Middle East Technical University
Department of Computer Engineering

CENG 140
Section 3

Spring 2018

Take Home Exam 1

Regulations

Due date: 02 April 2018, Monday (23:59) *(Not subject to postpone)*

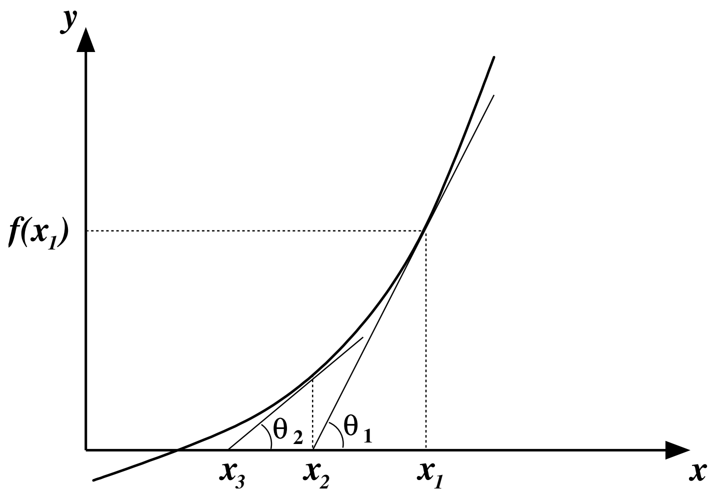
Submission: Electronically. You will be submitting your program source code written in a file which you will name as `the1.c` through the cow web system. Resubmission is allowed (till the last moment of the due date), The last will replace the previous.

Team: There is **no** teaming up. The take home exam has to be done/turned in individually.

Cheating: All parts involved (source(s) and receiver(s)) get zero. You will face disciplinary charges.

Introduction

This THE is about root finding. A so called Newton-Raphson method is a good and robust way to determine numerically the positions of the roots in a given interval. The technique makes use of the concept of the tangent line of a function. Unless the slope of a tangent line is zero, it is bound to intersect the x-axis.



This point of intersection is easily calculated:

$$\tan\theta_1 = \frac{f(x_1)}{x_1 - x_2} = f'(x_1)$$

Which yields

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

x_2 is closer to the root compared to x_1. It can be proven that if this process is iteratively continued in the limit x_n will meet the root. So, generalizing

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In the application of this iterative algorithm it is important to choose the initial point x_1 in the close proximity to the root.

The derivative is also to be calculated in numerical form. As you know the value of a derivative at a given point a is de defined by:

$$\left.\frac{df(x)}{dx}\right|_{x=a} = f'(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h}$$

Of course in real life to take the limit is unrealistic. h has to be chosen as a small value. To approximate the real limit, it is wise to take the average of the right hand side where h is first a positive small value and then a negative one. Let us assume the magnitude of this small value is δ, then the numeric derivative value at a is calculated as:

$$f'(a) = \frac{f(a+\delta) - f(a-\delta)}{2\delta}$$

Combining this tool of numerical differentiation with the Newton-Raphson method we are able to compute the roots of an analytic function in a given interval.

Problem

Your program will be linked with an object code that defines an analytic function with the name `f`.

Your program, when linked with the definition of `f()` and run, will read three floating point numbers from the input: the lower and upper limits of the search interval, and an tolerance value. Then it will search and print out the roots (in increasing order) of the function `f()` that fall in the given interval (including the end points). A printed root value will be accepted if it is in the range $[x_{real\_root} - tolerance, x_{real\_root} + tolerance]$.

There is no restriction on the form of the function. But you are assured the continuity of the function over the interval that will be given from input.

The prototype line (yes you have to type that into your `the1.c` program) for the function `f()` is:

```
double f(double x);
```

Specifications

- Use `double` for all floating point values.
- You do not have the freedom to change the prototyping of the function nor its name. Also, **do not** submit a code that actually defines an `f()`. (You are strongly advised **not to hard code** the function `f()` into the `the1.c` file even for your testing. Keep it in a separate file, compile it separately into an object and then link it with your the1 object)
- Here is an example function definition that you can use. As, said, even for test purpose compile it separately.

```
double f(double x)
{
   return x*x + sin(x+1) + x;
}
```

- You are **not allowed** to define any function other than the `main()` function and the `f()` function. (For `f()` the definition is only a prototype line).
- You are **not allowed** to use macros (`#define`) with arguments.
- Input is from standard input (keyboard) and output is to standard output (screen).
- You can assume that the input is error free.
- You are assured that there are no two roots closer than $10 \times Tolerance$
- The input consists of three floating points separated by (at least one) whitespaces:

  $Interval_{lower\_bound} \quad Interval_{upper\_bound} \quad Tolerance$

  Example:

  ```
  -8.7     11.1 0.001
  ```
- The output is the root values, printed in increasing order, separated by at least one blank. No additional make-up of the input, no beautification!
  Here is the example output for the example function and the input above (yours may deviate slightly):

  ```
  -7.142721 -4.372346 0.000008 2.545359 5.085252 8.542115
  ```
- Neither the actual functions that will be used for `f()` nor the actual test data will be provided.
- If your test function `f()` contains function calls of `expt, sqrt, log` or any trigonometric function don't forget to link it with `-lm`.

An example development

1. type into a file that you might name `cicifonksiyon.c` (you can chose another name)

   ```
   #include <math.h>
   double f(double x)
   {
     return x*x + sin(x+1) + x;
   }
   ```
2. Compile it into an object by:

   ```
   gcc  -c  cicifonksiyon.c
   ```
   This will create in the same directory the object file `cicifonksiyon.o`
3. Write/modify your solution in the file `the1.c`
   (This file should have the prototype line for `f()`, don't forget it)
   Now compile and also link with the `cicifonksiyon.o`:

   ```
   gcc  -o the1 the1.c  cicifonksiyon.o -lm
   ```

As said in the lecture, gcc is both a compiler and a linker. Because of the `-o` option the executable will not be given the name of `a.out` but `the1`. Furthermore, the suffix of `cicifonksiyon.o` tells gcc that this is an object file and after the compilation of `the1.c`, during the linking phase, this object has to be linked in as well. The `-lm` tells that possibly mathematical functions are used and they shall be linked in from the math object library.

4. Now you can run your program at the Unix prompt:

   ```
   ./the1
   ```

   The program run will pause and wait for you to input the input parameters.

5. If everything is ok, presumably you will test other function definitions and continue from (1), if you want to alter only `the1.c` then you continue from (3).

Grading

- Your program will be compiled and run N times with different inputs. The grade weights of the runs are equal. Testing is, as usual, black-box style.
- A run is expected to terminate in at most 1 second. Otherwise it will be graded as zero for that input. The proper implementation of the algorithm, for the example function above, with an input line of `-100 100 0.0001` has found 64 roots in 0.26 sec CPU time, on an AMD 2800, Linux.
- Announcing a root which is not in the range or is not a root at all will cost twice the points of a missing of a root. Multiple output of the same root will count as a 'not root case'.
- Any program that cannot collect 30 points will be graded by eye (glass-box evaluation). These eye evaluation are not open to question. They are final and decisive and are not subject to any objection or questioning.

BUGÜN NEWTON-RAPHSON UYGULAMASI VERDİLER.
BUNUN MUHABBETİ DE HİÇÇEKİLMİYO YAW...