

Exercise 6
Psych 254
Due Nov 14

Problem 1. In this exercise you will use a mathematical model to create simulated data, and then fit the model using a gradient search method.

First, implement a simplified version of Logan's (1988) instance theory of automaticity. This model proposes that task performance on a particular trial is determined by a horserace between two processes: an algorithm, which is guaranteed to succeed but does not improve with practice, and memory retrieval, which depends upon the number of memory traces (one of which is stored on each trial). Create a function called `XX_instance_model()` which takes two arguments:

Params: a 1 X 4 vector of parameters, which specify the following:

Params(1): mean algorithm finishing time

Params(2): standard deviation of algorithm finishing time

Params(3): mean memory retrieval time

Params(4): standard deviation of memory retrieval time

Trialnums: a scalar or vector containing trial numbers at which to run the model

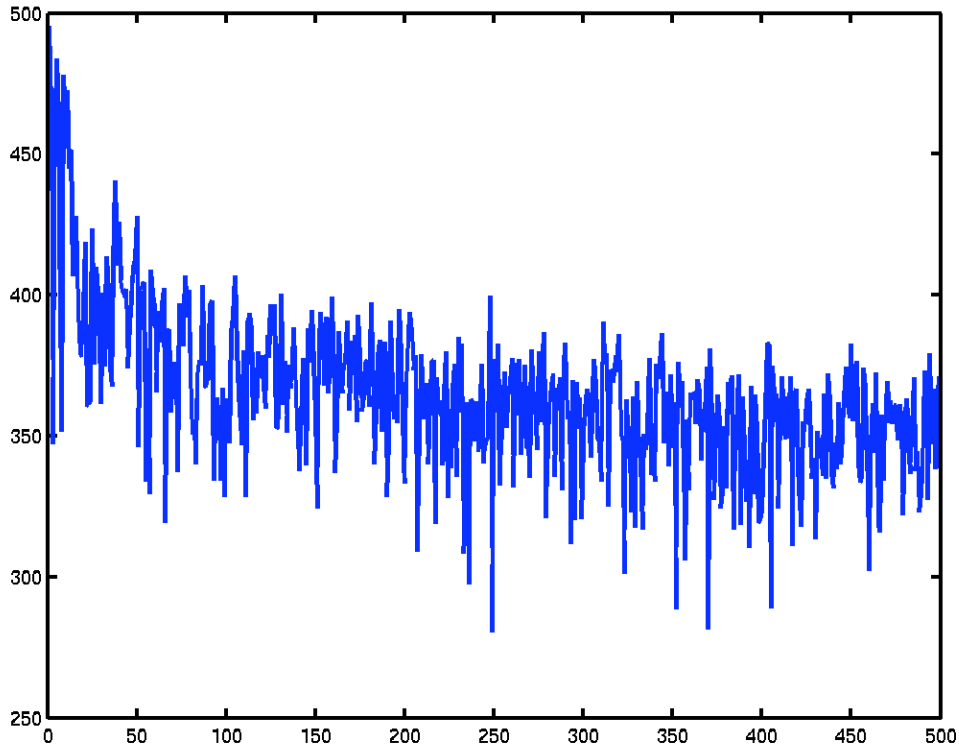
The function should take these parameters (checking for proper input specification) and then run the model using the following algorithm:

1. determine the algorithm finishing time by sampling from a normal distribution specified by the algorithm finishing time mean and SD
2. determine the memory retrieval finishing time by taking a number of samples from a normal distribution specified by the memory retrieval mean and SD. Because a memory trace is created after each trial, the total number of memory traces on each trial is `trialnums-1`. Note that this means that the algorithm is always the winner on trial 1. The memory retrieval finishing time on trials 2:n is the minimum memory retrieval time of all of the samples for the particular trial.
3. Return the minimum of the algorithm and memory retrieval finishing times as the reaction time for that trial. Because `trialnums` can be either a scalar or vector, you will need to loop through all values in the vector.
Hint: If you have a vector `trialnums=[1 2 3 4 5...]` and you want to loop through all values in the vector, you can use the following construct:

```
for x=trialnums,  
    ...  
end;
```

this will loop through, on each loop setting `x` to the next value in `trialnums`.

Test the model to see what the resulting data look like, using $b=[500 \ 10 \ 500 \ 50]$ as sample parameters. It should look something like this:



Run the model and plot the data several times, to see how much variability there is from run to run.

Problem 2. In the second problem you will fit several different functions to data from the instance model that you created in problem 1. Create a program called `instance_fit_<name>.m` to perform these operations.

Before starting this section, you will need to download from the web site two functions that will be used in fitting the data. The first, `power_fun(beta,N)` is a power function:

$$Y = a + b * N^{-c}$$

The second, `exp_fun(beta,N)` is an exponential function:

$$Y = a + b * e^{-cN}$$

In each case, a , b , and c are the three parameters to be estimated from the data, and N is the trial number. You can interpret a as the asymptote of the learning curve, b as the range of the learning curve, and c as the learning rate parameter. These parameters are contained in the β argument to the function, $\beta=[a \ b \ c]$.

First, run the model 100 times with $\beta = [450 \ 100 \ 400 \ 100]$ over the range of 1:100 trials, and average across these runs to create a mean dataset.

Next, use `nlinfit()` to estimate the best-fitting parameters for both the power and exponential functions from the mean RT data. Plot the data and the best-fitting curves on a single plot. You should present each of these in a different color, and provide a legend and x-axis and y-axis titles. The legend should include the R-squared value for each fitted function, which can be computed by squaring the correlation coefficient between the fitted and actual data obtained using `corrcoef()`.

The result should look something like this:

