Comp. Methods in Mech. Eng.

MCG 4127

Assignment # 4

Name: Usama Tariq

Student Number: 7362757

Professor: Catherine Mavriplis

TA: Fabien Giroux

Due Date: January 24, 2019

Introduction

In this assignment the purpose was to understand the time complexity for finding roots utilizing two different methods – namely the bi-sectional method and false position. These methods are used to solve a mechanical problem where the deflection of a nonlinear spring is observed. In FigureX, a block of mass, m is released a distance, h, above a nonlinear spring.

The resistance force, F, of the spring is given as:

$$F = -(k_1 d + k_2 d^{\frac{3}{2}})$$

Conservation of energy can be used to show that:

$$mgd + mgh = \frac{2}{5} k_2 d^{\frac{5}{2}} + \frac{1}{2} k_1 d^2$$

Given the following parameters, d can be solved using either method.

$$k_1 = 40{,}000 \frac{g}{s^2}$$

$$k_2 = 40 \frac{g}{s^2}$$

$$m = 95\ g$$

$$g = 9.81 \frac{m}{s^2}$$

$$h = 0.43\ m$$

Using the given values and solving for f we obtain:

$$f(d) = 16d^{2.5} + 20000d^2 - 931.95d - 400.33$$

Plotting this equation yields

Bi-Sectional Method

In the Bi-Sectional method, knowing to values of the function (with opposing signs) and repeatedly bisecting the interval to a smaller range each iteration. The equation of this method is:

$$midpoint = \frac{a+b}{2}$$

False Position Method

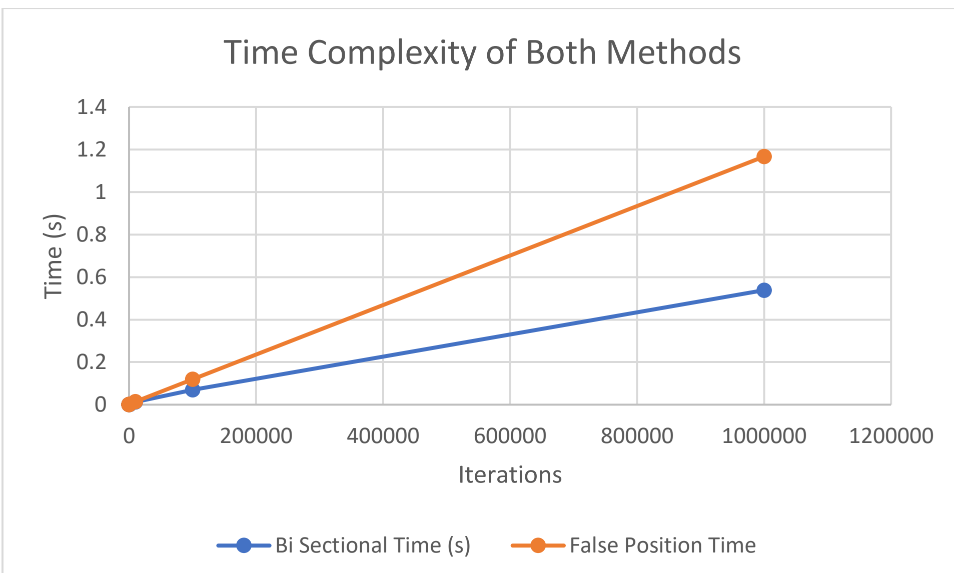
The false-position method also uses a trial and error method to find the root of a function and the equation it uses is as follows:

Given a function $f(x)$ continuous on an interval $[a, b]$ such that $f(a) * f(b) < 0$

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Results and Discussion

Iterations	Bi-Sectional	Time (s)	False Position	Time
5	0.166719	0	0.168326	0
8	0.16666	0	0.168326	0
9	0.16665	0	0.166652	0
10	0.166655	0	0.166652	0
11	0.166653	0	0.166652	0
12	0.166652	0	0.166652	0
900	0.166652	0.001036	0.166652	0.000996
10,000	0.166652	0.01215	0.166652	0.018351
100,000	0.166652	0.070351	0.166652	0.119706
1,000,000	0.166652	0.538294	0.166652	1.16678



In the table and graph above as the number of iterations increases the accuracy gets better in both methods. After a certain point (around 50 iterations) the accuracy of the results did not improve for up to 5 significant digits. Observing the time complexity, it turns out that the bi-sectional method takes less time than the false position. However, this does not matter too much because it depends on how the compiler goes through the code and there could be some stuff that could stay on the stack for longer than others.

As in terms of which method gets to the correct answer the fastest it is the false position method which gets to the answer at 9 iterations while the bi-sectional method gets to the same answer by bi-sectional method.

Varying Constants

Varying the constants such as k_1, k_2, m and h by 10-20% to see the effect on d should be studied.

	-20%	$+20\%$
k_1	0.18999	0.149997
k_2	0.166728	0.166716
m	0.146586	0.185483
h	0.152004	0.180067

Appendix A

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include <math.h>
#include <functional>
#include <ctime>
#include <chrono>

// Using typedefs for all functions
using function_type = std::function<double(double)>;

// Using a template returning bool which checks if
// two numbers have the same sign
// TEMPLATE DECLERATION
template <typename T>
inline bool SameSign(T x, T y);

// Template implementation
template <typename T>
inline bool SameSign(T x, T y){
  return (x >= 0) ^ (y < 0);
}

// Function computing the bi seciotional method
double bi_section_method(function_type f, double a, double b, double iterations, double
 tolerance){

  // Starting a timer to see how long this function takes
  auto start = std::chrono::high_resolution_clock::now();

  double mid;
  double result;
  for (int i = 0; i < iterations; i++){
    mid = (a+b)/2;
    // if (f(mid) == 0 || (b-a)/2 < tolerance){
    //   result = mid;
    // }

    if ( SameSign(f(mid), f(a)) ){
      a = mid;
    } else {b = mid;}

    if (i == iterations - 1){result = (a+b)/2;}
  }

  auto finish = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = finish - start;
  std::cout << "Elapsed time: " << elapsed.count() << '\n';

  std::cout << "Bi Section Method using " << iterations << " iterations: " << result <<
 std::endl;
  std::cout << '\n';
  return result;
}

// Function computing the false position method
```

```
double false_position (function_type f, double a, double b, double iterations, double tolerance){

  auto start = std::chrono::high_resolution_clock::now();

  double c;
  double result;
  for (int i = 0; i < iterations; i++){
    c = (a*f(b) - b*f(a))/(f(b) - f(a));
    // if (f(c) == 0 || (b-a)/2 < tolerance){
    //   result = c;
    // }

    if (f(a)*f(c) < 0){
      b = c;
    } else{ a = c; }

    if (i == iterations - 1){result = (a+b)/2;}

  }

  auto finish = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = finish - start;
  std::cout << "Elapsed time: " << elapsed.count() << '\n';

  std::cout << "Fale Position Method using " << iterations << " iterations: " << result << std::endl;
  std::cout << '\n';
  return c;
}



int main(){

  auto f = [](double d){
    return 16*pow(d,2.5) + 20000*pow(d,2) - 931.95*d - 480.886;
  };

  // VALUES
  double iterations = 1000;
  double a = 0.15;
  double b = 0.19;
  double tolerance = 0;

  std::cout << "------------------------------------------------------\n";
  std::cout << "------------------------------------------------------\n";
  std::cout << "|                  BI-SECTIONAL METHOD               |\n";
  std::cout << "------------------------------------------------------\n";
  std::cout << "------------------------------------------------------\n";

  bi_section_method (f, a, b, iterations, tolerance);

  std::cout << "------------------------------------------------------\n";
  std::cout << "------------------------------------------------------\n";
  std::cout << "|                  FALSE POSITION METHOD             |\n";
  std::cout << "------------------------------------------------------\n";
  std::cout << "------------------------------------------------------\n";
  false_position (f, a, b, iterations, tolerance);


```

```
        }
```