Ministry of Education and Science of the Russian Federation
Peter the Great St. Petersburg State Polytechnic University
Institute of Computer Sciences and Technologies
Graduate School of Cyber-Physical Systems and Control

Report 4

Least Squares Approximation
Discipline: Modern Problems of Informatics and Computer Science
31 March 2017

Student Group: 13541/8

Christopher W. Blake

Hubert Truchan

Professor

Rodionova E.A.

St. Petersburg
2017

Contents

Problem Description – Using C#

A program should be built using a common programming language, such as C#. This program should induce error onto a known objective function and then attempt to solve for the constant of the objective function use Lease Squares Error (MSE) and a minimization method such as gradient descent. (See appendix 1 for C# code.)

The objective function below is modified with random error (+- 2%) to represent noise. The values of "a" and "b" are then hidden and a minimization method is used approximate the values of "a" and "b" from the noisy data points.

$$f(a, b, x) = a * \ln(x) + b$$

Procedure

1.) **Choose concrete values of "a" and "b".**
a = 3.000
b = 8.000

2.) **Generate exact values for 100 points of "x". Store these in an array.**
origX[] = 0 to 100
origY[] = f(3,8, x)

3.) **Create a new array for noisy results, which is within 2%.**
noisyY[] = origY[] * random(0.98, 1.02)

4.) **Define the mean square error (MSE) equation.**
$$MSE = \sqrt{\sum_{x=0}^{100} (f(a, b, x) - noisyY[x])^2}$$

5.) **Process the MSE equation through gradient descent. "a" and "b" are variables.**
aFit = best approximation for "a" fitting through the noisy data.
bFit = best approximation for "b" fitting through the noisy data.

Solution

The chart, figure 1, shows the original (black line), noisy (grey points), and approximated curve (red points).

The approximated curve is visually similar. However, the percentage error of the constants is around 5%. (See below.)

% error A =
4.37% = (3.131 – 3.0)/3.0*100

% error B =
6.15% = (7.508 – 8.0)/8.0 * 100

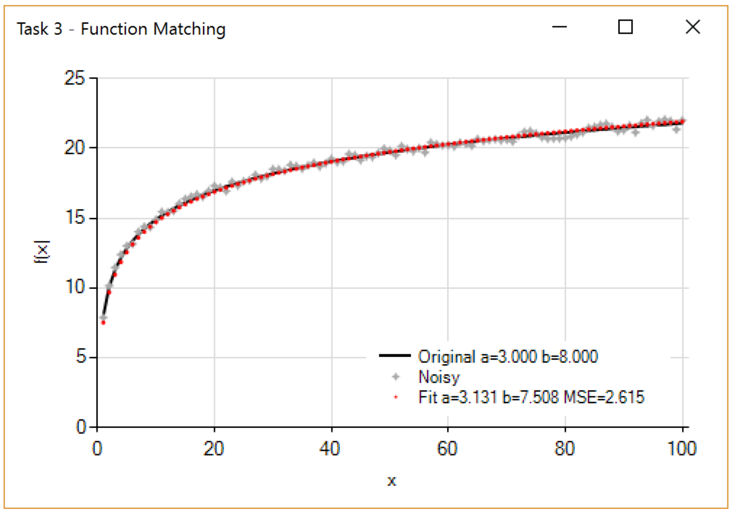


Figure 1: Least Squares fit to using C# program

Verification Description - using OriginPro

For checking purposes, we have decided to use the OriginPro program and its built-in non-linear capabilities. Origin's nonlinear regression method is based on the most widely used algorithm in nonlinear least squares fitting. Origin's nonlinear least squares function fitting is very flexible tool box and provides many useful features. For finding the minimum square error, it uses a gradient descent method. However, like many minimization algorithms, it is possible that only a local minimum will be found, rather than the global minimum.

- Least-squares fitting curve represents the assumed theoretical model.
- Least-squares method minimizes the sum of the squares of the deviations of the experimental given samples.
- The sum of the squares is computed as:

$$RSS = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} w_i\,(y - \hat{y}_i)^2 = \sum_{i=1}^{n} w_i\,[y_i - (\beta_0 + \beta_1 x_i)]^2$$

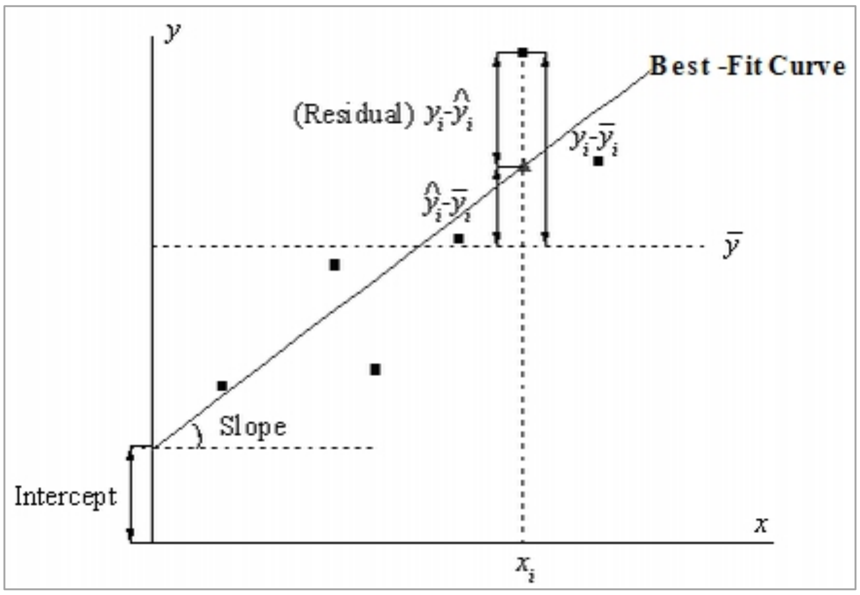


Figure 2: Least squares curve fitting visualization

Solution using OriginPro

The aim was to fit the function:

$$y = 3 * ln(x) + 8 + noise(x)$$

Where noise is:

$$noise(x) = random(0,1) * 0.02 * (3 * ln(x) + 8)$$

Where:

Residual Sum of Squares:	RSS= 1.22946
Degrees of Freedom:	υ=597
Reduced Chi-Squares:	$\chi_v^2 = \frac{RSS}{v} = \frac{1.22946}{597} = 0.00206$
Number of Points:	599

The calculated variables are:

a=8.001, b=3.029

With the standard error:

a $\sigma_i = 0.00492$

b $\sigma_i = 0.0019$

Where $\sigma_i = \sqrt{C_{ii}\chi^2}$

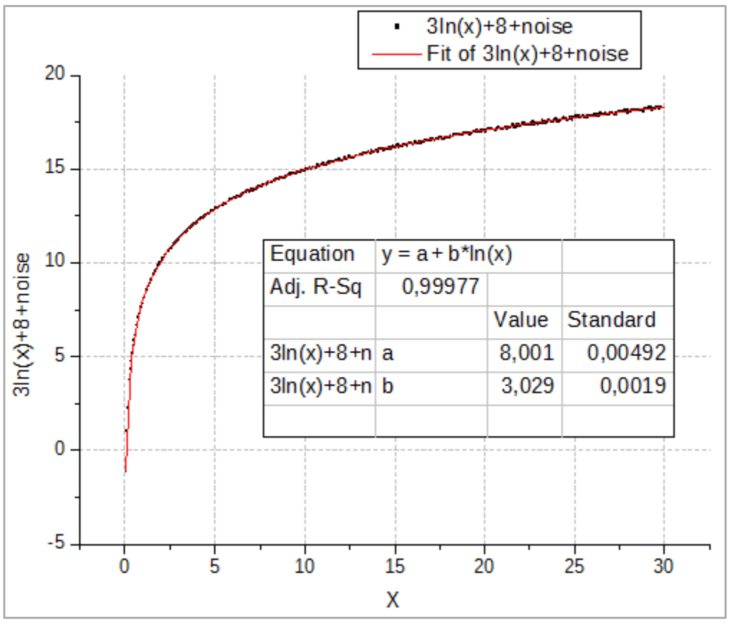
C_{ii}- is the diagonal element of the variance-covariance matrix defined as $C = (F' \otimes F)^{-1}$

Relative error :

a $\sigma = 0.0125\%$

b $\sigma = 0.0967\%$

where $\sigma = \frac{(y-y_i)}{y}$



Equation	y = a + b*ln(x)		
Adj. R-Sq	0,99977		
		Value	Standard
3ln(x)+8+n	a	8,001	0,00492
3ln(x)+8+n	b	3,029	0,0019

Figure 3: Graph of least squares fit using OriginPro.

Figure 4: Residual value of 3ln() + 8 + noise(x)

Appendix 1 – C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Optimization;
using DiffSharp.Interop.Float64;
using System.Windows.Forms.DataVisualization.Charting;

namespace Task3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            //Main objective function
            Func<DV, D> fOBJ = delegate (DV abx)
            {
                // y = a*ln(x) + b
                D a = abx[0];
                D b = abx[1];
                D x = abx[2];

                //Calculate result
                D r = a * AD.Log(x) + b;
                //D r = a * x*x+ b;
                return r;
            };

            //Original Dataset Variables
            const double aOrig = 3.0;
            const double bOrig = 8.0;
            double[] xOrig = new double[100];
            double[] yOrig = new double[100];

            #region Calculate Original data
            //Calculate original dataset
            for(int x=0; x<100; x++)
            {
                xOrig[x] = x+1;
                DV param = new DV(new D[] { aOrig, bOrig, xOrig[x] });
                yOrig[x] = fOBJ(param);
            }

            //Create series
            Series origSeries = new Series("Original")
            {
                ChartType = SeriesChartType.Line,
                Color = Color.Black,
                BorderDashStyle = ChartDashStyle.Solid,
                BorderWidth = 2,
                ChartArea = chartResults.ChartAreas[0].Name,
                LegendText = "Original a=" + aOrig.ToString("F3") + " b=" + bOrig.ToString("F3")

            };
            for (int i = 0; i < xOrig.Length; i++)
                origSeries.Points.AddXY(xOrig[i], yOrig[i]);
            #endregion

            #region Calculate random error version
            double[] yNoisy = new double[100];
            Random rand = new Random();

            //Add noise
            for (int i = 0; i < xOrig.Length; i++)
            {
                double percentError = 0.98 + (1.02 - 0.98)*rand.NextDouble() ;
                yNoisy[i] = yOrig[i] * percentError;
            }

            //Create series
            Series noisySeries = new Series("Noisy")
            {
                ChartType = SeriesChartType.Point,
                Color = Color.DarkGray,
                MarkerStyle = MarkerStyle.Star4,
                MarkerSize = 7,
                BorderWidth = 0,
                ChartArea = chartResults.ChartAreas[0].Name
            };
            for (int i = 0; i < xOrig.Length; i++)
                noisySeries.Points.AddXY(xOrig[i], yNoisy[i]);

            #endregion

            #region Calculate a and b using error function and gradient descent
            //Error equation
            Func<DV, D> fMSE = delegate (DV ab)
            {
                //Get parameters
                D a = ab[0];
                D b = ab[1];

                //Compute squared error for each point
                D errorSum = 0;
                for (int i=0; i<xOrig.Length; i++)
                {
                    //Compute object function value
                    DV param = new DV(new D[] {a, b, xOrig[i]});
                    D yCalc = fOBJ(param);

                    //Check for error
                    if (double.IsNaN(yCalc))
                        continue;

                    //Compute error between noisy version and calculated version
                    D err = AD.Pow(yNoisy[i] - yCalc, 2);
                    errorSum += err;
                }

                //Compute least square
                D mse = AD.Pow(errorSum, 0.5);

                //return results
                return mse;
            };

            //Calculate optimization for a and b
            DV startPoint = new DV(new D[] { 10, 5 });
            int calcsF;
            int calcsGradient;
            DV result = GradientDescent.FirstOrder_OneDimensionalMethod(fMSE, startPoint, 0.01, out calcsF, out 
calcsGradient);//, out calcsHessian);
            double aFit = result[0];
            double bFit = result[1];
            DV paramMseFit = new DV(new D[] { aFit, bFit});
            double mseFit = fMSE(paramMseFit);

            //Create series
            Series fitSeries = new Series("Fit")
            {
                ChartType = SeriesChartType.Point,
                Color = Color.Red,
                MarkerSize = 3,
                MarkerStyle = MarkerStyle.Circle,
                ChartArea = chartResults.ChartAreas[0].Name,
                LegendText = "Fit a=" + aFit.ToString("F3") + " b=" + bFit.ToString("F3") + " MSE=" + 
mseFit.ToString("F3")
            };
            for (int i = 0; i < xOrig.Length; i++)
            {
                DV param = new DV(new D[] { aFit, bFit, xOrig[i] });
                fitSeries.Points.AddXY(xOrig[i], fOBJ(param));
            }
            #endregion

            //Add series to chart
            chartResults.Series.Clear();
            chartResults.Series.Add(origSeries);
            chartResults.Series.Add(noisySeries);
            chartResults.Series.Add(fitSeries);

        }
    }
}
```