

Hypothesis testing

Contents

1	Student's t-test	2
1.1	Using the functions	2
1.2	t -test function	3
1.3	Right tailed test	4
1.4	Left tailed test	8
1.5	Two tailed test	11
2	F-test	15
2.1	Using the functions	16
2.2	F-test function	17
2.3	Right tailed test	18
2.4	Left tailed test	22
2.5	Two tailed test	25

1 Student's t -test

The main purpose of this document is to make difficult aspects of hypothesis testing easier to comprehend, in particular the significance levels and the values from the statistical table containing critical values. This table is also called the t table. The critical values are required to reject the null hypothesis of a statistical test. To make understanding of significance levels and critical values topics easier to understand, MATLAB is used to write functions that visually display these subjects. In the function files we will use the limiting distributions of the corresponding statistical tests.

When manually using the statistical table, two values are required to find the correct critical value of the hypothesis test in this table. The two required inputs are the significance level and the number of observations minus one, which is also known as the degrees of freedom. Therefore, the minimal information required for the functions to work are the significance level and the degrees of freedom. After filling in these values and running the code, a graph displaying the limiting distribution and the value from the statistical table will be shown. The user also has the option to give the manually calculated test statistic as input to the function file. In this case, this test statistic will also be displayed, from which the user is able to see if the corresponding null hypothesis can be rejected.

All the code can be found [here](#) on GitHub.

1.1 Using the functions

To help the learner understand the code, the `TTesting.m` file has been written. This file explains what input is required for the function and offers the ability to run the function files.

The t -test can be right tailed, left tailed or two tailed. The side of the test depends on the formulation of the null and alternative hypotheses. When the hypotheses are formulated as

$$\begin{aligned} H_0 : \hat{\beta}_1 &= \beta_1^0 \\ H_1 : \hat{\beta}_1 &\neq \beta_1^0 \end{aligned} \tag{1}$$

a two tailed test is required. In case that the alternative hypothesis H_1 is $\hat{\beta}_1 < \beta_1^0$, a left tailed test needs to be performed. The last case is a right tailed test, which is when the alternative hypothesis H_1 is formulated as $\hat{\beta}_1 > \beta_1^0$.

```
1 % -----
2 % When using TTest(SIDE, NU, ALPHA, TSTAT),
3 %
4 % SIDE indicates the side of the hypothesis test. Input for SIDE can be
5 % any of the three following:
6 %   - 'RightSided' for a right-tailed t-test
7 %   - 'LeftSided' for a left-tailed t-test
8 %   - 'TwoSided' for a two-tailed t-test
9 %
10 % NU is the degrees of freedom and can be any positive valued integer.
11 %
12 % ALPHA denotes the significance level. The input for ALPHA can be any
13 % real valued number on the interval (0,1).
14 %
15 % TSTAT is the manually derived t-statistic, which can take on any finite
16 % value. This input argument is optional and in case input is given, the
17 % function plots the manually calculated test statistic valued TSTAT and
```

```

18 % show whether the null hypothesis can or cannot be rejected.
19 %
20 % The size of the plot is predetermined and can be customized by changing
21 % the values of TTR.scale, TTL.scale, TTTS.scale in their corresponding
22 % function files.
23 %
24 % TTest(SIDE, NU, ALPHA, TSTAT)
25 % -----
26 % Example
27 TTest('TwoSided', 35, 0.05, -2.3)

```

As can be observed, an example command has been written on the last line of the function file. With that line of code, we visualize a two tailed t -test with degrees of freedom $\nu = 35$, a significance level $\alpha = 0.05$ and a test statistic valued -2.3 . The result is the following graph:

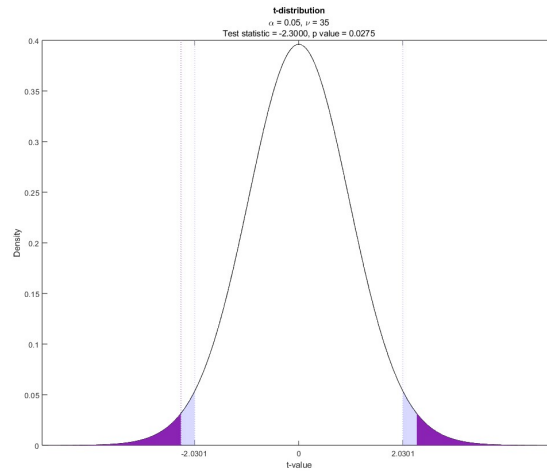


Figure 1: t -test example

1.2 t -test function

When a statistics learner wants to perform the hypothesis test corresponding to equation (1), they will use the Student's t -test. The function `TTest.m` helps the learner's understanding of the two difficult subjects by visualizing the underlying statistical meaning and motivation of the tests. The function `TTest.m` checks what side of the test the user wants to perform and then 'sends' the task to the corresponding function.

```

1 function TT = TTest(side, nu, alpha, tstat)
2 %TTEST Visualize a Student's t-test
3 % TTest(SIDE, NU, ALPHA, TSTAT) plots the theoretical
4 % Student's t-distribution with NU degrees of freedom. Depending on the
5 % input argument for SIDE, it calculates one or two critical values
6 % corresponding to a one or two sided t-test with NU degrees of freedom
7 % at an ALPHA level of significance and plots the related rejection
8 % region(s). A vertical line representing the manually calculated
9 % test statistic valued TSTAT will be plotted as well, this input
10 % argument is optional.

```

```

11 %
12 %     SIDE can be:
13 %     'RightSided',
14 %     'LeftSided',
15 %     'TwoSided'.
16
17 % -----
18 % Check if the user does not want to plot the test statistic, which
19 % happens when the number of input arguments is equal to three.
20 % -----
21 if (nargin == 3)
22     switch(side)
23         case "RightSided"
24             TTestRight(nu, alpha);
25         case "LeftSided"
26             TTestLeft(nu, alpha);
27         case "TwoSided"
28             TTestTwoSided(nu, alpha);
29         otherwise
30             uiwait(warndlg(['Please specify the correct side, right, ' ...
31                             'left or two sided']))
32             return
33     end
34 else
35     switch(side)
36         case "RightSided"
37             TTestRight(nu, alpha, tstat);
38         case "LeftSided"
39             TTestLeft(nu, alpha, tstat);
40         case "TwoSided"
41             TTestTwoSided(nu, alpha, tstat);
42         otherwise
43             uiwait(warndlg(['Please specify the correct side, right, ' ...
44                             'left or two sided']))
45             return
46     end
47 end

```

1.3 Right tailed test

When the alternative hypothesis H_1 is $\hat{\beta}_1 > \beta_1^0$, **TTestRight.m** can be used. A significance level of $\alpha = 0.05$ is commonly used in hypothesis testing. This means in for a right sided test, we want to find the 95th percentile.

When calculating the critical value of the t test, the inverse cumulative distribution function (ICDF) needs to be used. The ICDF is also known as the quantile function. The inverse CDF tells us what value of x is needed to make $F_X(x)$ return our required probability p .

We use this ICDF to calculate the critical value since we want to find what value of the test statistic is at least or at most required (depending on the alternative hypothesis) to reject the null hypothesis. Since we make use of the Student's t -distribution, the inverse cumulative distribution function also requires the degrees of freedom as input.

```

1 function TTR = TTestRight(nu, alpha, tstat)

```

```

2 %TTESTRIGHT Visualize a right sided Student's t-test
3 %   TTR = TTestRight(NU, ALPHA, TSTAT) plots the theoretical
4 %   student's t-distribution with NU degrees of freedom. It calculates the
5 %   critical value corresponding to a right sided t-test with NU degrees
6 %   of freedom at an ALPHA level of significance and plots the related
7 %   rejection region. A vertical line representing the manually calculated
8 %   test statistic valued TSTAT will be plotted, this input argument is
9 %   optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu <= 0)
15     uiwait(warndlg('The degrees of freedom should be larger than zero.'));
16     return
17 elseif (mod(nu, 1) ~= 0)
18     uiwait(warndlg(['Please fill in an integer for the degrees of ' ...
19         'freedom.']));
20     return
21 elseif (alpha <= 0 || alpha >= 1)
22     uiwait(warndlg(['Please fill in a value of alpha between zero and '...
23         'one.']));
24     return
25 end
26
27 % -----
28 % Check if the user wants to plot the test statistic and whether the input
29 % is valid.
30 % -----
31 if (nargin == 3)
32     TTR.Display = 1;
33     if (tstat < 0)
34         uiwait(warndlg(['The test statistic is negative valued.' ...
35             ' If this is intentional, please make use of the left' ...
36             ' tailed test.']));
37         return
38     end
39 else
40     TTR.Display = 0;
41 end
42
43 % -----
44 % Calculating the critical value.
45 % -----
46 TTR.CV = icdf('T', 1-alpha, nu);
47
48 % -----
49 % Determining the length of the horizontal axis, which depends on the
50 % critical value as having only one or two degrees of freedom can result
51 % in a large critical value. In most cases however, the critical value is
52 % relatively small and [-5, 5] is a good interval to display the curve of
53 % the student's t-distribution and the crititcal value. xmin uses -TTR.CV
54 % as the critical value is positive by construction.
55 % -----

```

```

56 TTR.xmin = min([-TTR.CV-1 -5]);
57 TTR.xmax = max([TTR.CV+1 5]);
58 TTR.x = TTR.xmin:0.01:TTR.xmax;
59
60 % -----
61 % Creating the density.
62 % -----
63 TTR.y = pdf('T', TTR.x, nu);
64
65 % -----
66 % Calculating the rejection region, as the area needs to be shown in the
67 % plot
68 % -----
69 TTR.xright = TTR.CV:0.001:TTR.xmax;
70 TTR.yright = pdf('T', TTR.xright, nu);
71
72 % -----
73 % Setting up the plot. To create a subtitle consisting of two lines, the
74 % sprintf() function is used in the subtitle() function. The subtitle will
75 % be split up in two lines when the user wants to plot the test statistic
76 % and calculate the p value. TTR.alphadec is used to determine the number
77 % of decimals for displaying alpha, which depends on the user and hence is
78 % dynamic. TTR.nodect is used for the degrees of freedom, which don't have
79 % decimals. The code then asks for the size of the monitor of the user to
80 % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
81 % used. TTR.scale scales the graph with respect to the monitor size of the
82 % user. xticks is used as it is necessary to show the exact critical value
83 % on the horizontal axis.
84 % -----
85 TTR.alphadec = sprintf('%%.%df', ...
86     length(char(extractAfter(string(alpha), '.'))));
87 TTR.nodect = sprintf('%%.%df', 0);
88 TTR.variables = sprintf(['\\alpha = ', TTR.alphadec, ', \\nu = ', ...
89     TTR.nodect], alpha, nu);
90
91 TTR.mp = get(0, 'MonitorPositions');
92 TTR.mwidth = TTR.mp(1, 3);
93 TTR.mheight = TTR.mp(1, 4);
94 TTR.scale = 0.45;
95
96 TTR.gwidth = TTR.scale*TTR.mwidth;
97 TTR.gheight = 0.75*TTR.gwidth;
98 TTR.x0 = 0.5*TTR.mwidth*(1 - TTR.scale);
99 TTR.y0 = (TTR.mheight - TTR.gheight - 84)*0.5;
100
101 figure
102 plot(TTR.x,TTR.y,'-black');
103 xticks([0 TTR.CV]);
104 title("t-distribution");
105 subtitle({TTR.variables}, 'Interpreter', 'tex');
106 xlabel("t-value");
107 ylabel("Density");
108 TTR.fig = gcf;
109 TTR.fig.Position = [TTR.x0, TTR.y0, TTR.gwidth, TTR.gheight];

```

```

110 hold on
111
112 % -----
113 % Marking the critical values in the plot.
114 % -----
115 xline([TTR.CV], 'LineStyle', ':', 'Color', '#9a9afc', 'LineWidth', 1.4);
116
117 % -----
118 % Filling the areas of the rejection region.
119 % -----
120 TTR.ar = area(TTR.xright, TTR.yright);
121 TTR.ar.FaceColor = 'blue';
122 TTR.ar.FaceAlpha = 0.15;
123 TTR.ar.EdgeColor = 'none';
124
125 % -----
126 % The user has the option to also plot the self calculated test statistic
127 % and compute the corresponding p value. This part of the code will only
128 % run when there is input for the third argument (the value of the test
129 % statistic). In the case that there is no input for the third argument,
130 % the p value will not be calculated and the function has finished
131 % running.
132 %
133 % The first nested if else statement will check, on the condition that the
134 % user gave input for the third argument, if the test statistic is smaller
135 % than the critical value. If this is the case, the null hypothesis cannot
136 % be rejected and a vertical light purple dotted line corresponding to the
137 % value of the test statistic and a light purple shaded area will be added
138 % to the plot. Else, the null can be rejected and a purple vertical dotted
139 % line corresponding to the value of the test statistic and a dark purple
140 % shaded area will be plotted.
141 %
142 % Afterwards the subtitle is updated and the code has finished
143 % running.
144 % -----
145 if (TTR.Display == 1)
146     if (tstat < TTR.CV)
147         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
148             1.4);
149         TTR.tint = tstat:0.001:TTR.CV;
150         TTR.ty = pdf('T', TTR.tint, nu);
151         TTR.tar = area(TTR.tint, TTR.ty);
152         TTR.tar.FaceColor = '#8a22b3';
153         TTR.tar.FaceAlpha = 0.04;
154         TTR.tar.EdgeColor = 'none';
155     else
156         xline(tstat, 'LineStyle', ':', 'Color', '#8a22b3', 'LineWidth', ...
157             1.4);
158         TTR.tint = tstat:0.001:TTR.xmax;
159         TTR.ty = pdf('T', TTR.tint, nu);
160         TTR.tar = area(TTR.tint, TTR.ty);
161         TTR.tar.FaceColor = '#8a22b3';
162         TTR.tar.FaceAlpha = 1;
163         TTR.tar.EdgeColor = 'none';

```

```

164     end
165     TTR.pval = 1-cdf('T', tstat, nu);
166     TTR.empdec = sprintf('%.%df', 4);
167     TTR.pdec = sprintf('%.%df', 2);
168     TTR.tp = sprintf(['Test statistic = ', TTR.empdec, ', p value = ', ...
169         TTR.pdec], tstat, TTR.pval);
170     subtitle({TTR.variables, TTR.tp}, 'Interpreter', 'tex');
171 end

```

1.4 Left tailed test

Instead of a right tailed test, it is also possible to perform a left tailed test. The alternative hypothesis H_1 then becomes $\hat{\beta}_1 < \beta_1^0$. Keeping the significance level of $\alpha = 0.05$, we now want to find the 5th percentile. Due to symmetry of the t -distribution at zero, this 5th percentile has the same value as the negative value of the 95th percentile, which is used in the function `TTestLeft.m`.

```

1 function TTL = TTestLeft(nu, alpha, tstat)
2 %TTESTLEFT Visualize a left sided Student's t-test
3 %   TTL = TTestLeft(NU, ALPHA, TSTAT) plots the theoretical
4 %   student's t-distribution with NU degrees of freedom. It calculates the
5 %   critical value corresponding to a left sided t-test with NU degrees of
6 %   freedom at an ALPHA level of significance and plots the related
7 %   rejection region. A vertical line representing the manually calculated
8 %   test statistic valued TSTAT will be plotted, this input argument is
9 %   optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu <= 0)
15     uiwait(warndlg('The degrees of freedom should be larger than zero.'));
16     return
17 elseif (mod(nu, 1) ~= 0)
18     uiwait(warndlg(['Please fill in an integer for the degrees of ' ...
19         'freedom.']));
20     return
21 elseif (alpha <= 0 || alpha >= 1)
22     uiwait(warndlg(['Please fill in a value of alpha between zero and ' ...
23         'one.']));
24     return
25 end
26
27 % -----
28 % Check if the user wants to plot the test statistic and whether the input
29 % is valid.
30 % -----
31 if (nargin == 3)
32     TTL.Display = 1;
33     if (tstat > 0)
34         uiwait(warndlg(['The test statistic is positive valued.' ...
35             ' If this is intentional, please make use of the right' ...
36             ' tailed test.']));

```



```

37         return
38     end
39 else
40     TTL.Display = 0;
41 end
42
43 % -----
44 % Calculating the critical value.
45 % -----
46 TTL.CV = -icdf('T', 1-alpha, nu);
47
48 % -----
49 % Determining the length of the horizontal axis, which depends on the
50 % critical value as having only one or two degrees of freedom can result
51 % in a large critical value. In most cases however, the critical value is
52 % relatively small and [-5, 5] is a good interval to display the curve of
53 % the student's t-distribution and the critical values. xmax uses -TTL.CV
54 % as the critical value is negative by construction.
55 % -----
56 TTL.xmin = min([TTL.CV-1 -5]);
57 TTL.xmax = max([-TTL.CV+1 5]);
58 TTL.x = TTL.xmin:0.01:TTL.xmax;
59
60 % -----
61 % Creating the density.
62 % -----
63 TTL.y = pdf('T', TTL.x, nu);
64
65 % -----
66 % Calculating the rejection region, since this area needs to be shown in
67 % the plot.
68 % -----
69 TTL.xleft = TTL.xmin:0.001:TTL.CV;
70 TTL.yleft = pdf('T', TTL.xleft, nu);
71
72 % -----
73 % Setting up the plot. To create a subtitle consisting of two lines, the
74 % sprintf() function is used in the subtitle() function. The subtitle will
75 % be split up in two lines when the user wants to plot the test statistic
76 % and calculate the p value. TTL.alphadec is used to determine the number
77 % of decimals for displaying alpha, which depends on the user and hence is
78 % dynamic. TTL.nodect is used for the degrees of freedom, which don't have
79 % decimals. The code then asks for the size of the monitor of the user to
80 % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
81 % used. TTL.scale scales the graph with respect to the monitor size of the
82 % user. xticks is used as it is necessary to show the exact critical value
83 % on the horizontal axis.
84 % -----
85 TTL.alphadec = sprintf('%.%df', ...
86     length(char(extractAfter(string(alpha), '.'))));
87 TTL.nodect = sprintf('%.%df', 0);
88 TTL.variables = sprintf(['\\alpha = ', TTL.alphadec, ', \\nu = ', ...
89     TTL.nodect], alpha, nu);
90

```

```

91 TTL.mp = get(0, 'MonitorPositions');
92 TTL.mwidth = TTL.mp(1, 3);
93 TTL.mheight = TTL.mp(1, 4);
94 TTL.scale = 0.45;
95
96 TTL.gwidth = TTL.scale*TTL.mwidth;
97 TTL.gheight = 0.75*TTL.gwidth;
98 TTL.x0 = 0.5*TTL.mwidth*(1 - TTL.scale);
99 TTL.y0 = (TTL.mheight - TTL.gheight - 84)*0.5;
100
101 figure
102 plot(TTL.x,TTL.y,'-black');
103 xticks([TTL.CV 0]);
104 title("t-distribution");
105 subtitle({TTL.variables}, 'Interpreter', 'tex');
106 xlabel("t-value");
107 ylabel("Density");
108 TTL.fig = gcf;
109 TTL.fig.Position = [TTL.x0, TTL.y0, TTL.gwidth, TTL.gheight];
110 hold on
111
112 % -----
113 % Marking the critical value in the plot.
114 % -----
115 xline([TTL.CV], 'LineStyle', ':', 'Color', '#9a9afc', 'LineWidth', 1.4);
116
117 % -----
118 % Filling the area of the rejection region.
119 % -----
120 TTL.ar = area(TTL.xleft, TTL.yleft);
121 TTL.ar.FaceColor = 'blue';
122 TTL.ar.FaceAlpha = 0.15;
123 TTL.ar.EdgeColor = 'none';
124
125 % -----
126 % The user has the option to also plot the self calculated test statistic
127 % and compute the corresponding p value. This part of the code will only
128 % run when there is input for the third argument (the value of the test
129 % statistic). In the case that there is no input for the third argument,
130 % the p value will not be calculated and the function has finished
131 % running.
132 %
133 % The first nested if else statement will check, on the condition that the
134 % user gave input for the third argument, if the test statistic is larger
135 % than the critical value. If this is the case, the null hypothesis cannot
136 % be rejected and a vertical light purple dotted line corresponding to the
137 % value of the test statistic and a light purple shaded area representing
138 % the p value will be added to the plot. Else, the null can be rejected
139 % and a purple vertical dotted line corresponding to the value of the test
140 % statistic and a dark purple shaded area displaying the p value will be
141 % plotted.
142 %
143 % Afterwards the subtitle will be updated and the code has finished
144 % running.

```

```

145 % -----
146 if (TTL.Display == 1)
147     if (tstat > TTL.CV)
148         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
149             1.4);
150         TTL.tint = TTL.CV:0.001:tstat;
151         TTL.ty = pdf('T', TTL.tint, nu);
152         TTL.tar = area(TTL.tint, TTL.ty);
153         TTL.tar.FaceColor = '#8a22b3';
154         TTL.tar.FaceAlpha = 0.04;
155         TTL.tar.EdgeColor = 'none';
156     else
157         xline(tstat, 'LineStyle', ':', 'Color', '#8a22b3', 'LineWidth', ...
158             1.4);
159         TTL.tint = TTL.xmin:0.001:tstat;
160         TTL.ty = pdf('T', TTL.tint, nu);
161         TTL.tar = area(TTL.tint, TTL.ty);
162         TTL.tar.FaceColor = '#8a22b3';
163         TTL.tar.FaceAlpha = 1;
164         TTL.tar.EdgeColor = 'none';
165     end
166     TTL.pval = cdf('T', tstat, nu);
167     TTL.empdec = sprintf('%%.4f', 4);
168     TTL.pdec = sprintf('%%.4f', 4);
169     TTL.tp = sprintf(['Test statistic = ', TTL.empdec, ' p value = ', ...
170         TTL.pdec], tstat, TTL.pval);
171     subtitle({TTL.variables, TTL.tp}, 'Interpreter', 'tex');
172 end

```

1.5 Two tailed test

If the alternative hypothesis H_1 is specified as in equation (1), a two tailed test needs to be performed. The function `TTestTwoSided.m` will do so. For a two tailed test, two critical values need to be calculated. If in this case we also use the significance level $\alpha = 0.05$, we need to calculate the 2.5th and 97.5th percentile: the significance level α is ‘equally divided’ for both sides.

The same approach goes for the p value: the p value of a two tailed t -test is equal to double the p value in one tail. To visualize the p value, the code needs to find on which intervals the tails of the distribution need to be shaded. To do this, the code will first evaluate whether the test statistic is negative or positive valued (i.e., lies in the left or right tail). Then, depending on the sign of the test statistic, the codes takes the negative or positive ‘counterpart’ of the test statistic and is now able to shade both tails. The method works because the Student’s t -distribution is symmetric around zero.

```

1 function TTTS = TTestTwoSided(nu, alpha, tstat)
2 %TTESTTWOSIDED Visualize a two sided Student's t-test
3 % TTTS = TTestTwoSided(NU, ALPHA, TSTAT) plots the theoretical
4 % student's t-distribution with NU degrees of freedom. It calculates the
5 % two critical values corresponding to a two sided t-test with NU
6 % degrees of freedom at an ALPHA level of significance and plots the
7 % related rejection regions. A vertical line representing the manually
8 % calculated test statistic valued TSTAT will be plotted, this input

```

```

9  % argument is optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu <= 0)
15     uiwait(warndlg('The degrees of freedom should be larger than zero.'));
16     return
17 elseif (mod(nu, 1) ~= 0)
18     uiwait(warndlg(['Please fill in an integer for the degrees of ' ...
19                     'freedom.']));
20     return
21 elseif (alpha <= 0 || alpha >= 1)
22     uiwait(warndlg(['Please fill in a value of alpha between zero and ' ...
23                     'one.']));
24     return
25 end
26
27 % -----
28 % Check if the user wants to plot the test statistic.
29 % -----
30 if (nargin == 3)
31     TTTS.Display = 1;
32 else
33     TTTS.Display = 0;
34 end
35
36 % -----
37 % Calculating the critical values.
38 % -----
39 TTTS.CV = icdf('T', 1-alpha/2, nu);
40 TTTS.CVleft = -TTTS.CV;
41 TTTS.CVright = TTTS.CV;
42
43 % -----
44 % Determining the length of the horizontal axis, which depends on the
45 % critical values as having only one or two degrees of freedom can result
46 % in large critical values. In most cases however, the critical values are
47 % relatively small and [-5, 5] is a good interval to display the curve of
48 % the student's t-distribution and the crititcal values.
49 % -----
50 TTTS.xmin = min([TTTS.CVleft-1 -5]);
51 TTTS.xmax = max([TTTS.CVright+1 5]);
52 TTTS.x = TTTS.xmin:0.01:TTTS.xmax;
53
54 % -----
55 % Creating the density.
56 % -----
57 TTTS.y = pdf('T', TTTS.x, nu);
58
59 % -----
60 % Calculating the rejection regions, as these areas need to be shown in
61 % the plot.
62 % -----

```

```

63 TTTS.xleft = TTTS.xmin:0.001:TTTS.CVleft;
64 TTTS.xright = TTTS.CVright:0.001:TTTS.xmax;
65 TTTS.yleft = pdf('T', TTTS.xleft, nu);
66 TTTS.yright = pdf('T', TTTS.xright, nu);
67
68 % -----
69 % Setting up the plot. To create a subtitle consisting of two lines, the
70 % sprintf() function is used in the subtitle() function. The subtitle will
71 % be split up in two lines when the user wants to plot the test statistic
72 % and calculate the p value. TTTS.alphadec is used to determine the number
73 % of decimals for displaying alpha, which depends on the user and hence is
74 % dynamic. TTTS.noddec is used for the degrees of freedom, which don't have
75 % decimals. The code then asks for the size of the monitor of the user to
76 % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
77 % used. TTTS.scale scales the graph with respect to the monitor size of
78 % the user. xticks is used as it is necessary to show the exact critical
79 % value on the horizontal axis.
80 % -----
81 TTTS.alphadec = sprintf('%%.%%df', ...
82     length(char(extractAfter(string(alpha), '.'))));
83 TTTS.noddec = sprintf('%%.%%df', 0);
84 TTTS.variables = sprintf(['\\alpha = ', TTTS.alphadec, ', \\nu = ', ...
85     TTTS.noddec], alpha, nu);
86
87 TTTS.mp = get(0, 'MonitorPositions');
88 TTTS.mwidth = TTTS.mp(1, 3);
89 TTTS.mheight = TTTS.mp(1, 4);
90 TTTS.scale = 0.45;
91
92 TTTS.gwidth = TTTS.scale*TTTS.mwidth;
93 TTTS.gheight = 0.75*TTTS.gwidth;
94 TTTS.x0 = 0.5*TTTS.mwidth*(1 - TTTS.scale);
95 TTTS.y0 = (TTTS.mheight - TTTS.gheight - 84)*0.5;
96
97 figure
98 plot(TTTS.x,TTTS.y,'-black');
99 xticks([TTTS.CVleft 0 TTTS.CVright]);
100 title("t-distribution");
101 subtitle({TTTS.variables}, 'Interpreter', 'tex');
102 xlabel("t-value");
103 ylabel("Density");
104 TTTS.fig = gcf;
105 TTTS.fig.Position = [TTTS.x0, TTTS.y0, TTTS.gwidth, TTTS.gheight];
106 hold on
107
108 % -----
109 % Marking the critical values in the plot.
110 % -----
111 xline([TTTS.CVleft TTTS.CVright], 'LineStyle', ':', 'Color', '#9a9afc',...
112     'LineWidth', 1.4);
113
114 % -----
115 % Filling the areas of the rejection region.
116 % -----

```

```

117 TTTS.arleft = area(TTTS.xleft, TTTS.yleft);
118 TTTS.arleft.FaceColor = 'blue';
119 TTTS.arleft.FaceAlpha = 0.15;
120 TTTS.arleft.EdgeColor = 'none';
121
122 TTTS.arright = area(TTTS.xright, TTTS.yright);
123 TTTS.arright.FaceColor = 'blue';
124 TTTS.arright.FaceAlpha = 0.15;
125 TTTS.arright.EdgeColor = 'none';
126
127 % -----
128 % The user has the option to also plot the self calculated test statistic
129 % and compute the corresponding p value. This part of the code will only
130 % run when there is input for the third argument (the value of the test
131 % statistic). In the case that there is no input for the third argument,
132 % the p value will not be calculated and the function has finished
133 % running.
134 %
135 % The first nested if else statement will check, on the condition that the
136 % user gave input for the third argument, if the absolute value of the
137 % test statistic is smaller than the critical value. If this is the case,
138 % the null hypothesis cannot be rejected and a vertical light purple
139 % dotted line corresponding to the value of the test statistic and a
140 % light purple shaded area will be added to the plot. Else, the null can
141 % be rejected and a purple vertical dotted line corresponding to the value
142 % of the test statistic and a dark purple shaded area will be plotted.
143 %
144 % Afterwards the subtitle will be updated and the code has finished
145 % running.
146 % -----
147 if (TTTS.Display == 1)
148     if (abs(tstat) < TTTS.CV)
149         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
150             1.4);
151         TTTS.tintleft = TTTS.CVleft:0.001:-abs(tstat);
152         TTTS.tintright = abs(tstat):0.001:TTTS.CVright;
153
154         TTTS.tyl = pdf('T', TTTS.tintleft, nu);
155         TTTS.tlar = area(TTTS.tintleft, TTTS.tyl);
156         TTTS.tlar.FaceColor = '#8a22b3';
157         TTTS.tlar.FaceAlpha = 0.04;
158         TTTS.tlar.EdgeColor = 'none';
159
160         TTTS.tyr = pdf('T', TTTS.tintright, nu);
161         TTTS.trar = area(TTTS.tintright, TTTS.tyr);
162         TTTS.trar.FaceColor = '#8a22b3';
163         TTTS.trar.FaceAlpha = 0.04;
164         TTTS.trar.EdgeColor = 'none';
165     else
166         xline(tstat, 'LineStyle', ':', 'Color', '#8a22b3', 'LineWidth', ...
167             1.4);
168         TTTS.tintleft = TTTS.xmin:0.001:-abs(tstat);
169         TTTS.tintright = abs(tstat):0.001:TTTS.xmax;
170

```

```

171     TTTS.tyl = pdf('T', TTTS.tintleft, nu);
172     TTTS.tlar = area(TTTS.tintleft, TTTS.tyl);
173     TTTS.tlar.FaceColor = '#8a22b3';
174     TTTS.tlar.FaceAlpha = 1;
175     TTTS.tlar.EdgeColor = 'none';
176
177     TTTS.tyr = pdf('T', TTTS.tinright, nu);
178     TTTS.trar = area(TTTS.tinright, TTTS.tyr);
179     TTTS.trar.FaceColor = '#8a22b3';
180     TTTS.trar.FaceAlpha = 1;
181     TTTS.trar.EdgeColor = 'none';
182 end
183 TTTS.pval = 2*cdf('T', -abs(tstat), nu);
184 TTTS.empdec = sprintf('%.%df', 4);
185 TTTS.pdec = sprintf('%.%df', 4);
186 TTTS.tp = sprintf(['Test statistic = ', TTTS.empdec, ', p value = ', ...
187     TTTS.pdec], tstat, TTTS.pval);
188 subtitle({TTTS.variables, TTTS.tp}, 'Interpreter', 'tex');
189 end

```

2 F-test

When the goal of the hypothesis test focuses on the variances of two groups, an F-test can be performed. Under the null hypothesis, the F statistics follows an F-distribution. If $X \sim \chi_{\nu_1}^2$ and $Y \sim \chi_{\nu_2}^2$ (Chi-squared distributed with ν_1 and ν_2 degrees of freedom respectively), then

$$Z = \frac{X/\nu_1}{Y/\nu_2} \sim F_{\nu_1, \nu_2}$$

Z is F-distributed with ν_1 and ν_2 degrees of freedom. ν_1 is also called the numerator degrees of freedom and ν_2 the denominator degrees of freedom. The usual form that the corresponding hypothesis test is used is:

$$\begin{aligned} H_0 : \sigma_1^2 &\leq \sigma_2^2 \\ H_1 : \sigma_1^2 &> \sigma_2^2 \end{aligned} \tag{2}$$

With this test, we determine whether there is enough ‘evidence’ that the variance of the first group is larger than the variance of the second group.

When manually using the statistical table, three values are required to find the correct critical value of the hypothesis test in this table. These three values are the significance level, the numerator degrees of freedom and the denominator degrees of freedom. For every significance level, a specific table exists. The column headings denote the numerator degrees of freedom and the row headings the denominator degrees of freedom. Therefore, the minimal information required for the functions to work are the significance level and the two degrees of freedom. After filling in these values and running the code, a graph displaying the limiting distribution and the value from the statistical table will be shown. The user also has the option to give the manually calculated test statistic as input to the function file. In this case, this test statistic will also be displayed, from which the user is able to see if the corresponding null hypothesis can be rejected.

2.1 Using the functions

To help the learner understand the code, the `FTesting.m` file has been written. This file explains what input is required for the function and offers the ability to run the function files.

The F-test can be right tailed, left tailed or two tailed. The side of the test depends on the formulation of the alternative hypothesis. When the alternative hypothesis is formulated as

$$H_1 : \sigma_1^2 \neq \sigma_2^2,$$

a two tailed test is required. In case that the alternative hypothesis H_1 is $\sigma_1^2 > \sigma_2^2$, a right tailed test needs to be performed. The last case is a left tailed test, which is when the alternative hypothesis H_1 is formulated as $\sigma_1^2 < \sigma_2^2$.

```
1 % -----
2 % When using FTest(SIDE, NU1, NU2, ALPHA, TSTAT),
3 %
4 % SIDE indicates the side of the hypothesis test. Input for SIDE can be
5 % any of the three following:
6 %   - 'RightSided' for a right-tailed F-test
7 %   - 'LeftSided' for a left-tailed F-test
8 %   - 'TwoSided' for a two-tailed F-test
9 %
10 % NU1 and NU2 are the degrees of freedom and can be any positive valued
11 % integer.
12 %
13 % ALPHA denotes the significance level. The input for ALPHA can be any
14 % real valued number on the interval (0,1).
15 %
16 % TSTAT is the manually derived F-statistic, which can take on any
17 % nonnegative value. This input argument is optional and in case input is
18 % given, the function plots the manually calculated test statistic valued
19 % TSTAT and show whether the null hypothesis can or cannot be rejected.
20 %
21 % The size of the plot is predetermined and can be customized by changing
22 % the values of FTR.scale, FTL.scale, FTTS.scale in their corresponding
23 % function files.
24 %
25 % FTest(SIDE, NU1, NU2, ALPHA, TSTAT)
26 % -----
27 % Example
28 FTest('RightSided', 30, 20, 0.05, 1.8)
```

When we run the example command that has been written on the last line of the function file, we visualize a right tailed F-test with numerator degrees of freedom $\nu_1 = 30$, denominator degrees of freedom $\nu_2 = 20$, a significance level $\alpha = 0.05$ and a test statistic valued 1.8. The result is the following graph:

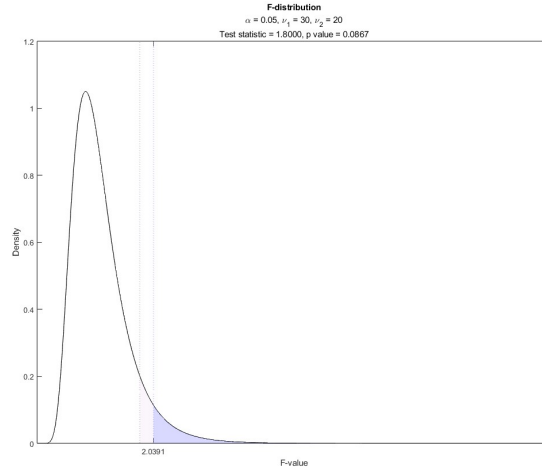


Figure 2: F-test example

2.2 F-test function

The function file `FTest.m` helps the learner's understanding of the two difficult subjects by visualizing the underlying statistical motivation and meaning of the tests. The function file `FTest.m` checks what side of the test the user wants to perform and then 'sends' the task to the corresponding function file.

```

1 function FT = FTest(side, nu1, nu2, alpha, tstat)
2 %FTEST Visualize an F-test
3 % FTest(SIDE, NU1, NU2, ALPHA, TSTAT) plots the theoretical
4 % F-distribution with NU1 and NU2 degrees of freedom. Depending on the
5 % input argument for SIDE, it calculates one or two critical values
6 % corresponding to a one or two sided F-test with NU1 and NU2 degrees of
7 % freedom at an ALPHA level of significance and plots the related
8 % rejection region(s). A vertical line representing the manually
9 % calculated test statistic valued TSTAT will be plotted as well, this
10 % input argument is optional.
11 %
12 % SIDE can be:
13 % 'RightSided',
14 % 'LeftSided',
15 % 'TwoSided'.
16
17 % -----
18 % Check if the user does not want to plot the test statistic, which
19 % happens when the number of input arguments is equal to four.
20 % -----
21 if (nargin == 4)
22     switch(side)
23     case "RightSided"
24         FTestRight(nu1, nu2, alpha);
25     case "LeftSided"
26         FTestLeft(nu1, nu2, alpha);

```

```

27         case "TwoSided"
28             FTestTwoSided(nu1, nu2, alpha);
29         otherwise
30             uiwait(warndlg(['Please specify the correct side, right, ' ...
31                 'left or two sided'])))
32             return
33     end
34 else
35     switch(side)
36         case "RightSided"
37             FTestRight(nu1, nu2, alpha, tstat);
38         case "LeftSided"
39             FTestLeft(nu1, nu2, alpha, tstat);
40         case "TwoSided"
41             FTestTwoSided(nu1, nu2, alpha, tstat);
42         otherwise
43             uiwait(warndlg(['Please specify the correct side, right, ' ...
44                 'left or two sided'])))
45             return
46     end
47 end

```

2.3 Right tailed test

The function file `FTestRight.m` can be used when the null and alternative hypotheses are in the form of equation (2).

```

1 function FTR = FTestRight(nu1, nu2, alpha, tstat)
2 %FTESTRIGHT Visualize a right sided F-test
3 % FTR = FTestLeft(NU1, NU2, ALPHA, TSTAT) plots the theoretical
4 % F-distribution with NU1 and NU2 degrees of freedom. It calculates the
5 % critical value corresponding to a right sided F-test with NU1 and NU2
6 % degrees of freedom at an ALPHA level of significance and plots the
7 % related rejection region. A vertical line representing the manually
8 % calculated test statistic valued TSTAT will be plotted, this input
9 % argument is optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu1 <= 0)
15     uiwait(warndlg(['The degrees of freedom of the first population' ...
16         ' should be larger than zero.']));
17     return
18 elseif (nu2 <= 0)
19     uiwait(warndlg(['The degrees of freedom of the second population' ...
20         ' should be larger than zero.']));
21     return
22 elseif (mod(nu1, 1) ~= 0)
23     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
24         ' freedom of the first population.']));
25     return
26 elseif (mod(nu2, 1) ~= 0)

```

```

27     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
28         ' freedom of the second population.'])));
29     return
30 elseif (alpha <= 0 || alpha >= 1)
31     uiwait(warndlg(['Please fill in a value of alpha between zero and' ...
32         ' one.'])));
33     return
34 end
35
36 % -----
37 % Check if the user wants to plot the test statistic and make sure that
38 % the test statistic is nonnegative valued.
39 % -----
40 if (nargin == 4)
41     if (tstat < 0)
42         uiwait(warndlg(['The test statistic cannot be negative valued,'...
43             ' please make sure that the test statistic is correctly' ...
44             ' calculated.'])));
45         return
46     end
47     FTR.Display = 1;
48 else
49     FTR.Display = 0;
50 end
51
52 % -----
53 % Calculating the critical value.
54 % -----
55 FTR.CV = icdf('F', 1-alpha, nu1, nu2);
56
57 % -----
58 % Determining the length of the horizontal axis, which depends on the
59 % critical value as having only one or two degrees of freedom can result
60 % in large critical value. In most cases however, the critical value is
61 % relatively small and [0, 9] is a good interval to display the curve of
62 % the F-distribution and the crititcal value.
63 % -----
64 FTR.xmin = 0;
65 FTR.xmax = max([FTR.CV+3 9]);
66 FTR.x = FTR.xmin:0.01:FTR.xmax;
67
68 % -----
69 % Creating the density.
70 % -----
71 FTR.y = pdf('F', FTR.x, nu1, nu2);
72
73 % -----
74 % Calculating the rejection region, as the area needs to be shown in the
75 % plot.
76 % -----
77 FTR.xright = FTR.CV:0.001:FTR.xmax;
78 FTR.yright = pdf('F', FTR.xright, nu1, nu2);
79
80 % -----

```

```

81 % Setting up the plot. To create a subtitle consisting of two lines, the
82 % sprintf() function is used in the subtitle() function. The subtitle will
83 % be split up in two lines when the user wants to plot the test statistic
84 % and calculate the p value. FTR.alphadec is used to determine the number
85 % of decimals for displaying alpha, which depends on the user and hence is
86 % dynamic. FTR.nodect is used for the degrees of freedom, which don't have
87 % decimals. The code then asks for the size of the monitor of the user to
88 % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
89 % used. FTR.scale scales the graph with respect to the monitor size of the
90 % user. xticks is used as it is necessary to show the exact critical value
91 % on the horizontal axis.
92 % -----
93 FTR.alphadec = sprintf('%.%df', ...
94     length(char(extractAfter(string(alpha), '.'))));
95 FTR.nodect = sprintf('%.%df', 0);
96 FTR.variables = sprintf(['\\alpha = ', FTR.alphadec, ', \\nu_{1} = ', ...
97     FTR.nodect, ', \\nu_{2} = ' FTR.nodect], alpha, nu1, nu2);
98
99 FTR.mp = get(0, 'MonitorPositions');
100 FTR.mwidth = FTR.mp(1, 3);
101 FTR.mheight = FTR.mp(1, 4);
102 FTR.scale = 0.45;
103
104 FTR.gwidth = FTR.scale*FTR.mwidth;
105 FTR.gheight = 0.75*FTR.gwidth;
106 FTR.x0 = 0.5*FTR.mwidth*(1 - FTR.scale);
107 FTR.y0 = (FTR.mheight - FTR.gheight - 84)*0.5;
108
109 figure
110 plot(FTR.x,FTR.y,'-black');
111 xticks([FTR.CV]);
112 title("F-distribution")
113 subtitle({FTR.variables}, 'Interpreter', 'tex');
114 xlabel("F-value");
115 ylabel("Density");
116 FTR.fig = gcf;
117 FTR.fig.Position = [FTR.x0, FTR.y0, FTR.gwidth, FTR.gheight];
118 hold on
119
120 % -----
121 % Marking the critical value in the plot.
122 % -----
123 xline([FTR.CV], 'LineStyle', ':', 'Color', '#9a9afc', 'LineWidth', 1.4);
124
125 % -----
126 % Filling the area of the rejection region.
127 % -----
128 FTR.ar = area(FTR.xright, FTR.yright);
129 FTR.ar.FaceColor = 'blue';
130 FTR.ar.FaceAlpha = 0.15;
131 FTR.ar.EdgeColor = 'none';
132
133 % -----
134 % The user has the option to also plot the self calculated test statistic

```

```

135 % and compute the corresponding p value. This part of the code will only
136 % run when there is input for the fourth argument (the value of the test
137 % statistic). In the case that there is no input for the fourth argument,
138 % the p value will not be calculated and the function has finished
139 % running.
140 %
141 % The first nested if else statement will check, on the condition that the
142 % user gave input for the fourth argument, if the test statistic is
143 % smaller than the critical value. If this is the case, the null
144 % hypothesis cannot be rejected and a vertical light purple dotted line
145 % corresponding to the value of the test statistic and a light purple
146 % shaded area will be added to the plot. Else, the null can be rejected
147 % and a purple vertical dotted line corresponding to the value of the
148 % test statistic and a dark purple shaded area will be plotted.
149 %
150 % Afterwards the subtitle is updated and the code has finished
151 % running.
152 % -----
153 if (FTR.Display == 1)
154     if (tstat < FTR.CV)
155         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
156             1.4);
157         FTR.tint = tstat:0.001:FTR.CV;
158         FTR.ty = pdf('F', FTR.tint, nu1, nu2);
159         FTR.tar = area(FTR.tint, FTR.ty);
160         FTR.tar.FaceColor = '#8a22b3';
161         FTR.tar.FaceAlpha = 0.04;
162         FTR.tar.EdgeColor = 'none';
163     else
164         xline(tstat, 'LineStyle', ':', 'Color', '#8a22b3', 'LineWidth', ...
165             1.4);
166         FTR.tint = tstat:0.001:FTR.xmax;
167         FTR.ty = pdf('F', FTR.tint, nu1, nu2);
168         FTR.tar = area(FTR.tint, FTR.ty);
169         FTR.tar.FaceColor = '#8a22b3';
170         FTR.tar.FaceAlpha = 1;
171         FTR.tar.EdgeColor = 'none';
172     end
173     FTR.pval = 1-cdf('F', tstat, nu1, nu2);
174     FTR.empdec = sprintf('%.4f', 4);
175     FTR.pdec = sprintf('%.4f', 4);
176     FTR.tp = sprintf(['Test statistic = ', FTR.empdec, ', p value = ', ...
177         FTR.pdec], tstat, FTR.pval);
178     subtitle({FTR.variables, FTR.tp}, 'Interpreter', 'tex');
179 end

```

2.4 Left tailed test

When the alternative hypothesis H_1 is defined as $\sigma_1^2 < \sigma_2^2$, the function file `FTestLeft.m` can be used.

```
1 function FTL = FTestLeft(nu1, nu2, alpha, tstat)
2 %FTESTLEFT Visualize a left sided F-test
3 %   FTL = FTestLeft(NU1, NU2, ALPHA, TSTAT) plots the theoretical
4 %   F-distribution with NU1 and NU2 degrees of freedom. It calculates the
5 %   critical value corresponding to a left sided F-test with NU1 and NU2
6 %   degrees of freedom at an ALPHA level of significance and plots the
7 %   related rejection region. A vertical line representing the manually
8 %   calculated test statistic valued TSTAT will be plotted, this input
9 %   argument is optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu1 <= 0)
15     uiwait(warndlg(['The degrees of freedom of the first population' ...
16         ' should be larger than zero.']));
17     return
18 elseif (nu2 <= 0)
19     uiwait(warndlg(['The degrees of freedom of the second population' ...
20         ' should be larger than zero.']));
21     return
22 elseif (mod(nu1, 1) ~= 0)
23     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
24         ' freedom of the first population.']));
25     return
26 elseif (mod(nu2, 1) ~= 0)
27     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
28         ' freedom of the second population.']));
29     return
30 elseif (alpha <= 0 || alpha >= 1)
31     uiwait(warndlg(['Please fill in a value of alpha between zero and' ...
32         ' one.']));
33     return
34 end
35
36 % -----
37 % Check if the user wants to plot the test statistic and make sure that
38 % the test statistic is nonnegative valued.
39 % -----
40 if (nargin == 4)
41     if (tstat < 0)
42         uiwait(warndlg(['The test statistic cannot be negative valued,'...
43             ' please make sure that the test statistic is correctly' ...
44             ' calculated.']));
45         return
46     end
47     FTL.Display = 1;
48 else
49     FTL.Display = 0;
50 end
```

```

51 |
52 | % -----
53 | % Calculating the critical value.
54 | % -----
55 | FTL.CV = icdf('F', alpha, nu1, nu2);
56 |
57 | % -----
58 | % Determining the length of the horizontal axis, which depends on the
59 | % critical value as having only one or two degrees of freedom can result
60 | % in large critical value. In most cases however, the critical value is
61 | % relatively small and [0, 9] is a good interval to display the curve of
62 | % the F-distribution and the critical value. To display the distribution
63 | % better in a dynamic way, the right end value of the interval depends on
64 | % the critical value of a right sided F-test. This is because the critical
65 | % value of a left sided F-test does not increase as much as the thickness
66 | % of the right sided tail of the F-distribution when the degrees of
67 | % freedom increase.
68 | % -----
69 | FTL.betterright = icdf('F', 1-alpha, nu2, nu1);
70 | FTL.xmin = 0;
71 | FTL.xmax = min([FTL.betterright+3 9]);
72 | FTL.x = FTL.xmin:0.01:FTL.xmax;
73 |
74 | % -----
75 | % Creating the density.
76 | % -----
77 | FTL.y = pdf('F', FTL.x, nu1, nu2);
78 |
79 | % -----
80 | % Calculating the rejection region, as the area needs to be shown in the
81 | % plot.
82 | % -----
83 | FTL.xleft = FTL.xmin:0.001:FTL.CV;
84 | FTL.yleft = pdf('F', FTL.xleft, nu1, nu2);
85 |
86 | % -----
87 | % Setting up the plot. To create a subtitle consisting of two lines, the
88 | % sprintf() function is used in the subtitle() function. The subtitle will
89 | % be split up in two lines when the user wants to plot the test statistic
90 | % and calculate the p value. FTL.alphadec is used to determine the number
91 | % of decimals for displaying alpha, which depends on the user and hence is
92 | % dynamic. FTL.noddec is used for the degrees of freedom, which don't have
93 | % decimals. The code then asks for the size of the monitor of the user to
94 | % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
95 | % used. FTL.scale scales the graph with respect to the monitor size of the
96 | % user. xticks is used as it is necessary to show the exact critical value
97 | % on the horizontal axis.
98 | % -----
99 | FTL.alphadec = sprintf('%.%df', ...
100 |     length(char(extractAfter(string(alpha), '.'))));
101 | FTL.noddec = sprintf('%.%df', 0);
102 | FTL.variables = sprintf(['\\alpha = ', FTL.alphadec, ', \\nu_{1} = ', ...
103 |     FTL.noddec, ', \\nu_{2} = ' FTL.noddec], alpha, nu1, nu2);
104 |

```

```

105 FTL.mp = get(0, 'MonitorPositions');
106 FTL.mwidth = FTL.mp(1, 3);
107 FTL.mheight = FTL.mp(1, 4);
108 FTL.scale = 0.45;
109
110 FTL.gwidth = FTL.scale*FTL.mwidth;
111 FTL.gheight = 0.75*FTL.gwidth;
112 FTL.x0 = 0.5*FTL.mwidth*(1 - FTL.scale);
113 FTL.y0 = (FTL.mheight - FTL.gheight - 84)*0.5;
114
115 figure
116 plot(FTL.x,FTL.y,'-black');
117 xticks([FTL.CV]);
118 title("F-distribution");
119 subtitle({FTL.variables}, 'Interpreter', 'tex');
120 xlabel("F-value");
121 ylabel("Density");
122 FTL.fig = gcf;
123 FTL.fig.Position = [FTL.x0, FTL.y0, FTL.gwidth, FTL.gheight];
124 hold on
125
126 % -----
127 % Marking the critical value in the plot.
128 % -----
129 xline([FTL.CV], 'LineStyle', ':', 'Color', '#9a9afc', 'LineWidth', 1.4);
130
131 % -----
132 % Filling the area of the rejection region.
133 % -----
134 FTL.ar = area(FTL.xleft, FTL.yleft);
135 FTL.ar.FaceColor = 'blue';
136 FTL.ar.FaceAlpha = 0.15;
137 FTL.ar.EdgeColor = 'none';
138
139 % -----
140 % The user has the option to also plot the self calculated test statistic
141 % and compute the corresponding p value. This part of the code will only
142 % run when there is input for the fourth argument (the value of the test
143 % statistic). In the case that there is no input for the fourth argument,
144 % the p value will not be calculated and the function has finished
145 % running.
146 %
147 % The first nested if else statement will check, on the condition that the
148 % user gave input for the fourth argument, if the test statistic is larger
149 % than the critical value. If this is the case, the null hypothesis cannot
150 % be rejected and a vertical light purple dotted line corresponding to the
151 % value of the test statistic and a light purple shaded area will be added
152 % to the plot. Else, the null can be rejected and a purple vertical dotted
153 % line corresponding to the value of the test statistic and a dark purple
154 % shaded area will be plotted.
155 %
156 % Afterwards the subtitle is updated and the code has finished
157 % running.
158 % -----

```



```

159 if (FTL.Display == 1)
160     if (tstat > FTL.CV)
161         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
162             1.4);
163         FTL.tint = FTL.CV:0.001:tstat;
164         FTL.ty = pdf('F', FTL.tint, nu1, nu2);
165         FTL.tar = area(FTL.tint, FTL.ty);
166         FTL.tar.FaceColor = '#8a22b3';
167         FTL.tar.FaceAlpha = 0.04;
168         FTL.tar.EdgeColor = 'none';
169     else
170         xline(tstat, 'LineStyle', ':', 'Color', '#8a22b3', 'LineWidth', ...
171             1.4);
172         FTL.tint = FTL.xmin:0.001:tstat;
173         FTL.ty = pdf('F', FTL.tint, nu1, nu2);
174         FTL.tar = area(FTL.tint, FTL.ty);
175         FTL.tar.FaceColor = '#8a22b3';
176         FTL.tar.FaceAlpha = 1;
177         FTL.tar.EdgeColor = 'none';
178     end
179     FTL.pval = cdf('F', tstat, nu1, nu2);
180     FTL.empdec = sprintf('%.4f', 4);
181     FTL.pdec = sprintf('%.4f', 4);
182     FTL.tp = sprintf(['Test statistic = ', FTL.empdec, ' p value = ', ...
183         FTL.pdec], tstat, FTL.pval);
184     subtitle({FTL.variables, FTL.tp}, 'Interpreter', 'tex');
185 end

```

2.5 Two tailed test

Lastly, it is also possible that the F-test is two tailed. This occurs when we have that $H_1: \sigma_1^2 \neq \sigma_2^2$. The function file `FTestTwoSided.m` has been written to visualize the two tailed F-test.

Similar to the two tailed t -test, the p value of the test statistic is derived by doubling the p value. However, since the F-distribution is not symmetric and its support is $[0, \infty)$, it is not possible to find the ‘counterpart’ of the test statistic by multiplying its value by negative one.

In the case that the degrees of freedom ν_1 and ν_2 are equal to each other, the ‘counterpart’ of the test statistic can be found by deriving its reciprocal.

If the two degrees of freedom are not equal to each other, taking the reciprocal will not give the correct value. This problem has been solved by noting that the p value is equally divided in the two tails. The code determines whether the percentile of the test statistic is larger than or equal to the 50th percentile. If this is the case, the ‘counterpart’ is somewhere below the 50th percentile. The code will then continue by calculating the p value of the tail which contains the test statistic. Depending on in which tail the ‘counterpart’ should be, the inverse cumulative function is used to derive the value of the ‘counterpart’. The code has now gathered all required information to shade the p value in both tails. This method also works when the two degrees of freedom are equal to each other.

```

1 function FTTS = FTestTwoSided(nu1, nu2, alpha, tstat)
2 %FTESTTWOSIDED Visualize a two sided F-test
3 % FTTS = FTestTwoSided(NU, ALPHA, TSTAT) plots the theoretical
4 % F-distribution with NU1 and NU2 degrees of freedom. It calculates the
5 % two critical values corresponding to a two sided F-test with NU1 and

```

```

6 % NU2 degrees of freedom at an ALPHA level of significance and plots the
7 % related rejection regions. A vertical line representing the manually
8 % calculated test statistic valued TSTAT will be plotted, this input
9 % argument is optional.
10
11 % -----
12 % Check whether the input is valid.
13 % -----
14 if (nu1 <= 0)
15     uiwait(warndlg(['The degrees of freedom of the first population' ...
16         ' should be larger than zero.']));
17     return
18 elseif (nu2 <= 0)
19     uiwait(warndlg(['The degrees of freedom of the second population' ...
20         ' should be larger than zero.']));
21     return
22 elseif (mod(nu1, 1) ~= 0)
23     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
24         ' freedom of the first population.']));
25     return
26 elseif (mod(nu2, 1) ~= 0)
27     uiwait(warndlg(['Please fill in an integer for the degrees of' ...
28         ' freedom of the second population.']));
29     return
30 elseif (alpha <= 0 || alpha >= 1)
31     uiwait(warndlg(['Please fill in a value of alpha between zero and' ...
32         ' one.']));
33     return
34 end
35
36 % -----
37 % Check if the user wants to plot the test statistic and make sure that
38 % the test statistic is nonnegative valued.
39 % -----
40 if (nargin == 4)
41     if (tstat < 0)
42         uiwait(warndlg(['The test statistic cannot be negative valued,'...
43             ' please make sure that the test statistic is correctly' ...
44             ' calculated.']));
45         return
46     end
47     FTTS.Display = 1;
48 else
49     FTTS.Display = 0;
50 end
51
52 % -----
53 % Calculating the critical values.
54 % -----
55 FTTS.CVleft = icdf('F', alpha/2, nu1, nu2);
56 FTTS.CVright = icdf('F', 1-alpha/2, nu1, nu2);
57
58 % -----
59 % Determining the length of the horizontal axis, which depends on the

```

```

60 % critical values as having only one or two degrees of freedom can result
61 % in large critical value. In most cases however, the critical values are
62 % relatively small and [0, 9] is a good interval to display the curve of
63 % the F-distribution and the critical values.
64 % -----
65 FTTS.xmin = 0;
66 FTTS.xmax = min([FTTS.CVright+3 9]);
67 FTTS.x = FTTS.xmin:0.01:FTTS.xmax;
68
69 % -----
70 % Creating the density.
71 % -----
72 FTTS.y = pdf('F', FTTS.x, nu1, nu2);
73
74 % -----
75 % Calculating the rejection regions, as these areas need to be shown in
76 % the plot.
77 % -----
78 FTTS.xleft = FTTS.xmin:0.01:FTTS.CVleft;
79 FTTS.xright = FTTS.CVright:0.01:FTTS.xmax;
80 FTTS.yleft = pdf('F', FTTS.xleft, nu1, nu2);
81 FTTS.yright = pdf('F', FTTS.xright, nu1, nu2);
82
83 % -----
84 % Setting up the plot. To create a subtitle consisting of two lines, the
85 % sprintf() function is used in the subtitle() function. The subtitle will
86 % be split up in two lines when the user wants to plot the test statistic
87 % and calculate the p value. FTTS.alphadec is used to determine the number
88 % of decimals for displaying alpha, which depends on the user and hence is
89 % dynamic. FTTS.noddec is used for the degrees of freedom, which don't have
90 % decimals. The code then asks for the size of the monitor of the user to
91 % calculate the size (in pixels) of the graph. The standard 4:3 ratio is
92 % used. FTTS.scale scales the graph with respect to the monitor size of
93 % the user. xticks is used as it is necessary to show the exact critical
94 % value on the horizontal axis.
95 % -----
96 FTTS.alphadec = sprintf('%.%df', ...
97     length(char(extractAfter(string(alpha), '.'))));
98 FTTS.noddec = sprintf('%.%df', 0);
99 FTTS.variables = sprintf(['\\alpha = ', FTTS.alphadec, ', \\nu_{1} = ', ...
100     FTTS.noddec, ', \\nu_{2} = ' FTTS.noddec], alpha, nu1, nu2);
101
102 FTTS.mp = get(0, 'MonitorPositions');
103 FTTS.mwidth = FTTS.mp(1, 3);
104 FTTS.mheight = FTTS.mp(1, 4);
105 FTTS.scale = 0.45;
106
107 FTTS.gwidth = FTTS.scale*FTTS.mwidth;
108 FTTS.gheight = 0.75*FTTS.gwidth;
109 FTTS.x0 = 0.5*FTTS.mwidth*(1 - FTTS.scale);
110 FTTS.y0 = (FTTS.mheight - FTTS.gheight - 84)*0.5;
111
112 figure
113 plot(FTTS.x, FTTS.y, '-black');

```

```

114 | xticks([FTTS.CVleft FTTS.CVright]);
115 | title("F-distribution");
116 | subtitle({FTTS.variables}, 'Interpreter', 'tex');
117 | xlabel("F-value");
118 | ylabel("Density");
119 | xlim([0 FTTS.xmax]);
120 | FTTS.fig = gcf;
121 | FTTS.fig.Position = [FTTS.x0, FTTS.y0, FTTS.gwidth, FTTS.gheight];
122 | hold on
123 |
124 | % -----
125 | % Marking the critical values in the plot.
126 | % -----
127 | xline([FTTS.CVleft FTTS.CVright], 'LineStyle', ':', 'Color', '#9a9afc',...
128 |     'LineWidth', 1.4);
129 |
130 | % -----
131 | % Filling the areas of the rejection region.
132 | % -----
133 | FTTS.arleft = area(FTTS.xleft, FTTS.yleft);
134 | FTTS.arleft.FaceColor = 'blue';
135 | FTTS.arleft.FaceAlpha = 0.15;
136 | FTTS.arleft.EdgeColor = 'none';
137 |
138 | FTTS.arright = area(FTTS.xright, FTTS.yright);
139 | FTTS.arright.FaceColor = 'blue';
140 | FTTS.arright.FaceAlpha = 0.15;
141 | FTTS.arright.EdgeColor = 'none';
142 |
143 | % -----
144 | % The user has the option to also plot the self calculated test statistic
145 | % and compute the corresponding p value. This part of the code will only
146 | % run when there is input for the fourth argument (the value of the test
147 | % statistic). In the case that there is no input for the fourth argument,
148 | % the p value will not be calculated and the function has finished
149 | % running.
150 | %
151 | % The first nested if else statement will check, on the condition that the
152 | % user gave input for the fourth argument, if the value of the test
153 | % statistic lies between the two critical values. If this is the case, the
154 | % null hypothesis cannot be rejected and a vertical light purple dotted
155 | % line corresponding to the value of the test statistic and a light purple
156 | % shaded area will be added to the plot. Else, the null can be rejected
157 | % and a purple vertical dotted line corresponding to the value of the test
158 | % statistic and a dark purple shaded area will be plotted.
159 | %
160 | % Afterwards the subtitle is updated and the code has finished running.
161 | % -----
162 | if (FTTS.Display == 1)
163 |     % Check if null can be rejected
164 |     if (tstat > FTTS.CVleft && tstat < FTTS.CVright)
165 |         xline(tstat, 'LineStyle', ':', 'Color', '#ae9ab5', 'LineWidth', ...
166 |             1.4);
167 |         % Determine the interval of the light purple shaded area

```

```

168     if (tstat >= icdf('F', 0.5, nu1, nu2))
169         FTTS.otherpside = 1 - cdf('F', tstat, nu1, nu2);
170         FTTS.otherstat = icdf('F', FTTS.otherpside, nu1, nu2);
171         FTTS.tintleft = FTTS.CVleft:0.001:FTTS.otherstat;
172         FTTS.tintright = tstat:0.001:FTTS.CVright;
173         FTTS.pval = (1 - cdf('F', tstat, nu1, nu2))*2;
174     else
175         FTTS.otherpside = cdf('F', tstat, nu1, nu2);
176         FTTS.otherstat = icdf('F', 1 - FTTS.otherpside, nu1, nu2);
177         FTTS.tintleft = FTTS.CVleft:0.001:tstat;
178         FTTS.tintright = FTTS.otherstat:0.001:FTTS.CVright;
179         FTTS.pval = 2*cdf('F', tstat, nu1, nu2);
180     end
181     FTTS.tyl = pdf('F', FTTS.tintleft, nu1, nu2);
182     FTTS.tlar = area(FTTS.tintleft, FTTS.tyl);
183     FTTS.tlar.FaceColor = '#8a22b3';
184     FTTS.tlar.FaceAlpha = 0.04;
185     FTTS.tlar.EdgeColor = 'none';
186
187     FTTS.tyr = pdf('F', FTTS.tintright, nu1, nu2);
188     FTTS.trar = area(FTTS.tintright, FTTS.tyr);
189     FTTS.trar.FaceColor = '#8a22b3';
190     FTTS.trar.FaceAlpha = 0.04;
191     FTTS.trar.EdgeColor = 'none';
192 else
193     xline(tstat, 'LineStyle', ':', 'Color','#8a22b3', 'LineWidth', ...
194         1.4);
195     % Determine the interval of the dark purple shaded area
196     if (tstat >= FTTS.CVright)
197         FTTS.otherpside = 1 - cdf('F', tstat, nu1, nu2);
198         FTTS.otherstat = icdf('F', FTTS.otherpside, nu1, nu2);
199         FTTS.tintleft = FTTS.xmin:0.001:FTTS.otherstat;
200         FTTS.tintright = tstat:0.001:FTTS.xmax;
201         FTTS.pval = (1 - cdf('F', tstat, nu1, nu2))*2;
202     else
203         FTTS.otherpside = cdf('F', tstat, nu1, nu2);
204         FTTS.otherstat = icdf('F', 1 - FTTS.otherpside, nu1, nu2);
205         FTTS.tintleft = FTTS.xmin:0.001:tstat;
206         FTTS.tintright = FTTS.otherstat:0.001:FTTS.xmax;
207         FTTS.pval = 2*cdf('F', tstat, nu1, nu2);
208     end
209     FTTS.tyl = pdf('F', FTTS.tintleft, nu1, nu2);
210     FTTS.tlar = area(FTTS.tintleft, FTTS.tyl);
211     FTTS.tlar.FaceColor = '#8a22b3';
212     FTTS.tlar.FaceAlpha = 1;
213     FTTS.tlar.EdgeColor = 'none';
214
215     FTTS.tyr = pdf('F', FTTS.tintright, nu1, nu2);
216     FTTS.trar = area(FTTS.tintright, FTTS.tyr);
217     FTTS.trar.FaceColor = '#8a22b3';
218     FTTS.trar.FaceAlpha = 1;
219     FTTS.trar.EdgeColor = 'none';
220 end
221 FTTS.empdec = sprintf('%.4f', 4);

```

```
222     FTTS.pdec = sprintf('%.%df', 4);
223     FTTS.tp = sprintf(['Test statistic = ', FTTS.empdec, ', p value = ', ...
224         FTTS.pdec], tstat, FTTS.pval);
225     subtitle({FTTS.variables, FTTS.tp}, 'Interpreter', 'tex');
226 end
```