COMP0118 - Computational modelling for biomedical imaging

Coursework 2 - Report

10/03/23

Part 1:

1. (a) Here are the experimental mean and standard deviation computed after sampling:
`Sample 1; Mean : 1.5105; Std : 0.18385`
`Sample 2; Mean : 1.9734; Std : 0.18559`
These are the expected values, our sampling works.

   (b) MATLAB's `ttest2` returns 1 if the null hypothesis is rejected at the 5% significance level, and 0 otherwise. In our case, the null hypothesis is that the data in the two samples come from independent random samples from normal distributions with equal means and equal but unknown variances.
Here is our result: `The result of ttest2 is: 1`.
This is the expected result, given that the means are different enough.

   (c) i. The design matrix for our model is $X = [X_1 X_2]$. We have simulated 40 data points, in two different groups of 20 points. If we look at them group by group, we can write the X matrix such as:

$$X = \begin{pmatrix} 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}$$

where we have 20 ones followed by 20 zeros for the first column, and 20 zeros followed by 20 ones for the second one, because the 20 first samples will come from group 1, and the 20 next from group 2. X is a 40x2 matrix.
One data point cannot be in two groups, and the two groups are populated (i.e there is no group without a data point attached to it), therefore it is clear that the two columns are linearly independent, so $\dim(X) = 2$.

   ii. According to slide 53 from lecture 2,

$$P_X = \frac{XX'}{X'X}$$

We have to verify two conditions: P_X must be idempotent and symmetric:
- $P_X^2 = X(X'X)^\dagger X'X(X'X)^\dagger X' = X((X'X)^\dagger X'X)(X'X)^\dagger X' = X(X'X)^\dagger X' = P_X$ by the associative property of real matrices
- $P_X' = (\frac{XX'}{X'X})' = \frac{(XX')'}{(X'X)'} = \frac{(X')'X'}{X'(X')'} = \frac{XX'}{X'X} = P_X$

Let's now compute P_X in our case:

$$P_X = \frac{1}{20}\begin{pmatrix} \mathbf{1_{20}} & \mathbf{0_{20}} \\ \mathbf{0_{20}} & \mathbf{1_{20}} \end{pmatrix}$$

where $\mathbf{0_{20}}$ is the 20x20 matrix filled with zeros and $\mathbf{1_{20}}$ is the 20x20 matrix filled with ones.
We can also compute $Tr(P_X) = 2$. The trace of the perpendicular projection operator corresponds to the dimension of the column space $C(X)$, which is expected.

   iii. Let's now compute the projection, using P_X as described above. Let's note that we have used as a stochastic component a normal sample with mean 0 and standard deviation 1.

$$\hat{Y} = \begin{pmatrix} 1.5449 \\ \vdots \\ 1.5449 \\ 1.6348 \\ \vdots \\ 1.6348 \end{pmatrix}$$

Looking closely at the form of P_X and $\hat{Y}$, we understand that the output vector shows the mean of the sample group for each data point. We can say it represents the "estimation" space because given that X represents the belonging to each group, we are trying to estimate the mean of each group. That is what is done in $\hat{Y}$.

iv. We can compute R_X:

$$R_X = I - P_X = \frac{1}{20}\begin{pmatrix} 19 & -1 & -1 & 0 & \dots & 0 \\ -1 & \ddots & -1 & \vdots & \ddots & \vdots \\ -1 & -1 & 19 & 0 & \dots & 0 \\ 0 & \dots & 0 & 19 & -1 & -1 \\ \vdots & \ddots & \vdots & -1 & \ddots & -1 \\ 0 & \dots & 0 & -1 & -1 & 19 \end{pmatrix}$$

Let's now verify the conditions for R_X to be a perpendicular projection operator:

- $R_X^2 = (I - P_X)(I - P_X) = I^2 - 2P_X + P_X^2 = I - 2P_X + P_X = I - P_X = R_X$ because I and P_X are indempotent.
- $R_X^{'} = (I - P_X)^{'} = I^{'} - P_X^{'} = I - P_X = R_X$ because I and P_X are symmetric

We have verified that R_X is a perpendicular projection operator.

v. Let's compute the projection on the error space (we will show only a few samples):

$$\hat{e} = \begin{pmatrix} \texttt{0.3795} \\ \vdots \\ \texttt{0.8913} \\ \texttt{2.2306} \\ \vdots \\ \texttt{-0.0492} \end{pmatrix}$$

We have two variables in our simulation (because we have two sample groups). This is a finite dimension space, therefore its subspaces $C(X)$ and $C(X)^{\perp}$ must verify the following formula: $\dim C(X) + \dim C(X)^{\perp} = 40$. It comes that $\dim C(X)^{\perp} = 38$.

vi. To determine the angle between $\hat{e}$ and $\hat{Y}$, we use the dot product to retrieve the cosine, then the angle.
Our simulation gives the result: `angle = 1.5708`, that is $\theta = \frac{\pi}{2}$ where θ represents the angle between $\hat{e}$ and $\hat{Y}$.
So, our two vectors are orthogonal. This is expected, given that P_X projects Y on $C(X)$, and R_X on $C(X)^{\perp}$, and that these two spaces are orthogonal.
As a sanity check, we have computed $(\hat{Y} + \hat{e}) - Y$, and we obtain the vector full of zeros, which proves that $\hat{Y}$ and $\hat{e}$ are the decomposition of Y in two orthogonal spaces.

vii. By definition: $\hat{Y} = X\hat{\beta}$, but we also have $\hat{Y} = P_X Y$, it follows that:

$$X\hat{\beta} = X(X^{'}X)^{\dagger}X^{'}Y \Longrightarrow \hat{\beta} = (X^{'}X)^{\dagger}X^{'}Y$$

where $X^{\dagger}$ represents the pseudo-inverse of X.
This is known as the least-squares estimate because when minimizing the sum of squares of some data, we also end up with the normal equation.
We use the formula to estimate the parameters in our simulation:

```
beta_est = [1.5449; 1.6348]
```

We find the means of the two groups, 1.5 is correctly estimated, 2.0 is not really retrieved.

viii. If we rather express $\hat{e}^{'}\hat{e}$ as $\sum_i {\hat{e}_i}^2$, we can easily see that $\hat{\sigma}^2$ is actually the sum of squared errors, but with the $\frac{1}{n - \dim X}$, we can call it mean of squared errors.
The computation gives us the following value: $\hat{\sigma}^2 = \texttt{1.0939} \Longrightarrow \hat{\sigma} = \texttt{1.0459}$. This is approximately the standard deviation we have used for our stochastic component.

ix. Here is our result:

$$S_{\hat{\beta}} = \begin{pmatrix} \texttt{0.0547} & \texttt{0} \\ \texttt{0} & \texttt{0.0547} \end{pmatrix}$$

The diagonal terms are the covariances, we use the square root to retrieve the standard deviation, and we find that our estimated model parameters have a standard deviation of `0.2338`, which is almost the 0.2 value we have used to sample our data.

x. We want to compare the group differences in the means, meaning we want to observe $\mu_1 - \mu_2$ (or $\mu_2 - \mu_1$, we will stick to the first formulation). Therefore, we write:

$$\lambda = \begin{pmatrix} 1 & -1 \end{pmatrix}$$

From this λ, we can derive the reduced model:

$$X_0 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

which is basically the column vector full of ones, because under the null hypothesis H_0, we assume that there are no differences between the two means.

xi. Let's compute the sum of squared errors for the reduced model, and then compute the F-statistic with the formula indicated in the lecture (part 3, slide 95). Our results are as follows:

$$P_{X_0} = \frac{1}{40}\begin{pmatrix} \mathbf{1_{20}} & \mathbf{1_{20}} \\ \mathbf{1_{20}} & \mathbf{1_{20}} \end{pmatrix} \Longrightarrow R_{X_0} = \frac{1}{20}\begin{pmatrix} 19 & -1 & -1 & -1 & \dots & -1 \\ -1 & \ddots & -1 & \vdots & \ddots & \vdots \\ -1 & -1 & 19 & -1 & \dots & -1 \\ -1 & \dots & -1 & 19 & -1 & -1 \\ \vdots & \ddots & \vdots & -1 & \ddots & -1 \\ -1 & \dots & -1 & -1 & -1 & 19 \end{pmatrix}$$

$$SSR(X_0) = 41.6479, \quad SSR(X) = 41.5671$$

$$F = 0.0738$$

The F-statistic being different than 1, we can deduce that the groups have different means. Although, it is not far from 1: our noisy estimates have made the means closer to each other. We find in our computation that the F-statistic has one degree of freedom for the numerator, and 38 degrees of freedom for the denominator.

xii. We compute the t-statistic with the formula indicated in the coursework sheet. We obtain: `t_stat = -0.2717`.
The magnitude of the t-statistic is different from zero, although it is relatively low (see remark made in c) xi.). Therefore, we have reason to believe that the null hypothesis is not verified i.e that the mean of both populations is different. It is the same conclusion as the one found in b).

xiii. The estimated model parameters in $\hat{\beta}$ represent the mean of each sample group. Their ground truth values should be 1.5 and 2.0 (see a)). We obtain satisfactory results with our estimate (see c) vii.).

xiv. Here are our results for the projection of e into $C(X)$:

$$e_{C(X)} = \begin{pmatrix} 0.0344 \\ \vdots \\ 0.0344 \\ -0.3386 \\ \vdots \\ -0.3386 \end{pmatrix}$$

The projection of the ground truth deviation onto the estimation space represents the level of noise that is still present in our estimates (i.e not in the error space). To check this fact, let's look at our $\hat{\beta}$ and let's remove the "estimation noise" found above. We obtain `[1.5105; 1.9734]`. We can see that the 1.5 mean stays approximately the same, but the 2.0 mean has been corrected compared to our former estimate.

xv. Here are our results for the projection of e into $C(X)^{\perp}$:

$$e_{C(X)^{\perp}} = \begin{pmatrix} 0.4371 \\ \vdots \\ 0.9292 \\ 2.2508 \\ \vdots \\ 0.0585 \end{pmatrix}$$

This projection represents how much stochastic noise has participated in the error part of the estimation.

(d) i. We add the first column full of ones, such as indicated in the coursework sheet:

$$X = \begin{pmatrix} 1 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 1 \end{pmatrix}$$

We can see that by adding the last two columns, we retrieve the first column. Therefore, $\dim(X) = 2$.

ii. We obtain the same P_X as in question c) ii. The estimation space should therefore be the same as in the previous part.

iii. Here is the appropriate contrast vector for our hypothesis:

$$\lambda = \begin{pmatrix} 0 & 1 & -1 \end{pmatrix}$$

iv. Using the same routine as before, we compute: `t_stat = -0.2717`. We obtain the same value as in 1(c)xii.

v. X_1 and X_2 still represent the belonging to each sampling group, and X_0 represent the mean of both groups, so that we can determine how close to one another they are (i.e their common effect).

(e) i. The model has not changed, we keep the same design matrix:

$$X = \begin{pmatrix} 1 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 1 \end{pmatrix}$$

As demonstrated above, $\dim(X) = 2$.

ii. Our hypothesis is that the second variable has no effect, i.e $\beta_2 = 0$, which can be translated by:

$$\lambda = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

We deduce the reduced model:

$$X_0 = \begin{pmatrix} 1 & 1 \\ \vdots & \vdots \\ 1 & 1 \\ 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \end{pmatrix}$$

iii. We have computed: `t_stat = 3.2979`. The value of the t-statistic being relatively high, we can understand that the null hypothesis of β_2 being 0 should be rejected.

iv. β_0 represents the mean of our samples, and β_1 represents the value we need to add to the mean (β_0) to retrieve the mean of group 1.

(f) There would be no point in testing an hypothesis for this new model, it only represents the mean of our samples. It would have made sense to compare one group with another, but here there is simply no group at all (or conversely, all the samples are part of the same group).

2. (a) We obtain the following result: `The result of ttest is: 1.` `ttest2` was looking at the mean of both groups, where `ttest` looks at the mean of the difference between the two groups. Anyway, we obtain the same results for the two functions, which is expected.

(b) i. Using the information given in the coursework sheet, here is the design matrix for this new GLM model:

$$X = \begin{pmatrix} 1 & 1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 0 & 0 & \dots & 1 \\ 1 & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 1 & 0 & \dots & 1 \end{pmatrix}$$

where the first three columns correspond to the same GLM model as previously, and the remainder of the matrix is two 20x20 identity matrices stacked on top of each other (because each measurement corresponds to a subject, and that the same subject is inspected twice). For the same reason as in 1(d)i, $\dim(X) = 2$.

ii. For this new model, we can write:

$$\lambda = \begin{pmatrix} 0 & 1 & -1 & 0 & \dots & 0 \end{pmatrix}$$

iii. We can compute: `t_stat = -0.3939`. That is a higher (in magnitude) t-statistic, but still relatively low, we cannot accept the null hypothesis. This is the same conclusion as in (a).

Part 2:

1. (a) Here are our results:
```
The result of ttest2 is: 1
The p-value is: 0.021221
```
Even with smaller sample sizes, it seems that `ttest2` can still discriminate between the two groups. Nevertheless, the p-value is quite small.

(b) i. See the code.

ii. See the code. We have used `nchoosek` first for 6 samples, then for 8 samples, and we have run a loop to stack vertically the 14 samples that we have obtained.

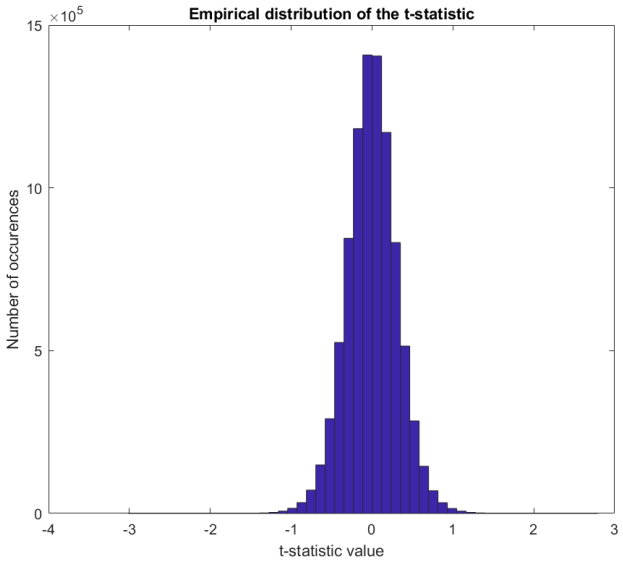

Figure 1: Empirical distribution of the t-statistic

iii.

iv. Here is the new p-value we have found: `0.7304`. It is quite high, and much higher than the one found in (a). Even though we had a small initial sample size, the permutation test has made clear the state of the null hypothesis for us: we should reject it (as expected from the last parts). The permutation test seems particularly useful when facing a situation with little data.

(c) We can already see from the graph (see Figure 2) that the distribution is more spread out, it is likely that we get high values. The new p-value we have obtained is: `0.8075`. Here, the difference of means is a notably fit statistic for our null hypothesis, and it is even more convincing than the t-statistic in proving us that we should not accept it.

(d) i. For building the 1000 permutations, we have used - as suggested - MATLAB's `randperm`. We just had to do a loop to convert the integer indices to the samples.
Our new p-value is: `0.6090`.

ii. This new p-value is still high, but relatively lower than the ones of (b) and (c). It can be explained by the number of permutations (only 1000 compared to 9018009 previously): we only have a sample of what can happen, so it is normal that our value is less accurate.

iii. The duplicates can affect the p-value (and there are duplicates in our permutations): because we see multiple times a certain configuration, we give it more importance, and we bias the model into thinking that the samples should behave in this way (although it can bias it towards higher or lower p-values!).

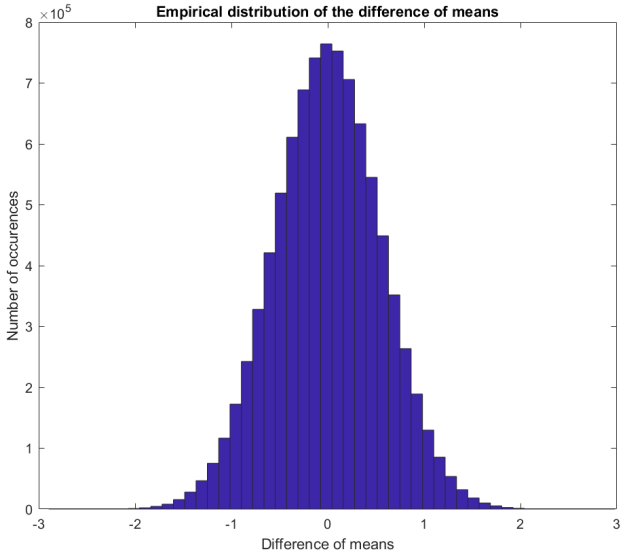

Figure 2: Empirical distribution of the difference of means

2. (a) First of all, we have to import the data. We have put it in a 16x40x40x40 array where the 8 first channels (in the first dimension) hold the data from the group 1 subjects, and the following 8 channels the data from the group 2 subjects. This way, we can process the data the same way we have done in the previous parts. We have not added noise as the maps may already be noisy. We will use the GLM model introduced in Part 1 1(c).
Our model has found that the maximum t-statistic among all the voxels is: `1.7451`.

   (b) We will rather use the approximate permutation-based p-value given that MATLAB cannot handle the full range of permutations (see 1(d) for the method). We do a permutation of the indices, and then at every voxel, we replace the indices by their actual value in the data vector. We are able to compute for 10000 permutations.

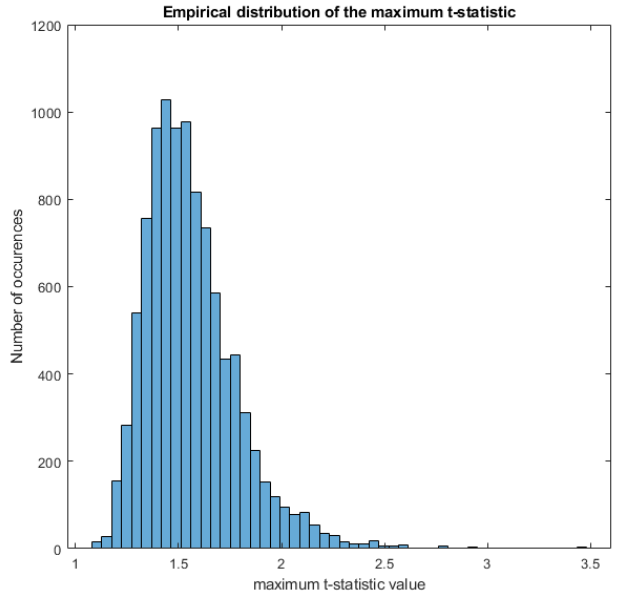

Figure 3: Empirical distribution of the maximum t-statistic

   (c) Our computation gives us a p-value of: `0.1767`. This value seems correct, because when looking at the graph, most maximum t-statistics are around 1.5, and our original p-value is relatively higher at 1.7451.

   (d) For the p-value to be 5%, we need to only retain the maximum t-statistics above a certain threshold. We will add the p-values starting from the right of the distribution (because we have to start with the highest, the p-value corresponding to the at least more extreme values). We will stop when we have collected 5% of the total number of t-statistics, and the minimum of what we will have

collected will be the threshold.
5% of 10000 is 500, our threshold maximum t-statistic is therefore located at the index $10000-500 = 9500$.
The maximum t-statistic threshold corresponding to a p-value of 5% is: `1.9787`.

Code listing:

```
close all;
clear;
clc;

%%% COMP0118
%%% Coursework 2


%% PART 1
%% Sampling
%% 1a - fixing the parameters, generating data

% Parameters of our simulation
sample_size = 20;
mean_1 = 1.5;
mean_2 = 2;
std_sample = 0.2;
rng(22182376);

sample_1 = normrnd(mean_1, std_sample, 20, 1);
sample_2 = normrnd(mean_2, std_sample, 20, 1);


%% 1a - verifying the data

disp('Sample 1');
disp('Mean : '+string(mean(sample_1)));
disp('Std : '+string(std(sample_1)));
disp('');
disp('Sample 2');
disp('Mean : '+string(mean(sample_2)));
disp('Std : '+string(std(sample_2)));


%% 1b - ttest2

disp('The result of ttest2 is: '+string(ttest2(sample_1, sample_2)));


%% 1ci - design matrix

ones_vector = ones(20, 1);
zeros_vector = zeros(20, 1);

X = [ones_vector zeros_vector; zeros_vector ones_vector];


%% 1cii - perpendicular projection operator

PX = (1/20)*[ones(20, 20) zeros(20, 20); zeros(20, 20) ones(20, 20)];


%% 1ciii - Y hat

Y_gt = [sample_1; sample_2];
```

```
e = normrnd(0, 1, 40, 1); % noise
Y = Y_gt + e;

Y_hat = PX * Y;


%% 1civ - Rx

RX = (1/20)*[20*eye(20)-ones(20, 20) zeros(20, 20); zeros(20, 20) 20*
    eye(20)-ones(20, 20)];


%% 1cv - e hat

e_hat = RX * Y;


%% 1cvi - angle Y_hat e_hat

% We start by normalizing the vectors, so we can retrieve the cosine
    of
% their angle via the dot product

Y_hat_norm = Y_hat/norm(Y_hat);
e_hat_norm = e_hat/norm(e_hat);

cos_angle = Y_hat_norm.' * e_hat_norm;
angle = acos(cos_angle);


%% 1cvii - beta_est

beta_est = (1/20)*X.'*Y;

disp('beta_est = '+string(beta_est));


%% 1cviii - stochastic component

sigma_hat_squared = (e_hat.' * e_hat)/(40-2);

sigma_hat = sqrt(sigma_hat_squared);


%% 1cix - covariance matrix

S_beta = sigma_hat_squared*pinv(X.' * X);


%% 1cxi - reduced model

X_0 = ones(40, 1);
PX_0 = (1/40)*ones(40, 40);
RX_0 = eye(40) - PX_0;

e_hat_0 = RX_0 * Y;

SSR = e_hat.' * e_hat;
SSR_0 = e_hat_0.' * e_hat_0;

% degrees of freedom
```

```
nu_1 = trace(PX - PX_0);
nu_2 = trace(eye(40) - PX);

F_stat = ((SSR_0 - SSR)/nu_1) / (SSR/nu_2);


%% 1cxii - empirical t-statistic

lambda_model = [1; -1];

t_stat = (lambda_model.' * beta_est) / (sqrt(lambda_model.' * S_beta *
    lambda_model));


%% 1cxiv - projection of e in C(X)

proj_e_cx = PX * e;


%% 1cxv - projection of e in C(X)orth

proj_e_cxorth = RX * e;


%% d
%% 1di - X

ones_vector = ones(20, 1);
zeros_vector = zeros(20, 1);

X = [ones_vector ones_vector zeros_vector; ones_vector zeros_vector
   ones_vector];


%% 1dii - PX

PX = X * pinv(X.' * X) * X.';


%% 1div

RX = eye(40) - PX;

e_hat = RX * Y;

beta_est = pinv(X.' * X)*X.'*Y;

disp('beta_est = '+string(beta_est));

sigma_hat_squared = (e_hat.' * e_hat)/(40-2);

sigma_hat = sqrt(sigma_hat_squared);

S_beta = sigma_hat_squared*pinv(X.' * X);

lambda_model = [0; 1; -1];

t_stat = (lambda_model.' * beta_est) * pinv((sqrt(lambda_model.' *
   S_beta * lambda_model)));

%% e
```

```
173
174 ones_vector = ones(20, 1);
175 zeros_vector = zeros(20, 1);
176
177 X = [ones_vector ones_vector zeros_vector; ones_vector zeros_vector
      ones_vector];
178
179 PX = X * pinv(X.' * X) * X.';
180
181 RX = eye(40) - PX;
182
183 e_hat = RX * Y;
184
185 beta_est = pinv(X.' * X)*X.'*Y;
186
187 disp('beta_est = '+string(beta_est));
188
189 sigma_hat_squared = (e_hat.' * e_hat)/(40-2);
190
191 sigma_hat = sqrt(sigma_hat_squared);
192
193 S_beta = sigma_hat_squared*pinv(X.' * X);
194
195 lambda_model = [0; 0; 1];
196
197 t_stat = (lambda_model.' * beta_est) * pinv((sqrt(lambda_model.' *
      S_beta * lambda_model)));
198
199
200 %% 2ai
201
202 disp('The result of ttest is: '+string(ttest(sample_1, sample_2)));
203
204
205 %% 2bi - new GLM model
206
207 X = zeros(40, 23);
208 X(:, 1) = [ones(1, 20) ones(1, 20)].';
209 X(:, 2) = [ones(1, 20) zeros(1, 20)].';
210 X(:, 3) = [zeros(1, 20) ones(1, 20)].';
211 X(1:20, 4:end) = eye(20);
212 X(21:end, 4:end) = eye(20);
213
214
215 %% 2bii - contrast vector
216
217 lambda_model = [0; 1; -1; zeros(20, 1)];
218
219
220 %% 2biii - t-stat
221
222 PX = X * pinv(X.' * X) * X.';
223
224 RX = eye(40) - PX;
225
226 e_hat = RX * Y;
227
228 beta_est = pinv(X.' * X)*X.'*Y;
229
230 disp('beta_est = '+string(beta_est));
231
```

```
sigma_hat_squared = (e_hat.' * e_hat)/(40-2);

sigma_hat = sqrt(sigma_hat_squared);

S_beta = sigma_hat_squared*pinv(X.' * X);

t_stat = (lambda_model.' * beta_est) * pinv((sqrt(lambda_model.' *
    S_beta * lambda_model)));



%% PART 2
%% 1a

mean_1 = 1.5;
mean_2 = 2;
std_sample = 0.2;

sample_1 = normrnd(mean_1, std_sample, 6, 1);
sample_2 = normrnd(mean_2, std_sample, 8, 1);

disp('Sample 1');
disp('Mean : '+string(mean(sample_1)));
disp('Std : '+string(std(sample_1)));
disp('');
disp('Sample 2');
disp('Mean : '+string(mean(sample_2)));
disp('Std : '+string(std(sample_2)));


%% 1a - tests

[ttest2_res, ttest2_pval] = ttest2(sample_1, sample_2);

disp('The result of ttest2 is: '+string(ttest2_res));
disp('The p-value is: '+string(ttest2_pval));


%% 1bi

D = [sample_1; sample_2];


%% 1bii - between groups permutations

perms_n1 = nchoosek(D, 6); %permutation for sample size 1 (=6)
perms_n2 = nchoosek(D, 8); %permutation for sample size 2 (=8)

D_perms = zeros(size(perms_n1, 1)*size(perms_n2, 1), 6+8);
idx_D_perms = 1;

for n1=1:size(perms_n1, 1)
    for n2=1:size(perms_n2, 1)
        D_perms(idx_D_perms, :) = [perms_n1(n1, :) perms_n2(n2, :)];
        idx_D_perms = idx_D_perms + 1;
    end
end


%% 1bii - adding noise
```

```
292
293 D_perms = D_perms + normrnd(0, 1, 9018009, 14);
294
295
296 %% 1biii - computation routine
297
298 X = [ones(6, 1) zeros(6, 1); zeros(8, 1) ones(8, 1)];
299
300 lambda_model = [1; -1];
301
302 PX = X * pinv(X.' * X) * X.';
303
304 RX = eye(14) - PX;
305
306 beta_vectors = pinv(X.' * X)*X.'*D_perms.';
307
308 numerator_full = lambda_model.' * beta_vectors;
309
310 res_vector = RX * D_perms';
311
312 sigma_squared_not_summed = D_perms.' .* res_vector;
313
314 sigma_squared_vector = sum(sigma_squared_not_summed);
315
316 pre_denominator = lambda_model.' * pinv(X.' * X) * lambda_model;
317
318 denominator_full = sqrt(pre_denominator * sigma_squared_vector);
319
320 t_stats_vector = numerator_full ./ denominator_full;
321
322
323 %% 1biii - histogram
324
325 hist(t_stats_vector, 50);
326 title('Empirical distribution of the t-statistic');
327 xlabel('t-statistic value');
328 ylabel('Number of occurences');
329
330
331 %% 1biv - p-value
332
333 original_t_stat = t_stats_vector(1);
334
335 new_p_value = sum((t_stats_vector>=original_t_stat))/(size(perms_n1,
      1)*size(perms_n2, 1));
336
337
338 %% 1c - diff of means as a stat
339
340 diff_of_means_stat = beta_vectors(1, :) - beta_vectors(2, :);
341
342
343 %% 1c - histogram
344
345 hist(diff_of_means_stat, 50);
346 title('Empirical distribution of the difference of means');
347 xlabel('Difference of means');
348 ylabel('Number of occurences');
349
350
351 %% 1c - p-value with diff of means
```

```
352
353 original_diff_of_means = diff_of_means_stat(1);
354
355 new_p_value = sum((diff_of_means_stat>=original_diff_of_means))/(size(
    perms_n1, 1)*size(perms_n2, 1));
356
357
358 %% 1d - approximate permutation-based p-val
359
360 nb_of_perms = 1000;
361
362 D_randperm = zeros(nb_of_perms, 14); %storage
363 D_randperm(1, :) = D.'; %should contain the original non-permuted
    values
364
365 for kk=2:1000
366     array_randperm = randperm(14);
367     for ll=1:14
368         D_randperm(kk, ll) = D(array_randperm(ll));
369     end
370 end
371
372 D_randperm = D_randperm + normrnd(0, 1, nb_of_perms, 14);
373
374
375 %% 1d - computation routine
376
377 X = [ones(6, 1) zeros(6, 1); zeros(8, 1) ones(8, 1)];
378 PX = X * pinv(X.' * X) * X.';
379 RX = eye(14) - PX;
380 lambda_model = [1; -1];
381 beta_vectors = pinv(X.' * X)*X.'*D_randperm.';
382 numerator_full = lambda_model.' * beta_vectors;
383 res_vector = RX * D_randperm';
384 sigma_squared_not_summed = D_randperm.' .* res_vector;
385 sigma_squared_vector = sum(sigma_squared_not_summed);
386 pre_denominator = lambda_model.' * pinv(X.' * X) * lambda_model;
387 denominator_full = sqrt(pre_denominator * sigma_squared_vector);
388 t_stats_vector = numerator_full ./ denominator_full;
389
390 %% 1d - new p_value
391
392 original_t_stat = t_stats_vector(1);
393 new_p_value = sum((t_stats_vector>=original_t_stat))/(size(D_randperm,
    1));
394
395
396 %% Last question
397 %% Importing and pre-processing of the data
398
399 % Importing the Y data
400 CPA_idx = [string(0)+string(4); string(0)+string(5); string(0)+string
    (6); string(0)+string(7); string(0)+string(8); string(0)+string(9);
    string(10); string(11)];
401 PPA_idx = [string(0)+string(3); string(0)+string(6); string(0)+string
    (9); string(10); string(13); string(14); string(15); string(16)];
402 % We will put the voxel data on top of each other
403 big_Y = zeros(16, 40, 40, 40);
404
405 for kk=1:8
406
```

```
    fid = fopen('data/CPA'+CPA_idx(kk)+'_diffeo_fa.img', 'r', 'l'); % little-endian
    dataCPA = fread(fid, 'float'); % 16-bit floating point
    dataCPA = reshape(dataCPA, [40 40 40]); % dimension 40x40x40

    big_Y(kk, :, :, :) = dataCPA;

    fid = fopen('data/PPA'+PPA_idx(kk)+'_diffeo_fa.img', 'r', 'l'); % little-endian
    dataPPA = fread(fid, 'float'); % 16-bit floating point
    dataPPA = reshape(dataPPA, [40 40 40]); % dimension 40x40x40

    big_Y(8+kk, :, :, :) = dataPPA;

end

% Importing the mask
fid = fopen('data/wm_mask.img', 'r', 'l'); % little-endian
data_mask = fread(fid, 'float'); % 16-bit floating point
data_mask = reshape(data_mask, [40 40 40]); % dimension 40x40x40


%% Computing the t-statistics

% Invariants of the routine
X = [ones(8, 1) zeros(8, 1); zeros(8, 1) ones(8, 1)];
lambda_model = [1; -1];
PX = X * pinv(X.' * X) * X.';
RX = eye(16) - PX;
pre_denominator = lambda_model.' * pinv(X.' * X) * lambda_model;

% Storage
t_stat_cube = zeros(40, 40, 40);

% Routine
for ii=1:40
    for jj=1:40
        for kk=1:40

            if(data_mask(ii, jj, kk) ~= 0) %ROI
                Y_kk = big_Y(:, ii, jj, kk);

                beta_vectors = pinv(X.' * X)*X.'*Y_kk;
                numerator_full = lambda_model.' * beta_vectors;
                res_vector = RX * Y_kk;
                sigma_squared_not_summed = Y_kk .* res_vector;
                sigma_squared_vector = sum(sigma_squared_not_summed);
                denominator_full = sqrt(pre_denominator *
                    sigma_squared_vector);
                t_stats_vector = numerator_full ./ denominator_full;
                t_stat_cube(ii, jj, kk) = t_stats_vector;
            end

        end
    end
end


%% Retrieving the biggest one

max_t_stat_all_voxels = max(abs(t_stat_cube(:)));
```

```
465
466
467 %% Permutations
468
469 nb_of_perms = 10000;
470
471 D_randperm = zeros(nb_of_perms, 16); %storage
472 D_randperm(1, :) = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]; %should
       contain the original non-permuted values
473
474 tic;
475 for kk=2:nb_of_perms
476     D_randperm(kk, :) = randperm(16);
477 end
478 toc;
479
480
481 %% Routine but with the permutations
482
483 % Invariants of the routine
484 X = [ones(8, 1) zeros(8, 1); zeros(8, 1) ones(8, 1)];
485 lambda_model = [1; -1];
486 PX = X * pinv(X.' * X) * X.';
487 RX = eye(16) - PX;
488 pre_denominator = lambda_model.' * pinv(X.' * X) * lambda_model;
489
490 % Storage
491 t_stat_cube = zeros(nb_of_perms, 40, 40, 40);
492
493 tic;
494 % Routine
495 for ii=1:40
496     for jj=1:40
497         for kk=1:40
498
499             if(data_mask(ii, jj, kk) ~= 0) %ROI
500                 Y_kk = big_Y(:, ii, jj, kk);
501                 Y_kk_perms = zeros(nb_of_perms, 16);
502                 % Building the permuted matrix
503                 for ll=1:nb_of_perms
504                     for zz=1:16
505                         Y_kk_perms(ll, zz) = Y_kk(D_randperm(ll, zz));
506                     end
507                 end
508
509                 beta_vectors = pinv(X.' * X)*X.'*Y_kk_perms.';
510                 numerator_full = lambda_model.' * beta_vectors;
511                 res_vector = RX * Y_kk_perms.';
512                 sigma_squared_not_summed = Y_kk_perms.' .* res_vector;
513                 sigma_squared_vector = sum(sigma_squared_not_summed);
514                 denominator_full = sqrt(pre_denominator *
                       sigma_squared_vector);
515                 t_stats_vector = numerator_full ./ denominator_full;
516                 t_stat_cube(:, ii, jj, kk) = t_stats_vector;
517             end
518
519         end
520     end
521 end
522 toc;
523
```

```
524
525 %% Finding the maximum t-stat among all voxels for every permutation
526
527 t_stat_empirical_perms = zeros(1, nb_of_perms);
528
529 for kk=1:nb_of_perms
530     t_stat_cube_slice = t_stat_cube(kk, :, :, :);
531     t_stat_empirical_perms(1, kk) = max(abs(t_stat_cube_slice(:)));
532 end
533
534
535 %% Histogram
536
537 histogram(t_stat_empirical_perms, 50);
538 title('Empirical distribution of the maximum t-statistic');
539 xlabel('maximum t-statistic value');
540 ylabel('Number of occurences');
541
542
543 %% p-value
544
545 new_p_value = sum((t_stat_empirical_perms>=max_t_stat_all_voxels))/(
       nb_of_perms);
546
547
548 %% maximum t-statistic threshold
549
550 % To retrieve the histogram distribution, we need to sort our
551 % t_stat_empirical_perms array
552
553 t_stat_empirical_perms_sorted = sort(t_stat_empirical_perms);
554
555 % 5% of 10000 is 500 => we are looking for the index 9500
556 disp('The maximum t-statistic threshold value is: '+string(
       t_stat_empirical_perms_sorted(9500)));
557
558
559 % End
```