Assignment 3: Nonlinear Regression with Matlab

Dhanush Wagle

06/10/2021

Problem 1

Investigate how to do Nonlinear regression with Matlab. There are at least three function that can be used:

lsqcurvefit: least squares parameter fitting, from the optimization toolbox.

fminunc: general unconstrained minimization.

nlinfit: fitting of parameters for nonlinear models, from the Statistics and Machine Learning Toolbox.

Select two of them and use them to find the parameters for the problem solved on the video NonLinearRegression.mp4 (in the video, the Excel solver is used).

Compare with the results obtained in the video.

Start from different initial values for the parameters, and notice how the functions provided in matlab can also be trapped in local minima.

Upload a brief report including name, matlab code, and results.

If you prefer to use Python, you should research and document the appropriate libraries and functions.

==========

The model and data for the problem are included below for reference.

==========

From the book: Regression Analysis, Concepts and Applications. % Franklin A Graybill, Hariharan K. Lyer. 1994.

Chapter 9: Nonlinear regresion. % Example 9.3.1

Model: y = B1 + B2*exp(-B3*x)

y	x
2.86	**0.0**
2.64	**0.0**
1.57	**1.0**
1.24	**1.0**
0.45	**2.0**
1.02	**2.0**
0.65	**3.0**
0.18	**3.0**
0.15	**4.0**
0.01	**4.0**
0.04	**5.0**
0.36	**5.0**

Solution:

$$y_f = B_1 + B_2 e^{-B_3 x}$$

Initial Guesses:

$$B_1 = 1.0$$

$$B_2 = 2.0$$

$$B_3 = 1.0$$

Using initial guesss we get, y as [3, 3, 1.73575888, 1.73575888, 1.27067057, 1.27067057, 1.09957414, 1.09957414, 1.03663128, 1.03663128, 1.01347589, 1.01347589]

For non linear regression we use python library name SciPy. SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. [1]

The sub-package we use in this problem is optimize, and the module used is curve_fit(), which takes three agruments: optimization function $y_f = B_1 + B_2 e^{-B_3 x}$, x and y

Now using, python optimization function we get the values of

$$B_1 = 0.0287529, B_2 = 2.72327684, B_3 = 0.68276193$$

And the new predicted values of y using optimization function is:

y=[2.75202974,2.75202974,1.40460596,1.40460596,0.72386095,0.72386095,0.37993517,0.37993517,0.20617709,0.2

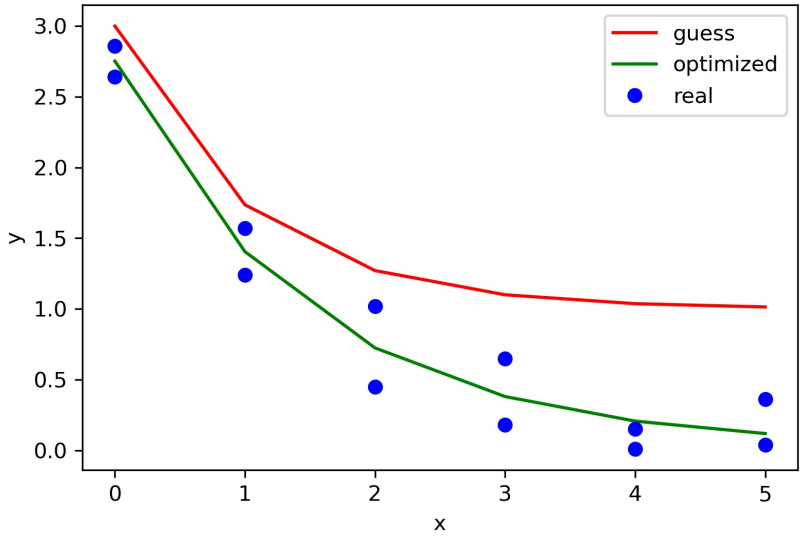

Figure 1: real vs guess vs optimized

References

[1] https://en.wikipedia.org/wiki/SciPy

Appendix

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

y = np.array([2.86,2.64,1.57,1.24,0.45,1.02, 0.65,0.18,0.15,0.01,0.04,0.36])

x = np.array([0.0,0.0,1.0,1.0,2.0,2.0,3.0,3.0,4.0,4.0,5.0,5.0])

def nlr(x,b1,b2,b3):
        return b1+b2*np.exp(-b3*x)
```

```
g = [1.0,2.0,1.0]

n = len(x)
y_pred = np.empty(n)
y_pred1 = np.empty(n)
i,j=0,0

for values in x:
        y_pred[i]=nlr(values,g[0],g[1],g[2])
        i+=1

c,cov = curve_fit(nlr,x,y)
print(c)

for values in x:
        y_pred1[j] = nlr(values,c[0],c[1],c[2])
        j+=1

print(y_pred)
print(y_pred1)

plt.plot(x,y_pred,'r-',label='guess')
plt.plot(x,y_pred1,'g-',label='optimized')
plt.plot(x,y,'bo',label='real')
plt.legend()
plt.show()
```