```matlab
%% Part 1
%% 1.1.a

% Select the sample size for both groups
sample_size = 20;

% Mean of the first group
mean1 = 1.5;

% Mean of the second group
mean2 = 2;

% Stochastic random component
mean_moise = 0;
sigma_noise = 0.2;

% Fix the random seed
rng(2500294)

% Compute the noises for both groups
noise1 = sigma_noise * randn(sample_size, 1);
noise2 = sigma_noise * randn(sample_size, 1);

% Responses
sample1 = mean1 + noise1;
sample2 = mean2 + noise2;

% Compute mean and standard deviation for the first sample
mean_sample1 = mean(sample1)
std_sample1 = std(sample1)

% Compute mean and standard deviation for the second sample
mean_sample2 = mean(sample2)
std_sample2 = std(sample2)

%% 1.1.b

% Compute the two-sample t-statistic
[h, ~, ~, stats] = ttest2(sample1, sample2)

%% 1.1.c.i

% Compute the design matrix
X1 = [ones(sample_size, 1); zeros(sample_size, 1)];
X2 = [zeros(sample_size,1); ones(sample_size, 1)];
X = [X1, X2]

% Dimension of the column space C(X)
dimX = rank(X)

%% 1.1.c.ii

% Compute the perpendicular projection operator
```

```
PX = X * inv(X' * X) * X'

% Compute the trace of PX
trace_PX = trace(PX)

% Verify the key properties of a perpendicular projection operator
idempotence = norm(PX * PX - PX, 'fro'); % Should be 0
symmetry = norm(PX - PX', 'fro'); % Should be 0

%% 1.1.c.iii

% Total response
Y = [sample1; sample2];

% Determine the projection of Y into C(C)
Y_hat = PX * Y

%% 1.1.c.iv

% Identity matrix
I = eye(size(PX));

% Compute Rx
RX = I - PX;

% Verify the key properties of a perpendicular projection operator
idempotence = norm(RX * RX - RX, 'fro'); % Should be 0
symmetry = norm(RX - RX', 'fro'); % Shou8ld be 0

%% 1.1.c.v

% Determine e_hat
e_hat = RX * Y

% Determine the dimension of C(X)⊥
dimX_perp = trace(I - PX);

%% 1.1.c.vi

% Determine the angle between e_hat and Y_hat in degrees
theta = acos( e_hat' * Y_hat / (norm(e_hat) * norm(Y_hat))) * (180 / pi)

%% 1.1.c.vii

% Determine the estimate to the model parameters of the GLM
beta_hat = inv(X' * X) * X' * Y

%% 1.1.c.viii

% Estimate the variance of the stochastic component e_hat
sigma2 = (e_hat' * e_hat) / (2*sample_size - dimX)

%% 1.1.c.ix
```

```
% Estimate the covariance matrix of beta_hat
cov_beta = sigma2 * inv(X' * X)

% Determine the standard deviation of the model parameters
std_beta_hat = sqrt(diag(cov_beta))

%% 1.1.c.x

% Contrast vector
lambda = [1; -1];

% Reduced model
X0 = ones(size(Y));

%% 1.1.c.xi

% Compute the perpendicula projection operator for the reduced model
PX0 = X0 * inv(X0' * X0) * X0';

% Compute errors for the original model
SSR_X = e_hat' * e_hat;

% Compute errors for the reduced model
e_hat_X0= Y - PX0 *Y;
SSR_X0 = e_hat_X0' * e_hat_X0;

% Compute the additional error as a result of the constraint
add_SSR = SSR_X0 - SSR_X

% Compute the degrees of freedom
v1 = trace(PX-PX0)
v2 = trace(eye(size(PX))-PX)

% Estimate the F-statistic
F_stat = (add_SSR / v1) / (SSR_X / v2)

%% 1.1.c.xii

% Determine the t-statistic
t_stat =  lambda' * beta_hat / sqrt(lambda' * cov_beta * lambda)


%% 1.1.c.xiv

% Gound truth deviation e
e_real = [noise1; noise2];

% Projection of e into C(X)
e_proj = PX * e_real

%% 1.1.c.xv
```

```
% Projection of e into C(X)⊥
e_proj_perp = (I - PX) * e_real

%% 1.1.d.i

% Compute the design matrix
X = [ones(2 * sample_size,1), X1, X2];

% Dimension of column space
dimX = rank(X)

%% 1.1.d.ii

% Compute the perpendicular projection operator
PX = X * pinv(X' * X) * X'

%% 1.1.d.iii

% Reduced model
X0 = [ones(2* sample_size,1), X1 + X2];

% Contrast vector
lambda = [0; 1; -1];

% Compute the perpendicular projection operator for the reduced model
PX0 = X0 * pinv(X0' * X0) * X0';


%% 1.1.d.iv

% Determine the estimate to the model parameters of the GLM
beta_hat = pinv(X' * X) * X' * Y;

% Identity matrix
I = eye(size(PX));

% Compute R_X
RX = I - PX;

% Compute e_hat
e_hat = RX * Y;

% Estimate the variance of the stochastic component e_hat
sigma2 = (e_hat' * e_hat) / (2* sample_size - trace(PX));

% Estimate the covariance matrix of B_hat
cov_beta = sigma2 * pinv(X' * X);

% Determine the t-statistic
t_stat = lambda' * beta_hat / sqrt(lambda' * cov_beta * lambda)

%% 1.1.e.i
```

```
% Compute the design matrix
X = [ones(2*sample_size,1), X1];

% Dimension of column space
dim_CX = rank(X);

%% 1.1.e.ii

% Contrast vector
lambda = [0; 1];

% Reduced model
X0 = ones(2*sample_size,1);

%% 1.1.e.iii

% Determine the estimate to the model parameters of the GLM
beta_hat = pinv(X' * X) * X' * Y;

% Identity matrix
I = eye(size(PX));

% Compute R_X
RX = I - PX;

% Compute e_hat
e_hat = RX * Y;

% Estimate the variance of the stochastic component e_hat
sigma2 = (e_hat' * e_hat) / (2* sample_size - trace(PX));

% Estimate the covariance matrix of B_hat
cov_beta = sigma2 * pinv(X' * X);

% CDetermine the t-statistic
t_stat = lambda' * beta_hat / sqrt(lambda' * cov_beta * lambda)


%% 1.2.a

% Compute the paired t-statistic
[h, ~, ~, stats] = ttest(sample1, sample2)

%% 1.2.b.i

% Constant variable
X0 = ones(2 * sample_size, 1);

% Explanatory variable for indicating different time points
X1 = [zeros(sample_size,1); ones(sample_size,1)];

% Construction of the matrix for the dummy variable for indicating if an observation ↙
is made on the
```

```
% subject i
S = eye(sample_size);
S = repmat(S, 2, 1);

% Design matrix
X = [X0, X1, S];
rank(X)

%% 1.2.b.ii

% Contrast vector
lambda = [0; 1; zeros(sample_size,1)]

%% 1.2.b.iii

% Compute the perpendicular projection operator
PX = X * pinv(X' * X) * X';

% Determine the estimate to the model parameters of the GLM
beta_hat = pinv(X' * X) * X' * Y;

% Identity matrix
I = eye(size(PX));

% Compute R_X
RX = I - PX;

% Compute e_hat
e_hat = RX * Y;

% Estimate the variance of the stochastic component e_hat
sigma2 = (e_hat' * e_hat) / (2* sample_size - trace(PX));

% Estimate the covariance matrix of B_hat
cov_beta = sigma2 * pinv(X' * X);

% CDetermine the t-statistic
t_stat = lambda' * beta_hat / sqrt(lambda' * cov_beta * lambda)


%% Part 2
%% 2.1.a

% Select the sample size for the first group
n1 = 6;

% Select the sample sixe for the second group
n2 = 8;

% Mean of the first group
mean1 = 1.5;

% Mean of the second group
```

```
mean2 = 2;

% Stochastic random component
mean_noise = 0;
sigma_noise = 0.2;

% Fix the random seed
rng(2500294)

% Compute the noises for both groups
noise1 = sigma_noise * randn(n1, 1);
noise2 = sigma_noise * randn(n2, 1);

% Responses
sample1 = mean1 + noise1;
sample2 = mean2 + noise2;

% Compute mean and standard deviation for the first sample
mean_sample1 = mean(sample1);
std_sample1 = std(sample1);

% Compute mean and standard deviation for the second sample
mean_sample2 = mean(sample2);
std_sample2 = std(sample2);

% Determine the t-statistic and p-value
[h, p_value_observed, ~, stats] = ttest2(sample1, sample2)
t_observed = stats.tstat

%% 2.1.b.i

% Create an array to store the observations for both groups
D = [sample1; sample2];
sample_size = length(D);

%% 2.1.b.ii

% Construct all the valid permutations of D maintaning the sample size of
% each group
permutations = nchoosek(1:sample_size, n1);
num_permutations = size(permutations, 1);
t_statistics = zeros(num_permutations, 1);

%% 2.1.b.iii

% Compute the t-statistics for all the permutations
for i = 1:num_permutations
    permutation_index = permutations(i, :);
    permutation_group1 = D(permutation_index);
    permutation_group2 = D(setdiff(1:sample_size, permutation_index));
    [~, ~, ~, permutation_stats] = ttest2(permutation_group1, permutation_group2);
    t_statistics(i) = permutation_stats.tstat;
end
```

```
%% 2.1.b.iv

% Determine the p-value
p_value = sum(abs(t_statistics) >= abs(t_observed)) / length(t_statistics)


%% 2.1.c

% Initialize
means_differences = zeros(num_permutations, 1);
mean_diff_observed = mean_sample1-mean_sample2

% Compute the difference between the means for all the permutations
for i = 1:num_permutations
    permutation_index = permutations(i, :);
    permutation_group1 = D(permutation_index);
    permutation_group2 = D(setdiff(1:sample_size, permutation_index));
    means_differences(i) = mean(permutation_group1) - mean(permutation_group2);
end

% Compute the p-value
p_value_permutation = sum(abs(means_differences) >= abs(mean_diff_observed)) / ↙
length(means_differences)

%% 2.1.d.i

% number of permutations


% Calcolo del t-statistic osservato
[h, p_value_observed, ~, stats] = ttest2(sample1, sample2);
t_observed = stats.tstat;

num_permutations = 1000;

% Initialize
t_statistics = zeros(num_permutations, 1);

% Always include the original labeling
t_statistics(1) = t_observed;

% compute the t-statistic for 999 random permutations
for i = 2:num_permutations
    permutation_index = randperm(sample_size);
    permutation_group1 = D(permutation_index(1:n1));
    permutation_group2 = D(permutation_index(n1+1:end));
    [~, ~, ~, permutation_stats] = ttest2(permutation_group1, permutation_group2);
    t_statistics(i) = permutation_stats.tstat;
end

% Calcolo del p-value approssimato
p_value = sum(abs(t_statistics) >= abs(t_observed)) / length(t_statistics)
```

```
%% 2.1.d.iii

% Check if there are any duplicates in the 1000 permutations
unique_permutations = unique(t_statistics);
num_duplicates = num_permutations - length(unique_permutations)

%% 2.2.a

% Initialize
dim = 40;
% Load ROI mask
fid = fopen('wm_mask.img', 'r', 'l');
wm_mask = fread(fid, 'float');
fclose(fid);
wm_mask = reshape(wm_mask, [dim, dim, dim]);

% Number of subjects
n1 = 8;
n2 = 8;
n = n1 + n2;

% Load FA images
FA_group1 = zeros(dim,dim,dim,n1);
FA_group2 = zeros(dim,dim,dim,n2);

% Load the maps for group 1
for i = 1:n1
    filename = sprintf('CPA%d_diffeo_fa.img', i);
    fid = fopen(filename, 'r', 'l');
    data = fread(fid, 'float');
    fclose(fid);
    FA_group1(:,:,:,i) = reshape(data, [dim, dim, dim]);
end

% Load the maps for group 2
for i = 1:n2
    filename = sprintf('PPA%d_diffeo_fa.img', i);
    fid = fopen(filename, 'r', 'l');
    data = fread(fid, 'float');
    fclose(fid);
    FA_group2(:,:,:,i) = reshape(data, [dim, dim, dim]);
end

% Construct the design matrix
X = [ones(n1,1), zeros(n1,1); zeros(n2,1), ones(n2,1)];

% Compute the two-sample t-statistic
t_values = Two_Sample_T_Stat(FA_group1, FA_group2, X,wm_mask, dim, dim, dim);

% Find the maximum t-statistic
[max_val, linear_idx] = max(t_values(:))
[i_max, j_max, k_max] = ind2sub(size(t_values), linear_idx);
```

```
%% 2.2.b

% Construct all the valid permutations maintaning the sample size of
% each group
perms = nchoosek(1:n, n1);
num_permutations = size(perms, 1);

% Initialize
max_t_val = zeros(num_permutations, 1);

% Concatenate the two groups
D = cat(4, FA_group1, FA_group2);

% Compute the two-sample t-statistic for all the permutations
parfor p = 1:num_permutations
    perm_idx = perms(p, :);
    FA_group1 = D(:, :, :, perm_idx);
    FA_group2 = D(:, :, :, setdiff(1:n, perm_idx));
    t_values = Two_Sample_T_Stat(FA_group1, FA_group2, X, wm_mask, dim, dim, dim);

    % Find the maximum t-statistic
    max_t_val(p) = max(t_values(:));
end

% Visualize the distribution
histogram(max_t_val, 50);
xlabel('Maximum t-statistic');
ylabel('Frequency');
title('Empirical Distribution of Maximum t-statistic' );

%% 2.2.c

% Determine the multiple-comparisons-corrected p-value
p_value_corrected = sum(max_t_val(:, : , :) >= max_val) / length(max_t_val)

%% 2.2.d

% Determine the maximum t-statistic threshold corresponding to p-value of 5%
p_value_threshold = prctile(max_t_val, 95)

%% Function

function t_values = Two_Sample_T_Stat(FA_group1, FA_group2, X, wm_mask, dim1, dim2, ↙
dim3)
%
% This function computes the voxel-wise two-sample t-statistic for comparing
% FA (Fractional Anisotropy) values between two subject groups using the
% General Linear Model (GLM).
%
% The GLM used is: Y = X1β1 + X2β2 + e
%
% INPUTS:
```

```
%   - FA_group1: 4D array (dim1 x dim2 x dim3 x n1) containing FA values for group 1
%   - FA_group2: 4D array (dim1 x dim2 x dim3 x n2) containing FA values for group 2
%   - X: Design matrix of size (n1 + n2) x 2, encoding group membership
%   - dim1, dim2, dim3: Dimensions of the 3D brain volume
%
% OUTPUT:
%   - t_values: 3D array (dim1 x dim2 x dim3) containing computed t-statistics for ↙
each voxel

% Initialize
t_values = zeros(dim1, dim2 , dim3);

% Compute the perpendicular projection operator
PX = X * pinv(X' * X) * X';

% Compute Rx
RX = eye(size(PX)) - PX;

% Compute the two-sample t-statistic
for i = 1 : dim1
    for j = 1 : dim2
        for k = 1 : dim3
            if wm_mask(i, j, k) > 0

                % Total response
                Y = [squeeze(FA_group1(i, j, k, :)); squeeze(FA_group2(i, j, k, ↙
:))];

                % Determine e_hat
                e_hat = RX * Y;

                % Determine the estimate to the model parameters of the GLM
                beta_hat = pinv(X' * X) * X' * Y;

                % Estimate the variance of the stochastic component e_hat
                sigma2 = sum(e_hat.^2) / (size(Y,1) - rank(X));

                % Estimate the covariance matrix of beta_hat
                cov_beta = sigma2 * pinv(X' * X);

                % Contrast vector
                lambda = [1; -1];

                % Determine the t-statistic
                t_values(i, j, k) = (lambda' * beta_hat) ./ sqrt(lambda' * cov_beta ↙
* lambda);
            end
        end
    end
end
end
```