Tues Jan 29

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

Announcements:
- HW problem §2.6 #4 is postponed 'til next week
- pick up graded HW.
- comment on nuclear waste problem w3.3

$$mv'(t) = -W + B + F_R$$

Let "up" be positive dir

missing minus sign

buoyancy force

linear drag force of resistance

Warm-up Exercise: Solve the linear DE & IVP:

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

[We need the solution in class]
[So someone needs to solve it. :)]

$$y' - y = 1 - 3x$$

$$e^{-x}[y' - y] = (1-3x)e^{-x}$$

$$\ldots \quad y(x) = 3x + 2 + Ce^{x}$$

IVP $y(x) = 3x + 2 - 2e^{x}$

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 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

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

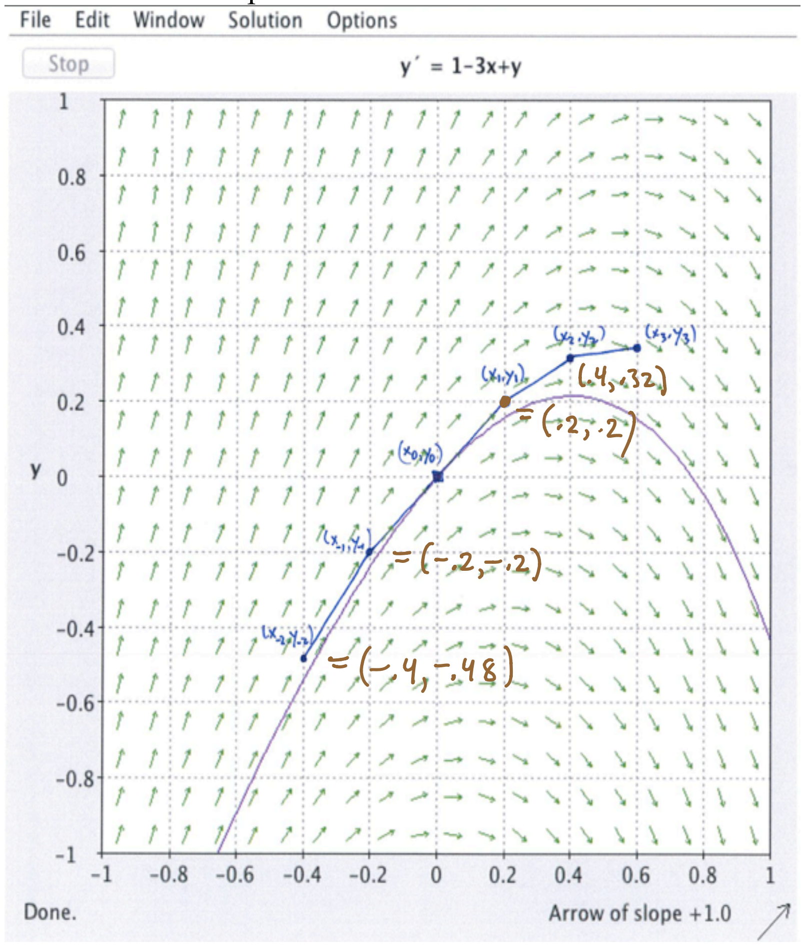
...etc.

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.



$y' = 1 - 3x + y = f(x,y)$

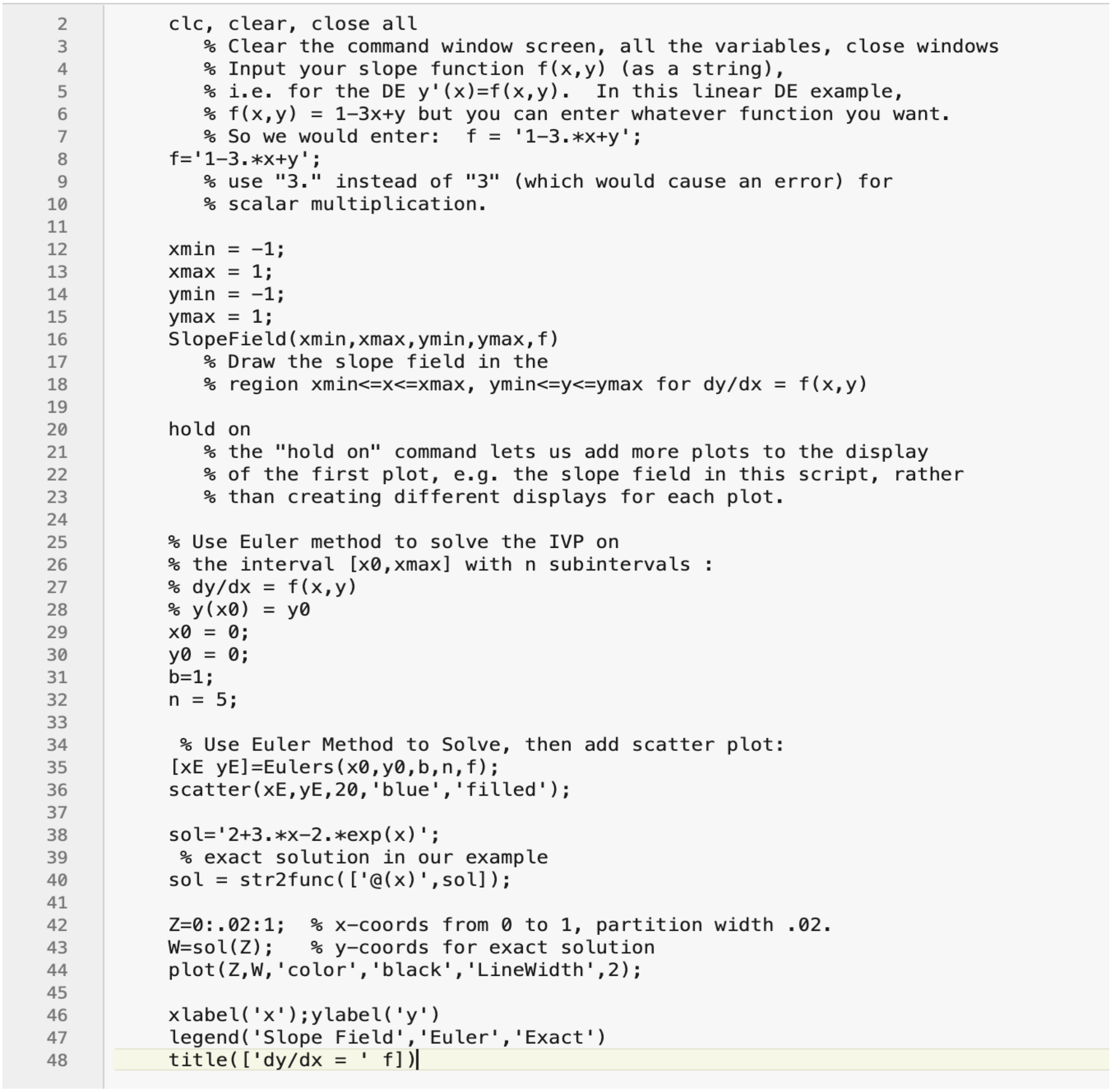
$y(0) = 0$

$\Delta x = h = .2$

i	x_i	y_i	$f(x_i,y_i)$	$\Delta y = f(x_i,y_i)\Delta x$	new x	new $y = y_i + \Delta y$
0	0	0	$1-0+0=1$	$1(.2)$	.2	.2
1	.2	.2	$1-.6+.2=.6$	$.6(.2) = .12$	.4	.32
2	.4	.32				
0	0	0	1	$1(-.2)$	−.2	−.2
−1	−.2	−.2	$1+.6-.2=1.4$	$1.4(-.2) = -.28$	−.4	−.48
−2	−.4	−.48				

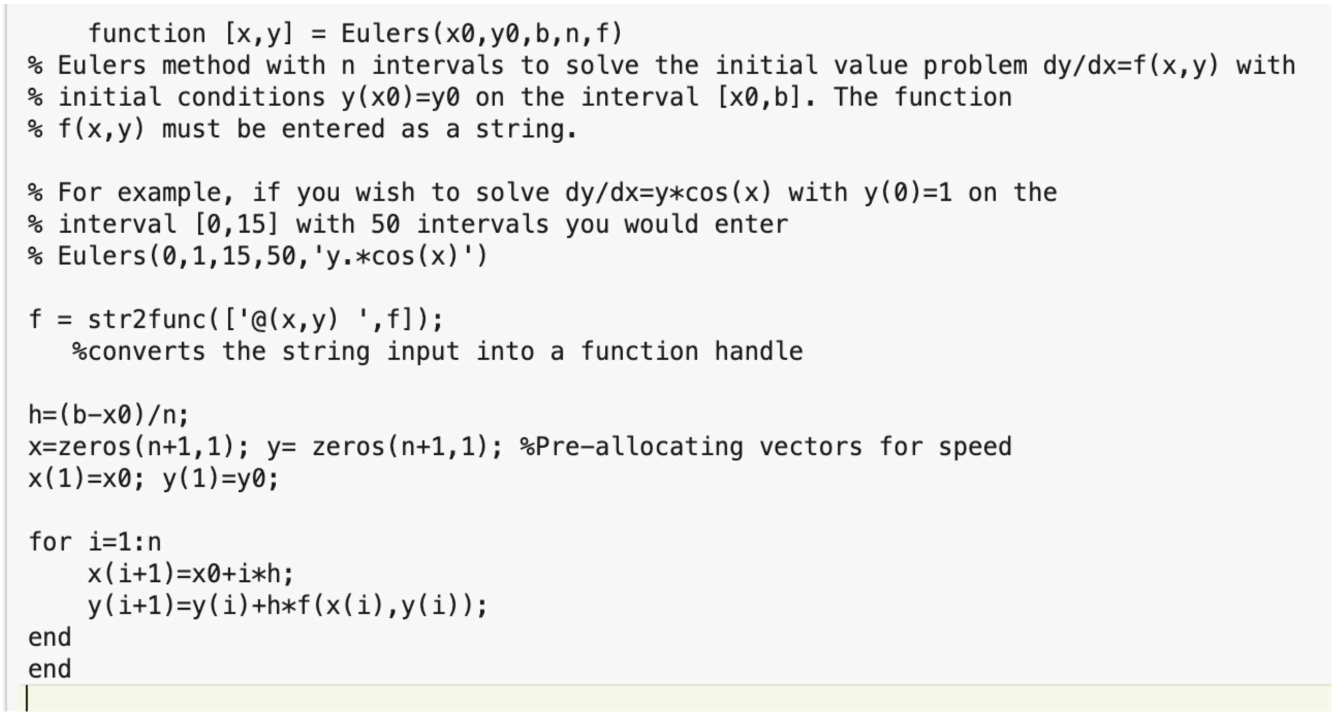
We can automate this process....we'll illustrate using Matlab. For simplicity and because that's usually what we care about in applications we'll just increment in the positive x-direction.

We'll go through the following Matlab script:

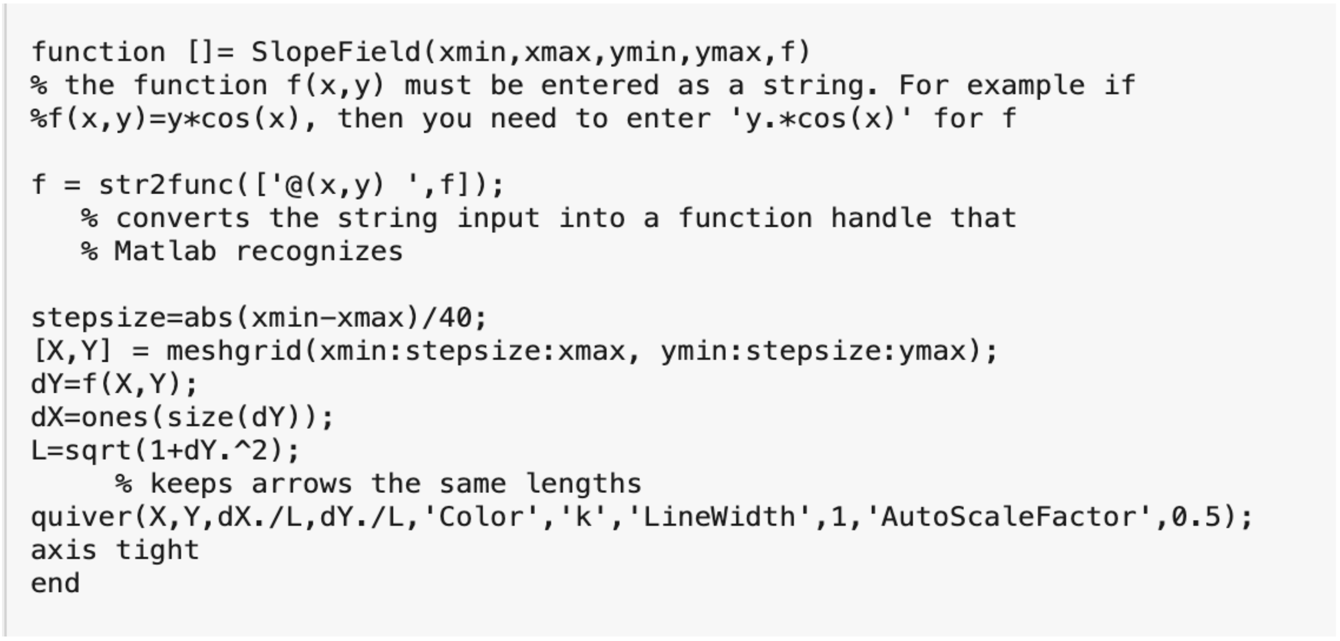


```
clc, clear, close all
   % Clear the command window screen, all the variables, close windows
   % Input your slope function f(x,y) (as a string),
   % i.e. for the DE y'(x)=f(x,y).  In this linear DE example,
   % f(x,y) = 1-3x+y but you can enter whatever function you want.
   % So we would enter:  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: Octave

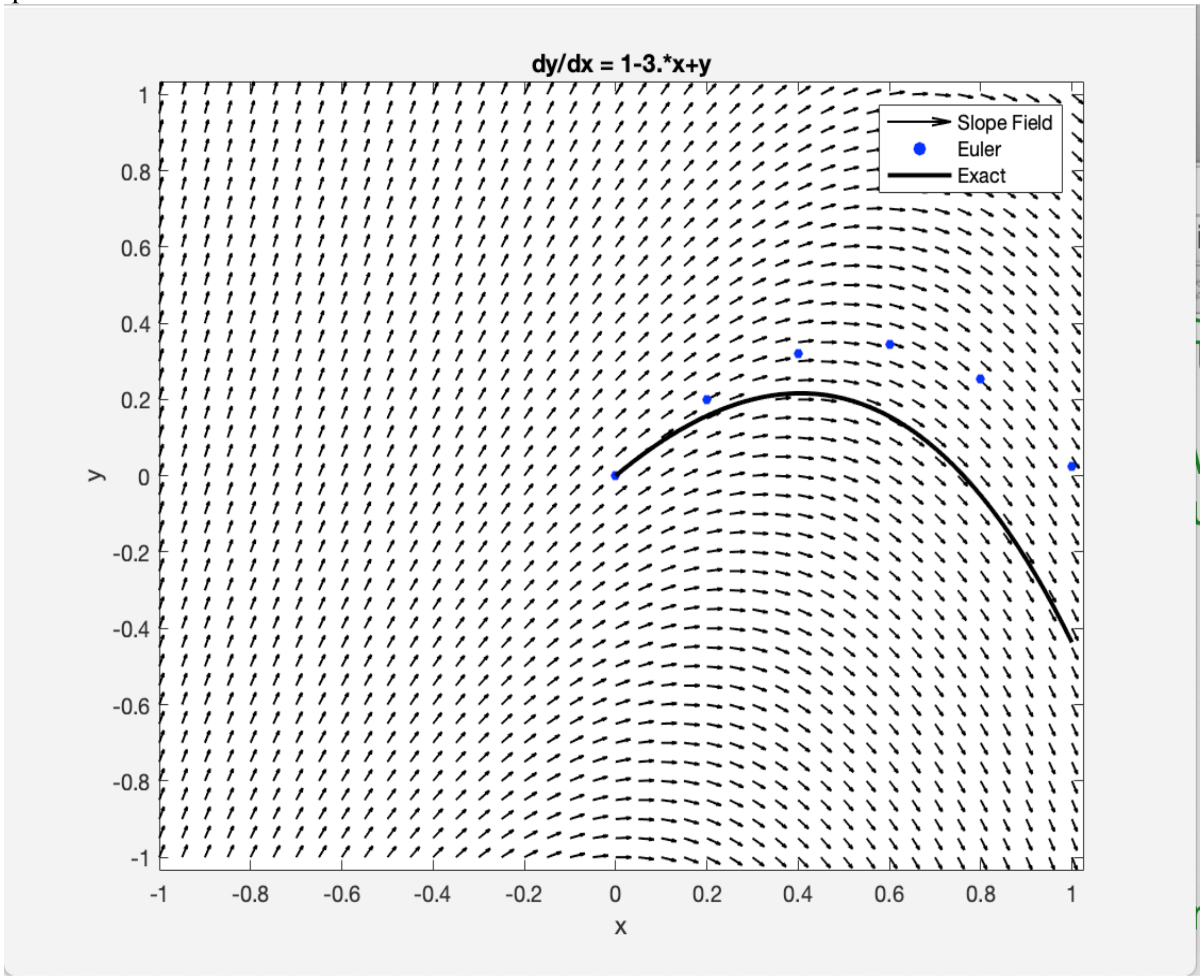


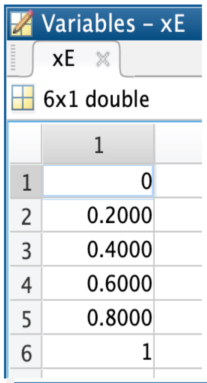
```
    function [x,y] = Eulers(x0,y0,b,n,f)
% Eulers method with n intervals to solve the initial value problem dy/dx=f(x,y) with
% initial conditions y(x0)=y0 on the interval [x0,b]. The function
% f(x,y) must be entered as a string.

% 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:

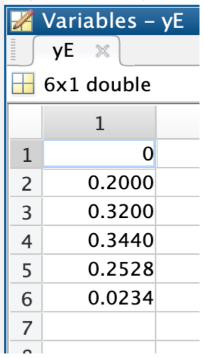



Variables – xE

xE

6x1 double

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

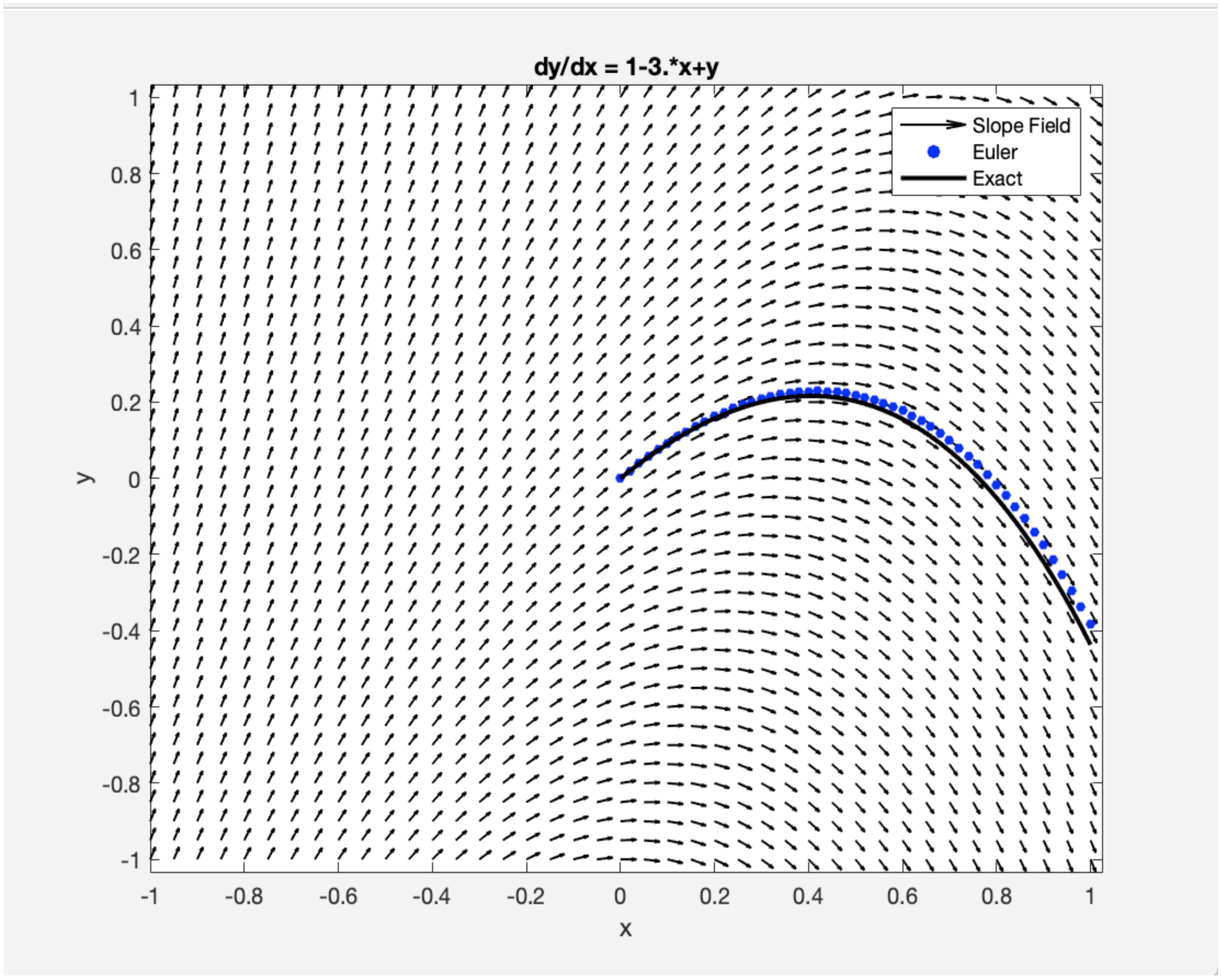

Variables – yE

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:

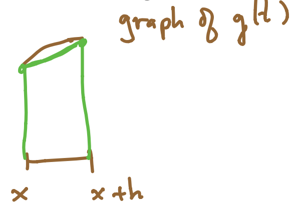
Suppose we already knew the solution $y(x)$ to the initial value problem

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

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

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

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



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

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

Recall, the trapezoid rule to approximate the integral

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

would be

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

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
$k_2 = f(x_j + h, \underbrace{y_j + h k_1}_{\text{Euler for } y_{j+1}})$     #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 $\underline{y(x) = e^x}$ as its solution:

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

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

Euler.

$x_0 = 0$

$y_0 = 1$

$f(0,1) = 1$

$x_1 = 1 \qquad (h = 1)$

$y_1 = y_0 + \underset{\substack{\uparrow \\ f(0,1)\Delta x}}{\Delta y} = 1 + 1 \cdot 1 = 2$ red picture

Improved Euler.

$x_0 = 0$

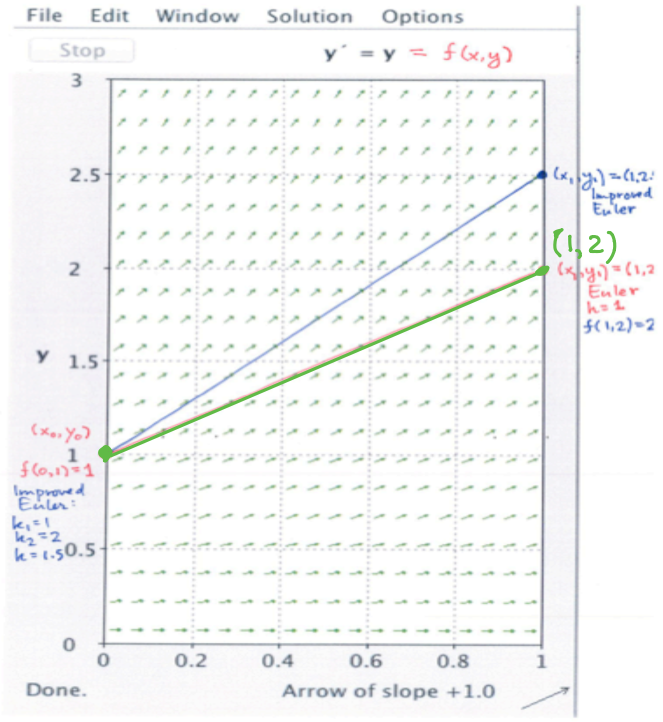
$y_0 = 1$

$k_1 = f(x_0, y_0) = 1$

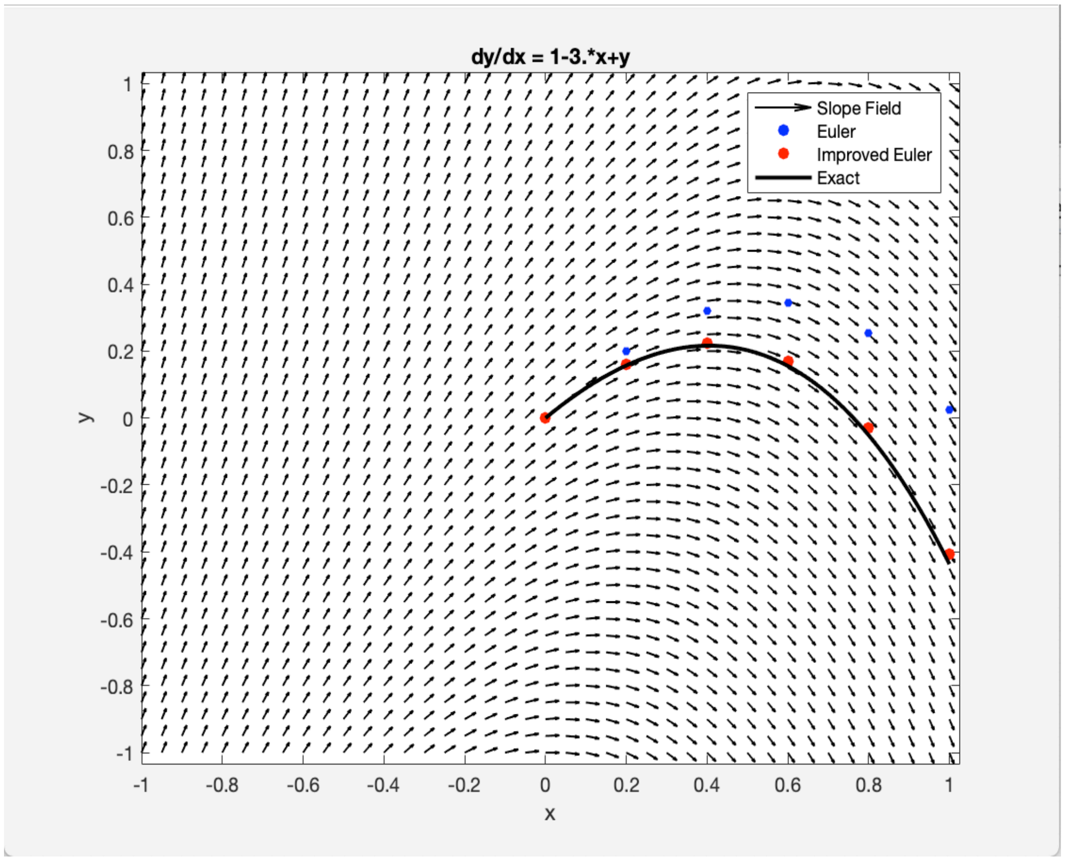
$k_2 = f(x_1, y_1) = f(1, \underset{\substack{\uparrow \\ \text{Euler}}}{y_0 + f(x_0, y_0) h}) = f(1,2) = 2$

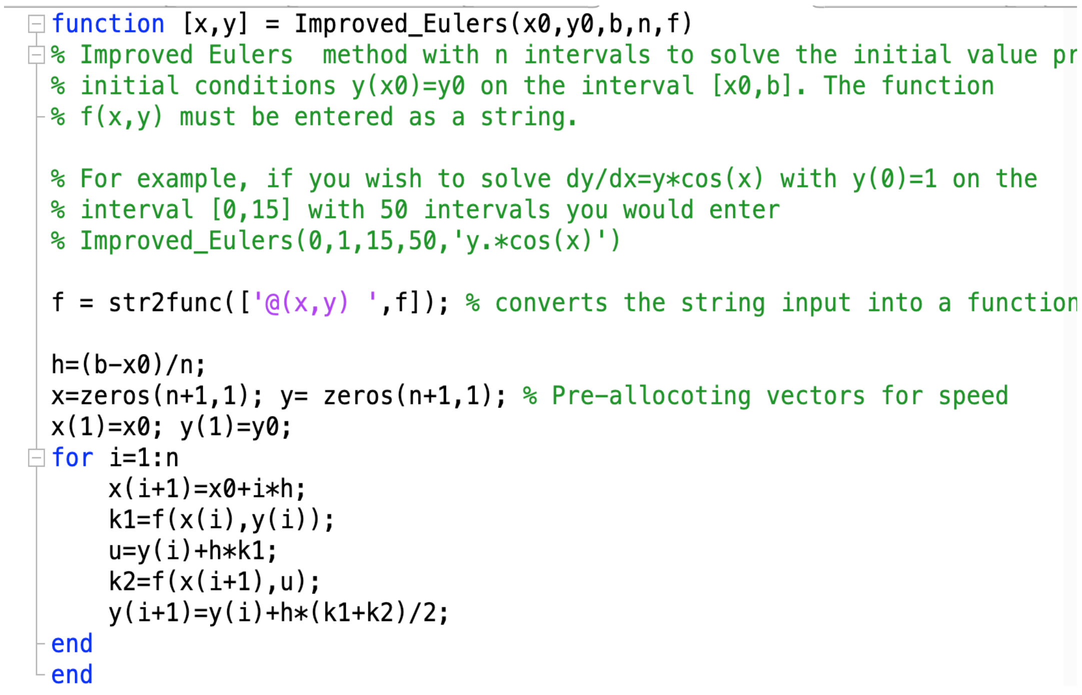
$k = \frac{k_1 + k_2}{2} = 1.5.$

$y_1 = y_0 + hk = 1 + 1 \cdot 1.5 = 2.5.$ blue picture



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
```