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 3

Penalty Functions
Discipline: Methods of Optimization
6 December 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 is used for locating the point of minimization. However, this objective function is also constrained to a certain allowable answer range, by means of the following constraint inequalities. Hence, a new modified objective function combined with penalty functions is developed. To determine this minimization point, a previously develop minimization method of gradient descent is utilized.

Objective Function:

$$f(x_1, x_2) = (1 - x_1)^2 - 10(x_2 - x_1^2)^2 + x_1^2 - 2x_1x_2 + e^{(-x_1 - x_2)}$$

Constraints:

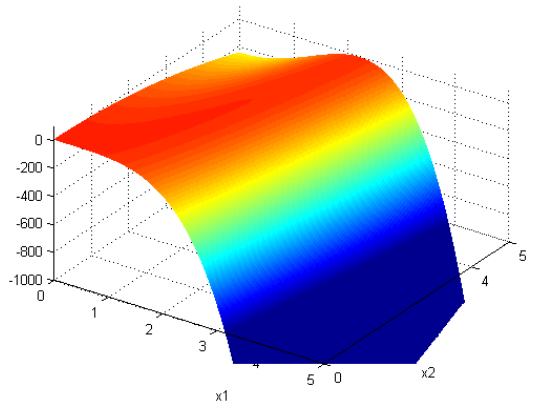
$$x_1^2 + x_2^2 \leq 16$$

$$(x_2 - x_1)^2 + x_1 \leq 6$$

$$x_1 + x_2 \geq 2$$

Convexity

Before search methods were be performed, the unconstrained objective function was graphically and analytically tested for convexity. It can be seen that the objective function is **not convex**. This is graphically clear in the right image. Additionally the second gradient for both the x1 and x2 variables is not always positive. This, by definition, states that the objective function is **not convex.**



First Order Gradient

Respect to x1

$$\nabla f(x_{1,}x_2) = e^{-x_1 - x_2} - 10(x_2 - x_1^2)^2 + x_1^2 - 2x_2x_1 + (1 - x_1)^2$$

Respect to x2

$$\nabla f(x_{1,}x_2) = e^{-x_1 - x_2} - 20(x_2 - x_1^2) - 2x_1$$

Second Order Gradient

Respect to x1

$$\nabla^2 f(x_{1,}x_2) = e^{-x_1 - x_2} - 80x_1^2 + 40(x_2 - x_1^2) + 4$$

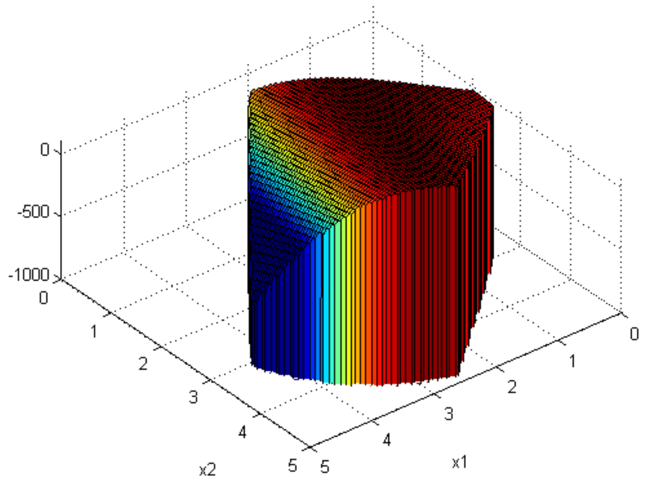
Respect to x2

$$\nabla^2 f(x_{1,}x_2) = e^{-x_1 - x_2} - 20$$

$$\boldsymbol{\nabla^2 f(x) \nsucc 0} \quad \textbf{therefore} \quad \boldsymbol{f(x)} \quad \textbf{is not convex}$$

The constrained objective function was again graphically tested. However, this also shows that the constrained objective function is **not convex**.

There is clearly a minimum in the area of position x≈(4,3) as well as a minimum in the area of position x≈(0.5,3).



Penalty Function Method

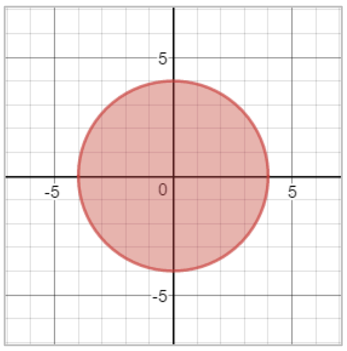
Description

A second order penalty function is selected to control the range of possible results. This is to ensure a second derivative. This penalty function is combined with the constraints and added to the original objective function. By doing so, when a combination of x1 and x2 is selected outside of the constrained area, the value will be unrealistically increased, guaranteeing the position not to be a minimum.

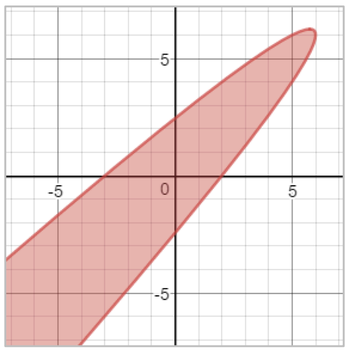
$$penalty(t) = \begin{cases} 0, & t < 0 \\ 100000 * t^2, & t \geq 0 \end{cases}$$

Below are the constraints and how each graphically reduces the target area.

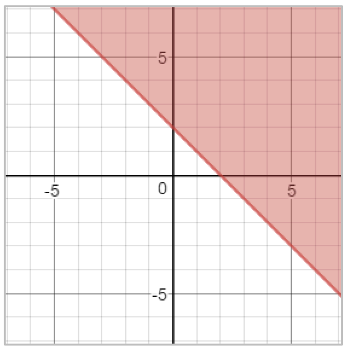
$\boldsymbol{x_1^2 + x_2^2 \leq 16}$



$\boldsymbol{(x_2 - x_1)^2 + x_1 \leq 6}$



$\boldsymbol{x_1 + x_2 \geq 2}$



Combined Objective

The above constraints are now added to the original objective function.

$$f(x_1, x_2) = (1 - x_1)^2 - 10(x_2 - x_1^2)^2 + x_1^2 - 2x_1x_2 + e^{(-x_1 - x_2)}$$

$$+ penalty(x_1^2 + x_2^2 - 16)$$

$$+ penalty((x_2 - x_1)^2 + x_1 - 6)$$

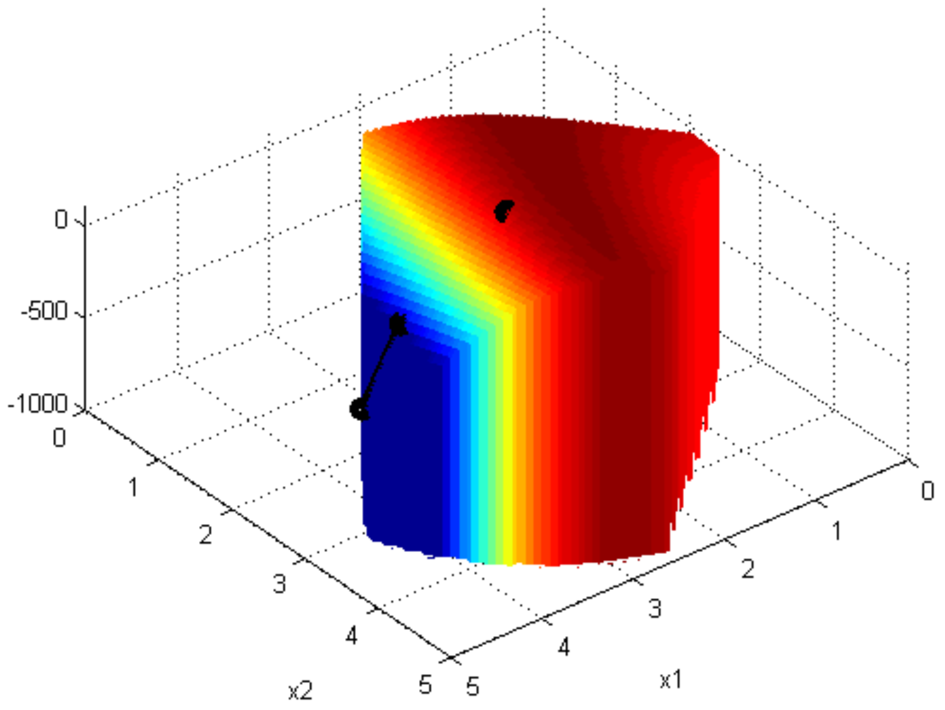
$$+ penalty(2 - x_1 - x_2)$$

Results

The results for two different minimization points are shown. This is required because, as earlier proven, the function is not convex. A graphical depiction of the steps taken during a gradient descent, as well as a table performing the result for different accuracy requirements are shown below.

Minimization Point 1

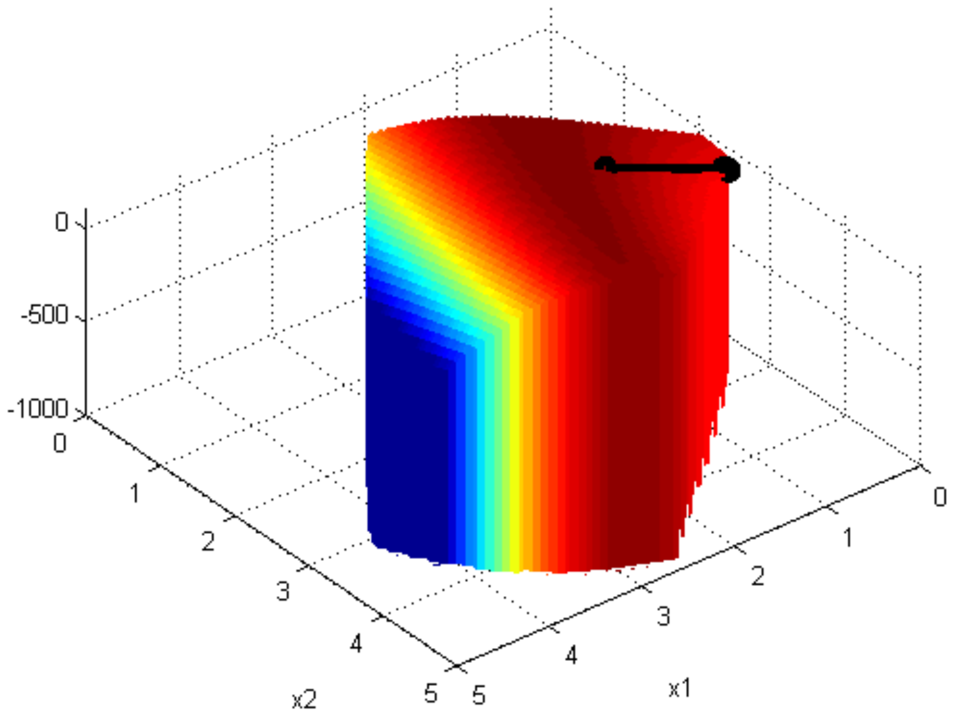
A start point was picked slightly on the low side of the main hill. This is to ensure that the minimization point goes toward the global minimum. It can be seen that a standard gradient descent algorithm was able to locate this minimization point in 3 steps.



Accuracy	X1	X2	f(x)	Calcs f(x)	Calcs f'(x)	Total Calcs
0.1	3.32	1.75	-851.69	25	3	28
0.01	3.348	1.718	-851.167	40	3	43
0.001	3.3465	1.7180	-893.6656	55	3	58

Minimization Point 2

A start point was picked slightly on the low side of the smaller hill. This is to ensure that the minimization point goes toward the local minimum. It can be seen that a standard gradient descent algorithm was able to locate this minimization point in just 2 steps.



Accuracy	X1	X2	f(x)	Calcs f(x)	Calcs f'(x)	Total Calcs
0.1	0.15	2.58	1100.11	17	2	19
0.01	0.131	2.555	5.937	40	3	43
0.001	0.1285	2.5519	-61.5846	55	3	58

Conclusion

From the results, it can be seen that the objective function has two minimization points, a global minimum and a local minimum, when restricted by a series of constraints. The local minimum is at position (0.129, 2.55) and the global minimum is at position (3.347, 1.718).

Additionally adding constraints to the system was easily achievable by use of penalty functions. By use of these penalty functions, disallowed values for inputs x1 and x2 cause drastic increases in the value of f(x), thereby effectively constraining the possible result area.

Christopher W. Blake
December 6, 2016

Appendix A: C# Program

Main

```
static void Main(string[] args)
        {
            //Required accuracy values
            List<double> epsValues = new List<double> { 0.1, 0.01, 0.001 }; //accuracy

            //Functions
            Func<DV, D> objFunc = delegate (DV x)
            {
                D x1 = x[0];
                D x2 = x[1];

                //(1-x1)^2 - 10(x2-x1^2)^2 + x1^2 - 2x1*x2 + exp(-x1-x2)
                return AD.Pow(1.0 - x1, 2) //(1-x1)^2
                        - 10*AD.Pow(x2 - AD.Pow(x1, 2), 2) // - 10(x2-x1^2)^2
                        + AD.Pow(x1, 2) // + x1^2
                        - 2*x1*x2 // - 2x1*x2
                        + AD.Exp(-x1 - x2); //exp(-x1 - x2)
            };
            Func<D, D> penalty = delegate (D t)
            {
                if (t < 0)
                    return 0;
                else
                {
                    return 1000000 * AD.Pow(t, 2);
                }
            };
            Func<DV,D> objFunc_penalized = delegate (DV x)
            {
                D x1 = x[0];
                D x2 = x[1];

                //Constraints
                //Example: x >= 1  ----------------------->  1 - x <= 0
                //x1^2 + x2^2 <= 16     --------->  x1^2 + x2^2 - 16 <= 0
                //(x2 - x1)^2 + x1 <= 6 --------->  (x2 - x1) ^ 2 + x1 - 6 <= 0
                //x1 + x2 >= 2          --------->  2 - x1 - x2 <= 0

                //Combine objective function and penalty functions
                return 0
                    + objFunc(x)
                    + penalty(AD.Pow(x1,2) + AD.Pow(x2,2) - 16)
                    + penalty(AD.Pow(x2 - x1, 2) + x1 - 6)
                    + penalty(2 - x1 - x2)
                    ;
            };

            //Get results
            int calcsF;
            int calcsGradient;
            int calcsHessian;
            DV[] xLocations = null;
            double[] fx = null;

            epsValues = new List<double> {0.1, 0.01, 0.001}; //accuracy
            #region 1.)  Penalty Function, First Order, One-Dimensional Method

            //Show the table header
            Console.WriteLine("----- Gradient Search, First Order, One-Dimensional Method -----");
            Console.WriteLine("       eps        X1        X2        f(x)     Calcs F     Calcs Gr");
            foreach (double eps in epsValues)
            {
                //Perform calculation
                //DV startPoint = new DV(new D[] { 2, 2 });
                DV startPoint = new DV(new D[] { 1, 2 });
                var xMin = Optimization.GradientDescent.FirstOrder_OneDimensionalMethod(objFunc_penalized,
startPoint, eps, out calcsF, out calcsGradient, out xLocations, out fx);

                //determine number of decimal places to show
                int dp = BitConverter.GetBytes(decimal.GetBits((decimal)eps)[3])[2] + 1;

                //Display result on console
                if (xMin != null)
                Console.WriteLine("{0,10}{1,10:F" + dp + "}{2,10:F" + dp + "}{3,12:F" + dp + "}{4,10}{5,10}", eps,
(double)xMin[0], (double)xMin[1], (double)objFunc_penalized(xMin), calcsF, calcsGradient);
            }

            #endregion

            Console.ReadKey();
        }
```