component of a deadbolt shot at an angle. With drag, one would need to study a more complicated system of coupled differential equations for the horizontal and vertical motions, if you didn't shoot the bolt straight up. So we're shooting it straight up.

No drag:

$$v'(t) = -g \approx -9.8 \ \frac{m}{s^2}$$

$$v(t) = -g\,t + v_0 = -g\,t + 5\,g = g\cdot(-t+5) \qquad \frac{m}{s}$$

$$x(t) = -\frac{1}{2}g\,t^2 + v_0\,t + x_0 = -\frac{1}{2}g\,t^2 + 5\,g\,t = g\,t\left(-\frac{1}{2}t + 5\right) \qquad m$$

So our deadbolt goes up for 5 seconds, then drops for 5 seconds until it hits the ground. Its maximum height is given by

$$x(5) = \frac{g\cdot 5\cdot 5}{2} = 122.5\ m$$

Linear drag: The same deadbolt, with the same initial velocity with numerical value $5\,g = 49 \ \frac{m}{s}$. We're told that our deadbolt has a measured terminal velocity of $v_\tau = -245 \ \frac{m}{s}$ which is the numerical value of $-25\,g$. The initial value problem for velocity is

$$v'(t) = -g - \rho\,v$$
$$v(0) = v_0 = 25\,g = 245.$$

So, in these letters the terminal velocity is (easily recoverable by setting $v'(t) = 0$) and is given by

$$v_\tau = -\frac{g}{\rho} = -25\,g \Rightarrow \rho = .04\ .$$

So, from our earlier work: Substituting $\alpha = -g$ into the formulas for terminal velocity, velocity, and height:

$$v(t) = v_\tau + \left(v_0 - v_\tau\right)e^{-\rho\,t} = -245 + 294\,e^{-.04\,t}\ .$$
$$y(t) = -245\,t + \frac{294}{.04}\left(1 - e^{-.04\,t}\right)\ .$$

The maximum height occurs when $v(t) = 0$,

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

which yields $t = 4.56$ sec:

```
> -ln(245./294.)/.04;
```

$$4.558038920 \quad \textbf{(1)}$$

And the maximum height is 108.3 m:

```
> y := t -> -245.*t + 294/.04*(1-e^(-.04*t));
  y(4.558038920);
```

$$y := t \to -245.\cdot t + \frac{294}{.04}\left(1 - e^{-.04\cdot t}\right)$$

$$y := t \to (-1)\cdot 245.\, t + \frac{294\left(1 - e^{(-1)\cdot 0.04\, t}\right)}{0.04}$$

$$108.280465 \quad \textbf{(2)}$$

So the drag caused the deadbolt to stop going up sooner ($t = 4.56$ vs. $t = 5$ sec) and to not get as high (108.3 *vs* 122.5 *m*). This makes sense. It's also interesting what happens on the way down - the drag makes the descent longer than the ascent: 4.85 seconds on the descent, vs. 4.56 on the ascent.

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

$$9.410949931,\ 0. \quad \textbf{(3)}$$

```
> 9.411 - 4.558;
```

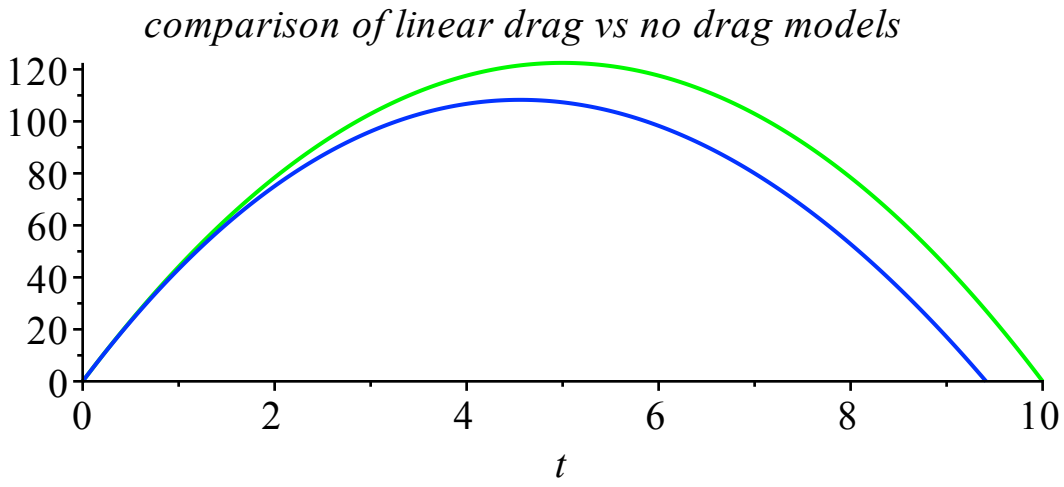
$$4.853 \quad \textbf{(4)}$$

IMPORTANT to note that we needed to use a "solve" command (or something sophisticated like Newton's method) to find when the deadbolt landed. You cannot isolate the t algebraically when trying to solve $y(t) = 0$ for t. This situation will also happen in some of the lab and/or homework problems this week.

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

picture:

```
> z := t→49 t − 4.9·t^2 :
  with(plots) :
  plot1 := plot(z(t), t = 0 ..10, color = green) :
  plot2 := plot(y(t), t = 0 ..9.4110, color = blue) :
  display({plot1, plot2}, title = `comparison of linear drag vs no drag models`);
```




```
>
```

Tues Jan 29

2.4-2.5 Euler's method and improved Euler for numerically solving first order differential equations

Announcements:

Warm-up Exercise:

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 Matlab scripts for parts of these notes, and your next homework assignment will include a technology portion for which you will use Matlab or other software (e.g. Python, Maple), if you're more comfortable. Today and tomorrow we'll focus on Euler's method, section 2.4, and improved Euler, section 2.5. Tomorrow we'll add Runge Kutta (section 2.6) to the discussion.

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

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

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

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

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" fixed step size, $\Delta x := h$. 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

$$x_1 = x_0 + h$$

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

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

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

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

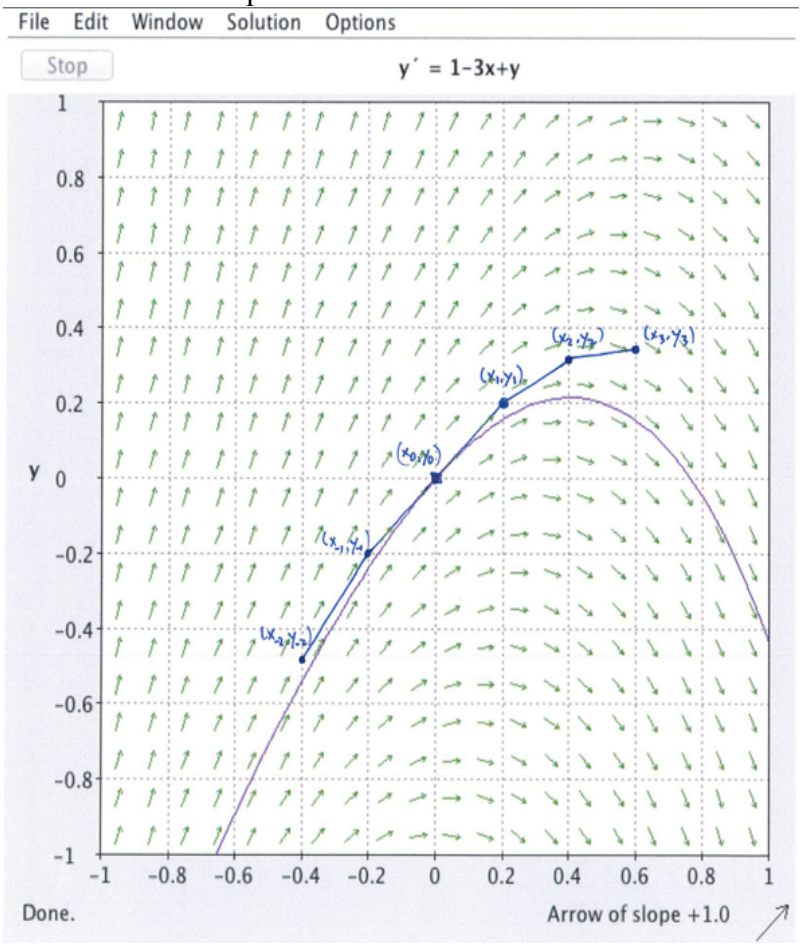
...etc.

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

$$y'(x) = 1 - 3x + y$$
$$y(0) = 0$$

with step size $h = 0.2$.

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



We can automate this process....we'll illustrate using Matlab. For simplicity and because that's usually what we care about in applications we'll just increment in the positive x-direction.

We'll go through the following Matlab script:

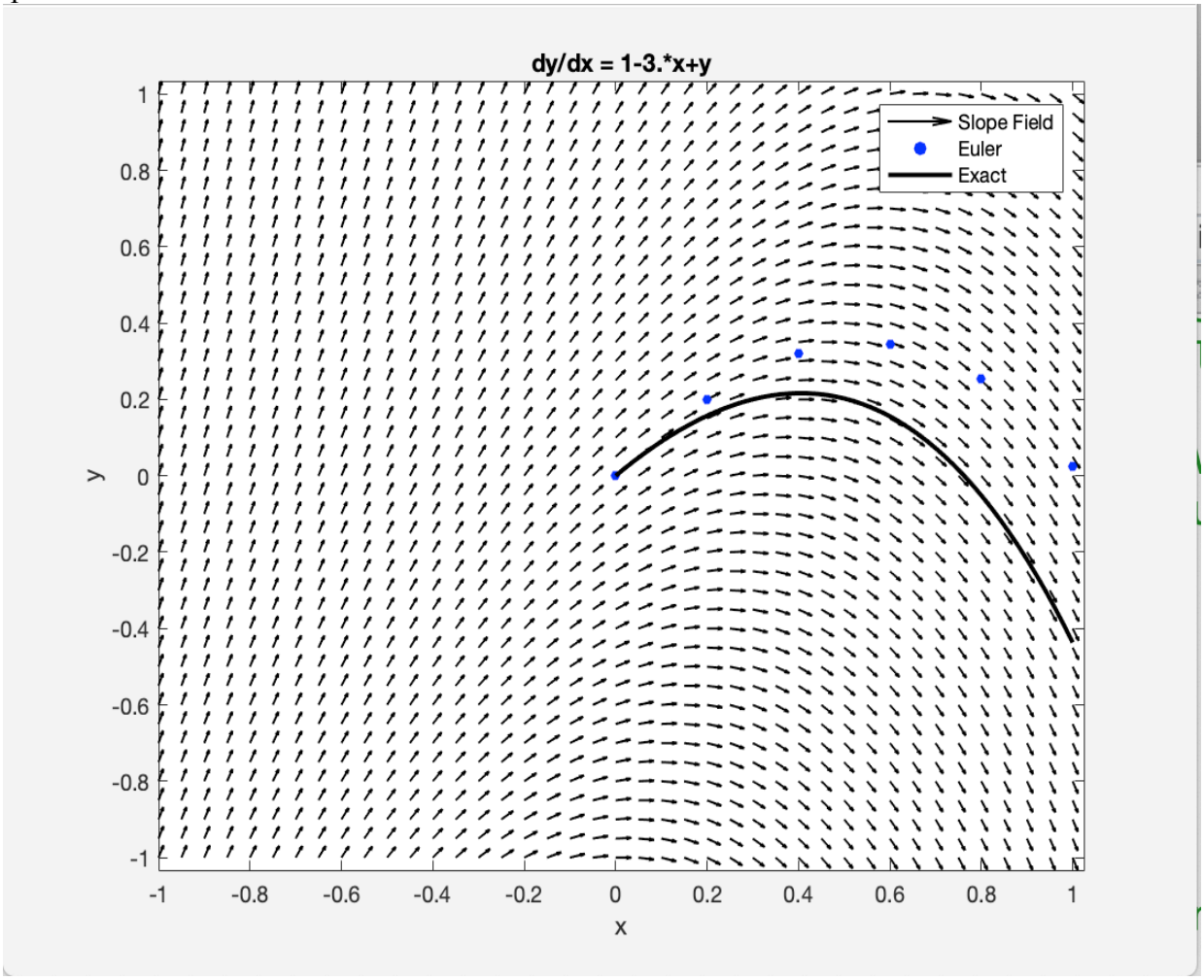
```
clc, clear, close all
   % Clear the command window screen, all the variables, close windows
   % Input your slope function f(x,y) (as a string),
   % i.e. for the DE y'(x)=f(x,y).  In this linear DE example,
   % f(x,y) = 1-3x+y but you can enter whatever function you want.
   % So we would enter:  f = '1-3.*x+y';
f='1-3.*x+y';
   % use "3." instead of "3" (which would cause an error) for
   % scalar multiplication.

xmin = -1;
xmax = 1;
ymin = -1;
ymax = 1;
SlopeField(xmin,xmax,ymin,ymax,f)
   % Draw the slope field in the
   % region xmin<=x<=xmax, ymin<=y<=ymax for dy/dx = f(x,y)

hold on
   % the "hold on" command lets us add more plots to the display
   % of the first plot, e.g. the slope field in this script, rather
   % than creating different displays for each plot.

% Use Euler method to solve the IVP on
% the interval [x0,xmax] with n subintervals :
% dy/dx = f(x,y)
% y(x0) = y0
x0 = 0;
y0 = 0;
b=1;
n = 5;

 % Use Euler Method to Solve, then add scatter plot:
[xE yE]=Eulers(x0,y0,b,n,f);
scatter(xE,yE,20,'blue','filled');

sol='2+3.*x-2.*exp(x)';
 % exact solution in our example
sol = str2func(['@(x)',sol]);

Z=0:.02:1;  % x-coords from 0 to 1, partition width .02.
W=sol(Z);   % y-coords for exact solution
plot(Z,W,'color','black','LineWidth',2);

xlabel('x');ylabel('y')
legend('Slope Field','Euler','Exact')
title(['dy/dx = ' f])
```

Which called the following two functions:

```
    function [x,y] = Eulers(x0,y0,b,n,f)
% Eulers method with n intervals to solve the initial value problem dy/dx=f(x,y) with
% initial conditions y(x0)=y0 on the interval [x0,b]. The function
% f(x,y) must be entered as a string.

% For example, if you wish to solve dy/dx=y*cos(x) with y(0)=1 on the
% interval [0,15] with 50 intervals you would enter
% Eulers(0,1,15,50,'y.*cos(x)')

f = str2func(['@(x,y) ',f]);
   %converts the string input into a function handle

h=(b-x0)/n;
x=zeros(n+1,1); y= zeros(n+1,1); %Pre-allocating vectors for speed
x(1)=x0; y(1)=y0;

for i=1:n
    x(i+1)=x0+i*h;
    y(i+1)=y(i)+h*f(x(i),y(i));
end
end
```

```
function []= SlopeField(xmin,xmax,ymin,ymax,f)
% the function f(x,y) must be entered as a string. For example if
%f(x,y)=y*cos(x), then you need to enter 'y.*cos(x)' for f

f = str2func(['@(x,y) ',f]);
   % converts the string input into a function handle that
   % Matlab recognizes

stepsize=abs(xmin-xmax)/40;
[X,Y] = meshgrid(xmin:stepsize:xmax, ymin:stepsize:ymax);
dY=f(X,Y);
dX=ones(size(dY));
L=sqrt(1+dY.^2);
     % keeps arrows the same lengths
quiver(X,Y,dX./L,dY./L,'Color','k','LineWidth',1,'AutoScaleFactor',0.5);
axis tight
end
```

And produced this:



Variables - xE

xE

6x1 double

	1
1	0
2	0.2000
3	0.4000
4	0.6000
5	0.8000
6	1

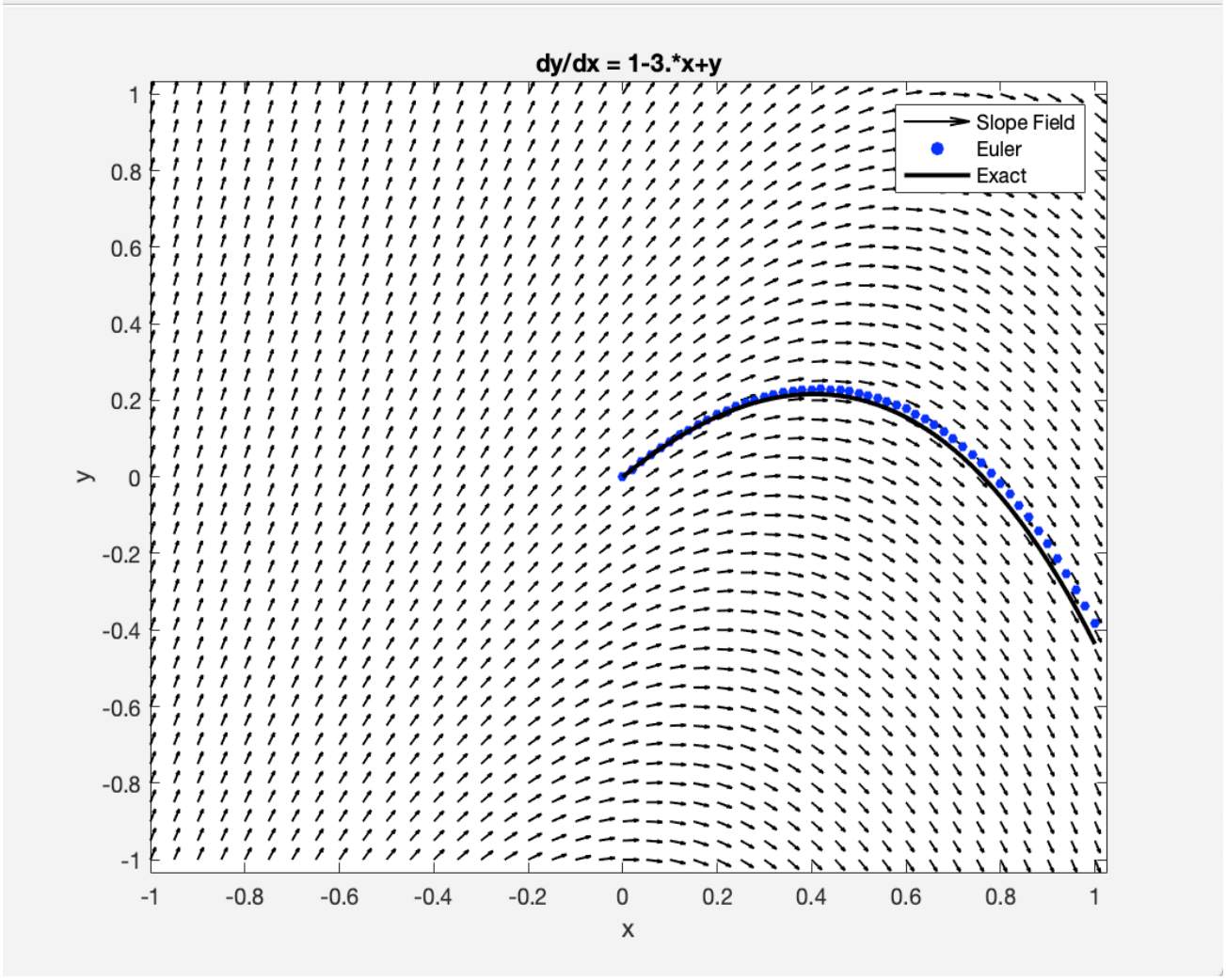
Variables - yE

yE

6x1 double

	1
1	0
2	0.2000
3	0.3200
4	0.3440
5	0.2528
6	0.0234
7	

just changed "n" to 50:



Actually, that's not so great, considering how much computation got done....

In more complicated differential equations it is a very serious issue to find relatively efficient ways of approximating solutions. An entire field of mathematics, ``numerical analysis" deals with such issues for a variety of mathematical problems. Our text explores improvements to Euler in sections 2.5 and 2.6, in particular it discusses improved Euler, and Runge Kutta. Runge Kutta-type codes are actually used in commerical numerical packages, e.g. in Wofram, Matlab, Maple etc.

Let's talk about improved Euler, which is an extension of the Trapezoid rule for integration:

Suppose we already knew the solution $y(x)$ to the initial value problem

$$y'(x) = f(x, y)$$
$$y(x_0) = y_0.$$

If we integrate the DE from x to $x + h$ and apply the Fundamental Theorem of Calculus, we get

$$y(x + h) - y(x) = \int_x^{x+h} f(t, y(t))\, dt \text{, i.e.}$$
$$y(x + h) = y(x) + \int_x^{x+h} f(t, y(t))\, dt .$$

One problem with Euler is that we approximate this integral above by $h \cdot f(x, y(x))$, i.e. we use the value at the left-hand endpoint as our approximation of the integrand, on the entire interval from x to $x + h$. This causes errors that are larger than they need to be, and these errors accumulate as we move from subinterval to subinterval and as our approximate solution diverges from the actual solution. The improvements to Euler depend on better approximations to the integral. These are subtle, because we don't yet have an approximation for $y(t)$ when t is greater than x, so also not for the integrand $f(t, y(t))$ on the interval $[x, x + h]$.

Improved Euler uses an approximation to the Trapezoid Rule to compute the integral in the formula

$$y(x+h) = y(x) + \int_x^{x+h} f(t, y(t))\ dt\,.$$

Recall, the trapezoid rule to approximate the integral

$$\int_x^{x+h} f(t, y(t))\ dt$$

would be

$$\frac{1}{2} h \cdot (f(x, y(x)) + f(x+h, y(x+h)))\,.$$

Since we don't know $y(x+h)$ we approximate its value with unimproved Euler, and substitute into the formula above. This leads to the improved Euler "pseudocode" for how to increment approximate solutions.

Improved Euler pseudocode:

$k_1 = f(x_j, y_j)$ #current slope

$k_2 = f(x_j + h, y_j + h\,k_1)$ #use unimproved Euler guess for $y(x_j + h)$ to estimate slope function when $x = x_j + h$

$k = \frac{1}{2}(k_1 + k_2)$ #average of two slopes

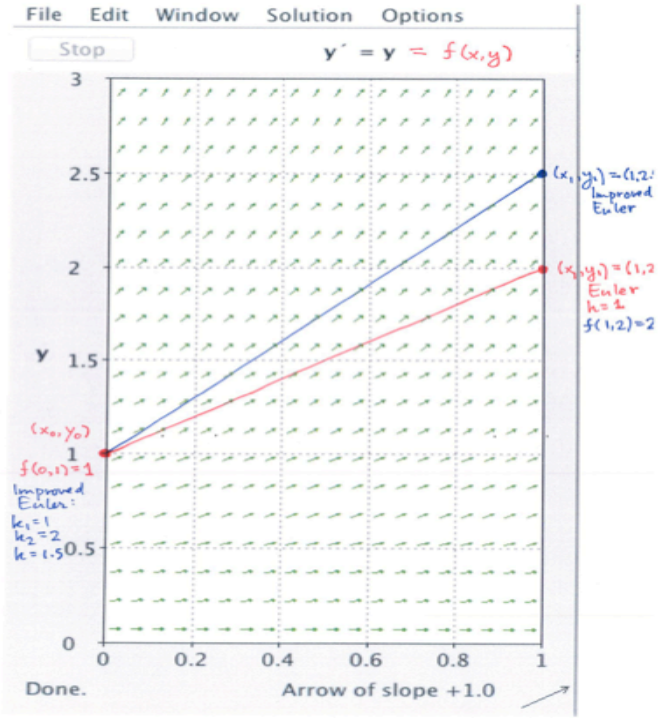
$x_{j+1} = x_j + h$ #increment x

$y_{j+1} = y_j + h\,k$ #increment y

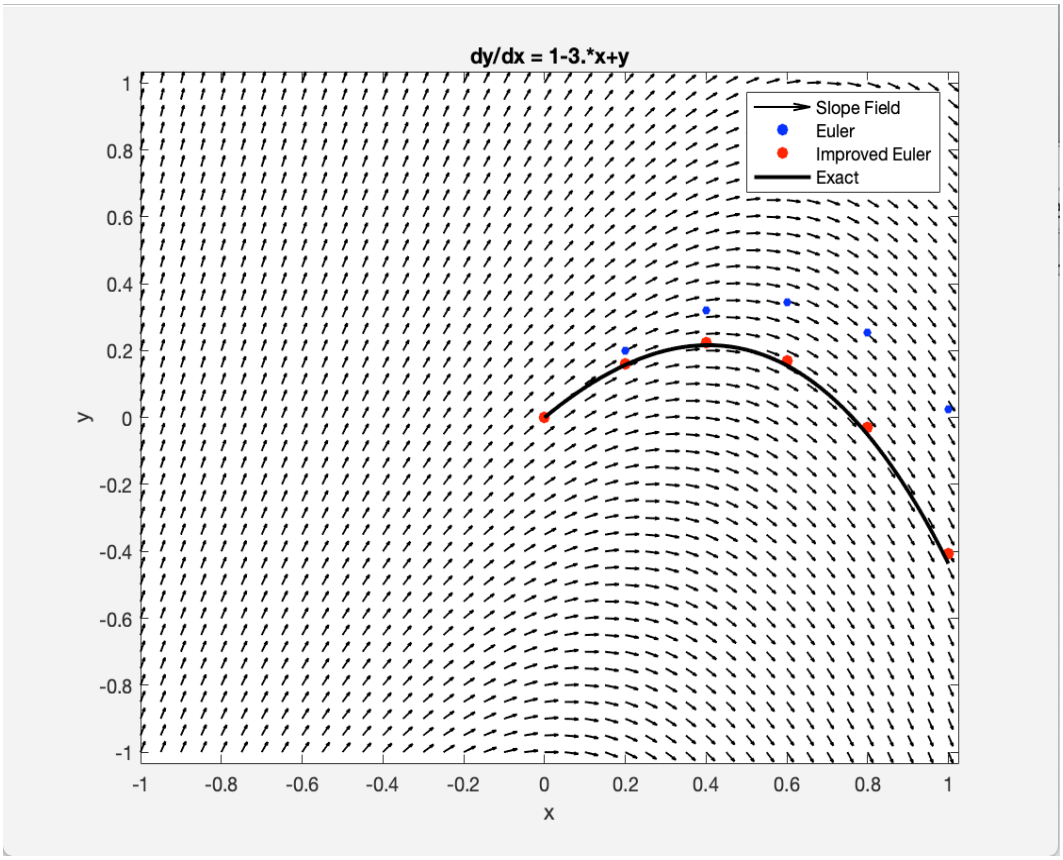
Exercise 2 Contrast Euler and Improved Euler for the following IVP which has $y(x) = e^x$ as its solution:

$$y'(x) = y$$
$$y(0) = 1$$

to estimate $y(1) \approx e$ with one step of size $h = 1$. Your computations should mirror the picture below.



If we call the "improved Euler" function in the previous Matlab script, the result is much improved, even with $n = 5$:




```
function [x,y] = Improved_Eulers(x0,y0,b,n,f)
% Improved Eulers  method with n intervals to solve the initial value pr
% initial conditions y(x0)=y0 on the interval [x0,b]. The function
% f(x,y) must be entered as a string.

% For example, if you wish to solve dy/dx=y*cos(x) with y(0)=1 on the
% interval [0,15] with 50 intervals you would enter
% Improved_Eulers(0,1,15,50,'y.*cos(x)')

f = str2func(['@(x,y) ',f]); % converts the string input into a function

h=(b-x0)/n;
x=zeros(n+1,1); y= zeros(n+1,1); % Pre-allocoting vectors for speed
x(1)=x0; y(1)=y0;
for i=1:n
    x(i+1)=x0+i*h;
    k1=f(x(i),y(i));
    u=y(i)+h*k1;
    k2=f(x(i+1),u);
    y(i+1)=y(i)+h*(k1+k2)/2;
end
end
```

Wed Jan 30

2.5-2.6 Improved Euler and Runge Kutta.

Announcements:

Warm-up Exercise:

Runge Kutta

In the same vein as ``improved Euler" we can use the Simpson approximation for the integral for Δy instead of the Trapezoid rule, and this leads to the Runge-Kutta method. You may or may not have talked about Simpson's Parabolic Rule for approximating definite integrals in Calculus. It is based on a parabolic approximation to the integrand , whereas the Trapezoid rule is based on an approximationg of the graph with trapezoids, on small subintervals.

Simpson's Rule:

Consider two numbers $x_0 < x_1$, with interval width $h = x_1 - x_0$ and interval midpoint $\underline{x} = \frac{1}{2}(x_0 + x_1)$.

If you fit a parabola $p(x)$ to the three points

$$\left(x_0, g\left(x_0\right)\right),\ \ (\underline{x}, g(\underline{x})),\ \ \left(x_1, g\left(x_1\right)\right)$$

then

$$\int_{x_0}^{x_1} p(t)\mathrm{d}t = \frac{h}{6}\left(g\left(x_0\right) + 4\cdot g(\underline{x}) + g\left(x_1\right)\right).$$

(You will check this fact in your homework this week!) This formula is the basis for Simpson's rule, which you can also review in your Calculus text or at Wikipedia. (Wikipedia also has a good entry on Runge-Kutta.) Applying the Simpson's rule approximation for our DE, and if we already knew the solution function $y(x)$, we would have

$$y(x+h) = y(x) + \int_x^{x+h} f(t, y(t))\,dt$$

$$\approx y(x) + \frac{h}{6}\cdot\left(f(x, y(x)) + 4f\left(x + \frac{h}{2}, y\left(x + \frac{h}{2}\right)\right) + f(x+h, y(x+h))\right) .$$

However, we have the same issue as in Trapezoid - that we've only approximated up to $y(x)$ so far. Here's the magic pseudo-code for how Runge-Kutta takes care of this. (See also section 2.6 to the text.)

Runge-Kutta pseudocode:

$k_1 = f\left(x_j, y_j\right)$ # left endpoint slope

$k_2 = f\left(x_j + \frac{h}{2}, y_j + \frac{h}{2}\,k_1\right)$ # first midpoint slope estimate, using k_1 to increment y_j

$k_3 = f\left(x_j + \frac{h}{2}, y_j + \frac{h}{2}\,k_2\right)$ # second midpoint slope estimate using k_2 to increment y_j

$k_4 = f\left(x_j + h, y_j + h\,k_3\right)$ # right hand slope estimate, using k_3 to increment y_j

$k = \frac{1}{6}\left(k_1 + 2\,k_2 + 2\,k_3 + k_4\right)$ #weighted average of all four slope estimates, consistent with Simpson's rule

$x_{j+1} = x_j + h$ # increment x

$y_{j+1} = y_j + h\,k$ # increment y

Exercise 1 Contrast Euler and Improved Euler to Runge-Kutta for our IVP

$$y'(x) = y$$
$$y(0) = 1$$

to estimate $y(1) \approx e$ with one step of size $h = 1$. Your computations should mirror the picture below. Note how amazingly accurate your prediction is, considering you only did a single step. It's actually more accurate than regular Euler with 100 steps of size 0.01.

Runge-Kutta pseudocode:

$k_1 = f(x_j, y_j)$ # left endpoint slope

$k_2 = f\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} k_1\right)$ # first midpoint slope estimate, using k_1 to increment y_j

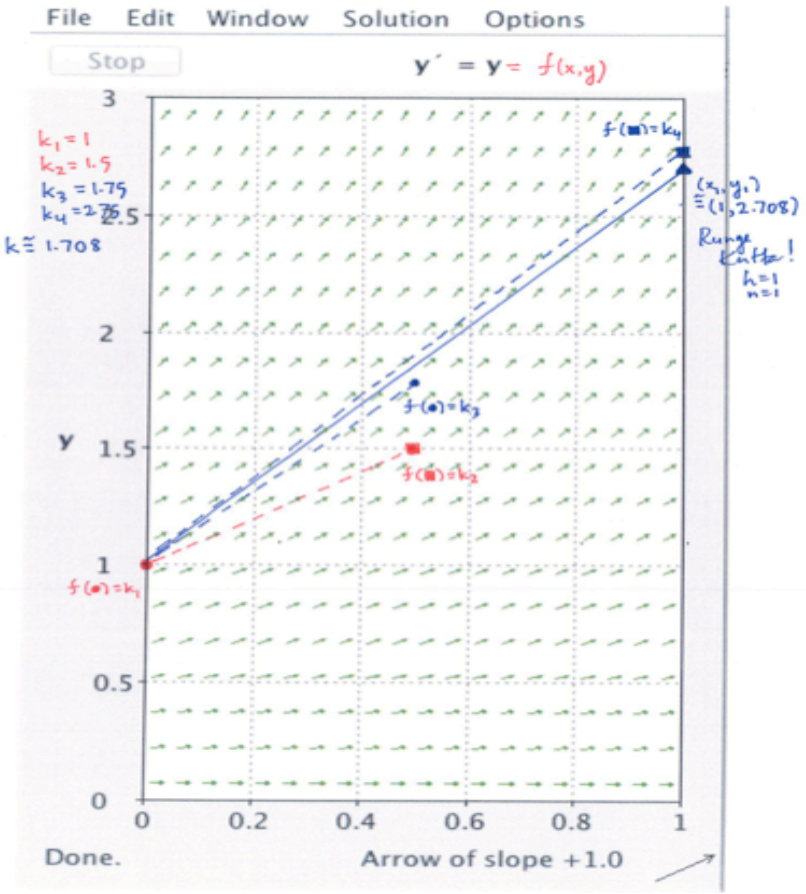
$k_3 = f\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} k_2\right)$ # second midpoint slope estimate using k_2 to increment y_j

$k_4 = f(x_j + h, y_j + h k_3)$ # right hand slope estimate, using k_3 to increment y_j

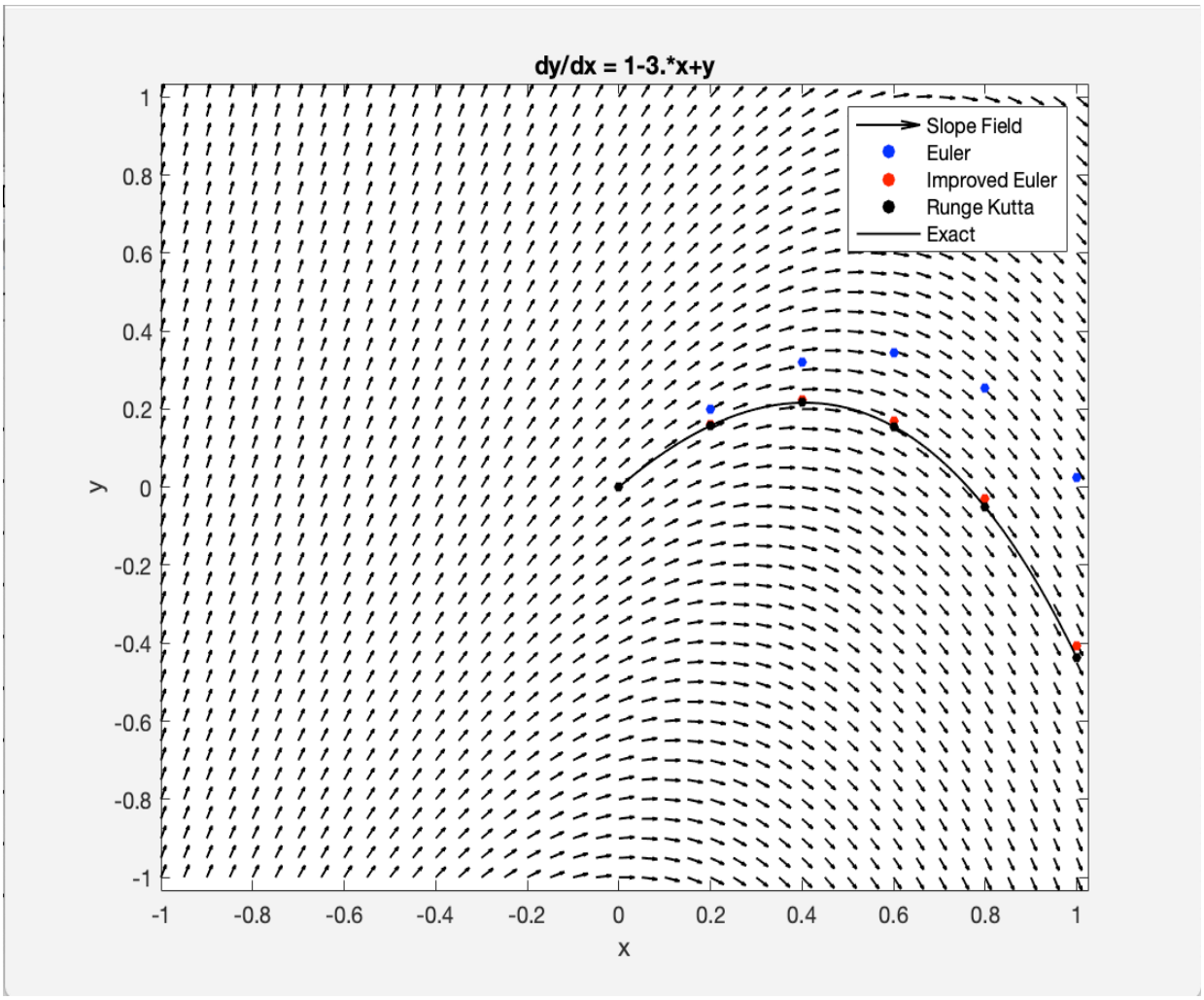
$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ #weighted average of all four slope estimates, consistent with Simpson's rule

$x_{j+1} = x_j + h$ # increment x

$y_{j+1} = y_j + h k$ # increment y



Using the Runge-Kutta function on our running example:



```
function [x,y] = Runge_Kutta(x0,y0,b,n,f)
% Runge Kutta method with n subintervals to solve the initial value
% problem dy/dx=f(x,y) with
% initial conditions y(x0)=y0 on the interval [x0,b]. The function
% f(x,y) must be entered as a string.

% For example, if you wish to solve dy/dx=y*cos(x) with y(0)=1 on the
% interval [0,15] with 50 intervals you would enter
% Runge_Kutta(0,1,15,50,'y.*cos(x)')

f = str2func(['@(x,y) ',f]); % converts the string input into a function handle

h=(b-x0)/n;
x=zeros(n+1,1); y= zeros(n+1,1); % Pre-allocoting vectors for speed
x(1)=x0; y(1)=y0;
for i=1:n
    x(i+1)=x0+i*h;
    k1=f(x(i),y(i));
    k2=f(x(i)+h/2,y(i)+h/2*k1);
    k3=f(x(i)+h/2,y(i)+h/2*k2);
    k4=f(x(i+1),y(i)+h*k3);
    y(i+1)=y(i)+h*(k1+2*k2+2*k3+k4)/6;
end
end
```

Numerical experiments (In Maple, which has an easier time than Matlab in adjusting significant digits.)

Estimate e *by estimating* $y(1)$ for the solution to the IVP

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

Apply Runge-Kutta with n = 10, 20, 40 ... subintervals, successively doubling the number of subintervals until you obtain the target number below - rounded to 9 decimal digits - twice in succession.

```
> evalf(e);
```

2.718281828459045 **(5)**

It worked with $n = 40$:

Initialize

```
> restart : # clear any memory from earlier work
> Digits := 15 : # we need lots of digits for "famous numbers"
>
> unassign( `x`, `y` ) : # in case you used the letters elsewhere, and came back to this piece of code
> f := (x, y) → y : # slope field function for our DE i.e. for y'(x)=f(x,y)
> x[0] := 0 : y[0] := 1 : #initial point
  h := 0.025 : n := 40 : #step size and number of steps - first attempt for "famous number e"
```

Runge Kutta loop. WARNING: ONLY RUN THIS CODE AFTER INITIALIZING

```
> 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] ); #print iteration step, current x,y, values, and exact solution value
    end if:
    k1 := f(x[i], y[i]); #current slope function value
    k2 := f( x[i] + h/2, y[i] + h/2 · k1 );
      # first estimate for slope at right endpoint of midpoint of subinterval
    k3 := f( x[i] + h/2, y[i] + h/2 · k2 ); #second estimate for midpoint slope
    k4 := f(x[i] + h, y[i] + h·k3); #estimate for right-hand slope
    k := (k1 + 2· k2 + 2·k3 + k4)/6; # Runge Kutta estimate for rate of change of y
    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

10, 0.250, 1.28402541566434

20, 0.500, 1.64872126807197

30, 0.750, 2.11700001155073

40, 1.000, 2.71828181979283 **(6)**

Math 2280-002
Friday Feb 1

3.1 Second order linear differential equations and vector space theory connections to Math 2270

Announcements:

Warm-up Exercise:

3.1 Second order linear differential equations, and vector space theory connections to Math 2270:

Definition: A vector space is a collection of objects together with an "addition" operation "+", and a "scalar multiplication" operation, so that the rules below all hold.

(α) Whenever $f, g \in V$ then $f+g \in V$. (closure with respect to addition)
(β) Whenever $f \in V$ and $c \in \mathbb{R}$, then $c \cdot f \in V$. (closure with respect to scalar multiplication)

As well as:
(a) $f+g=g+f$ (commutative property)
(b) $f+(g+h)=(f+g)+h$ (associative property)
(c) $\exists\ 0 \in V$ so that $f+0=f$ is always true.
(d) $\forall f \in V\ \exists\ -f \in V$ so that $f+(-f)=0$ (additive inverses)
(e) $c \cdot (f+g)=c \cdot f+c \cdot g$ (scalar multiplication distributes over vector addition)
(f) $(c_1+c_2) \cdot f=c_1 \cdot f+c_2 \cdot f$ (scalar addition distributes over scalar multiplication)
(g) $c_1 \cdot (c_2 \cdot f)=(c_1 c_2) \cdot f$ (associative property)
(h) $1 \cdot f=f,\ (-1) \cdot f=-f,\ 0 \cdot f=0$ (these last two actually follow from the others).

Examples you've seen in Math 2270:
(1) $\mathbb{R}^m$, with the usual vector addition and scalar multiplication, defined component-wise
(2) subspaces W of $\mathbb{R}^m$, which satisfy (α),(β), and therefore automatically satisfy (a)-(h), because the vectors in W also lie in $\mathbb{R}^m$.

Maybe you've also seen ...
Exercise 1) In Chapter 3 we focus on the vector space

$$V=C(\mathbb{R}) := \{f: \mathbb{R} \to \mathbb{R}\ s.t.\ f\ is\ a\ continuous\ function\}$$

and its subspaces. Verify that the vector space axioms for linear combinations are satisfied for this space of functions. Recall that the function $f+g$ is defined by $(f+g)(x) := f(x)+g(x)$ and the scalar multiple $cf(x)$ is defined by $(cf)(x) := cf(x)$. What is the zero vector for functions?

Recall that the vector space axioms are exactly the arithmetic rules we use to work with linear combination equations. In particular the following concepts are defined in any vector space V.

- the span of a finite collection of functions $f_1, f_2, \ldots f_n$.
- linear independence/dependence for a collection of functions $f_1, f_2, \ldots f_n$.
- subspaces of V
- bases and dimension for finite dimensional subspaces. (The function space V itself is infinite dimensional, meaning that no finite collection of functions spans it.)

Definition: A second order linear differential equation for a function $y(x)$ is a differential equation that can be written in the form

$$A(x)y'' + B(x)y' + C(x)y = F(x) .$$

We search for solution functions $y(x)$ defined on some specified interval I of the form $a < x < b$, or (a, ∞), $(-\infty, a)$ or (usually) the entire real line $(-\infty, \infty)$. In this chapter we assume the function $A(x) \neq 0$ on I, and divide by it in order to rewrite the differential equation in the standard form

$$y'' + p(x)y' + q(x)y = f(x) .$$

Definition: The DE above is called homogeneous if the right hand side $f(x)$ is the zero function, $f(x) \equiv 0$. If f is not the zero function, the DE is called nonhomogeneous (or inhomogeneous).

One reason the DE above is called linear is that the "operator" L defined by

$$L(y) := y'' + p(x)y' + q(x)y$$

satisfies the so-called linearity properties

(1) $L(y_1 + y_2) = L(y_1) + L(y_2)$

(2) $L(c\,y) = c\,L(y), c \in \mathbb{R}$.

(Recall that the matrix multiplication function $L(\mathbf{x}) := A\,\mathbf{x}$ satisfies the analogous properties. Any time we have have a transformation L satisfying (1),(2), we say it is a linear transformation.)

Exercise 2a) Check the linearity properties (1),(2) for the differential operator $L(y) := y'' + p(x)y' + q(x)y$.

2b) Use these linearity properties to show that

Theorem 0 the solution space to the homogeneous second order linear DE

$$y'' + p(x)y' + q(x)y = 0$$

is closed under addition and scalar multiplication, i.e. it is a subspace. Notice that this is the "same" proof one uses to show that the solution space to a homogeneous matrix equation $A\,\mathbf{x} = \mathbf{0}$ is a subspace.

Exercise 3) As an example, find the solution space to the following homogeneous differential equation for $y(x)$

$$y'' + 2y' = 0$$

on the x-interval $-\infty < x < \infty$. Notice that the solution space is the span of two functions. Hint: This is really a first order DE for $v = y'$.

Exercise 4) Use the linearity properties to show

Theorem 1 All solutions to the nonhomogeneous second order linear DE

$$y'' + p(x)y' + q(x)y = f(x)$$

are of the form $y = y_P + y_H$ where y_P is any single particular solution and y_H is some solution to the homogeneous DE. (y_H is called y_c, for complementary solution, in the text). Thus, if you can find a single particular solution to the nonhomogeneous DE, and all solutions to the homogeneous DE, you've actually found all solutions to the nonhomogeneous DE.

Theorem 2 (Existence-Uniqueness Theorem): Let $p(x), q(x), f(x)$ be specified continuous functions on the interval I, and let $x_0 \in I$. Then there is a unique solution $y(x)$ to the initial value problem

$$y'' + p(x)y' + q(x)y = f(x)$$
$$y(x_0) = b_0$$
$$y'(x_0) = b_1$$

and $y(x)$ exists and is twice continuously differentiable on the entire interval I.

Exercise 5) Verify Theorems 1 and 2 for the interval $I = (-\infty, \infty)$ and the IVP

$$y'' + 2y' = 3$$
$$y(0) = b_0$$
$$y'(0) = b_1$$

Unlike in the previous example, and unlike what was true for the first order linear differential equation

$$y' + p(x)y = q(x)$$

there is not a clever integrating factor formula that will always work to find the general solution of the second order linear differential equation

$$y'' + p(x)y' + q(x)y = f(x).$$

Rather, we will usually resort to vector space theory and algorithms based on clever guessing to solve these differential equations. It will help to know

Theorem 3: The solution space to the second order homogeneous linear differential equation

$$y'' + p(x)y' + q(x)y = 0$$

is 2-dimensional.

This Theorem is illustrated in Exercise 2 that we completed earlier. Theorem 3 and the techniques we'll actually be using going forward are illustrated by

Exercise 6) Consider the homogeneous linear DE for $y(x)$

$$y'' - 2y' - 3y = 0$$

6a) Find two exponential functions $y_1(x) = e^{rx}$, $y_2(x) = e^{\rho x}$ that solve this DE. Deduce that arbitrary linear combinations of y_1, y_2 also solve the DE.

6b) Show that every IVP

$$y'' - 2y' - 3y = 0$$
$$y(0) = b_0$$
$$y'(0) = b_1$$

can be solved with a unique linear combination $y(x) = c_1 y_1(x) + c_2 y_2(x)$.

6c) Use your work from part b to explain why the solution space is two-dimensional.

6d) Now consider the nonhomogeneous DE

$$y'' - 2y' - 3y = 9$$

Notice that $y_P(x) = -3$ is a particular solution. Use this information and superposition (linearity) to find the solution to the initial value problem

$$y'' - 2y' - 3y = 9$$
$$y(0) = 6$$
$$y'(0) = -2.$$