Programming Assignment 1

Due Monday, September 14 at 6 pm

Write a complete C program that implements Newton's method for finding a solution to the equation $f(x) = 0$. Input to the program will be an initial "guess," the desired maximum error tolerance, and the maximum number of iterations. Output should be a message stating

- Whether the iteration converged — i.e., whether the program found a solution,
- The number of iterations,
- The approximate solution, and
- The value of the function at the approximate solution.

Newton's Method

Newton's method is an algorithm for finding solutions to $f(x) = 0$. The method begins with an initial "guess," x_0. It proceeds by looking at the tangent line to the graph of $f(x)$ at $(x_0, f(x_0))$ and guessing that where the tangent line crosses the x-axis will be a better approximation to the actual solution of $f(x) = 0$. Say x_1 is the point where the graph of the tangent line crosses the x-axis. If x_1 isn't a good enough approximation to the actual solution of $f(x) = 0$, the process is repeated with x_1 replacing x_0. Of course, we can continue this process, each time we replace the old guess with the new guess. In general then, we have an old guess x_n, and to get the new guess, x_{n+1}, we look at where the tangent line to the graph of f crosses the x-axis. See Figure 1[1].

This suggests the following basic algorithm:

```
x_new = x_0
Repeat until x_new is good enough {
   x_old = x_new
   Find equation of tangent line at x_old
   x_new = Point where tangent line crosses x-axis
}
```

We can convert this basic algorithm into a more useful form by making the following observations.

[1]The diagram is taken from an old version of the Wikipedia article on Newton's method.

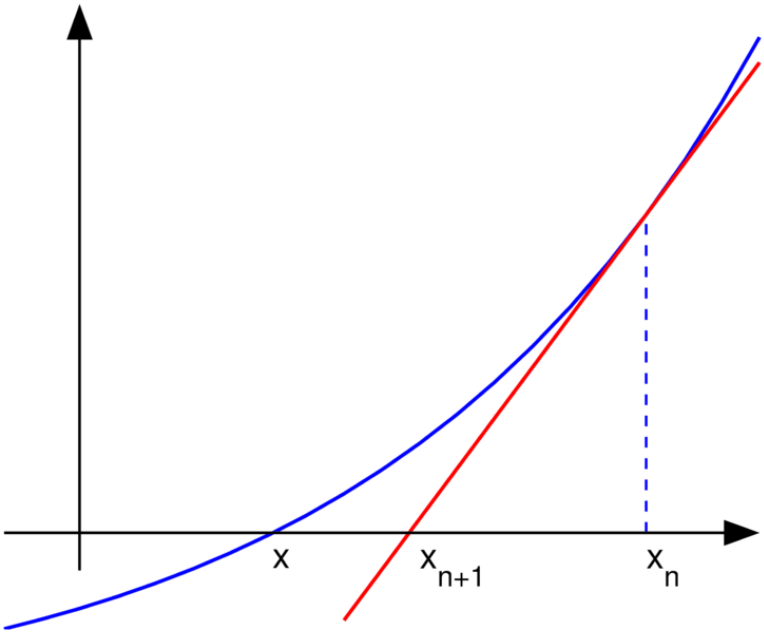

Figure 1: One Iteration of Newton's Method

- At the point (`x_old`, f(`x_old`)), the slope of the tangent line is given by f'(`x_old`). This is just the *derivative* of $f(x)$ at `x_old`.
- If the slope of the tangent line is f'(`x_old`) and and the point (`x_old`, f(`x_old`)) is on the line, then an equation of the tangent line is given by

$$y = f'(\texttt{x\_old})(x - \texttt{x\_old}) + f(\texttt{x\_old}).$$

- Where the tangent line crosses the x-axis can be found by solving the equation

$$0 = f'(\texttt{x\_old})(\texttt{x\_new} - \texttt{x\_old}) - f(\texttt{x\_old}),$$

  which has solution

$$\texttt{x\_new} = \texttt{x\_old} - \frac{f(\texttt{x\_old})}{f'(\texttt{x\_old})},$$

  provided f'(x_old) is nonzero.

So we can revise our algorithm as follows.

```
x_new = x_0
Repeat until x_new is good enough {
   x_old = x_new
```

```
   if (f'(x_old) == 0) {
      print "f'(x_old) is zero!"
      break;  /* Get out of here */
   }
   x_new = x_old - f(x_old)/f'(x_old)
}
```

So we really just need need to decide when `x_new` is "good enough." It turns out that a good upper bound on the difference between `x_new` and the solution is given by the difference

$$|\texttt{x\_new} - \texttt{x\_old}|.$$

So we can revise our algorithm a third time to get

```
x_new = x_0;
do {
   x_old = x_new;
   if (f'(x_old) == 0) {
      print "f'(x_old) is zero!";
      break;  /* Get out of here */
   }
   x_new = x_old - f(x_old)/f'(x_old);
} while (fabs(x_new - x_old) > error_tol);
```

Here `fabs` is the C library function for computing absolute values.

The Program

As noted above, the input will be the initial guess `x_0`, the maximum error tolerance, and the maximum number of iterations through the `do-while` loop. (The function f and its derivative f' are hardcoded in the source.) Also as noted above, output should include

- Whether the method converged – i.e., whether a solution was found within the required number of iterations
- The number of iterations
- The approximate solution
- The function value of the approximate solution (i.e., $f($`x_new`$)$ after the final iteration).

You *must* use a C function to implement Newton's method. It should have the following properties:

- The return value is the approximate solution

- Input args: `x_0, error_tol, max_iters`
- Output args: number of iterations and whether the iteration converged.

You can use more arguments *if* there's a good reason for them.

You *must* also have a function that computes $f(x)$ and a function that computes $f'(x)$.

Note that our algorithm doesn't include the code needed for finding the number of iterations and terminating the `do-while` if there have been too many iterations. You may want to write a version of the program that doesn't count iterations first, and then write a revised version after you get the first version working.

Development

You *should* compile and test your program using Linux, since the parallel computers we have access to all use Linux (rather than Windows or MacOS X).

You will probably want to link your program with C's math library. I also encourage you to get in the habit of compiling with the `-Wall, -g,` and `-o <file>` options. The first option gives extensive warnings. The second creates a symbol table which can be very useful when using a debugger, and the last stores the executable in a file other than `a.out`. Combining all these options you would compile your program with something like

```
gcc -g -Wall -o newton newton.c -lm
```

Here are some functions you can test your code with.

- $f(x) = x^2 - 2$, $f'(x) = 2x$, $x_0 = 2$, $x = 1.4142135623730950$.
- $f(x) = \sin(x)$, $f'(x) = \cos(x)$, $x_0 = 3$, $x = 3.1415926535897932$.
- $f(x) = (x-1)^3$, $f'(x) = 3(x-1)^2$, $x_0 = 2$, $x = 1.0$.
- $f(x) = (x-1)^{1/3}$, $f'(x) = \frac{1}{3}(x-1)^{-2/3}$, $x_0 = 0.5$, $x = 1.0$. To compute the value of this last function, you can use the C math library function, `pow,` which raises a number to a power. However, you need to be careful with the sign of $x - 1$. If it's negative,

  ```
  y = pow(x-1, 1.0/3.0);
  ```

  will result in `nan's`. To correct this, you can use

  ```
  y = pow(fabs(x-1), 1.0/3.0)
  if (x-1 < 0) y = -y;
  ```

Extra Help

I have office hours Mondays and Fridays from 4:45 to 5:45 and Wednesdays from 10:30 to 11:30. Erika has office hours on Thursdays from 3 to 4, and Mark has office hours on Thursdays from 4:30 to 5:30.

Submission

You should develop your program in the `p1` subdirectory of your *local* SVN tree. After you first create your source file, type

```
$ svn add newton.c
$ svn commit newton.c -m "create program 1 source file"
```

It's a good idea to frequently update this by typing something like

```
$ svn commit newton.c -m "modified program to do XXX"
```

This frequent updating is very important: if you accidentally corrupt or destroy your copy of the program, you can always retrieve the most recent version from the SVN server.

Be sure to commit your final version of the source code by 6 pm on Monday, September 14.

```
$ svn commit newton.c -m "final testing completed"
```

Grading

1. Correctness will be 50% of your grade. Does your program find the correct solution given correct input? Does it handle all of the possible input conditions?

2. Documentation will be 20% of your grade. Does your header documentation include the author's name, the purpose of the program, and a description of how to use the program? Are the identifiers meaningful? Are any obscure constructs clearly explained? Does the function header documentation explain the purpose of the function, its arguments and its return value?

3. Source format will be 20% of your grade. Is the indentation consistent? Have blank lines been used so that the program is easy to read? Is the use of capitalization consistent? Are there any lines of source that are longer than 80 characters (i.e., wrap around the screen)?

4. Quality of solution will be 10% of your grade. Are any of your functions more than 40 lines (not including blank lines, curly braces, or comments)? Are there multipurpose functions? Is your solution too clever – i.e., has the solution been condensed to the point where it's incomprehensible?

Academic Honesty

Remember that you can discuss the program with your classmates, but you cannot look at anyone else's pseudo-code or source code. (Of course, this includes code that you might find on the internet.)