**Numerical Optimization**

**Programming #1**

**Electrical Engineering & Computer Science**

**20171014 Young-Chul Yoon**

1. **Introduction**

**1) Goal of the assignment**

- Implement Bisection, Newton’s, Secant, Regula-Falsi methods  
- Discuss their comparative performance for at least five different problems

**2) Description about implementation and setting**

I implemented my code using C/C++. I tried to use clock() to compare the time comsumption of each method at first. But, it was hard to discriminate. I guess this is because of fastness of C/C++. So, I also use the **number of iteration** for a criteria of performance comparison.   
I set the stopping condition as (new\_x – prev\_x) < or f(x) == 0.  
Five problems were defined as below.  
1)   
2)   
3)   
4)   
5)

1. **Result in program**

Bisection, a = 0.5, b = 2  


Newton, x = 0.5  


Secant, x1 = 0.5, x0 = 2  


Regul Falsi Method, a = 0.5, b = 2  


Bisection, a = -0.5, b = 5  


Newton, x = -0.5  


Secant, x1 = -0.5, x0 = 5  


Regula Falsi Method, a = -0.5, b = 5  


Bisection, a = -1, b = 2  


Newton, x = -1  


Secant, a = -1, b = 2  


Regula Falsi Method, a = -1, b = 2  


Bisection, a=-10, b=5  


Newton, x = -10  


Secant, x1 = -10, b = 5  


Regula Falsi Method, a = -10, b = 5  


Bisection, a=0.3, b=10  


Newton, x=0.3  


Secant, x1=0.3, x2=10  


Regula Falsi Method, a=0.3, b=10  


1. **Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Function** | **Initial points** | **Solution** | **Iteration** |
| **Bisection** |  | a=0.5, b=2 | 0.578644 | 13 |
|  | a=-0.5, b=5 | -0.38192 | 15 |
|  | a=-1, b=2 | -0.688446 | 14 |
|  | a=-10, b=5 | 1.72762 | 17 |
|  | a=0.3, b=10 | 0.653078 | 16 |
| **Newton** |  | x=0.5 | 0.578714 | 2 |
|  | x=-0.5 | -0.381966 | 2 |
|  | x=-1 | -0.688368 | 4 |
|  | x=-10 | -9.97552 | 2 |
|  | x=0.3 | 0.653012 | 3 |
| **Secant** |  | x1=0.5, x0=2 | 0.578714 | 4 |
|  | x1=-0.5, x0=5 | -0.381966 | 4 |
|  | x1=-1, x0=2 | -0.688368 | 6 |
|  | x1=-10, x0=5 | 1.97897e+06 | **224** |
|  | x1=0.3, x0=10 | 0.653015 | 4 |
| **Regula Falsi Method** |  | a=0.5, b=2 | 0.578715 | 4 |
|  | a=-0.5, b=5 | -0.382205 | **17** |
|  | a=-1, b=2 | -0.692776 | **166** |
|  | a=-10, b=5 | 4.37975 | 5 |
|  | a=0.3, b=10 | 0.651321 | **91** |

**Fastness** : Newton > Secant > (Regula Falsi) > Bisection > (Regular Falsi)

The result shows that Newton and Secant method have similar speed to converge.  
Newton is just slightly faster than Secant. (This could be reversed according to initial x-points). As a bold number in iteration column of table above, Secant could diverge or may take huge iteration in some situation.  
Bisection is normally slower than Newton and Secant. Regula Falsi Method shows inconsistent speed. It converges as fast as Secant in best case. But in bad case, it is slower than Bisection. This fluctuation of performance is because of its additional process. Regula Falsi Method check the sign and substitute the one of two x-points with new one like bisection method. This process guarantee the safety of convergence, but simultaneously could lower the performance.

1. **Code Implementation**

**< Setting and Main function >**

|  |
| --- |
| #include <iostream>  #include <cmath>  #include <ctime>  using namespace std;  double Diff(double(\*f)(double), double);  void ValidCheck(double);  double func1(double);  double func2(double);  double func3(double);  double func4(double);  double func5(double);  double Bisection(double(\*f)(double), double, double, int);  double Newton(double(\*f)(double), double, int);  double Secant(double(\*f)(double), double, double, int);  double RFM(double(\*f)(double), double, double, int);  void main() {  double start = clock();  //double sol = Bisection(func5, 0.3, 10, 0);  //double sol = Newton(func5, 0.3, 0);  //double sol = Secant(func5, 0.3, 10, 0);  double sol = RFM(func5, 0.3, 10, 0);  double end = clock();  cout << "Answer : " << sol << endl;  cout << "Time : " << (double)(end-start)/CLOCKS\_PER\_SEC << endl;  }  // Differentiation function  double Diff(double(\*f)(double), double x) {  double eps = 0.0000001;  return (f(x + eps) - f(x)) / eps;  }  void ValidCheck(double x) {  if (isnan(x)) {  cout << "Invalid number error!!" << endl;  exit(1);  }  return;  }  double func1(double x) {  double f = log(x) + sin(x);  ValidCheck(f);  return f;  }  double func2(double x) {  double f = pow(x, 2) + 3 \* x + 1;  ValidCheck(f);  return f;  }  double func3(double x) {  double f = pow(x, 5) + 2 \* pow(x, 3) + exp(pow(x, 2) + x);  ValidCheck(f);  return f;  }  double func4(double x) {  double f = cos(pow(x, 2) + x);  ValidCheck(f);  return f;  }  double func5(double x) {  double f = pow(x, 3) + log(pow(x, 3)) + 1;  ValidCheck(f);  return f;  } |

**< Bisection method >**

|  |
| --- |
| double Bisection(double(\*f)(double), double a, double b, int iter) {  /\* Bisection method  \* f = function  \* a = first\_x  \* b = second x  \* iter = number of iteration  \*/  if (iter == 0 && f(a)\*f(b) > 0) {  cout << "Invalid input error" << endl;  exit(1);  }  double new\_x = (a + b) / 2;  if (f(new\_x) == 0 || abs(new\_x - a) < 0.0001){  cout << "Iteration : " << iter << endl;  return new\_x;  }  else if (f(a)\*f(new\_x) < 0)  return Bisection(f, new\_x, a, iter+=1);  else  return Bisection(f, new\_x, b, iter+=1);  } |

**< Newton method >**

|  |
| --- |
| double Newton(double(\*f)(double), double x, int iter) {  /\* Newton method  \* f = function  \* x = first\_x  \* iter = number of iteration  \*/  double new\_x = x - f(x) / Diff(f, x);    if (f(new\_x) == 0 || abs(x - new\_x) < 0.0001) {  cout << "Iteration : " << iter << endl;  return new\_x;  }  else  return Newton(f, new\_x, iter+=1);  } |

**< Secant method >**

|  |
| --- |
| double Secant(double(\*f)(double), double x1, double x0, int iter) {  /\* Secant method  \* f = function  \* x1 = recent\_x  \* x0 = prev\_x  \* iter = number of iteration  \*/  double new\_x = x1 - (f(x1) / ((f(x1) - f(x0)) / (x1 - x0)));    if (f(new\_x) == 0 || abs(x1 - new\_x) < 0.0001) {  cout << "Iteration : " << iter << endl;  return new\_x;  }  else  return Secant(f, new\_x, x1, iter+=1);  } |

**< Regula Falsi Method >**

|  |
| --- |
| double RFM(double(\*f)(double), double a, double b, int iter) {  /\* Regula Falsi method  \* f = function  \* a = recent\_x  \* b = prev\_x  \* iter = number of iteration  \*/  if (iter == 0 && f(a)\*f(b) > 0) {  cout << "Invalid input error" << endl;  exit(1);  }  double new\_x = a - (f(a) / ((f(a) - f(b)) / (a - b)));  if (f(new\_x) == 0 || abs(a - new\_x) < 0.0001) {  cout << "Iteration : " << iter << endl;  return new\_x;  }  else if (f(a)\*f(new\_x) < 0)  return RFM(f, new\_x, a, iter+=1);  else  return RFM(f, new\_x, b, iter+=1);  } |