**Lab 9: Model Fitting**

# Linear Regression for Polynomial Models

As we discussed in class, linear regression can be used to fit more than just linear models. In particular, linear regression can also be used to fit polynomial models.

We will begin this lab by writing a function to fit polynomial models of “arbitrary degree.” That means that the function should be able to fit a line, a parabola, a cubic expression, or higher-order polynomials, depending on which one you ask it to do.

As in previous labs, you have been provided m-files and a mat-file. You should put all of these files in your MATLAB working directory or another location where MATLAB will be able to find them.

Open the provided m-file “fitPolynomial.m”. Edit the function to fill in the sections for generating the independent variable matrix *xMat* and the line for computing the best fit parameter values. You may want to consult the lecture 9 slides if you are unsure of how to build *xMat*. Paste your code below.

Code for generating *xMat*:

for i=1:(degree+1)

xMat(:,i)=x.^i;

end

Code for computing parameters:

beta = xMat\y;

Now let’s test your new function by fitting the following sample data:

>> xvals=[0 1 2 3 4 5];

>> yvals=[3 4.2 4.9 5.3 6.2 7.4];

What parameters values do you find when you fit a 1st order polynomial (a line)?

3.1381

3.9495

4.7610

5.5724

6.3838

7.1952.

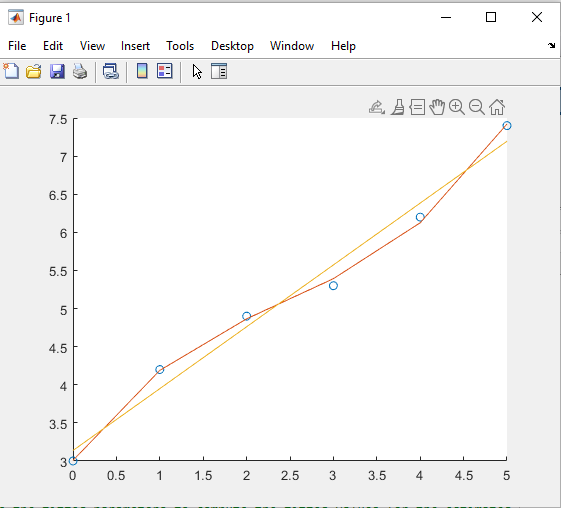
What parameters values do you find when you fit a 3st order polynomial (a cubic curve)?

3.0079 5.3921

4.1889 6.1254

4.8651 7.420

Generate a plot in which you scatter the provided data (*xvals*, *yvals*) and then overlay each of the two above fits. Paste your plot below:



# Fitting an ODE model: Predator-Prey

Many times in studying biology, we have real data and we have a model (or a few models) that we think could accurately describe the process. To assess whether our model is capable of explaining the data, it useful to fit the model parameters.

MATLAB has several built-in functions that we could use to do this. We will explore using the *nlinfit* function to fit parameters for Predator-Prey data. This function performs nonlinear regression to optimize parameter values. It takes as inputs the data (for example *t* and *y* for time series data), the model function, and an initial guess for the parameter values.

Load the experimental time series data from the provided file “PredatorPreyTimeData.mat” with the following command:

>> load('PredatorPreyTimeData.mat','t','yToFit');

OR by dragging the file onto the workspace.

View the time course data by plotting the data:

>> plot(t,yToFit, 'o');

In the above command, what did the third input to the function (‘o’) do?

‘o’ is the format of each data point. If it’s changed to ‘x’, data points appear as x’s.

We will now try to fit the data using the Lotka-Volterra model. Open the provided m-file “predatorPreyForFitting.m”. This is the same m-file I showed you during the lecture on Wednesday, and we can use it to fit the model. Review the code and make sure it makes sense.

How are the **initial conditions** for simulating the timecourse passed into the function?

y(0) is p(5) and p(6).

In terms of the fitting process, what does that mean? More specifically, will *nlinfit* be able to adjust these initial conditions to optimize the fit to the data?

Yes, nlinfit adjusts the parameters to fit the data.

Notice that this function only uses the prey data for fitting. The predator data will be calculated, but it is not included in the function’s output.

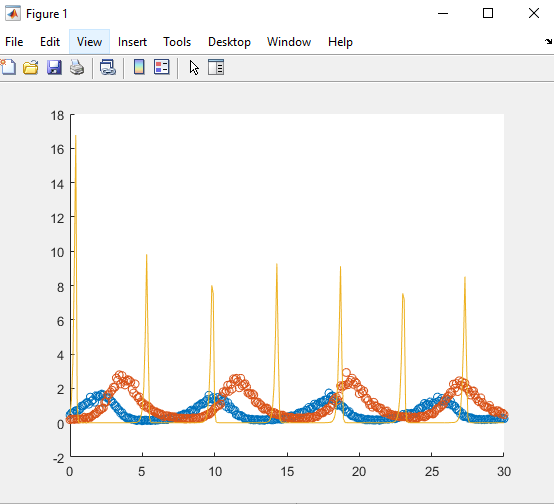
Try fitting the data with the following command:

>> p=nlinfit(t,yToFit(:,1),@predatorPreyForFitting,[10 1 1 2 0.5 0.2])

We can visually inspect to see how well the fit worked. Plot the provided data as you did above, with circle markers for each data point. We will then overlay the fitted model output on top. You can compute the fitted values (or model predictions) using the *predatorPreyForFitting* function and the parameter values stored in *p*. What command should you use to compute the model output?

>> yFit= nlinfit(t,yToFit(:,1),@predatorPreyForFitting

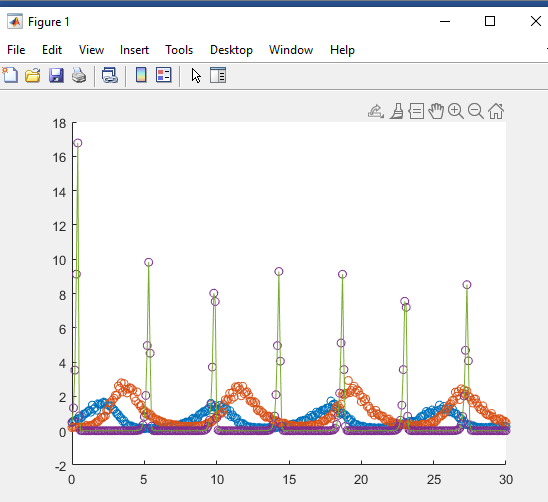
Overlay the fitted values on your plot and paste a screen capture below. Plot the simulated data as a smooth curve (rather than circle markers).



Does it look like a good fit? What do you think might have gone wrong?

No, initial parameters may have been off.

Try the fit again, but this time use [1 1 1 2 0.5 0.2] as your initial parameter guesses. Again, make an overlay plot of the data and the fit. Paste a screen capture below.



Does that fit look better?

Not really.

Even so, you may have gotten a warning about the Jacobian being ill-conditioned. That is a common enough problem with models containing a lot of parameters. We can try to solve the issue by reducing the number of parameters. Here we will assume that we know or can accurately estimate the initial values. To do that, and explore models for this data, open the other provided m-file “fitting\_Predator\_Prey\_models.m”

In Block 1, we will fit the same model but this time **assuming that we know the initial conditions** and fitting only the model parameters.

We can do that by defining a new function “fitFunc” that we will use for fitting. Notice that *fitFunc* takes inputs *p* and *t* (as expected for our model function), and then it calls *predatorPreyForFitting* after appending the initial conditions to *p*. In this way, the initial conditions sent as parameters for *predatorPreyForFitting*, but they are not parameters for *fitFunc* (and therefore they will not be adjusted during the fitting process).

When you run this block, do you get the same values for the first four parameters as you found above? If not, why not?

No, they’re different because the initial conditions sent as parameters for *predatorPreyForFitting*, but they are not parameters for *fitFunc* (and therefore they will not be adjusted during the fitting process).

# Fitting an alternate model

You just fit the experimental data to one possible predator-prey model, but how do you know that that is the best model? It is hard to definitively determine if you have chosen the best possible model. However, it is much more feasible to compare two potential models and assess which one better describes the data.

We will work to answer this question by fitting the same data with a second model. Our second model will be the “updated” predator prey model that you explored in a previous homework assignment. This model adds two extra parameters which describe saturation of the prey population (the carrying capacity for the prey) and saturation of predator’s ability to eat prey.

Use Block 2 to fit the data to the second model using *nlinfit*. Again, we are using a temporary function (this time called *fitFunc2*) to fit only the model parameters and not the initial conditions.

You may need to try a few different initial parameter guesses to make sure that the fitting process worked well (you can adjust just the last two values in p0). Visually inspect the quality of the fit and adjust p0 as needed. When you think you have a good fit, generate a plot of the data with the fit overlaid as you did above. Paste a screen capture below.

# Model Assessment

We discussed in class one approach for assessing the quality of fit of different models that differ in their number of parameters: to evaluate the models on different data from the data used to fit the model.

In this case, we can use the predator data as a test for the model, since our fit was based only on the prey data.

By modifying the two model functions that were provided, create and save two new functions that simulate the same models, but return the predator output, instead of the prey output. You should call them *predatorPreyForFitting2* and *predatorPreyForFittingUpdated2*. Remember that both the name of the function at the top of the file and the filename for your new function should have the new name.

Use Block 3 to simulate each model using the parameters (*p1* and *p2*) that you fit for each model before.

This block will also compute an r2 value for each model’s fit using the *corrcoef* function, and you can use those values to help you assess the quality of the fit.

What were the parameter values and r2 value for the fit with the first model (Lotka-Volterra)?

Parameters: a= p(1);

b= p(2);

c= p(3);

d= p(4);

.

r2: 0.927

What were the parameter values and r2 value for the fit with the second model (with *K* and *M*)?

Parameters: a= p(1);

b= p(2);

c= p(3);

d= p(4);

K= p(5);

M= p(6);

r2: 0.976

The data you fit was actually generated with one of the two models (plus simulated noise). Which model do you think was used to generate the data?

I’d expect Model 2, since it’s a better fit.

Can you think of another method you could use to evaluate which model gives a better description of the data (without generating any more data)?

Cross-validation between training data and testing data.