Lab Assignment #9

Due 4/1/2016 at 4pm on bCourses

The assignment will be partially graded by an autograder which will check the values of the required variables, so it is important that your function names are exactly what they are supposed to be and the input and output arguments are in the correct order (names ARE case-sensitive). Instructions for submitting your assignment are included at the end of this document.

1 Systems of Linear Equations

Write a function with header

```
function [x] = mySols(A,b)
```

to solve the system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a $m \times n$ matrix, $\mathbf{x}$ is a $n \times 1$ vector, and $\mathbf{b}$ is a $m \times 1$ vector.

- If there is no solution, `x` should be empty (defined as `[ ]`) and the function should display: "There is no solution".
- If there is one solution, `x` should contain this solution and the function should display: "There is one solution".
- If there is an infinite number of solutions, `x` should contain one of the solutions and the function should display: "There is an infinite number of solutions". Note that the solution you show can be different than the one shown below in the test case.

The discussion in section 12.6 and figure 12.1 of the book should be helpful.
Test cases:

```
>> x = mySols([1 2 3; 0 3 1; 1 14 7],[1;2;3])
There is no solution

x =

     []
>> x = mySols([1 2 3; 0 3 1; -1 14 7],[1;2;3])
There is one solution

x =

    3.0000
    1.1429
   -1.4286

>> x = mySols([1 2 3 4 5 6; 2 3 4 5 6 7; 3 4 5 6 7 8],[21;27;33])
There is an infinite number of solutions

x =
```

```
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
```

2 Polynomial derivatives

Let $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ be an arbitrary 3^{rd} degree polynomial. If you think about it, the coefficients completely determine this polynomial, and so we can package all relevant information about $f(x)$ in a convenient column vector form $\mathbf{f} = [a_0, a_1, a_2, a_3]^T$. Why might this be convenient? Well, we can represent common operations like differentiation with a simple matrix multiplication, as you'll have a chance to show below.

1. Write a Matlab function with the header
   `function [cubicDf, cubicD] = cubicPolyDiff(f)`
   which differentiates an arbitrary 3^{rd} degree polynomial $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, as represented by the input argument `f = [a0; a1; a2; a3]`. The output `cubicDf` should also be a length 4 column vector, and should be the coefficient vector of the polynomial $f'(x)$. The output `cubicD` should be the differentiation matrix, such that `isequal(cubicD*f, cubicDf)` evaluates to TRUE. Test case:

```
>> f = [1; 2; 3; 4];
>> [cubicDf, cubicD] = cubicPolyDiff(f)

cubicDf =

    2
    6
    12
    0

cubicD =

    0     1     0     0
    0     0     2     0
    0     0     0     3
    0     0     0     0



>> isequal(cubicD*f, cubicDf)

ans =

    1
```

2. Write a MATLAB function with header
`function [Df, D] = polyDiff(f)`
which differentiates an arbitrary nth degree polynomial $f(x) = a_0+a_1x+a_2x^2+...+a_nx^n$, as represented by its coefficient vector, `f = [a_0; a_1; a_2; ...; a_n]` (the length of `f` will be $n+1$). The output `Df` should also be a length $n+1$, and should be the coefficient vector of the polynomial $f'(x)$. The output, `D` should be the differentiation matrix, such that `isequal(D*f, Df)` evaluates to TRUE. Test case:

```
    >> f = [0; 6; 3; 0; 9; 4];

>> [ Df D ] = polyDiff( f )

Df =

     6
     6
     0
    36
    20
     0

D =

     0     1     0     0     0     0
     0     0     2     0     0     0
     0     0     0     3     0     0
     0     0     0     0     4     0
     0     0     0     0     0     5
     0     0     0     0     0     0

>> isequal(D*f, Df)

ans =

    1
```

3. Write a Matlab function with header
`function [Dmf, Dm] = polyDiffm(f, m)`
which finds the mth derivative of the arbitrary nth degree polynomial $f(x) = a_0+a_1x+a_2x^2+...+a_nx^n$, as represented by its coefficient vector, `f = [a_0; a_1; a_2;...; a_n]` (the length of `f` will be $n+1$). The output, `Dmf` should also be a length $n+1$, and should be the coefficient vector of the polynomial $f^m(x)$. The output, `Dm` should be the mth derivative matrix, such that `isequal(Dm*f, Dmf)` evaluates to TRUE. [Hint: use your function `polyDiff` to solve this problem].
Test case:

```
>> f = [0; 6; 3; 0; 9; 4];
>> [Dmf, Dm] = polyDiffm(f, 2)
```

```
Dmf =

     6
     0
   108
    80
     0
     0

Dm =

     0     0     2     0     0     0
     0     0     0     6     0     0
     0     0     0     0    12     0
     0     0     0     0     0    20
     0     0     0     0     0     0
     0     0     0     0     0     0

>> isequal(Dm*f, Dmf)

ans =

    1
```

3 Regression and model selection

In engineering, model selection is the task of selecting a model among a set of candidate models, given a data set. This is really useful for linking a series of observations to a mathematical model predicting these observations.

3.1 Linear law

First, we will consider observations derived from a linear model:

$$y = ax \tag{1}$$

Where x and y are some observable input and output, respectively, and a is a coefficient to be determined from the observations. Write a function with header

```
function [a, RMSE] = myLinearRegression(x,y)
```

that performs linear regression with inputs `x` and `y`, each of which is a column vector of class `double`. The output `a` is a scalar double which represents the best estimate for the coefficient a in Equation 1. `RMSE` is a scalar double which represents the root mean square error (RMSE), which is a measure of how good the fit is (the lower the RMSE, the better) and is calculated as follows:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}$$

where m is the number of points in the data set, and $\hat{y}_i$ is the estimation of y_i (in this case, ax_i). To visualize the performance of your function, you can plot the data points and the estimation results your function outputs (see image below).

You will test out your function with data related to Ohm's law, which is an example of such a linear law: $V = IR$, where V is the voltage (in volts), R the resistance (in ohms) and I the current (in amperes). Using measurements of I and V contained in the `Ohm.mat` file, your function will find the conductance G (the inverse of the resistance) using the data from this example electrical circuit. Your function should work on any set of data in the appropriate format.

```
>> load('Ohm.mat')
>> [G, RMSE] = myLinearRegression(V,I)

G =

    0.0627


RMSE =

   0.0261

>> xvec = 0:0.1:6;
>> yvec = G*xvec;
>> plot(V, I, 'b.', xvec, yvec, 'r-')
>> legend('measured', 'estimated')
```

3.2 General model

In this problem, we will consider observations to be derived from a general model:

$$y = af_1(x) + bf_2(x) + c \tag{2}$$

where the possible models are one of three options:

1. $f_1(x) = \sin(x)$, $f_2(x) = \cos(x)$
2. $f_1(x) = x^2$, $f_2(x) = x$
3. $f_1(x) = \exp(x)$, $f_2(x) = 0$

Write a function with header

```
function [a, b, c, modelNum] = myBestRegression(x,y)
```

where inputs `x` and `y` are column vectors of class `double` of observed data pairs. Your function should estimate `a`, `b`, and `c` from Equation 2 for all three models listed above. Your function should also compute RMSE for the best fit from each of the three models. Finally, your function should select the best model (the one with the lowest RMSE), and return the three coefficients for that model, as well as identifying the best model with the output `modelNum`. `modelNum` is a scalar double with a value of 1, 2, or 3 corresponding to the best

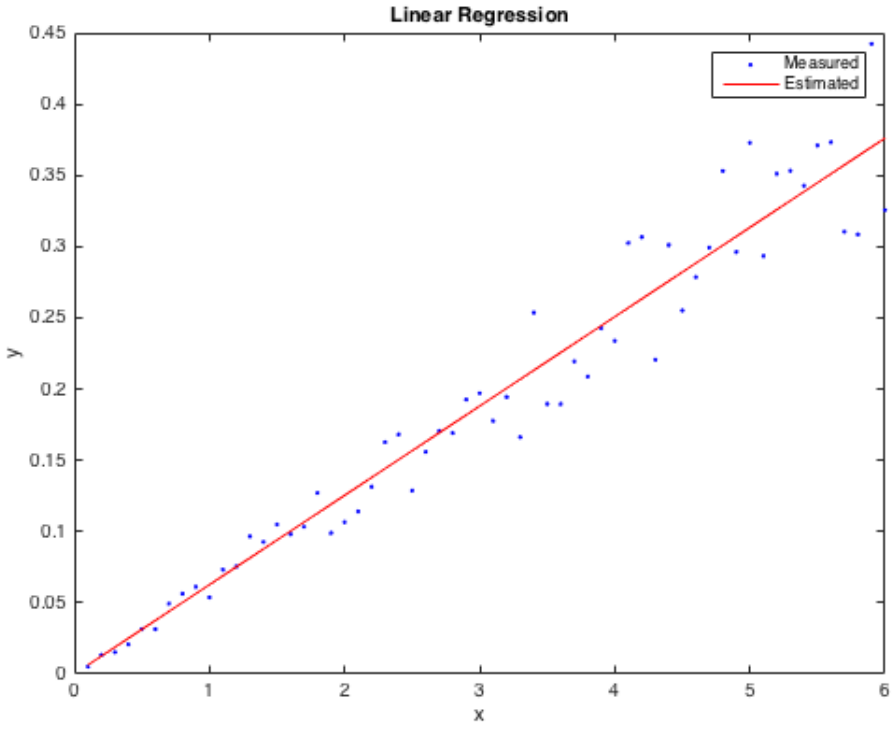

Figure 1: Ohm's law and linear regression

model from the list above. Note that for model 3, there is no coefficient b. If model 3 is the best model, the output `b` should be a scalar double whose value is zero.

Your function should work for any dataset involving two column vectors (`x` and `y`). Two example datasets can be found in the files `Projectile.mat` and `Pendulum.mat` found on bCourses.

`Projectile.mat` contains observations of the position (y) of a projectile along with the time of each observation (t). We know projectile motion is governed by the equation $y(t) = -\frac{1}{2}gt^2 + v_0t + y_0$, and hence your function should be able to determine that model 2 is best, and provide estimates of $-\frac{g}{2}$, v_0, and y_0 (outputs `a`, `b`, and `c`). Note that notation $y = f(x)$ is used to describe the general relationship between model inputs and outputs, even though the variables can have different names (in this example we have $y = f(t)$).

`Pendulum.mat` contains observations of the angle of a pendulum as observed at different points in time ($\theta = f(t)$). We know pendulum motion is governed by the equation $\theta(t) = a\sin(t) + b\sin(t) + \theta_0$, and hence your function should be able to determine that model number 1 is best, as well as provide estimates for a, b, and θ_0.

One example of the third model listed above is the calculation of compound interest. Compound interest at a constant interest rate r provides exponential growth from the initial capital x_0: at a time t, the current value $x(t)$ is given by $x(t) = x_0(1+r)^t = x_0 * e^{t\ln(1+r)}$.

You are not provided with a dataset for this example. You are encouraged to create your own dataset and use it to test your function's ability to fit model 3 above.

```
>> load('Projectile.mat');
>> [a,b,c,modelNum] = myBestRegression(t_proj, y)

a =

   -4.8961

b =

   50.0073

c =

    9.8852

modelNum =

     2

>> load('Pendulum.mat');
>> [a,b,c,modelNum] = myBestRegression(t_pend, theta)

a =

    0.0051

b =

    2.9946

c =

    0.0263

modelNum =

     1
```

4 Streamflow recession analysis and over-determined systems

Perhaps the most convenient feature in MATLAB is the backslash operator, ' \'. In this problem, we will use the backslash operator to solve an over-determined system and fit a well known model to some streamflow data.

When rain falls on a watershed, water storage in the landscape increases and streamflow levels generally increase. Soon after rain stops falling, the discharge levels in streams will

decline in a process known as the streamflow recession. Below is a plot of the volumetric flowrate (discharge) during and after a single rainfall event, from the Middle Fork of the Eel River in northern California. Notice that the discharge starts off rather slowly, peaks, then recedes gradually.

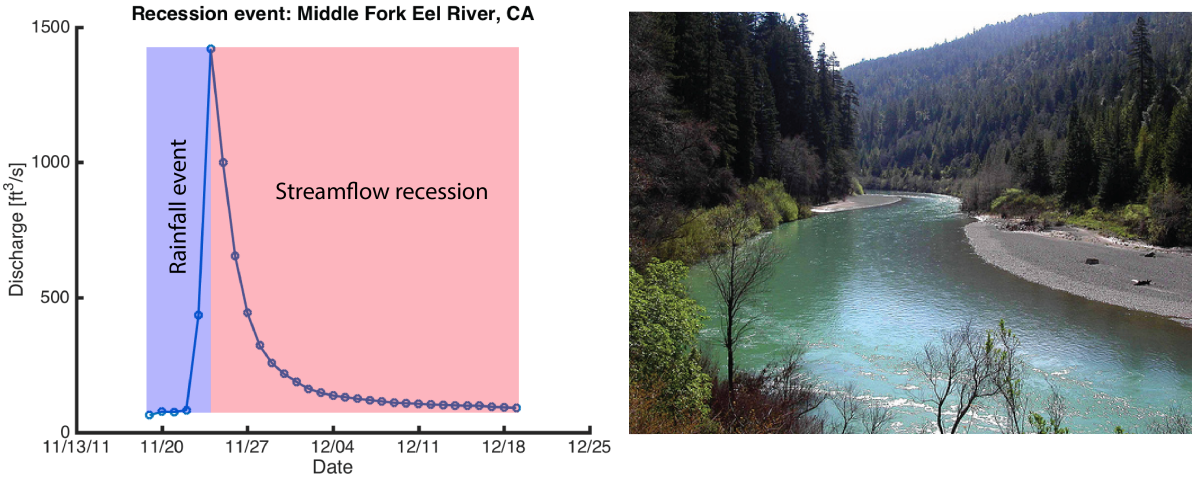

Figure 2: Recession event from the Middle Fork of the Eel River, CA in 2011

One of the most popular mathematical models for the streamflow recession is the so-called 'power-law' recession model, which defines the *rate of decline of streamflow* (i.e. the derivative of streamflow) as:

$$-\frac{dQ}{dt} = aQ^b. \tag{3}$$

Although you will not be required to obtain the solution of such an equation yourself, it can be solved to yield a highly nonlinear function for the form of the streamflow recession:

$$Q(t) = \left[(b-1)\left(\frac{Q_0^{1-b}}{b-1} + at\right)\right]^{\frac{1}{1-b}} \tag{4}$$

where Q_0 is the initial streamflow value on the first day of the recession. Instead of attempting to fit this nonlinear model to streamflow data, many researchers will instead go back to Equation 3 and log transform the relationship:

$$-\frac{dQ}{dt} = aQ^b \implies \log\left(-\frac{dQ}{dt}\right) = \log a + b\log Q. \tag{5}$$

As you can see, if you were to plot $\log\left(-\frac{dQ}{dt}\right)$ vs. $\log Q$, the result would be a linear function with a slope of b and a y-intercept of $\log a$.

To find a and b using Matlab we will perform four steps: 1) Compute a collection of values of the derivative, or time rate of change, of the discharge, 2) Construct and solve an over-determined system of equations, which will tell us the best fit values for a and b, 3) Determine how good the fit is by calculating R^2, and 4) Plot the best fit equation on top of the streamflow recession data.

1. For this problem, assume you are given N daily streamflow recession values, $Q(0), Q(1), Q(2), ..., Q(N-1)$, represented in Matlab as a column vector `discharge` with N entries. You can find a sample streamflow recession time series in the file `discharge.mat`, located in the assignment directory (this recession is the decreasing segment in the figure above). The first entry of `discharge`, $Q(0)$, represents flow on the first day, when $t = 0$, $Q(1)$ on the second day when $t = 1$, and so on. (Note that you will have to find a way to do the indexing such that you don't use a zero index in Matlab! Be careful with your indices in this problem!) Write a function with the header,

   ```
   function [dQdt avgQ] = getDerivative(discharge)
   ```

   that returns a vector of values of the derivative, or the time rate of change, of discharge, and the discharge values corresponding to each computed value of the derivative. As an approximation for the derivative on day $t = \tau$, your function should compute the following:

$$\frac{dQ(\tau)}{dt} = \frac{Q(\tau+1) - Q(\tau)}{(\tau+1) - (\tau)} = Q(\tau+1) - Q(\tau) \tag{6}$$

   where the simplification in the denominator can be made because t is in given in one day intervals. You should note that the length of the vector $\frac{dQ}{dt}$ will be $N-1$, instead of N. Your function will *also* return the two day average of discharge (`avgQ`) for each computed value of $\frac{dQ}{dt}$, that is:

$$Q_{avg}(\tau) = \frac{Q(\tau+1) + Q(\tau)}{2} \tag{7}$$

   which is also of length $N-1$. (Doing this simply stores the discharge at the temporal midpoint where the derivative is calculated so that Q and $\frac{dQ}{dt}$ are computed at the same point in time.)

2. Now, let's set up the over-determined system of equations that we will solve to get the parameters of our recession model, a and b.

   For each day, $t = \tau$, we will have a single equation

$$\log a + b \log\left(Q_{avg}(\tau)\right) = \log\left(-\frac{dQ(\tau)}{dt}\right) \tag{8}$$

   Yielding a total of $N-1$ equations for only two unknowns, a and b. How can we convert the above system of equations into a single matrix equation? Well, the two unknowns we are looking to find are b and $\log a$, and so we can call our vector of unknowns $\mathbf{x} = \begin{bmatrix} \log a \\ b \end{bmatrix}$. Next, we need the right hand side (RHS) of the equation in

vector form, which will be

$$\mathbf{y} = \begin{bmatrix} \log\left(-\frac{dQ(0)}{dt}\right) \\ \log\left(-\frac{dQ(1)}{dt}\right) \\ \vdots \\ \log\left(-\frac{dQ(N-2)}{dt}\right) \end{bmatrix} \tag{9}$$

Finally, to connect the left and right hand sides, we set up the full equation:

$$\mathbf{Ax} = \begin{bmatrix} 1 & \log\left(Q_{avg}(0)\right) \\ 1 & \log\left(Q_{avg}(1)\right) \\ \vdots & \vdots \\ 1 & \log\left(Q_{avg}(N-2)\right) \end{bmatrix} \mathbf{x} = \mathbf{y} \tag{10}$$

Clearly such a system will generally be over-determined; there are more equations than unknowns. However, Matlab's backslash operator is smart! It automatically knows how to solve over-determined systems using the least squares methods from section 13.2 in your textbook. In Matlab, the solution (in a least squares sense) to an equation like the system above is simply `x = A \ y`.

For this second part, write a function with the header,
`function [a b] = getRecessionParams(discharge)`
that returns the power law recession parameters a and b. You should use the function `getDerivative` from the previous question (make sure to *carefully* double check that it works before using it for this function). Inside the function `getRecessionParams`, you should set up the matrix equations demonstrated above and use the backslash operator to find the solutions, $\log a$ and b. Note that your function should return the values a and b, and so you will need to transform $\log a$ into a using the identity, $a = e^{\log a}$.
Test case:

```
>> load('discharge.mat')
>> [a b] = getRecessionParams(discharge)

a =
   5.6376e-05

b =
    2.3699
```

3. If you read section 13.2 in your textbook closely, you'll see that solving the over-determined system in the previous part is exactly the equivalent of performing a linear regression. Essentially, you fit the data with a best fit linear function of the form:

$$\log\left(-\frac{dQ}{dt}\right) = \log a + b \log Q \tag{11}$$

You can think of $\log\left(-\frac{dQ}{dt}\right)$ as the dependent variable (y), $\log Q$ as the independent variable (x), $\log a$ as the y-intercept of the best fit line, and b as the slope.

For any modeling exercise, it's good practice to report some measure of 'goodness of fit', that is, how well your model performs. There are a number of different measures of goodness of fit, but one of the most popular for linear regression is the coefficient of determination, commonly referred to as R^2.

In practical terms, R^2 is simply a measure of how much better your best fit *linear* function is than a best fit *constant* function, or mean model, $y = \text{mean}(y_{data})$. R^2 is computed with the following equation:

$$R^2 = 1 - \frac{SSE_{linear\,model}}{SSE_{mean\,model}}, \tag{12}$$

where $SSE_{linear\,model}$ is the sum of squared errors of the linear model, and $SSE_{mean\,model}$ is the sum of squared errors of the mean model. Sum of squared errors is simply the sum of the squares of the differences between the model and the data. Looking at the equation for R^2, you can see that the smaller the errors of the linear model *relative* to the mean model, the closer R^2 will get to 1. So, the *perfect* linear fit has an $R^2 = 1$.

To illustrate, consider the very simple dataset,
`xdata = [1, 2, 3, 3.5]; ydata = [1.2, 3, 2.8, 3.9];` (blue points below). Also plotted below are both the best fit linear model (using linear regression) and the best fit mean model to the data:

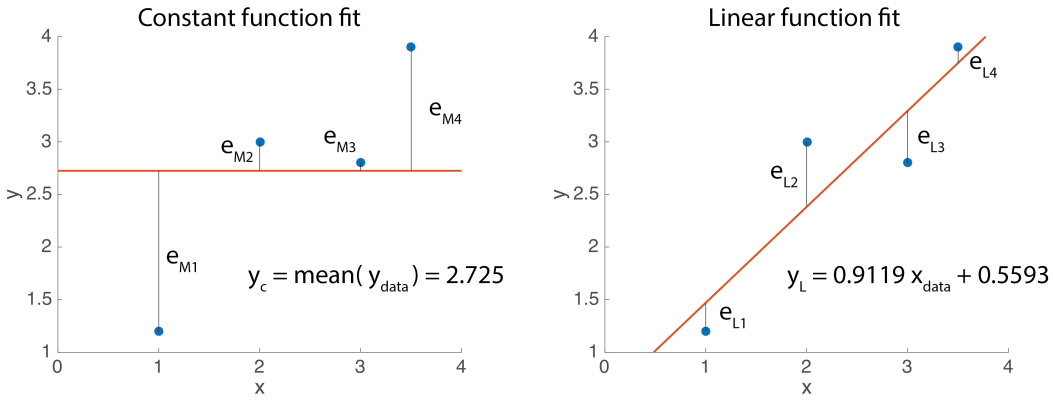

Figure 3: Linear model fit vs. constant (mean) model fit

The individual errors are simply the vertical lengths of the fine black lines, representing the distance from the data to each model. In this case, the sum of squared errors can be computed as:

$$SSE_{linear\,model} = {e_{L1}}^2 + {e_{L2}}^2 + {e_{L3}}^2 + {e_{L4}}^2 \tag{13}$$
$$SSE_{mean\,model} = {e_{M1}}^2 + {e_{M2}}^2 + {e_{M3}}^2 + {e_{M4}}^2 \tag{14}$$

Therefore, $R^2 = 1 - \frac{SSE_{linear\,model}}{SSE_{mean\,model}} = 0.8095$. Give it a try and see if you get the same numbers.

For this part, you will write a function with header
`function [ rsq ] = rsquaredRecessionModel(discharge)`
to compute the R^2 goodness of fit of your streamflow recession model. Note, for the data points in your R^2 calculation, you should use the variables `dQdt` and `avgQ` outputted from the function `getDerivative(discharge)`. Also, you should find the R^2 value for the *linear* fit in log-log space, that is, find R^2 for the fitted line $\log\left(-\frac{dQ}{dt}\right) = \log a + b \log Q$; while it is possible to compute an R^2 value for the non-linear form of the model (Equation 4), we will not do that here. Test case using the **`discharge.mat`** file:

```
>> load('discharge.mat')
>> [ rsq ] = rsquaredRecessionModel(discharge)

rsq =

   0.9100
```

4. For the final part of this problem, you will write the function
`function [ ] = plotRecessionModel(a, b, discharge)`
which will take a pair of recession parameters (a and b), a discharge time series (`discharge`), and plot the recession model on top of the discharge data. You should pair Equation 4 with the recession parameter values to plot the model. The function should also include the values of the recession parameters in the plot title. Note that Q_0 in Equation 4 will simply be the first element in the vector `discharge`, and the independent variable t will be a simple length N vector, `time = 0:(N-1)`. For the test case above, your `plotRecessionModel` function should generate a plot similar to the one below. Although it does not need to look exactly the same (you can choose different colors, markers, etc), you should properly label your axes, the legend, and the title.

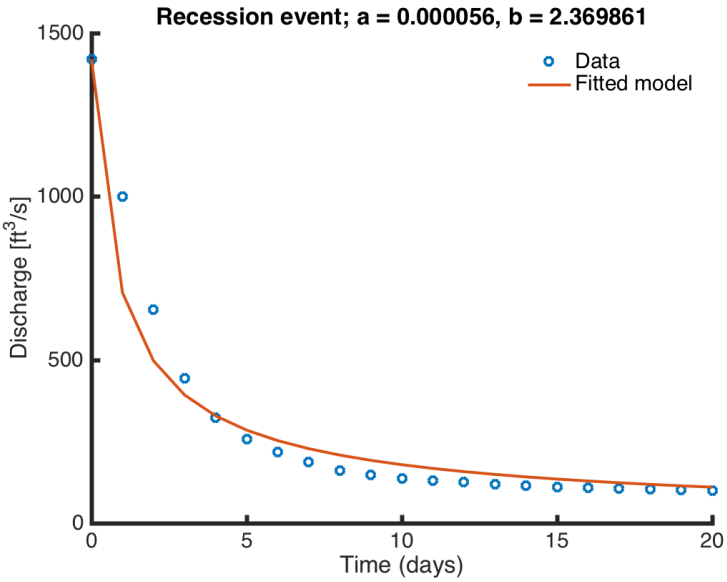

Figure 4: Power law recession model fitted to discharge data.

Submission Instructions

Your submission should include the following function files in a single zip file named Lab9.zip

- mySols.m
- cubicPolyDiff.m
- polyDiff.m
- polyDiffm.m
- myLinearRegression.m
- myBestRegression.m
- getDerivative.m
- getRecessionParams.m
- rsquaredRecessionModel.m
- plotRecessionModel.m

Don't forget that the function headers and file names you use must appear *exactly* as they do in this problem set. So, make sure the variables have the right capitalization, the functions have the correct name, and the inputs and outputs are in the correct order.