Lesson Plan 18 - Direction Fields, Euler's Method 7.2

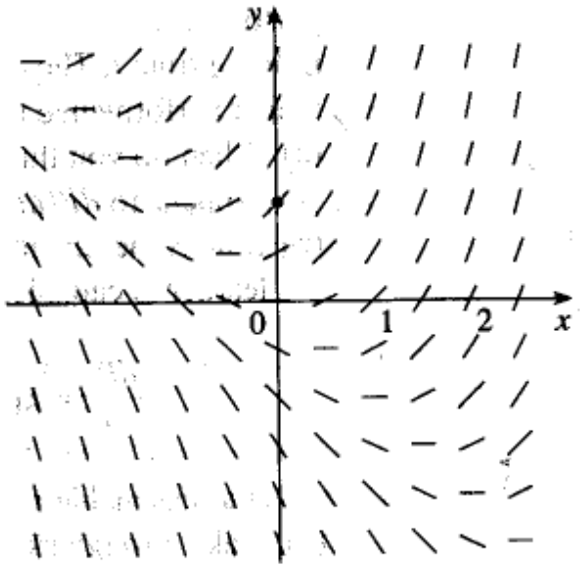
1) Take attendance
2) Return Quiz, questions?
3) Homework questions?

Direction Fields

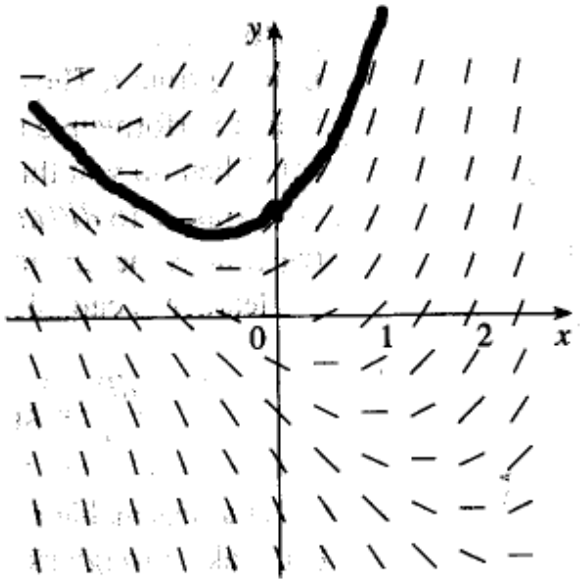
A direction field is a graph designed to help visualize the solution to a differential equation. As an example, we look at the equation:

$y' = x + y$ with the initial condition that $y(0) = 1$

First we draw in set of short lines at some lattice points on the graph.



We can then get a feel for the curve that goes through the point (0,1).



Provide Handout and have them try it and go over it.

Euler's method,

Alternatively we can use an iterative computational method that works like this:

1) Start at (0,1), letting $x_0 = 0,\ y_0 = 1$

2) Find a linear approximation for the curve at (x_i, y_i)

$m = y' = x_i + y_i$

$y = mx + b$

Using the current point we calculate *b*

$b = y_i - mx_i$

3) Using this equation and a small Δx, $x_{i+1} = x_i + \Delta x$ calculate y_{i+1}

$y_{i+1} = m(x_{i+1}) + b$

4) Repeat at stop 2 as many times as necessary.

Provide Program handout

Note: This is not a computer programming class, so of course you are not responsible for the following program.

A computer program that might perform this operation is as follows:

```
#include <libc.h>
#include <math.h>
int main(int argc, char **argv)
{
      double x1,y1;
      double x2,y2;
      double dx;
      double m;
      double b;
      int n;
      int i;

      //
      // Print out a nice usage message
      //
      if (argc != 5)
      {
            fprintf(stderr,"use: euler x1 y1 dx n\n");
            exit(-1);
      }
```

```
	//
	// Initialize Variables from input parameters
	//
	x1 = atof(argv[1]);
	y1 = atof(argv[2]);
	dx = atof(argv[3]);
	n = atoi(argv[4]);

	fprintf(stdout,"Starting at (%f,%f) delta-x: %f  n=%d\n",
x1, y1, dx, n);

	for(i=0;i<n;i++)
	{
		//
		// Print Current Point
		//
		fprintf(stdout,"%3d (%f, %f)\n",i,x1,y1);

		//
		// Calculate slope
		//
		m = x1+y1;

		//
		// Find the next x
		//
		x2 = x1+dx;

		//
		// Calculate b
		//
		b = y1 - (x1+y1)*x1;

		//
		// Calculate the new y
		//
		y2 = m*x2 + b;

		//
		// Reset the variables
		//
		x1 = x2;
		y1 = y2;
	}
}
```

Example Output

The nice thing about doing this programmatically is that you can easily check the accuracy with various dx's

dx = .1	dx = .01	dx=.001
0 (0.000000, 1.000000) 1 (0.100000, 1.100000) 2 (0.200000, 1.220000) 3 (0.300000, 1.362000) 4 (0.400000, 1.528200) 5 (0.500000, 1.721020) 6 (0.600000, 1.943122) 7 (0.700000, 2.197434) 8 (0.800000, 2.487178) 9 (0.900000, 2.815895) 10 (1.000000, 3.187485) 11 (1.100000, 3.606233) 12 (1.200000, 4.076857)	0 (0.000000, 1.000000) 10 (0.100000, 1.109244) 20 (0.200000, 1.240380) 30 (0.300000, 1.395698) 40 (0.400000, 1.577727) 50 (0.500000, 1.789264) 60 (0.600000, 2.033393) 70 (0.700000, 2.313527) 80 (0.800000, 2.633430) 90 (0.900000, 2.997265) 100 (1.000000, 3.409628) 110 (1.100000, 3.875594) 120 (1.200000, 4.400774)	0 (0.000000, 1.000000) 100 (0.100000, 1.110231) 200 (0.200000, 1.242561) 300 (0.300000, 1.399313) 400 (0.400000, 1.583053) 500 (0.500000, 1.796619) 600 (0.600000, 2.043145) 700 (0.700000, 2.326097) 800 (0.800000, 2.649303) 900 (0.900000, 3.016995) 1000 (1.000000, 3.433848) 1100 (1.100000, 3.905031) 1200 (1.200000, 4.436254)
dx = .0001 (.01s) 0 (0.000000, 1.000000) 1000 (0.100000, 1.110331) 2000 (0.200000, 1.242781) 3000 (0.300000, 1.399677) 4000 (0.400000, 1.583590) 5000 (0.500000, 1.797360) 6000 (0.600000, 2.044128) 7000 (0.700000, 2.327364) 8000 (0.800000, 2.650904) 9000 (0.900000, 3.018985) 10000 (1.000000, 3.436292) 11000 (1.100000, 3.908002) 12000 (1.200000, 4.439835)		