Ministry of Education and Science of the Russian Federation
Peter the Great St. Petersburg State Polytechnic University
Institute of Computer Science and Control Systems
Control Systems and Technology Department

Report for Laboratory 1

One-Dimensional Minimization Methods
Discipline: Methods of Optimization
22 November 2016

Student Group: 13541/8

Christopher W. Blake

Professor

Rodionova E.A.

St. Petersburg
2016

Contents

Introduction

The following objective function is provided, and a search algorithm should be used for locating the point of minimization. To determine this minimization point, three different methods are utilized and compared.

Objective Function:

f(x) = 10*x*ln(x)-x^2/2

where x is in the range [0.2 1]

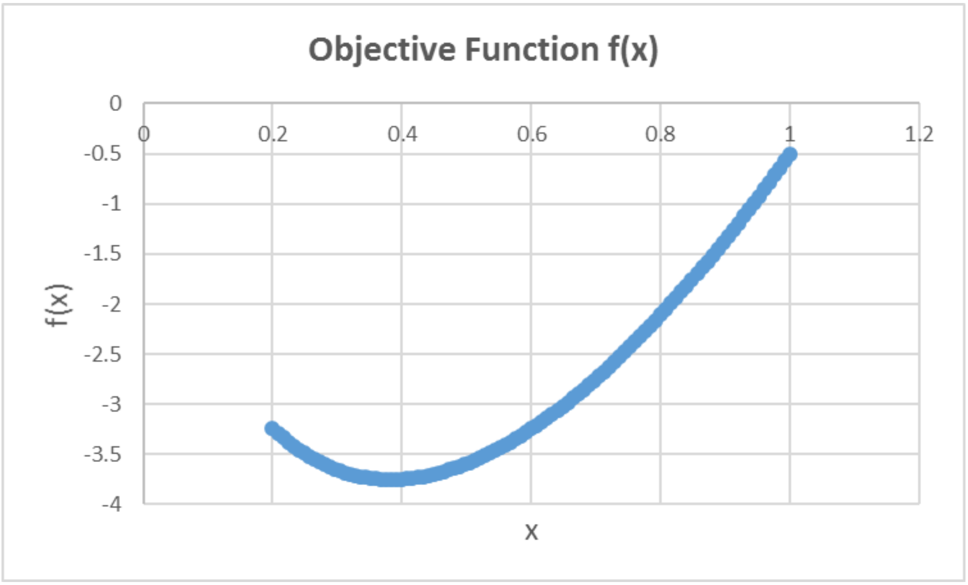
Methods

1. Direct Uniform Search
2. Dichotomy
3. Golden Ratio

Unimodal

Before such search methods can be performed, the objective function must be tested if the objective function is unimodal or not. (ie there can only be one minimum in the range.)

It can be seen from the below graph that the function is unimodal for the specified range.



Method Descriptions

Direct Uniform Search Method

Description

A range for the possible solutions is split into “n” intervals. The value of the objective function is calculated from the beginning of this interval until the minimization point is located. The program then repeats on this new narrow interval, until the new range is smaller than the required accuracy.

Program Outline

1.) Specify inputs

   a. f(x) – Objective Function

   b. eps – Accuracy Value
   c. a – Range Start
   d. b – Range End
   e. n – Number of intervals in range
2.) Determine the step size [h]. Example : h = (b-a)/n;
3.) Compute value at current position [k].
4.) Check if the value f(x[k]) is greater than at [k-1].
5.) If the value at [k] is greater the minimum point has been found. If not found, move to next [k] position.
6.) After the minimum position of [k] is found, change the range values (a and b) to the existing [k-1] and [k+1] values.
7.) Check if the new range (a-b) is smaller than the accuracy (eps.) If so, end the search and continue to step 8. If not, repeat steps 2 through 6.
8.) Return the results
   a. Final range: [a b]
   b. Final calculations: [f(a) f(b)]
   c. Number of calculations of f(x) required to reach the answer.

Dichotomy Method

Description

A point is chosen in the middle of the provide range. This point is compared to two points slightly on each side to determine which half of the range the solution is in, thereby eliminating half of the range for each cycle. The program then repeats on this new narrow interval, until the new range is smaller than the required accuracy.

Program Outline

1.) Specify inputs
   a. f(x) – Objective Function
   b. eps – Accuracy Value
   c. a – Range Start
   d. b – Range End
2.) Determine how large of a step to check on both sides of the middle point.
   a. delta = (a-b)*0.01;
3.) Pick the point in the middle of the range.
   a. Xi = (a+b)/2;
4.) Pick the point just before and just after this middle point
   a. X1 = Xi - delta; X2 = Xi + delta;
5.) Change the range according to Xi, X1, and X2. This will eliminate half of the range.
   a. If X1 > Xi, then b = X2;
   b. If X2 > Xi, then a = Xi;
6.) Check if the new range (a-b) is smaller than the accuracy (eps.) If so, end the search and continue to step 7. If not, repeat steps 2 through 5.
7.) Return the results
   a. Final range: [a b]
   b. Final calculations: [f(a) f(b)]
   c. Number of calculations of f(x) required to reach the answer.

Golden Search Method

Description

Using the "Golden Ratio" (phi) of 1.618 (0.618), two points are selected within the range. These two points represent overlapping portions of the existing range. These two points are then compared to each other to determine which new range the solution is in, thereby eliminating 61.8% of the range for each cycle. The program then repeats on this new narrow interval, until the new range is smaller than the required accuracy.

Program Outline

1.) Specify inputs
   a. f(x) – Objective Function
   b. eps – Accuracy Value
   c. a – Range Start
   d. b – Range End
2.) Determine the initial two points
   a. X1 = b – (b – a)/phi
   b. X2 = a + (b – a)/phi
3.) Change the range, X1, and X2 according to f(X1) and f(X2).
   a. If f(X1) < f(X2)
      i. b = X2
      ii. x2 = X1
      iii. X1 = b – (b – a)/phi
   b. If f(X1) < f(X2)
      i. a = X1
      ii. X1 = X2
      iii. X2 = a + (b – a)/phi
4.) Check if the new range (a-b) is smaller than the accuracy (eps.) If so, end the search and continue to step 5. If not, repeat steps 3 through 4.
5.) Return the results
   a. Final range: [a b]
   b. Final calculations: [f(a) f(b)]
   c. Number of calculations of f(x) required to reach the answer.

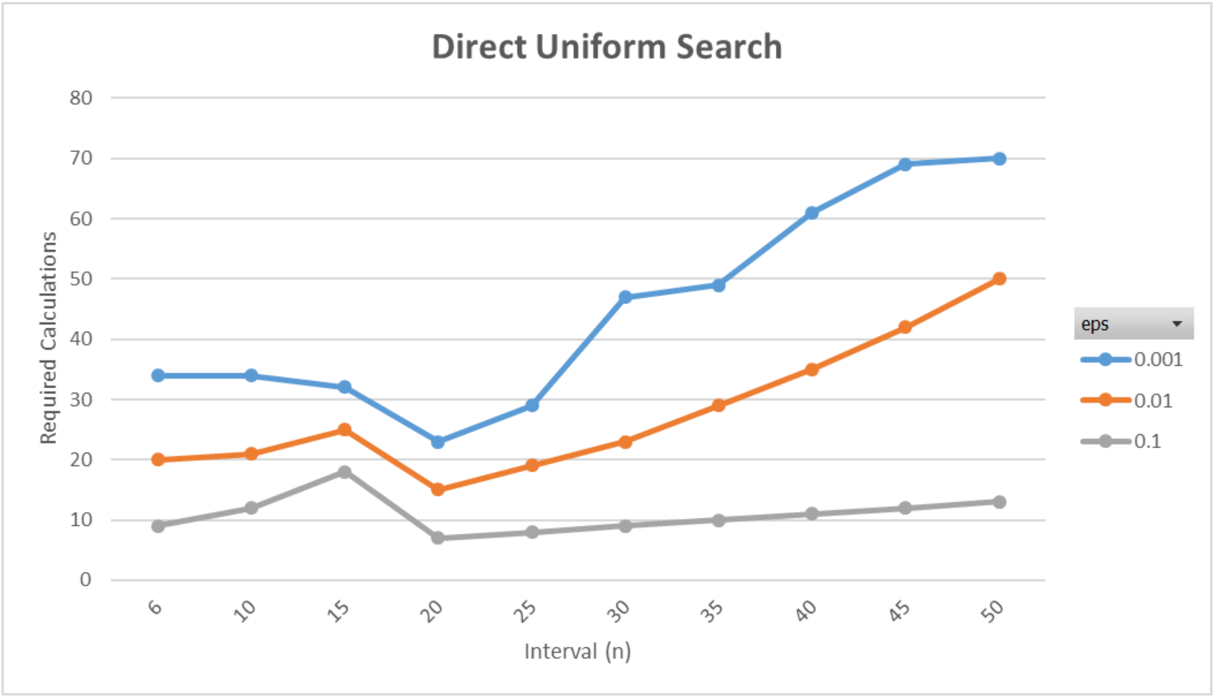
Results

The results for each of the three methods are shown in the following sections. The number of required calculations of f(x) to find the solution is used as the comparison.

Direct Search Method

Unlike the other methods (Dichotomy and Golden), the Direct Search method has an additional variable of "number of intervals" (n). As such it brings to question how this additional variable affects the efficiency of the algorithm. To accurately display this affect, many computations were completed with combinations of accuracy (eps) and interval (n).

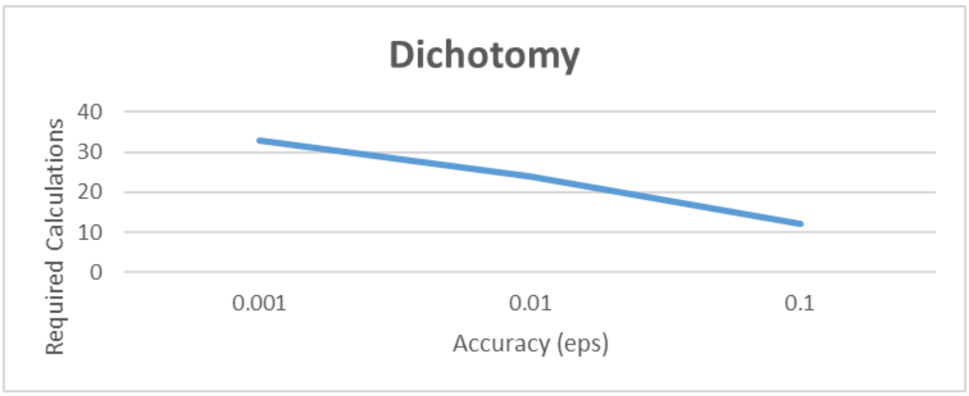
It can be seen in the below chart that, there is an optimal value for the interval parameter around n=20. Beyond this point the required number of calculations to achieve the solution quickly increases. Sample data for n = 10 and 20 is included below.



Orig A	Orig B	n	eps	Final A	Final B	f(a)	f(b)	Calcs
0.2	1	10	0.1	0.36	0.39	-3.74	-3.75	12
0.2	1	20	0.1	0.36	0.44	-3.74	-3.71	7
0.2	1	10	0.01	0.379	0.386	-3.749	-3.749	21
0.2	1	20	0.01	0.380	0.388	-3.749	-3.749	15
0.2	1	10	0.001	0.3821	0.3824	-3.7491	-3.7491	34
0.2	1	20	0.001	0.3820	0.3828	-3.7491	-3.7491	23

Dichotomy Method

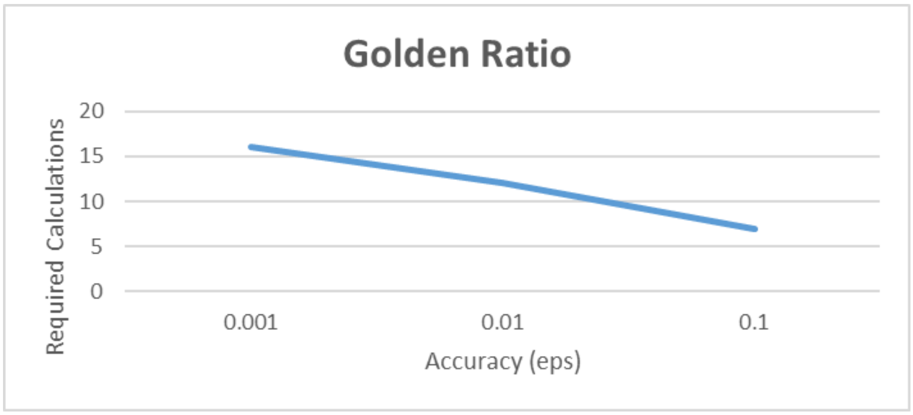
The below graph shows that the number of required calculations to find the solution as the accuracy is modified. The data table for this graph is included below.



Original A	Original B	eps	Final A	Final B	f(a)	f(b)	Calculations
0.2	1	0.1	0.30	0.4	-3.73	-3.74	12
0.2	1	0.01	0.381	0.388	-3.749	-3.749	24
0.2	1	0.001	0.3820	0.3828	-3.7491	-3.7491	33

Golden Search Method

The below graph shows that the number of required calculations to find the solution as the accuracy is modified. The data table for this graph is included below.



Original A	Original B	eps	Final A	Final B	f(a)	f(b)	Calculations
0.2	1	0.1	0.36	0.43	-3.74	-3.72	7
0.2	1	0.01	0.378	0.385	-3.749	-3.749	12
0.2	1	0.001	0.3818	0.3827	-3.7491	-3.7491	16

Conclusion

From the results, it can be seen that the minimum point of the objective function is at f(0.382) = 3.749.

From the various methods, the most effective solution is the Golden Section method. This method, required the least number of calculations of f(x) to find the solution.

Below is a table comparing the various methods. The highlighted value is the cell with the least number of calculations for a given accuracy requirement.

Method	**Accuracy (eps)**	**Required Calculations**
Direct Uniform Search*	0.1	7
Dichotomy	0.1	12
Golden Ratio	0.1	7
Direct Uniform Search*	0.01	15
Dichotomy	0.01	24
Golden Ratio	0.01	12
Direct Uniform Search*	0.001	23
Dichotomy	0.001	33
Golden Ratio	0.001	16

* using best interval (n) = 20

Appendix A: C# Program

Main

```
static void Main(string[] args)
{
#region Intro
Console.WriteLine(@"
///////////////////////////////////////
Task: Lab 1 - One Dimensional Minimization  Methods
Written By: Christopher W. Blake
Date: 11 Nov. 2016
///////////////////////////////////////
Description:
Creates 3 different seach algorithms for finding
the minimum of a given function f(x) in a range. It then tests
these results and prints them to the console.

1.) Direct search - compares side by side items, starting
from the beginning of the range.
2.) Dichotomy - Computes the middle point to determin
in which half of the range the minimum point is located.
The range is modified, and this process is repeated until
the range is smaller than the tolerance (eps).
3.) Golden Ratio - This is similar to Dichotomy however
it uses the golden ratio instead of cutting in half.
///////////////////////////////////////
");
#endregion


//Objective function for class      //f(x) = 10*x*ln(x)-x^2/2,  x E[0.2,1]
List<double> epsValues = new List<double> { 0.1, 0.01, 0.001 }; //accuracy
double a = 0.2, b = 1; //range
ObjectiveFunction f = delegate (double x)
{
    double Fx = 10 * x * Math.Log(x) - Math.Pow(x, 2) / 2;
    return Fx;
};


#region Direct Uniform Search
//Open file for output data
StreamWriter outputFile = new StreamWriter(@"..\..\DirectUniformSearch.txt"); //file to
save results

//Show the table header
Console.WriteLine("-----Direct Uniform Search-----");
Console.WriteLine((new SearchResult()).getTableHeader()); // console
outputFile.WriteLine((new SearchResult()).getTableHeader()); //in file

//Loop through all combinations of accuracy and interval
List<int> intervals = new List<int> { 6, 10, 15, 20, 25, 30, 35, 40, 45, 50 };
foreach (double eps in epsValues)
foreach(int n in intervals)
{
    //Calculate results
    SearchResult d = directUniformSearch(f, a, b, n, eps);

    //Display results on console
    Console.WriteLine(d.getTabbedResults());

    //Save results to file
    outputFile.WriteLine(d.getTabbedResults());
}
```

```
//Save the results
outputFile.Close();
#endregion

#region Dichotomy Search
StreamWriter outputFile2 = new StreamWriter(@"..\..\DichotomySearch.txt"); //file to save
results

Console.WriteLine();
Console.WriteLine("-----Dichotomy Search-----");
Console.WriteLine((new SearchResult()).getTableHeader()); // console
outputFile2.WriteLine((new SearchResult()).getTableHeader()); //in file

//Loop through all accuracy options
foreach (double eps in epsValues)
{
    //Calculate results
    SearchResult d = dichotomySearch(f, a, b, eps);

    //Display results on console
    Console.WriteLine(d.getTabbedResults());

    //Save results to file
    outputFile2.WriteLine(d.getTabbedResults());

}

//Save results
outputFile2.Close();
#endregion

#region Golden Ration Search
StreamWriter outputFile3 = new StreamWriter(@"..\..\GoldenRatioSearch.txt"); //file to save
results

Console.WriteLine();
Console.WriteLine("-----Golden Ratio Search-----");
Console.WriteLine((new SearchResult()).getTableHeader()); // console
outputFile3.WriteLine((new SearchResult()).getTableHeader()); //in file

//Loop through all accuracy options
foreach (double eps in epsValues)
{
    //Calculate results
    SearchResult d = goldenRatioSearch(f, a, b, eps);

    //Display results on console
    Console.WriteLine(d.getTabbedResults());

    //Save results to file
    outputFile3.WriteLine(d.getTabbedResults());

}

//Save results
outputFile3.Close();
#endregion

//Wait for user to click to exit
Console.ReadKey();
}
```

Method – Direct Uniform Search

```
public static SearchResult directUniformSearch(ObjectiveFunction f, double rangeStart, double rangeEnd,
int nIntervals, double accuracy)
        {
            // Relationship to variables from classroom
            double a = rangeStart;
            double b = rangeEnd;
            int n = nIntervals;
            double eps = accuracy;

            //Counter
            int fCounter = 0;

            //Loop until a solution is found
            while (true)
            {
                //Determine the step size (h)
                double h = (b-a)/n;

                //For storing previous results
                double[] x = new double[n];
                double[] fX = new double[n];

                //Caclulate first two points
                int i = 0;
                x[i] = a + h * i; //i=0;
                fX[i] = f(x[i]); fCounter++;
                i = 1;
                x[i] = a + h * i; //i=1;
                fX[i] = f(x[i]); fCounter++;

                //Loop through each additional step until f(x) on both sides is larger
                for (i=2; i<n; i++)
                {
                    //Calculate x and f(x)
                    x[i] = a + h*i;
                    fX[i] = f(x[i]); fCounter++;

                    //Check if f(x) is greater on both sides
                    double fxBefore = fX[i-2];
                    double fxMiddle = fX[i-1];
                    double fxAfter = fX[i];
                    if ((fxBefore > fxMiddle) && (fxMiddle < fxAfter))
                    {
                        //Change the interval
                        a = x[i-2];
                        b = x[i];

                        //Check if the accuracy has been achieved. If so, return the result.
                        if (Math.Abs(b - a) < eps)
                        {
                            return new SearchResult
                            {
                                //Inputs
                                rangeStart = rangeStart,
                                rangeEnd = rangeEnd,
                                nIntervals = nIntervals,
                                accuracy = accuracy,

                                //Results
                                a = x[i-2],
                                b = x[i],
                                Fa = fX[i - 2],
                                Fb = fX[i],
                                CalculationsUntilAnswer = fCounter
                            };
                        }
                        //stop this loop.
                        break;
                    }
                }
            }
        }
```

Method – Dichotomy Search

```
public static SearchResult dichotomySearch(ObjectiveFunction f, double rangeStart,
double rangeEnd, double accuracy)
        {
            //Relationship to variables from classroom
            double a = rangeStart;
            double b = rangeEnd;
            double eps = accuracy;

            //Counter
            int fCounter = 0;

            //Variables for processing
            double x1, x, x2;
            double f1, fX, f2;

            //Loop until solution found
            while (true)
            {
                //Calculate delta
                double delta = Math.Abs(a-b) * 0.01;

                //Create X values
                x = (a+b)/2;
                x1 = x - delta;
                x2 = x + delta;

                //Calculate Y values
                f1 = f(x1); fCounter++;
                fX = f(x); fCounter++;
                f2 = f(x2); fCounter++;

                //Check if solution found
                if (Math.Abs(b - a) < eps)
                {
                    //Return the current values
                    return new SearchResult
                    {
                        //Inputs
                        rangeStart = rangeStart,
                        rangeEnd = rangeEnd,
                        nIntervals = 0,
                        accuracy = accuracy,

                        //Results
                        a = a,
                        b = b,
                        Fa = f1,
                        Fb = f2,
                        CalculationsUntilAnswer = fCounter
                    };
                }

                //Change the range and repeat
                if (f1 > fX)
                    a = x;
                else if (f2 > fX)
                    b = x;
            }
        }
```

Method – Golden Ratio Search

```
public static SearchResult goldenRatioSearch(ObjectiveFunction f, double rangeStart, double
rangeEnd, double accuracy)
        {   // https://en.wikipedia.org/wiki/Golden_section_search
            // Relationship to variables from classroom
            double a = rangeStart; double b = rangeEnd; double eps = accuracy;

            //For storing results
            int fCounter = 0;
            Dictionary<double, double> fX = new Dictionary<double, double>();

            //Calculate probe points (explained on website)
            double phi = (1 + Math.Sqrt(5)) / 2.0; //golden ratio
            double x1 = b - (b - a) / phi; // (b-a) = distance between x2 and x4
            double x2 = a + (b - a) / phi;

            //Calculate first 2 points
            fX.Add(x1, f(x1)); fCounter++;
            fX.Add(x2, f(x2)); fCounter++;

            //loop until results found
            while (Math.Abs(b - a) > eps)
            {
                //Modify the range
                if (fX[x1] < fX[x2])
                {
                    //Change range
                    b = x2;

                    //Set new probe point
                    x2 = x1;
                    x1 = b - (b - a) / phi;

                    //Calculate at new x1
                    fX.Add(x1, f(x1)); fCounter++;
                }
                else
                {
                    //Change range
                    a = x1;

                    //Set new probe point
                    x1 = x2;
                    x2 = a + (b - a) / phi;

                    //Calculate at new x2
                    fX.Add(x2, f(x2)); fCounter++;
                }
            }
            //Return the current values
            return new SearchResult
            {
                //Inputs
                rangeStart = rangeStart,
                rangeEnd = rangeEnd,
                nIntervals = 0, //not used by this method
                accuracy = accuracy,
                //Results
                a = a,
                b = b,
                Fa = f(a),
                Fb = f(b),
                CalculationsUntilAnswer = fCounter
            };
        }
```

Class – SearchResult

```
public class SearchResult
        {
            //Input parameters
            public double rangeStart;
            public double rangeEnd;
            public int nIntervals;
            public double accuracy;

            //X
            public double a;
            public double b;

            //F(x)
            public double Fa;
            public double Fb;
            public int CalculationsUntilAnswer;

            //Methods
            public string getTableHeader()
            {
                string s = "";
                //Input parameters
                s += "origA";
                s += "\t" + "origB";
                s += "\t" + "n";
                s += "\t" + "eps";

                //Results
                s += "\t" + "finalA";
                s += "\t" + "finalB";
                s += "\t" + "f(a)";
                s += "\t" + "f(b)";
                s += "\t" + "calcs";

                return s;
            }
            public string getTabbedResults()
            {
                //determine number of decimal places to show
                int decPlaces =
BitConverter.GetBytes(decimal.GetBits((decimal)accuracy)[3])[2] + 1;

                string s = "";
                //Input parameters
                s += rangeStart;
                s += "\t" + rangeEnd;
                s += "\t" + nIntervals;
                s += "\t" + accuracy;

                //Results
                s += "\t" + a.ToString("F"+ decPlaces);
                s += "\t" + b.ToString("F" + decPlaces);
                s += "\t" + Fa.ToString("F" + decPlaces);
                s += "\t" + Fb.ToString("F" + decPlaces);
                s += "\t" + CalculationsUntilAnswer;

                return s;
            }
        }
```