CS3081 – Computational Mathematics

Tadhg Riordan - 12309240

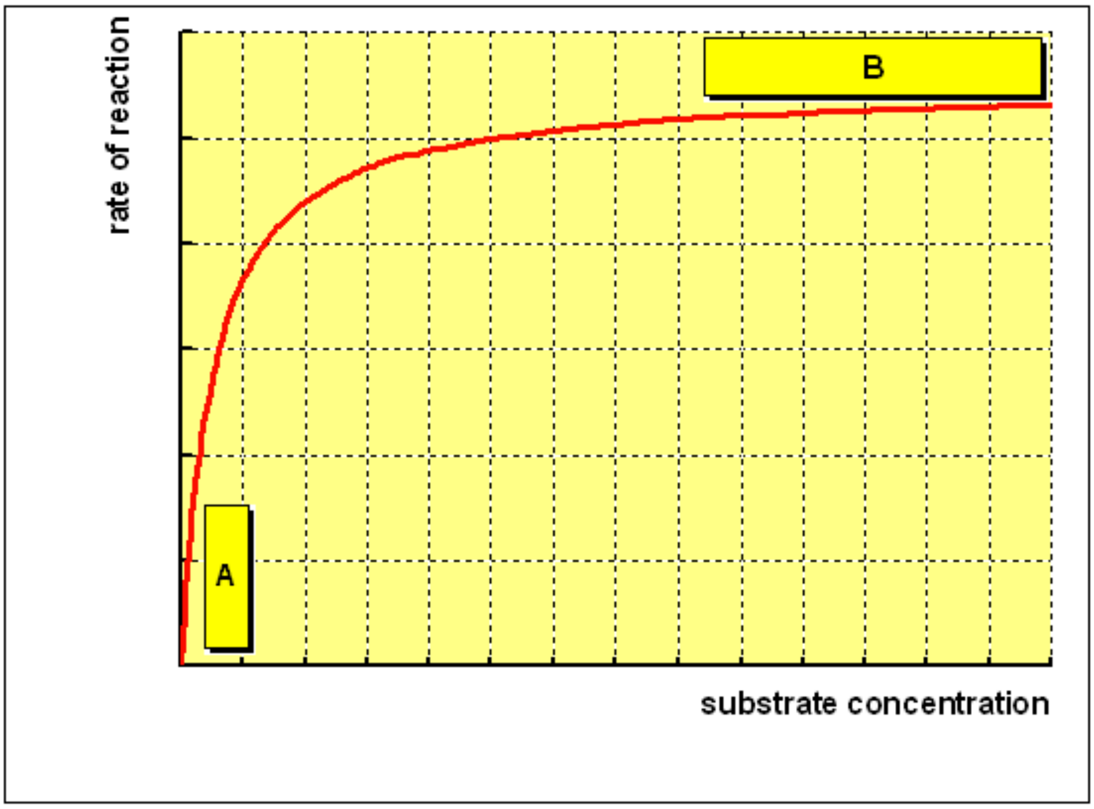
Coursework Assignment

Dr. Fergal Shevlin

Proposed assignment: **“use appropriate methods from SciPy to find an approximate solution for an overdetermined system of equations which are non-linear in the unknowns.”**

I gathered data from a biology experiment, the outcome of which to study the effect of substrate concentration on enzyme activity.

For an enzyme-catalyzed reaction, there is usually a hyperbolic relationship between the rate of reaction and the concentration of substrate. At low concentration of substrate, there is a steep increase in the rate of reaction with increasing substrate concentration. As the concentration of substrate increases, the enzyme becomes saturated with substrate and the curve levels off, as shown in the following graph:



The data gathered for the substrate concentration [*S*]:
[0.038,0.194,425,626,1.253,2.500,3.740]

The data gathered for the rate of reaction:
[0.050,0.127,0.094,0.2122,0.2729,0.2665,0.3317]

The goal here is to use the following model function for the approximation (best fit) of the curve for the data gathered. This model will be used to construct our system of overdetermined equations. There will 7 equations (equating to the number of data points gathered) with two non linear unknowns; Vmax, which is the rate of reaction when the enzyme is saturated with substrate, i.e. the maximum rate of reaction, and Km, the concentration of substrate which permits the enzyme to achieve half of Vmax.

$$\text{rate} = \frac{V_{\max}[S]}{K_M + [S]}$$

I will show how I approximated this curve using the Gauss Newton Method for solving non-linear systems.

The two vectors I must initially calculate are the vector of residuals and from that the Jacobian matrix.
The residual vector is a single column of size equal to the number of equations. Each value is the model function used in terms of its x and y coefficients and our unknowns, now called B1 and B2 for clarity. At each iteration of the Gauss Newton algorithm, the residual vector is recalculated, using the same values for x and y from the gathered data and the current values for B1 and B2. Our residual function is of the form:

$$r_i = y_i - \frac{\beta_1 x_i}{\beta_2 + x_i}$$

The Jacobian matrix is of size (equations * unknowns) where matrix organization is rows by columns. Each row here calculates the partial derivative of the residual function with respect to the unknown at its respective column. It then uses the current values of B1 and B2, as well as the corresponding data values at that position, to calculate a numerical value for that position. The partial derivative of B1 and B2 are,

$$\frac{\partial r_i}{\partial \beta_1} = -\frac{x_i}{\beta_2 + x_i}, \quad \frac{\partial r_i}{\partial \beta_2} = \frac{\beta_1 x_i}{(\beta_2 + x_i)^2}.$$

respectively.

With these vectors defined, we must now define how we will proceed through the method. Gauss Newton iteration is defined as follows:

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} - \left(\mathbf{J_r}^{\mathsf{T}}\mathbf{J_r}\right)^{-1} \mathbf{J_r}^{\mathsf{T}}\mathbf{r}(\boldsymbol{\beta}^{(s)})$$

Where:

$\boldsymbol{\beta}^{(s+1)}$: The next set of values for Beta

$\boldsymbol{\beta}^{(s)}$: The current set of values for Beta

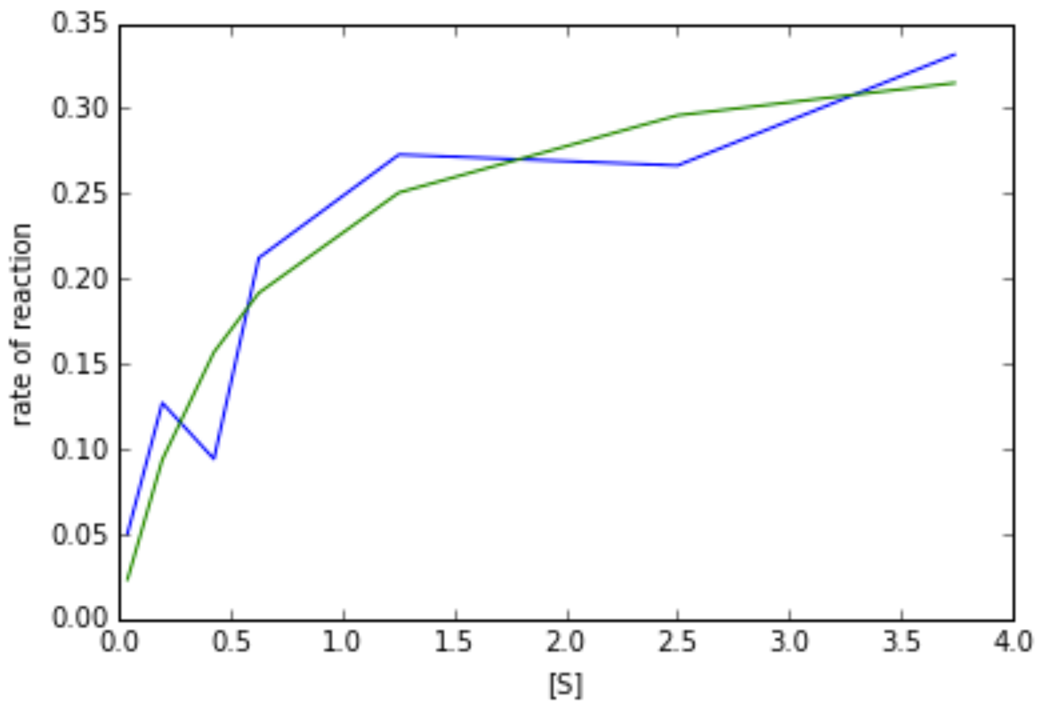
$\mathbf{J_r}^\mathsf{T}$: The transpose of the Jacobian matrix of residuals

$\left(\mathbf{J_r}^\mathsf{T}\mathbf{J_r}\right)^{-1}$:the inverse of the transpose by the Jacobian

$\mathbf{r}(\boldsymbol{\beta}^{(s)})$: the vector of residuals using Beta current

We must choose starting values for the unknowns so as to be able to iterate through the method. With this in place we must set a value for the number of iterations. We want to iterate until the sum of squares of residuals, ie. Each value in the residual vector squared and added, is sufficiently small. After an initial value of 1.44549658151, after 5 iterations this is reduced to 0.0078441826108, a sufficiently small value. Our Beta vector of unknowns now contains it's optimal values. We must now calculate our values for the curve from these optimal values. Our x axis will stay the same and the y axis points will be generated from the model function.

Here is a plot of the output. The blue line is the original data and the green line is our calculated curve:



SOURCES

The information regarding the system meaning was gathered from: http://www.ucl.ac.uk/~ucbcdab/enzass/substrate.htm

The data, application of Gauss Newton method and function images were gathered from Wikipedia

Code

```
import scipy
import numpy as np
import math
import scipy.misc
from numpy.linalg import inv
from matplotlib import pyplot as plt
#from pylab import *
%pylab inline

S = [0.038,0.194,.425,.626,1.253,2.500,3.740]
rates = [0.050,0.127,0.094,0.2122,0.2729,0.2665,0.3317]
iterations = 5 # max iterations
rows = 7 #no. of data points
cols = 2 #no. of unknowns in model function

B = np.matrix([[.9],[.2]]) # original guess for unknowns
Jr = np.zeros((rows,cols)) # Jacobian matrix from r
r = np.zeros((rows,1)) #r equations

def model(Vmax, Km, Sval): # model function
   return ((Vmax * Sval) / (Km + Sval))

def partialDerB1(B2,xi): # partial derivative of residual with respect to B1
   return round(-(xi/(B2+xi)),10)

def partialDerB2(B1,B2,xi): # partial derivative of residual with respect to B2
   return round(((B1*xi)/((B2+xi)*(B2+xi))),10)

def residual(x,y,B1,B2): # residual function
```

```
   return (y - ((B1*x)/(B2+x)))


for i in range(0,iterations): Gauss Newton. iterate "iterations" times

   sumOfResid=0
   #calculate Jr,r and the sum of residuals for this iteration.
   for j in range(0,rows):
      r[j,0] = residual(S[j],rates[j],B[0],B[1])
      sumOfResid = sumOfResid + (r[j,0] * r[j,0])
      Jr[j,0] = partialDerB1(B[1],S[j])
      Jr[j,1] = partialDerB2(B[0],B[1],S[j])

   Jrt =  np.transpose(Jr) # transposition of

   B = B - np.dot(np.dot(inv(np.dot(Jrt,Jr)),Jrt),r) # calc next Beta vector. B(next) = B(this) - inv(Jr^T*Jr)Jr^Tr(B(this)

   print "sum of the squares of the residuals after iteration",(i+1), ":"
   print sumOfResid
   print "Current Beta:"
   print B
#contruct y values for curve from original model function
curveRate = [0]*rows
for i in range(0,rows):
        curveRate[i] = model(B[0,0],B[1,0],S[i])
#plot output
plt.xlabel("[S]")
plt.ylabel("rate of reaction")
plt.plot(S,rates)
plt.plot(S,curveRate)
```