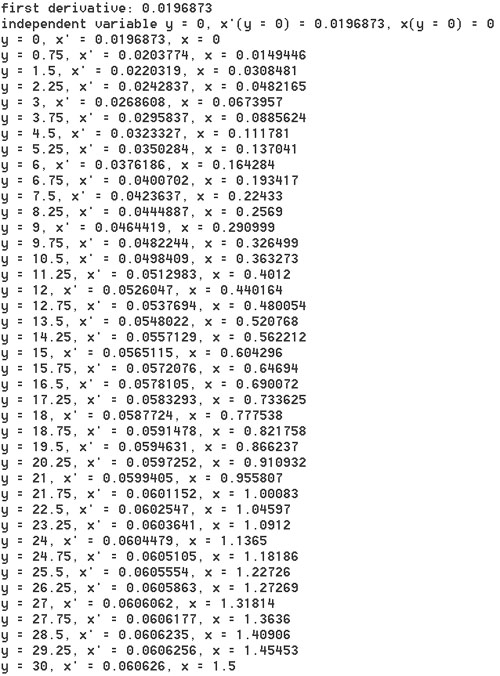
Program #4 The Shooting Method **11.25%**  
Due: 12/02/2016  
Assignment:



The above image shows my result for Problem 28-20 (the sailboat mast deflection problem) on page 831 with a specified target deflection of 1.5m at the top of the 30m mast. The first derivative initial condition (at y = 0, x' = .0196873) was obtained using the shooting method. With that first derivative initial condition, the deflection x of the sailboat mast at each y can be obtained. Note that y is the independent variable in this problem as y is the height up the sailboat mast, and x is the deflection of the sailboat mast from vertical.

When writing the evaluate method for SailboatMast, use the following convention:

* the input variables (xy array) are ordered y (independent variable), z (first derivative), x
* the array that is returned contains the function evaluations fz, fx in that order

**Multi-Step Methods for Ordinary Differential Equations**

In this program, you will extend the discussion of the **Non-Self-Starting Heun** method (page 756) to include the solution to a system of equations (see page 737). You will need to implement the following methods in**Heun.cpp**:

* static double\* rk4Multiple(MultiVarAndEqnsFunction\* f, double h, double\* xy, int len\_xy) //needed for non-self-starting Heun
* static double\* heunMultiple(MultiVarAndEqnsFunction\* f, double h, double\* xy0, double\* xy1, int numeq, double tol)
* static double\* stepsHeunMultiple(MultiVarAndEqnsFunction\* f, int steps, double h, double\* xy, int numeq, double tol) //needs two previous steps

where **h** is the step size, **xy** is an array containing the current values for all of the variables (including the independent variable, the first derivative of the dependent variable, and the dependent variable), numeq is the number of equations, tol is the tolerance to stop the correction iteration, and **steps** is the total number of steps to perform. The array that is returned contains the final values of all of the variables after one or several steps have been computed. **For consistency, let the independent variable be the first index in your returned array, and the dependent variable be the last index**.

Using your NSS Heun implementation, solve Problem 28.8 on page 828 (use **Diffusion.cpp**). Select an approximate value for the first derivative of the concentation of A, which you will compute precisely using shooting method described next.

**Shooting Method**

You will need to use the boundary conditions given in the problem to figure out the correct initial condition for the first derivative of the concentration of A (A\_initial = 0.1, A\_final = 0.0). Once you have Heun ODE solver working, add the **shooting-method computation** to your program as discussed on page 780.

* double evaluate(double z\_init) //called by false position

This method will be called by a root finding algorithm (see below). The parameter **z\_init** is a trial **initial first derivative, A'(x = 0)**. Given a trial first derivative, the evaluate method runs through the entire Heun computation and determines the resulting concentration of A at 4 cm. You need to allow the user to specify a target final concentration, set through the Diffusion constructor. Thus, the above evaluate method computes the difference between the concentration of A for the given trial first derivative and the desired target concentration of A at 4 cm. This effectively recasts the problem as a roots problem.

* double getFirstDerivative(double low, double high) //uses a root finding algorithm

This method uses low and high to run a root finding calculation, which will call the evaluate method just described. The result of this computation is the initial first derivative that gives the specified final concentration. Use **false position** as your root finding algorithm, and place it in FalsePosition.cpp.

* C++ files:
  + [**prog4.zip**](http://mboshart.dyndns.org/~mboshart/3020programs/prog4.zip)