

Fitting a Logistic Curve to Data

David Arnold

February 24, 2002

1 Introduction

This activity is based on an excellent article, *Fitting a Logistic Curve to Data*, by Fabio Cavallini, which appears in the *College Mathematics Journal*, 1993, Volume 24, Number 3, Pages: 247-253. In his article, Dr. Cavallini describes a number of *Mathematica* routines he designed to fit a logistic curve to a given set of data. In this activity, we will design Matlab routines to accomplish a similar fitting of the logistic equation to Dr. Cavallini's data set.

2 The Data Set

To quote Dr. Cavallini, "Ecological problems are nowadays of general concern. In particular, in Italy much attention is being paid to the problem of algal blooms in the Adriatic Sea and this has led also to an increase of interest in mathematical ecology at all levels, from high school teaching to advanced research. The logistic differential equation, dealt with in the next section, is a classical but still useful model for describing the dynamics of a one-species population in an environment with limited resources. For example, the data in Table 1 represent the time evolution of an algal sample taken in the Adriatic Sea and do seem to follow a logistic curve."

time	11	15	18	23	26	31
biomass	0.00476	0.0105	0.0207	0.0619	0.337	0.74
time	39	44	54	64	74	
biomass	1.7	2.45	3.5	4.5	5.09	

Table 1: Measured data. Time is expressed in days and biomass is expressed in mm^2 , since what is actually measured is the surface covered by biomass in a microscope sample.

We begin by obtaining a plot of the data. First, enter the data in Matlab.

```
T=[11,15,18,23,26,31,39,44,54,64,74]
```

```
M=[0.00476,0.0105,0.0207,0.0619,0.337,0.74,1.7,2.45,3.5,4.5,5.09]
```

For upcoming calculations, we need to insure that our data is entered in column vectors. This is accomplished with Matlab's transpose operator.

```
T=T' ;
```

```
M=M' ;
```

Next, plot the data as discrete points with the following command.

```
plot(T,M,'o')
```

A title and axis labels add a professional touch.

```
title('Time evolution of algal sample.')
xlabel('Time (days)')
ylabel('Biomass (mm^2)')
```

These commands will create an image similar to that shown in Figure 1.

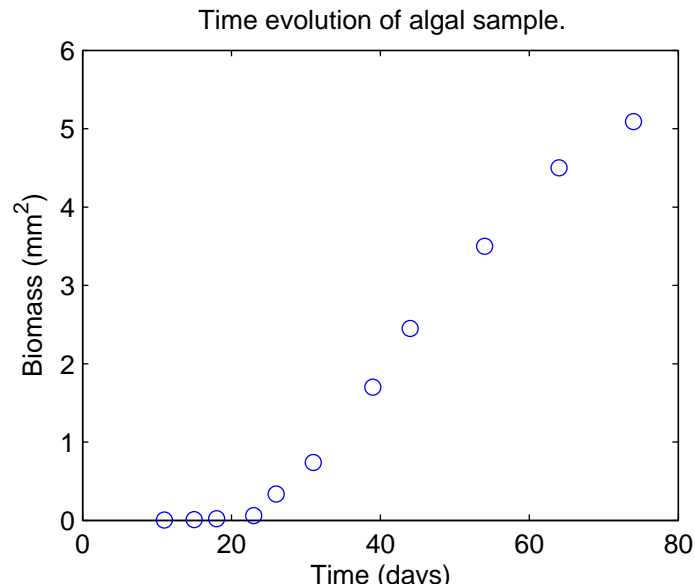


Figure 1: Plot of the data listed in Table 1

3 The Logistic Equation

Dr. Cavallini uses the *logistic equation* in the form

$$\frac{dm}{dt} = rm \left(1 - \frac{m}{K}\right), \quad (1)$$

where t is time, $m = m(t)$ is the biomass, and r and K are positive parameters. Using separation of variables, it can be shown that the solution of the logistic equation is

$$m(t) = \frac{K}{1 + Ce^{-rt}}, \quad (2)$$

where C is an arbitrary constant. It is interesting to allow Matlab to provide a solution.

```
syms m r k t
m=dsolve('Dm=r*m*(1-m/K)', 't')
```

Matlab responds with the following solution.

m =

$K/(1+\exp(-r*t)*C1*K)$

Letting $C = KC_1$ in this result provides equation (2).

A somewhat tricky calculation provides the second derivative of m with respect to t .

$$m''(t) = \frac{CKr^2e^{rt}(C - e^{rt})}{(C + e^{rt})^3}. \quad (3)$$

Of course, you can also allow Matlab to do this calculation for you.

```
m2=diff(m,t,2)
```

m2 =

$2*K^3/(1+\exp(-r*t)*C1*K)^3*r^2*\exp(-r*t)^2*C1^2-K^2/(1+\exp(-r*t)*C1*K)^2*r^2*\exp(-r*t)*C1$

Of course, you will want to simplify this result.

```
m2=simple(m2)
```

m2 =

$K^2*r^2*\exp(-r*t)*C1*(\exp(-r*t)*C1*K-1)/(1+\exp(-r*t)*C1*K)^3$

A little algebraic manipulation (let $C = KC_1$ and multiply numerator and denominator by e^{3rt}) will reveal that this is identical to equation (3).

The graph of $m = m(t)$ has a point of inflection when the second derivative in equation (3) is zero. The second derivative provided by equation (3) is zero when its numerator is zero; that is, when $C = e^{rt}$. So, if we put $C = e^{rt_0}$, where t_0 is the time when the point of inflection occurs, then the solution (2) become

$$m(t) = \frac{K}{1 + e^{-r(t-t_0)}}. \quad (4)$$

Note that the biomass m approaches the *carrying capacity* K as $t \rightarrow \infty$.

4 The Curve Fitting Algorithm

Dr. Cavallini now states: “We now show a procedure for fitting, in the least squares sense, a logistic curve (4) to a given data set (t_i, m_i) for $i = 1, 2, \dots, n$. In symbols, the problem is to minimize, for K, r and t_0 varying in the real line, the error

$$e = \sum_{i=1}^n (m(t_i) - m_i)^2. \quad (5)$$

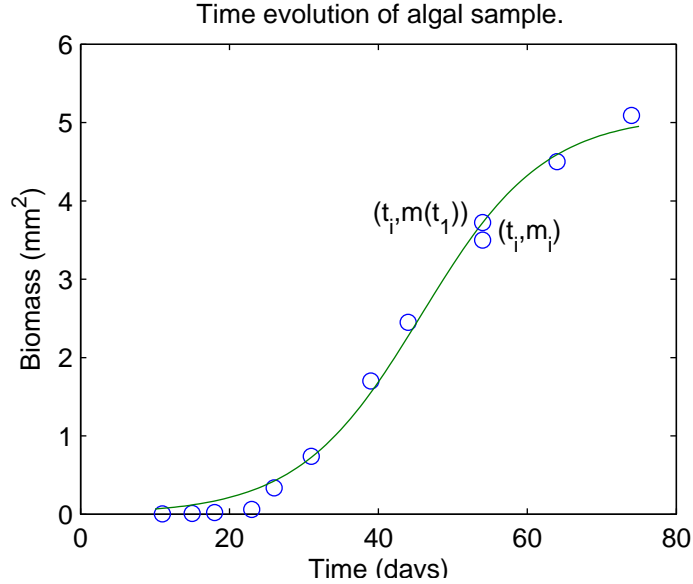


Figure 2: Finding the square of the error.

This warrants some explanation. In Figure 2, we’ve plotted the data set as discrete points and overlaid a potential solution to our curve fitting goal. Note that the coordinates of the given data point are (t_i, m_i) , while the coordinates of the point with the same abscissa on the fitted curve are $(t_i, m(t_i))$. The “error” is $m(t_i) - m_i$. Because some of the data points lie below the fitted curve while others lie above, we square to insure that our error is positive. That is, the squared error made at the time value t_i is

$$(m(t_i) - m_i)^2. \tag{6}$$

Thus, the total squared error in fitting the curve to our n data points is given by equation (5). Clearly, the object of the game is to *minimize* the total least squared error presented by equation (5). This is why we say that we are fitting a logistic to the data set in a “least squares sense.”

4.1 Eliminating a Parameter

The difficulty inherent in solving this least squares problem is evident in the nonlinear equation (4). There are three parameters. Most numerical optimization routines require that the user make a guess at the solution before the routine proceeds. If we eliminate K as one of the parameters, then there will remain only two, r and t_0 . As we shall see, with only two parameters, there are some nice Matlab routines that we can apply to find an estimate of the least squares solution.

So, we let

$$m(t) = Kh(t), \tag{7}$$

where

$$h(t) = \frac{1}{1 + e^{-r(t-t_0)}}. \tag{8}$$

4.2 The Power of Linear Algebra

At this point, Dr. Cavallini takes advantage of the power of linear algebra to simplify the calculations. He starts by saying that the error given in equation (5) can be written

$$e = K^2 \langle \mathbf{H}, \mathbf{H} \rangle - 2K \langle \mathbf{H}, \mathbf{M} \rangle + \langle \mathbf{M}, \mathbf{M} \rangle, \quad (9)$$

where \mathbf{H} and \mathbf{M} are the vectors

$$\begin{aligned} \mathbf{H} &= \langle h(t_1), h(t_2), \dots, h(t_n) \rangle, \text{ and} \\ \mathbf{M} &= \langle m_1, m_2, \dots, m_n \rangle. \end{aligned}$$

This statement definitely warrants some explanation, particularly if you haven't taken linear algebra as yet.

First, consider two vectors

$$\begin{aligned} \mathbf{a} &= \langle a_1, a_2, \dots, a_n \rangle, \text{ and} \\ \mathbf{b} &= \langle b_1, b_2, \dots, b_n \rangle. \end{aligned}$$

The *dot product* of \mathbf{a} and \mathbf{b} , written $\langle \mathbf{a}, \mathbf{b} \rangle$, is defined as

$$\langle \mathbf{a}, \mathbf{b} \rangle = a_1 b_1 + a_2 b_2 + \dots + a_n b_n. \quad (10)$$

The dot product possesses a number of useful algebraic properties.

1. The dot product is commutative.

$$\langle \mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{b}, \mathbf{a} \rangle$$

2. A scalar can be moved about according to the following rule.

$$\langle c\mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{a}, c\mathbf{b} \rangle = c \langle \mathbf{a}, \mathbf{b} \rangle$$

3. The dot product is distributive with respect to addition.

$$\langle \mathbf{a}, \mathbf{b} + \mathbf{c} \rangle = \langle \mathbf{a}, \mathbf{b} \rangle + \langle \mathbf{a}, \mathbf{c} \rangle$$

The Pythagorean Theorem holds equally well in n dimensions and we define the length of a vector \mathbf{a} , denoted by $\|\mathbf{a}\|$, by

$$\|\mathbf{a}\|^2 = a_1^2 + a_2^2 + \dots + a_n^2. \quad (11)$$

It is important to note that

$$\|\mathbf{a}\|^2 = \langle \mathbf{a}, \mathbf{a} \rangle. \quad (12)$$

We are now in a position to explain Dr. Cavallini's statement in equation (9). First, note that

$$\begin{aligned} e &= \sum_{i=1}^n (m(t_i) - m_i)^2 \\ &= (m(t_1) - m_1)^2 + \dots + (m(t_n) - m_n)^2 \\ &= (Kh(t_1) - m_1)^2 + \dots + (Kh(t_n) - m_n)^2 \\ &= \|\langle Kh(t_1) - m_1, \dots, Kh(t_n) - m_n \rangle\|^2 \\ &= \|K \langle h(t_1), \dots, h(t_n) \rangle - \langle m_1, \dots, m_n \rangle\|^2 \\ &= \|K\mathbf{H} - \mathbf{M}\|^2. \end{aligned}$$

Now, using equation (11) and the algebraic properties of the dot product, we can write

$$\begin{aligned}
 e &= \|K\mathbf{H} - \mathbf{M}\|^2 \\
 &= \langle K\mathbf{H} - \mathbf{M}, K\mathbf{H} - \mathbf{M} \rangle \\
 &= K^2\langle \mathbf{H}, \mathbf{H} \rangle - K\langle \mathbf{H}, \mathbf{M} \rangle - K\langle \mathbf{M}, \mathbf{H} \rangle + \langle \mathbf{M}, \mathbf{M} \rangle \\
 &= K^2\langle \mathbf{H}, \mathbf{H} \rangle - 2K\langle \mathbf{H}, \mathbf{M} \rangle + \langle \mathbf{M}, \mathbf{M} \rangle
 \end{aligned}$$

4.3 Minimizing

In equation (9), it's important to note that the total squared error e still contains three parameters, K , r , and t_0 . The idea is to adjust these parameters so as to minimize e . In single variable calculus, you find a minimum by taking the first derivative and setting it equal to zero. It is no different in multivariable calculus, the only difference being that we must take partial derivatives with respect to each parameter. In this case, we set $\partial e / \partial K$ equal to zero and solve for K .

$$\begin{aligned}
 \frac{\partial e}{\partial K} &= 0 \\
 2K\langle \mathbf{H}, \mathbf{H} \rangle - 2\langle \mathbf{H}, \mathbf{M} \rangle &= 0 \\
 K &= \frac{\langle \mathbf{H}, \mathbf{M} \rangle}{\langle \mathbf{H}, \mathbf{H} \rangle}
 \end{aligned} \tag{13}$$

Now, substitute this result in equation (9) to get

$$e = \langle \mathbf{M}, \mathbf{M} \rangle - \frac{\langle \mathbf{H}, \mathbf{M} \rangle^2}{\langle \mathbf{H}, \mathbf{H} \rangle}. \tag{14}$$

Equation (14) contains just two parameters, r and t_0 , the parameter K being eliminated. This equation, we shall find, is much easier to deal with, due to the reduction in the number of parameters present.

4.4 Writing a Function to Evaluate e

We will now plot the graph e , as defined by equation (14), as a function of r and t_0 . The graph will be a surface in three dimensions and we locate the solution to our least squares error by locating the lowest point on this "error surface." We begin by defining a domain in the rt_0 -plane.

It is a simple matter to determine an interval containing t_0 , as it must lie in the range of the given time data. Thus, considering the data in Table 1, the inflection point occurs at t_0 , where t_0 is some value such that $11 \leq t_0 \leq 74$.

A tougher project is to determine a likely range for the reproductive growth rate r . However, taking the derivative of the function in equation (4),

$$m'(t) = \frac{KRe^{-r(t-t_0)}}{(1 + e^{-r(t-t_0)})^2}. \tag{15}$$

Thus,

$$m'(t_0) = \frac{Kr}{4},$$

or, equivalently,

$$r = \frac{4m'(t_0)}{K}. \quad (16)$$

If we examine the data plotted in Figure 1, it would appear that the point of inflection occurs near the point having t -value $t = 44$, so let's take $t_0 = 44$. We can get a close approximation for the slope at t_0 , that is, $m'(t_0)$, by calculating the slope of the line passing through the points immediately preceding and following the point of inflection, that is, (39, 1.7) and (54, 3.5). Thus,

$$m'(t_0) \approx \frac{3.5 - 1.7}{54 - 39} \approx 0.12. \quad (17)$$

Substitute this result in equation (16), using the largest biomass as an approximation for K , that is $K \approx 5.09$. Thus,

$$r \approx \frac{4(0.12)}{5.09} \approx 0.0943. \quad (18)$$

Thus, as a starting point, let's suppose that r falls somewhere in the range defined by $0.01 \leq r \leq 0.6$. If we find this approximation inadequate, we will adjust and try again.

```
r=linspace(0.01,0.6,40);
t0=linspace(11,74,40);
[r,t0]=meshgrid(r,t0);
```

The `meshgrid` command creates matrices r and t_0 that define a “grid” of points, with r -values running from 0.01 to 0.6 in 40 equal increments, and t_0 values running from 11 to 74 in 40 equal increments. After the command `[r,t0]=meshgrid(r,t0)` both r and t_0 are matrices having 40 rows and 40 columns, each containing 1600 entries.

Next, we need to evaluate e at each point of the grid. We will do this by writing a Matlab function to do the work for us. However, because our intent is to later call one of Matlab's optimization routines to find the minimal e -value on our surface, we must write the function in a very special manner.

```
function e=myerror(x,t,m)
r=x(1);
t0=x(2);
h=1./(1+exp(-r*(t-t0)));
e=m'*m-(h'*m)^2/(h'*h);
```

This routine warrants some additional explanation.

First, note that the inputs to the routine, x , t , and m , are special. The variable x contains a column vector, the first entry of which is r , and the second entry contains t_0 . Note that the first two lines in the function pluck these values from the vector x and store them in the variables r and t_0 , respectively.

Secondly, the variables t and m contain the time and biomass data. Note that these vectors will hold the time and biomass data presented in Table 1.

Next, the *vector* h is computed using equation (8). Note the use of Matlab's array operator `./` as this expression divides 1 by a vector of values. In this manner, we compute $h(t)$ for each time value in Table 1.

The last line of the routine is especially tricky, but not quite as tricky if you know the pertinent fact from linear algebra. That is, if \mathbf{a} and \mathbf{b} are *column vectors*,

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix},$$

then

$$\begin{aligned} \langle \mathbf{a}, \mathbf{b} \rangle &= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \\ &= (a_1 \quad a_2 \quad \cdots \quad a_n) \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \\ &= \mathbf{a}^T \mathbf{b}. \end{aligned}$$

Thus, $\mathbf{m}' * \mathbf{m}$ computes the dot product $\langle \mathbf{m}, \mathbf{m} \rangle$, $\mathbf{h}' * \mathbf{m}$ computes the dot product $\langle \mathbf{h}, \mathbf{m} \rangle$, and $\mathbf{h}' * \mathbf{h}$ computes the dot product $\langle \mathbf{h}, \mathbf{h} \rangle$. Thus, the line $\mathbf{e} = \mathbf{m}' * \mathbf{m} - (\mathbf{h}' * \mathbf{m})^2 / (\mathbf{h}' * \mathbf{h})$ computes the error as defined by equation (14).

4.5 Plotting the Error Surface

We now need to plot the error surface by evaluating the function `myerror` at each point of the grid we defined earlier. We begin by determining the size of the matrices r and t_0 (they are both the same size, so determining the size of either one is sufficient).

```
[m,n]=size(r);
```

Next, prepare a matrix having the same size as matrices r and t_0 to contain the error at each (r, t_0) pair.

```
e=zeros(size(r));
```

Now, run a double loop to evaluate the error at each point in the grid defined by matrices r and t_0 .

```
for i=1:m
    for j=1:n
        e(i,j)=myerror([r(i,j);t0(i,j)],T,M);
    end
end
```

This last piece of code warrants a bit more explanation. Note that we call the function `myerror` by passing it three arguments. The first argument is a column vector containing the current value of r as its first entry and the current value of t_0 as its second entry. Then we pass the time and biomass vectors T and M that contain the data provided in Table 1. The function responds by calculating the error, then places the answer in the appropriate position of the matrix e .

We can now easily plot the error surface.

```
mesh(r,t0,e)
```

We add a title and some appropriate axis labels.

```
xlabel('r')  
ylabel('t_0')  
zlabel('e')  
title('Plotting the error versus the parameters r and t_0')
```

These commands produce the image shown in Figure 3. The solution to the least squares problem

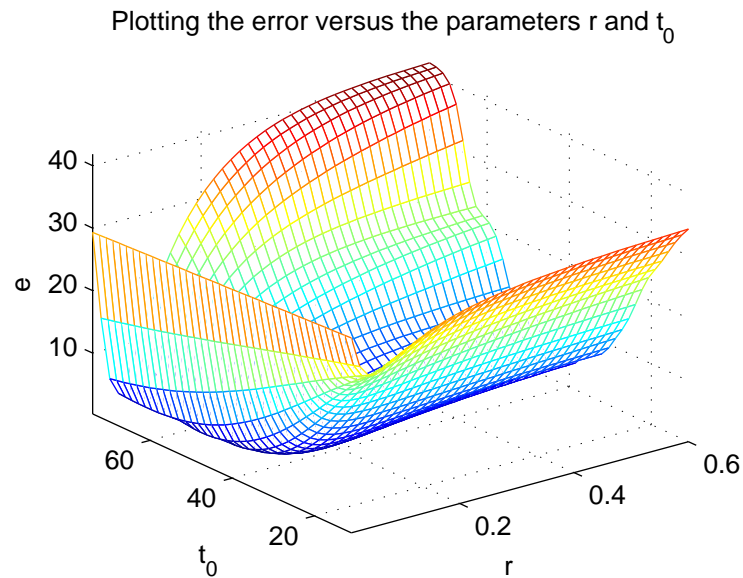


Figure 3: A plot of the “error surface.”

lies in our ability to locate the minimum error on this surface; i.e, the lowest point on this surface on the given domain. This can be approximated by clicking the “rotate” icon in the figure window and examining the surface at different viewing angles by dragging the figure with the mouse.

However, we can get a better indication of where the minimum lies on this surface by crafting a contour plot. This is easy to do in Matlab and requires no further preparation on our part. We feed Matlab’s `contour` command the same data we gave the `mesh` command, but we also pass another parameter which forces the drawing of 40 contours.

```
contour(r,t0,e,40)
```

Again, we add labels and a title.

```
xlabel('r')  
ylabel('t_0')  
title('A contour map of the error versus r and t_0.')
```

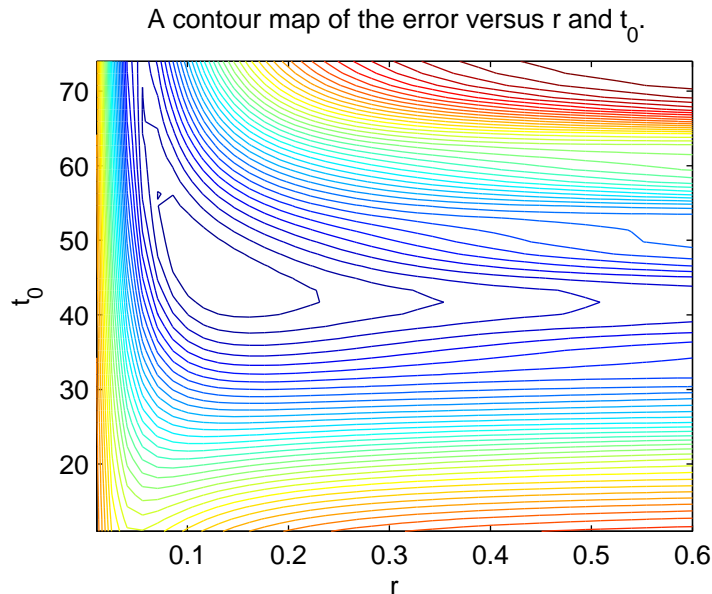


Figure 4: A contour map of the error surface.

These commands were used to craft the image in Figure 4. The contour map argues for the existence of a minimum near the point $(r, t_0) = (0.1, 50)$. You might easily be led to a better approximation by refining the contour map, adding more contours, or specifying values at which you want the contours drawn. Type `help contour` at the Matlab prompt to get a full discussion on the capabilities of Matlab's `contour` command. We only need a rough estimate to use as a starting point to Matlab's sophisticated optimization routines, so we'll settle for $(r, t_0) = (0.1, 50)$.

4.6 Finding the Minimum Error

We will now use Matlab's `fminsearch` command to find the minimal point on the error surface. For a full description of the capabilities of `fminsearch`, type `help fminsearch` at Matlab's prompt. The calling syntax the we will use follows.

```
X = FMINSEARCH(FUN,X0,OPTIONS,P1,P2,...)
```

The arguments to `fminsearch` need some explanation. First, the variable `FUN` contains the name used to define the function to be minimized. In our case, this is `myerror`. Then comes the argument `X0`, a column vector containing the initial guess. In our case, we send the column vector `[0.1;50]`. Next, the variable `OPTIONS` contains a structure containing options sent to the solver. These options are set with Matlab's `optimset` command. On our first effort, we will not use this feature and pass an empty matrix to `OPTIONS`. Finally, we can pass parameters after passing options, and it is our intent to pass the vector containing the time and biomass data from Table 1. That is why we defined the function `myerror` as we did.

There is little to do at this point as all preparation have already been completed. We need only call the optimization routine which will pass back the minimum error in the variable `min`.

```
min=fminsearch(@myerror,[0.1;50],[ ],T,M)
```

```
min =
```

```
    0.1213  
   45.7748
```

Hence, $r = 0.1213$ and $t_0 = 45.7748$ is the location of the minimum error on the error surface.

Finally, we use equation (13) to compute K . First, we pluck r and t_0 from the variable `min`.

```
r=min(1);  
t0=min(2);
```

Next, we compute the vector \mathbf{H} .

```
H=1./(1+exp(-r*(T-t0)));
```

Finally, we compute the carrying capacity K using equation (13).

```
K=(H'*M)/(H'*H)
```

```
K =
```

```
    5.0949
```

4.7 Plotting the Resulting Logistic

All that remains to be done is the plotting of the fitted logistic on the plot containing the time and biomass data from Table 1. This is a simple matter, now that we know that $r = 0.1213$, $t_0 = 45.7748$, and $K = 5.0949$ are the parameter values that will minimize the least square error.

```
t=linspace(10,75);  
y=K./(1+exp(-r*(t-t0)));  
plot(T,M,'o',t,y)
```

We add axis labels and a title.

```
title('Time evolution of algal sample.')
```

```
xlabel('Time (days)')
```

```
ylabel('Biomass (mm2)')
```

These commands produce an image similar to that in Figure 5.

Well, that's a pretty impressive fit!

5 Homework

For homework, use the technique of this activity to fit the logistic equation to the United State population data given in Section 3.2, page 140, Table 2, in *Differential Equations*, Polking, Boggess, and Arnold.

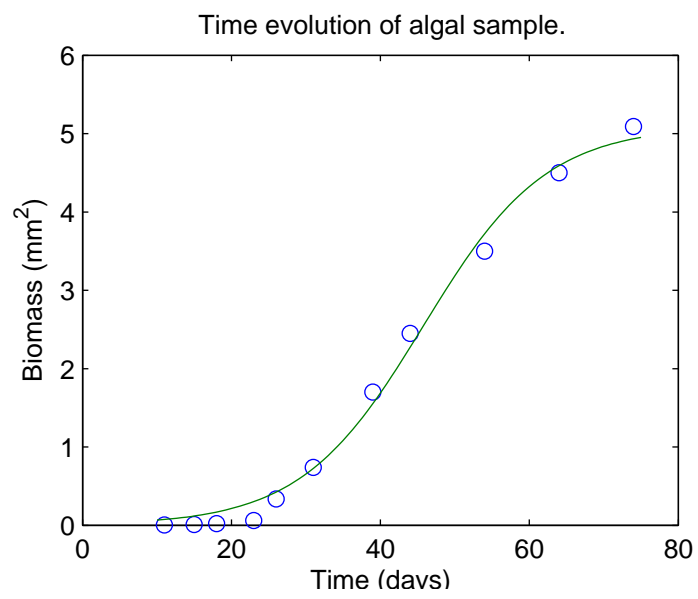


Figure 5: Fitting the logistic.