# Newton’s Method, Extended

Due: November 7, 2014

## CSC 150 Program 2

**Overview:**

Your second program will find as many roots as possible of a polynomial function that the user chooses, within a specific interval.

**Background:**

In Program 1, we learned about Newton’s Method as a way to find roots (zeroes) of a function. Recall the Newton’s Method algorithm:

```
Set the initial guess to be $x_0$
while $|f(x_n)| > err$ do
    $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ for $n = 0, 1, 2, 3..$
end while
```

In the Program 1 implementation of this method, we used the interval only as a way of finding a starting point. In Program 2, we will limit our search for roots to within the desired interval. Additionally, we will continue the search beyond just one root to find all roots within the desired interval.

Note that for a polynomial function of degree $d$, there can be no more than $d$ roots.

Refer to the Program 1 assignment document for additional background on Newton’s Method.

**Assignment Overview**

Your program will allow the user to enter a polynomial function and find all roots of the function within a specified interval, using Newton’s Method.

**Functional Requirements**

Input

- Your program will display a brief, descriptive message when it first begins. The purpose of the program and how to use it should be clearly stated for anyone who wants to use this program.
- The program prompts for the degree of the polynomial equation (e.g. 3 for a cubic polynomial).
- The program prompts for the polynomial equation. For each degree of the equation, beginning with the highest, prompt the user to enter the associated coefficient value.
    - For example, $f(x) = 4x^3 + x^2 - x - 0.5$ is represented by the inputs 4, 1, -1, -0.5.
    - If a term is missing from the equation, the user must enter 0 for that term.

Error Checking

- Check to make sure that the user enters a positive integer for the polynomial degree. Otherwise, display a message and exit the program.
- You may assume that the user will enter floating point numbers at the remaining input prompts.

Other Required Values

- You are provided with a header file, constants.h, which contains default values that you may use in your program, or modify to add or change your program's functionality. Recall that our first implementation of Newton's Method required a starting value, error tolerance, and maximum number of iterations to be input by the user. In this implementation, constants.h provides named constants to use for the starting value, error tolerance, and maximum number of iterations. In addition, constants.h provides named constants for the start and finish of the interval we are searching within, a step value that determines the size of the sub-intervals, and the maximum degree of the polynomial equation.

Algorithm

- Your program will step over the interval given by the named constants, using START, FINISH, and STEP, to examine small sub-intervals of the search interval.
- In each sub-interval, check for a sign change:
  $f(a) * f(b) < 0$ for an interval $[a, b]$
- If the function has a sign change within the sub-interval, use Newton's Method to search for a root of the function, using the sub-interval values to obtain an initial guess $\frac{(a+b)}{2}$.
- In addition to checking the error tolerance (TOLERANCE) and number of iterations (N_ITER), your program may stop the search when the maximum number of roots is found.

Output

- Your program must output the number of roots found, the value of each root, and the search interval used. Sample outputs appear below.

Implementation Requirements

- Your program must be decomposed into the C++ functions whose prototypes appear in constants.h.
- Your program submission file must be named `prog2.cpp`.
- To use constants.h in your program:
    - After you have created your project and program files, right-click on "Header Files" within Solution Explorer, choose Add New Item, Header File, and name it constants.h.
    - Copy and paste the contents of the constants.h file in the Program 2 folder on the website into this file.
    - Go back to your program file and add the following line near the top of your file, with your other `#include` statements:
      `#include "constants.h"`

**Program Notes**

- Your program will use arrays to represent the polynomial equations and associated roots. Since array sizes must be constant, you may use the included constant MAX_DEGREE for your array sizes.
- Since the function prototypes are already in constants.h, you do not need to have them again in your program file.
- Programs will be graded using the provided constants.h file. Do not change any part of the constants.h file, unless you have already submitted your completed program and wish to add functionality.

**Suggested Tasks**

- First, read and understand the assignment and what concepts you will use in solving it. A great resource to help you ask the right kinds of questions can be found online: search for "Basic Strategy for Algorithmic Problem Solving".
- Draw a flowchart that shows which functions will be called throughout the program, the order in which they will be called, and what pieces of information are passed between each function.
- Write detailed pseudocode for each of the individual functions, including main. Include a table of all variables used in each function and what those variables represent.
- Translate your pseudocode to C++ code and test your program.
- Use debugging cout statements and the Visual Studio debugger to find and fix issues. Refer to pre-lab 5 and book section 5.5 for reminders on how to do this.
- Don't forget to write your code according to the CSC 150 Coding Standard, including Doxygen headers for **every** function definition.
- Submit your prog2.cpp at http://www.mcs.sdsmt.edu/submit by 11:59 pm on the due date.

**Sample Program Runs**

Sample Run 1

```
This program finds the roots of polynomial functions.

Enter the degree of the polynomial equation: 3

Enter the coefficients for the equation.
Enter the coefficient for x^3: 1
Enter the coefficient for x^2: 4.5
Enter the coefficient for x^1: -1
Enter the constant term: -12

There are 2 roots in the interval (-3 , 4): -2  1.5
```

Sample Run 2

```
This program finds the roots of polynomial functions.

Enter the degree of the polynomial equation: 1

Enter the coefficients for the equation.
Enter the coefficient for x^1: 1
Enter the constant term: 1

There are 1 roots in the interval (-3 , 4): -1
```

Sample Run 3

```
This program finds the roots of polynomial functions.

Enter the degree of the polynomial equation: 2

Enter the coefficients for the equation.
Enter the coefficient for x^2: 1
Enter the coefficient for x^1: -0.5
Enter the constant term: -0.1875

There are 2 roots in the interval (-3 , 4): -0.25  0.75
```

Sample Run 4

```
This program finds the roots of polynomial functions.

Enter the degree of the polynomial equation: 2

Enter the coefficients for the equation.
Enter the coefficient for x^2: 1
Enter the coefficient for x^1: 0
Enter the constant term: -4

There are 2 roots in the interval (-3 , 4): -2  2
```

Error Sample Run

```
This program finds the roots of polynomial functions.

Enter the degree of the polynomial equation: 0

Degree must be positive.  Program exiting.
```

Additional Sample Runs

1. Download http://www.mcs.sdsmt.edu/csc150/Assignments/Programs/Program2/prog2.exe to your computer.
2. Go to the parent directory of the directory that contains the downloaded prog2.exe file.
3. Hold Shift key and right click the icon of the directory with prog2.exe, then select “Open command window here” from the pop-up menu.
4. Type prog2.exe in the console and hit Enter key to execute the demo program.