Revisiting area.cpp w/ a C++ class

1.) I am not exactly sure what a destructor is.
- They are created by the Circle function (eg Circle my_first_circle(my_radius))
- I think they are destroyed outside The {} because it is outside the scope of the class
- Because in Circle.cpp get/set radius are the public variables that allow calling on the private variable in the class

2). they are destroyed in that order because Circle 2's {}'s are nested in circle 1

3). After a bit of struggle yes.

4). because .get_y & .x are public variables in the class that can be accessed where .y private & ysg are private.
it is only in get_y that y is publicly accessed because it uses the private variable y to allow public access b calling on it w/in the class but does not allow direct access to it, ie using the class to define y as "value passed"

Optimization 101: squaring a

3). pow(x,2) = 5.37341 sec, X*X = 0.44531 sec., X*X is much faster

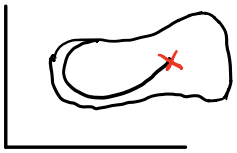
4). It is slower than X*X by ~0.33 sec, I would say the overhead is worthwhile when the expression is more complicated.

5) This one is faster than X*X by ~0.02 sec

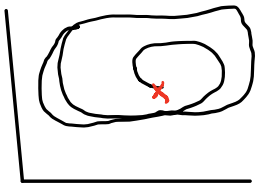
6) inline function is about the same as square it.
so in conclusion I would #define function as it is fastest & allows more complicated expressions

GSL Diff Eqn Solver

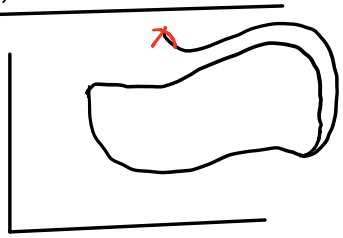
4. $x_0=1$ $v_0=0$



$x_0=0.1$ $v_0=0$



$x_0=-1.5$ $v_0=2.0$



x = start

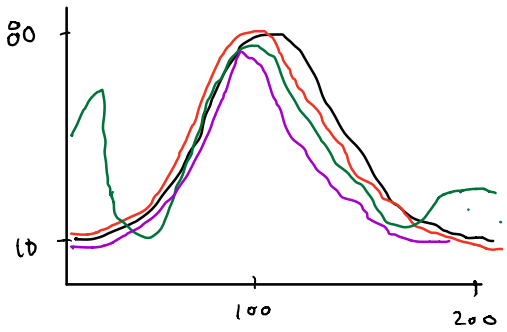
first two start @ zero velocity so go to -x values, 3rd has positive so @ first goes to +x values, all equilibriate @ same values.

GSL Interpolation Routines

1). no questions

4). Yes I modified GslSpline.cpp to include gsl_interp_polynomial

5



- exact — cubic
- poly — linear

- cubic fits the best in all regions, polynomial does okay around the peak but is very poor otherwise. This makes sense as in the note it says to only use polynomial for small regions linear is overall good but not as good as cubic.

Command line mystery

The murderer is Jeremy Bowers.

Python Scripts for C++ Programs.

1. Yes it is just an array

2. Yes

3. Not really I am relatively familiar with numpy.

Cubic Splining

3. I made a loop that generates X and Y value arrays for different # of point. Then I spline the function. I used the central difference function from Simple_derivate.cpp and looped over npts using h = steps. I set an error threshold and terminated the program having it print npts & step

I found I needed 347 points w/ a step size of 0.0115