GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

Outline

Plotting

- C does not have built-in plotting functions.
- There are various third-party libraries developed in C for plotting mathematical functions. But these libraries are usually developed for specific OS, and can hardly be used across different OS's.
- In real-world work place, one rarely plots using C. Rather we use other more convenient tools.
- In this course, whenever you are asked to plot, you can use any third-party tool to plot, for example, here is an online plotting tool:
  `http://gnuplot.respawned.com/`
- To plot, you can prepare you data (as a collection of (x, y) coordinates) in an ASCII file (or simply print to the console, if you are not asked explicitly to create the file), and load/paste the data in your choice of plotting software.

Outline

Designing a Function (Proptotype)

Suppose that in the `main` function or any other function, we need to call a function `foo` with input X to to obtain some information, say Y, about X. To do this, there are two ways:

1. Make `foo` return Y.
2. Create a variable in the calling function (e.g. `main`), that serves as a "receiver" for Y; pass the address of the receiver into the function (i.e., via pass by reference). In `foo`, compute the value Y and save it to the receiver through its address.

- A limitation of the first approach is that that Y can only be a single variable (since a function can only return one variable). When Y includes multiple variables, if they can be "naturally bundled", one can create a structure including these variables as the members of one structure-typed variable.
- It is also possible to combine the two approaches, making some components of Y returned from `foo` and sending other components of Y via pass by reference.

Outline

- In engineering, we are often interested in solving equations in the form of

$$f(x) = 0.$$

  where f is an arbitrary function.
- Via computer programming, usually we can very rapidly evaluate the function (i.e., compute the value of $f(x)$ for any x).
- Can we leverage this fact to solve equation $f(x) = 0$?
- We now consider the case where $f(x)$ is a polynomial in x, namely, $f(x)$ in the form of

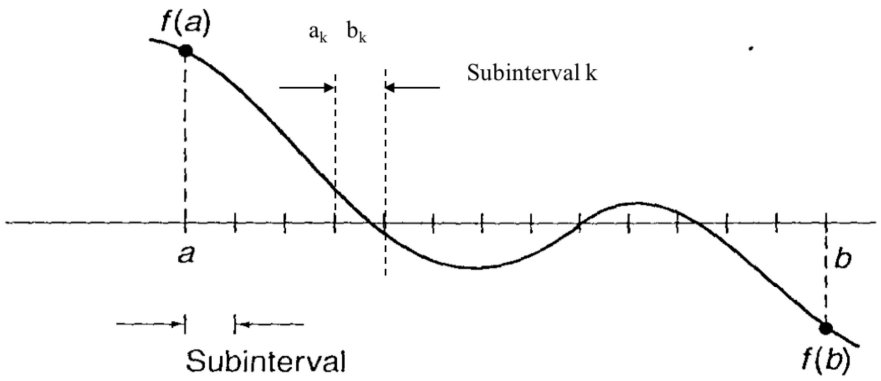
$$f(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c^n x^n$$

- The method we will develop apply in general to other kinds of functions $f(x)$.

- In polynomial $f(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c^n x^n$, the highest power (n) of x is called the degree of the polynomial.
- Solutions to polynomial equation $f(x) = 0$ is called the roots or zeros of polynomial $f(x)$.
- It is known that a polynomial with degree n has maximally n real roots.
  - $x + 3$ has one root, -3.
  - $x^2 - 1$ has two real roots, ± 1.
  - $x^2 + 1$ has no real roots.
  - $x^2 - 2x + 1 = (x - 1)^2$ has two repeated roots, $+1$ and $+1$.

- **Problem Statement:**
  Given a polynomial $c_0 + c_1x + c_2x^2 + \ldots + c^nx^n$ and an interval (a, b), find all real roots of the polynomial in the interval.
- **I/O Description**:
  - Input:
    - coefficients $c_0, c_2, \ldots, c_n$,
    - two end points of the interval, a and b
  - Output:
    - the roots of polynomial $c_0 + c_1x + c_2x^2 + \ldots + c^nx^n$ in (a, b)

Root Finding by Incremental Search



- Divide the interval (a, b) to many tiny subintervals.
- For each sub-interval, say (a_k, b_k), test if $f(a_k)$ and $f(b_k)$ take opposite signs, or, if $f(a_k) \cdot f(b_k) < 0$.
  - If yes, there is a root in (a_k, b_k); we then use the mid-point of the sub-interval, $\frac{a_k + b_k}{2}$, as the approximation of the root.
  - When the interval is sufficiently small, the approximation is very accurate.

Caution

- What if at one end of a sub-interval (a_k, b_k), the function has value 0? For example $f(a_k) = 0$, then we would have missed a_k as the root.
  - We should also check if $f(a_k)$ is zero.
    - It is sufficient to only check the left end of the sub-interval, since the right end is the left end of the next sub-interval.
  - But in programming, we never check if two floating-point (say, `double` typed) values are exactly equal, since even when they meant to be exactly equal, they usually aren't, due to finite precision in the representation of floating-point numbers.
  - In practice, checking two floating-type values, say x and y are equal is implemented as checking if `fabs(x-y)<ALMOST_ZERO`, where `ALMOST_ZERO` is an extremely small positive number.

```
#include <stdio.h>
#include <math.h>
#define ALMOST_ZERO 1e-20
#define SUBINTERVAL_SIZE 0.0001

double evalPoly(int polyDegree, double *polyCoeffs, double x)
{
   double result=0;
   int i;
   for (i=0; i<=polyDegree; i++)
   result = result + polyCoeffs[i]*pow(x, i);
   return result;
}
```

```
int findARootInSubInterval(int polyDegree, double *polyCoeffs, double leftEnd,
double rightEnd, double *ptrFoundRoot)
{
   double leftVal, rightVal;
   int found=0;
   leftVal=evalPoly(polyDegree, polyCoeffs, leftEnd);
   rightVal=evalPoly(polyDegree, polyCoeffs, rightEnd);
   if (fabs(leftVal)<ALMOST_ZERO)
   {
     *ptrFoundRoot = leftEnd;
     found=1;
   }
   else
   {
     if (leftVal*rightVal<0)
     {
       *ptrFoundRoot = 0.5*(leftEnd+rightEnd);
       found=1;
     }
   }
   return found;
}
```

```
int findAllRootsByIncSearch(int polyDegree, double *polyCoeffs, double xStart,
double xEnd, double *ptrAllRootsFound)
{
   int numOfRootsFound=0;
   double left, right;
   int rootFound;
   left=xStart;
   while (left<xEnd)
   {
     right=left+SUBINTERVAL_SIZE;
     rootFound=0;
     rootFound=findARootInSubInterval(polyDegree, polyCoeffs, left, right,
       &ptrAllRootsFound[numOfRootsFound]);
     if (rootFound)
       numOfRootsFound++;
     left=right;
   }
   return numOfRootsFound;
}
```

Potential Issues with Incremental Search

- No roots may be found when the range of search does not include any root.
- Found roots are not accurate if sub-interval size is not sufficiently small.
  - Incremental search can be slow if we demand very high accuracy.
- Incremental search may fail on repeated roots, since in this case $f(a_k) \cdot (b_k)$ is positive.
- When `ALMOST_ZERO` is not sufficiently small and the sub-interval size is too small, a root might be identified as several roots. This usually only happens when the polynomial coefficients have too small amplitudes so that its curve intersects with the x axis at at tiny angle. A fix for this is to scale up all coefficients of the polynomial.

Bisection Root Finding

- Suppose that for a given interval (a, b) we know there is exactly one root and we have observe $f(a)f(b) < 0$. In this case, we can use bisection search to find the root.
- Algorithm:
  - Keep cutting the interval into two halves, and identify the half that contains the root. When the interval is small enough, take its mid-point as the root.
  - Also caution the case where the root falls on an endpoint of the interval.

In-Class Exercise: