BIO782P - Week 2 Assignment

Ana Penedos (UID 1932)

14th December 2018

Part A

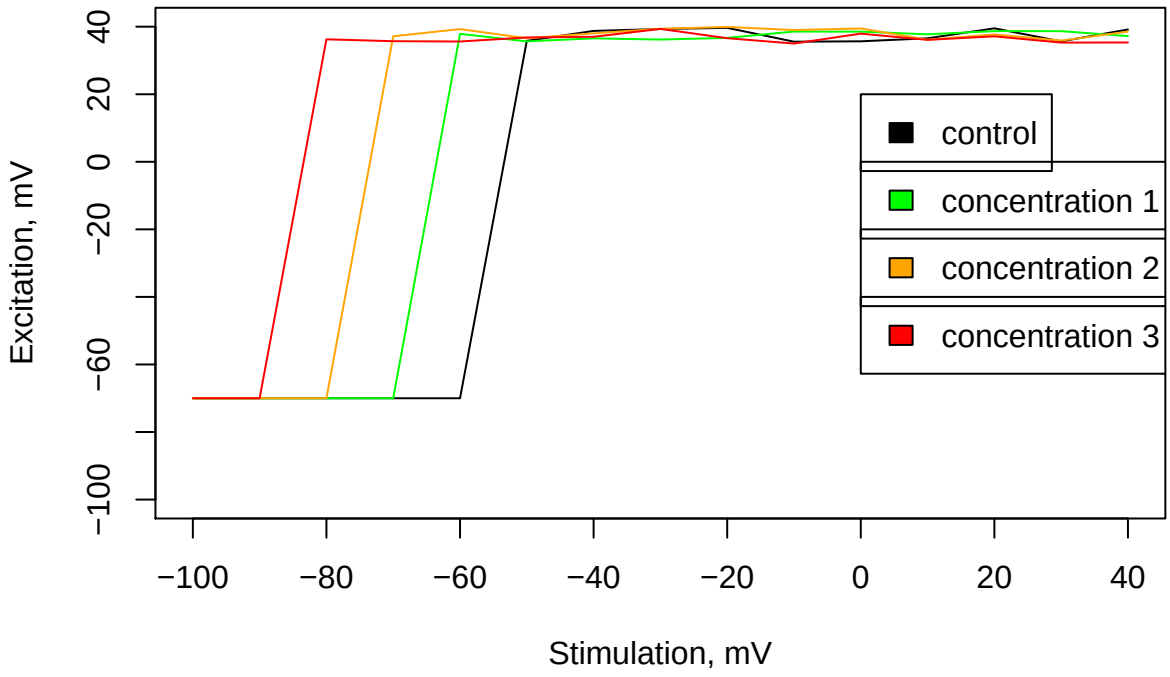
- effect of nicotine concentration on the nervous system
- giant squid axons submitted to different nicotine concentrations
- electric stimulus of increasing voltages was applied to each axon
- peak response voltage recorded for 3ms after each electric stimulus
- measured action potentials (in mV)

input data

```
stimulation_potential = seq(-100, 40, by=10)
response_voltage_control = c(-70.00000,-70.00000,-70.00000,-70.00000,-70.00000,
                             35.79871,38.75082,39.30423,39.65390,35.55835,
                             35.66762,36.55754,39.51027,35.65274,39.17013)
response_voltage_concN_1 = c(-70.00000,-70.00000,-70.00000,-70.00000,37.88144,
                             35.64345,36.54319,36.18773,36.67542,38.56634,
                             38.53102,37.73421,38.69826,38.67744,37.21896)
response_voltage_concN_2 = c(-70.00000,-70.00000,-70.00000,37.16110,39.28471,
                             36.57506,38.03037,39.37853,39.92131,39.01116,
                             39.46393,36.16073,37.70418,35.88598,38.53137)
response_voltage_concN_3 = c(-70.00000,-70.00000,36.25729,35.70035,35.61478,
                             36.81245,37.01299,39.34139,36.59520,35.00749,
                             37.94440,36.10611,37.15617,35.29182,35.32412)

plot(stimulation_potential,stimulation_potential,
     type='n',xlab='Stimulation, mV',ylab='Excitation, mV')
lines(stimulation_potential,response_voltage_control)
lines(stimulation_potential,response_voltage_concN_1,col="green")
lines(stimulation_potential,response_voltage_concN_2,col="orange")
lines(stimulation_potential,response_voltage_concN_3,col="red")

legend(0,20,"control",fill="black")
legend(0,00,"concentration 1",fill="green")
legend(0,-20,"concentration 2",fill="orange")
legend(0,-40,"concentration 3",fill="red")
```



1. Describe what the inputs, outputs and role for each of the functions predict, calculate_errors and fit_threshold are.

```
predict = function(activation_threshold,stimulation_voltage){
  # Produces a vector of predicted response values (mV) based on whether
  # the stimulation voltage is less or equal (-70) or greater (40) than
  # the activation threshold.
  #
  # Args:
  #   activation_threshold - the voltage at which the axon becomes
  #                          activated (numeric).
  #   stimulation_voltage - the voltages at which the axon was stimulated
  #                         (vector of numeric values).
  #
  # Returns:
  #   Vector of same length as the stimulation_voltage vector argument
  #   with value -70 when stimulation <= threshold and 40 when > threshold.

  # produces a vector of '-70's the same length as the variable
  # 'stimulation_voltage'
  activation = rep(-70,length(stimulation_voltage))
  # Values in position x of activation are changed based on
  # the value in position x of stimulation_voltage.
  # When stimulation_voltage <= activation_threshold, value is left at -70
  # When stimulation_voltage > activation_threshold, value is set to 40
  activation[stimulation_voltage>activation_threshold] = 40
  # returns vector of predicted values
  return(activation)
}
```

```r
# checking of the predict function
predict(-80, -90)
```

```
## [1] -70
```

```r
predict(-80, -80)
```

```
## [1] -70
```

```r
predict(-80, -75)
```

```
## [1] 40
```

```r
predict(-80, -70)
```

```
## [1] 40
```

```r
predict(-80, -40)
```

```
## [1] 40
```

```r
predict(-80, c(-90, -85, -80, -75, -70))
```

```
## [1] -70 -70 -70  40  40
```

```r
?rep()
test_rep <- rep(-70, 10)
test_rep
```

```
##  [1] -70 -70 -70 -70 -70 -70 -70 -70 -70 -70
```

```r
test_a_vector <- 1:10
test_a_vector
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
test_rep[test_a_vector > 5] = 40
test_rep
```

```
##  [1] -70 -70 -70 -70 -70  40  40  40  40  40
```

The `predict` function takes an activation threshold and a vector of stimulations used and produces a vector of the same length of the stimulation vector, with -70 in the positions corresponding to stimulation under the threshold and 40 in those where the stimulation voltage was above the threshold.

```r
calculate_errors = function(predicted, observed){
  # Calculates residual sum of squares between observed and predicted values.
  #
  # Args:
  #    2 numeric vectors of the same length
  #    predicted - vector containing the predicted values (numeric vector)
  #    observed - vector containing the observed values (numeric vector)
  #
  # Returns:
  #    Residual sum of squares (float) between two numeric vectors of
  #    observed and predicted values.
  total_errors = sum((observed - predicted)^2)
  return(total_errors)
}
```

The `calculate_errors` function calculates the residual sum of squares

$$sum((observed - predicted)^2)$$

estimating the total deviation of a vector of measured values from a vector of predicted values.

```
fit_threshold = function(input_values_stimulation,
                         input_values_response,
                         threshold){
  # Calculates the fit error using observed response values and response
  # values predicted from stimulation values and threshold.
  #
  # Args:
  #    input_values_stimulation - stimulation voltages used
  #                               (vector of numeric values)
  #    input_values_response - responses measured (vector of numeric values)
  #    threshold - activation threshold for the axon (numeric)
  #
  # Returns:
  #    Residual sum of squares between 'input_values_response' and predicted
  #    response values from 'input_values_stimulation' and 'threshold'.

  # predict response values from activation threshold and stimulation values
  predicted_values = predict(threshold,input_values_stimulation)
  # calculate residual sum of squares between predicted and observed values
  fit_errors = calculate_errors(predicted_values,input_values_response)
  return(fit_errors)
}
```

The `fit_threshold` function takes a vector of the stimulation voltages, a vector of observed responses and a value for threshold. It uses the `predict` function to calculate a vector of predicted responses and the `calculate_errors` function to determine the total sum of squares between the response values and the predicted response vector obatined.

2. How is the data modelled?

The data is modelled based on a step function. When the stimulation voltage take values less than or euqal to an activation threshold, response voltage values take a negative value of -70 mV. After the activation threshold is reached, response values are equal to 40 mV.

3. How many degrees of freedom are present in (a) the model, and (b) the residuals?

There are 10 data points, hence we start with 10 degrees of freedom (no total or mean value is calculated in the context of the modelling). The model sets a threshold value, hence using a degree of freedom. Hence there are 9 degrees of freedom left for the residuals.

4. Describe what happens when the code is executed, in statistical terms.

The code attempts to estimate the model parameter (activation threshold) by minimising the residual sum of squares between the response values observed and predicted, which is used as a measure of the goodness of fit of the model.

5. How is the model fit to the data optimised?

```
# Generate a random threshold from a uniform distribution between -100 and 40
fitted_threshold = runif(1,-100,40)
# Calculate the error between the response values predicted with the fitted
# threshold
error = fit_threshold(stimulation_potential,
                      response_voltage_control,
                      fitted_threshold)

# 20 iterations
for(i in 1:20){
  # generate a new random threshold from a uniform distribution
  # between -100 and 40
  new_threshold = runif(1,-100,40)
  # calculate the error associated to the new threshold
  new_error     = fit_threshold(stimulation_potential,
                                response_voltage_control,
                                new_threshold)
  if(new_error < error){
    # keep the new threshold and error if improvement from previous
    fitted_threshold = new_threshold
    error = new_error
  }
}

fitted_threshold
```

```
## [1] -56.95467
```

```
error
```

```
## [1] 89.99066
```

An initial threshold is randomly generated from a uniform distribution that can take values between -100 and 40. The initial error is then calculated between the observed values and the predicted responses from the random threshold and stimulation values. The process is repeated over 20 iterations and if a lower error is found between the predicted values for the random threshold and the response values, the script keeps the new threshold and error values obtained. At the end of the 20 iterations, the threshold associated with the smallest error calculated and the corresponding error are returned.

6. How could you improve the optimisation?

- More iterations: the simplest (though not more efficient) way to improve the optimisation is to simply increase the number of iterations.
- Use a normal distribution around the fitted_threshold with lower error: this way, when a better threshold is found (i.e., one with lower residual sum of squares) the new proposed threshold will more likely be closer to the improved value. By always picking from a uniform distribution, the process of optimisation is effectively restarted in each iteration, with no "learning" from prior cycles.
- Use several different start points: if we implemented the previous step (centering new guesses around the best guess so far), we would risk finding a local optimum rather than the global one, so it would be safer to implement an outer loop picking values from a uniform distribution and an inner loop selecting thresholds from a normal distribution. In this way, even if one of the inner loops returned a local optimum, we are more likely to find the optimal threshold through one of the other loops.

- Improve start value: instead of randomly picking a random initial value from a uniform distribution, we could take an educated guess that the threshold would fall closer to the median stimulation voltages used.

How would you modify the code to:

a) fit the data given under the three experimental nicotine treatments and

We could simply replace the "response_voltage_control" argument given to the `fit_threshold` function in the optimisation by "response_voltage_concN_1", "response_voltage_concN_2" and "response_voltage_concN_3". To aautomate this and avoid substitution mistakes or repeating the same lines of code, we could enclose the optimisation into a loop iterating through the 4 sets of responses measured (control and concentrations 1 to 3 of nicotine), where each set of responses is attributed to a "observed_response" variable that is then used within the optimisation as an argument for the `fit_threshold` function.

b) compare the goodness of fit amongst these fitted models to determine whether each nicotine treatment lowers the activation threshold, compared to the control treatment

Each fitted model for axon response to an electric stimulus in the presence of varying concentrations of nicotine could be compared to the responses predicted for the control using an ANOVA. If the p-value obtained for each predicted model was below the critical value previously defined (normally 0.05), we could accept that the threshold is affected by that nicotine concentration.

c) what is the null hypothesis in this case?

The null hypothesis, H0, is that there is no difference in threshold between the control and the treatment with concentration X (1, 2 or 3) of nicotine.

Part B

- code modified to print fits to logfiles (fit_01, fit_02, ...) at each iteration
- fitted_threshold and errors saved to final.csv
- use user-specified random seed to set initial threshold

Dockerfile:

```
# Docker R base image
FROM r-base

COPY . /usr/local/src/myscripts
WORKDIR /usr/local/src/myscripts

CMD ["Rscript", "activation-thresholds.R"]
```

1. How would we use this code to run our analysis on a grid? Outline what steps we should take.

1. Build our image: from the directory containing the Dockerfile and the scripts we wish to include in our image, we would build an image using `docker build -t activationContainer .`. The `-t` option can be used to give our image an informative name.

2. Getting the image: we could either
   i) simply copy the image into a directory on the grid or
   ii) upload it to a remote repository, in which case we would: ii1) tag the image using `docker tag activationContainer some_user/some_repository:activationContainer`; ii2) login to docker hub with `docker login`; ii3) upload our image with `docker push some_username/some_repository:activationContainer`; ii4) login to the grid and obtain the image using `module load singularity` (or `module load docker`) and then `singularity pull docker://some_username/some_repository:activationContainer` (or `docker pull some_username/some_repository:activationContainer`);
3. Running the container: either by (i) directly running `singularity run activationContainer.img input_data.csv` (or `docker run activationContainer.img input_data.csv`) or (ii) creating a jobfile like the one shown in question 2 and submitting it to a job queue.

Jobfile:

```
#$ -cwd
#$ -S /bin/bash
#$ -j y
#$ -pe smp 1
#$ -l h_vmem=2G
module load singularity
singularity run activationContainer.img input_data.csv
```

2. How would you rewrite this jobfile to achieve the following:

a) Move the final outputs to a directory (../output)?

I would modify my R script to accept an output directory given by a `-o` option (e.g., `-o results_directory`) and then change the last line of the jobfile to: `singularity run -B ../output:/results_directory activationContainer.img input_data.csv -o /results_directory`

b) Change random seed (set with the -p [some integer] argument)?

I would add a line of code generating a random integer from the shell and storing it as a variable before or after loading singularity: `seed=$RANDOM` I would then modify the line running singularity to:

```
singularity run -B ../output:/results_directory activationContainer.img \
input_data.csv -o /results_directory -p $seed
```

c) Run as an array job, with 42 replicates?

I would add the following two lines to the jobfile setup section:

```
#$ -t 1-42  # sets an array job with 42 tasks
#$ -tc 42  # determines that all tasks can happen concurrently
```

In order to prevent the overwriting of outputs when multiple tasks are run, I would need to slightly modify the script and the singularity run command:

```
mkdir ../output/results_directory_$seed
singularity run -B ../output/results_directory_$seed:/results_directory \
activationContainer.img input_data.csv -o /results_directory -p $seed
```