All MATLAB code for the project “computing high order dependencies between the different levels and features in limit order books”

Matthew Newton

Contents

Main Script All

Introduction 2
User Select Parameters 2
Select outputs 3
Import data 4
Functions 5

Data

Import Data 9

Estimator Functions

Standard Correlation 11
Brownian Estimator 13
Fourier Estimator 16
Copula Estimator 19

Global Functions

Select Features 25
Calculate All Dependencies Vary Time 26
Calculate All Dependencies Vary Window 27
Calculate All Dependencies Vary Resampling 29
Find DateTime Time Interval 30
Find DateTime Time Window 33
Find DateTime No Interval 36
Resample Data 36
Filter Data 38

Largest R 38
Find Intervals 39
Feature Name List 42
Relate Feature List 42

Plotting Functions

Plot Levels Scatter 43
Plot Levels Scatter Sub 44
Plot Compare Sampling 46
Plot Merits Time 49
Plot Merits Scale 52
Plot Merits Resampling 54
Summarise Results Time 46
Summarise Results Scale 62
Summarise Results Resampling 68
Visualise Matrices Time 72
Visualise Matrices Scale 76
Visualise Matrices Resampling 81

Introduction

```
% This main script is to compare all the methods of calculating dependency.
% It is possible for the user to selected which features are compared and a variety of graphs
that compare the different methods.
```

User Select Parameters

```
% Set the overall time window for data that is to be analysied.
% Set date in format 'dd-MMM-yyyy hh:mm:ss' e.g. '03-Oct-2016 07:33:57'.
% Time must be between 07:30:00' and 16:30:00'.
% Date must be between 03-Oct-2016 and 28-Oct-2016.
Set_Date_1 = '03-Oct-2016 12:35:00';
Set_Date_2 = '03-Oct-2016 16:25:00';

% Set the number of intervals, that is the number of seperate time intervals to analysis the
data seperately.
% Set as 1 if comparing the whole period.
num_intervals = 3;

% Choose which features that will be computed and hence dependencies calculated.
% The number in each catergory corresponds to the level.
% Values must be between 1 and 10 or empty.
select_ask_prices = [1:5];
select_bid_prices = [1:5];
```

```
select_ask_sizes = [1:5];
select_bid_sizes = [1:5];

% Function that re-orders the choosen features to make the code easier to work with.
[selected_features] = select_features...
(select_ask_prices,select_bid_prices,select_ask_sizes,select_bid_sizes);

% Set sampling frequency or time interval for the resample function.
% Time interval is in seconds.
time_interval = 1;
fs = 1/time_interval;
% Type = 0 ('event') takes the most recent event in the data.
% Type = 1 ('inter') linear interpolates the data.
type = 0;

% For the varying time frequency function, select the range and the number of frequencies to
be evalulated.
fs_range = [0.01,0.1];
num_freq = 10;

% Define frequency fraction and order for the butterworth filter that is applied on the
resampled data.
freq_fraction = 0.25;
filter_order = 2;

% A list is produced containing the most significant dependecies.
% Select the proportional number of terms to be included in this list (or the maximum
number).
prop = 0.5;
num_max = round(prop*(max(size(selected_features)))^2);

% Parameter to determine the circle size on the plots, (the size of the circles corresponds
to bid/ask size).
circle_size = 0.55;
```

Select outputs

```
% A "1" indicates that the graph/method is to be plotted/included and a "0"
% indicates that it is not.

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are super-imposed on the same plot.
scatter_plot = 0;

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are ploted on seperate subplots.
scatter_plot_sub = 0;

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% Each subplot compares the effect sampling and filtering to raw data.
compare_sampling = 0;

% A plots of the log determinant of each of the dependency matricies (merits) as a function
of time.
```

```
merits_time = 0;

% A plots of the log determinant of each of the dependency matricies (merits) where the x-
axis represents time as an expanding window (effectively a cumulative merit).
merits_scale = 0;

% A plots of the log determinant of each of the dependency matricies
% (merits) as a function of sampling frequency (only for resample and
% resample and filter corrleations)
merits_resampling = 0;

% A figure with a visualisation of each of the dependency matricies.
% The time updates every 3 seconds, so to close the figure, call "ctrl-C" in the command
window.
visualise_matrix = 0;

    % Select whether to evaulate varying: time, time window or resample.
    % Can only have one figure running at a time, so it is recommeneded that only one
variable below is set to a "1".
    visualise_time = 0;
    visualise_scale = 0;
    visualise_resampling = 0;

% A summary of all the results with each dependency method in a seperate figure.
summarise_data = 0;

    % Select which methods to evaluate.
    summarise_standard = 1;
    summarise_standard_resample = 1;
    summarise_standard_filter = 1;
    summarise_brownian = 1;
    summarise_fourier = 1;
    summarise_copula = 1;
    summarise_copula_resample = 1;
    summarise_copula_filter = 1;

    % Select whether to evaulate varying: time, time window or resample.
    summarise_time = 1;
    summarise_scale = 0;
    summarise_resampling = 0;
```

Import data

```
% Saves time when data is already imported.
if exist('XOMMarketDepthOct2016_Values','var') == 0

disp('Importing data fromm csv file...');

% Imports data from csv file.
[XOMMarketDepthOct2016_Values,XOMMarketDepthOct2016_DateTime,...
XOMTransactionsOct2016,XOMTransactionsOct2016_DateTime] = ...
import_data();

% Convert date and time to standard format.
DateTime_Con = datetime(XOMMarketDepthOct2016_DateTime, 'InputFormat',...
'yyyy-MM-dd''T''HH:mm:ss.SSS''Z', 'TimeZone', 'UTC');
```

```
disp('Data imported');

else

disp('Data already imported');

end
```

Functions

```
% Calculates all dependecies along with a list of their most significant values, uses
seperate intervals to compute dependencies.
% Also creates a list of intervals to be used in the plots.
if merits_time == 1 || (visualise_matrix == 1 && visualise_time == 1)...
   ||(summarise_data == 1 && summarise_time == 1)

disp('Calculating all dependencies for varying time...');

[R_St,R_Sresamplet,R_Sfiltert,R_Bt,R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,...
largest_R_St,largest_R_Sresamplet,largest_R_Sfiltert,largest_R_Bt,...
largest_R_Ft,largest_R_Ct,largest_R_Cresamplet,largest_R_Cfiltert] ...
= calculate_all_dependencies_vary_time...
(selected_features,num_intervals,num_max,Set_Date_1,Set_Date_2,...
XOMMarketDepthOct2016_Values,DateTime_Con,...
fs,type,freq_fraction,filter_order);

disp('All dependencies for varying time calculated');

end

% Calculates all dependecies along with a list of their most significant values, uses
seperate intervals to compute dependencies.
if merits_scale == 1 || (visualise_matrix == 1 && visualise_scale == 1)...
   || (summarise_data == 1 && summarise_scale == 1)

disp('Calculating all dependencies for a varying time window...');

[R_SWt,R_SresampleWt,R_SfilterWt,R_BWt,R_FWt,R_CWt,R_CresampleWt,...
R_CfilterWt,largest_R_SWt,largest_R_SresampleWt,largest_R_SfilterWt,...
largest_R_BWt,largest_R_FWt,largest_R_CWt,largest_R_CresampleWt,...
largest_R_CfilterWt] = ...
calculate_all_dependencies_vary_window...
(selected_features,num_intervals,...
num_max,Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,...
fs,type,freq_fraction,filter_order);

disp('All dependencies for a varying time window calculated');

end

% Calculates all dependecies along with a list of their most significant values, uses
different resample rates to compute dependencies.
if merits_resampling == 1 ||...
   (visualise_matrix == 1 && visualise_resampling == 1) ||...
   (summarise_data == 1 && summarise_resampling == 1)

disp('Calculating all dependencies for a varying sampling frequency...');
```

```
[R_SresampleSt,R_SfilterSt,R_CresampleSt,R_CfilterSt,largest_R_SresampleSt,...
largest_R_SfilterSt,largest_R_CresampleSt,largest_R_CfilterSt] = ...
calculate_all_dependencies_vary_resampling...
(selected_features,num_max,Set_Date_1,Set_Date_2,...
XOMMarketDepthOct2016_Values,DateTime_Con,type,...
freq_fraction,filter_order,fs_range,num_freq);

disp('All dependencies for a varying sampling frequency calculated');

end

% Contains a list of names of each feature.
[feature_name_list] = feature_name_list();

% Relates the row number to the feature number and name.
clear relate_feature_list;  % Without "clear" there will be an error.
[relate_feature_list] = relate_feature_list...
(selected_features,feature_name_list);

% Lists the date and time of the intervals.
[interval_list] = find_intervals...
(Set_Date_1,Set_Date_2,DateTime_Con,num_intervals);

disp('Creating required plots and compiling results...');

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are super-imposed on the same plot.
% Dotted line represents the interval spacing.
if scatter_plot == 1

plot_levels_scatter...
(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,...
circle_size,selected_features,feature_name_list,interval_list);

end

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are ploted on seperate subplots.
% Dotted line represents the interval spacing.
if scatter_plot_sub == 1

plot_levels_scatter_sub...
(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,...
circle_size,selected_features,feature_name_list,interval_list);

end

% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% Each subplot compares the effect sampling and filtering to raw data.
if compare_sampling == 1

plot_compare_sampling...
(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,...
circle_size,selected_features,feature_name_list,interval_list,fs,...
```

```
type,filter_order,freq_fraction);

end

% A plot of the log determinant of each of the dependency matricies (merits) as a function of
time.
if merits_time == 1 || (summarise_time == 1 && summarise_data == 1)

[merit_R_St,merit_R_Sresamplet,merit_R_Sfiltert,merit_R_Bt,merit_R_Ft,...
merit_R_Ct,merit_R_Cresamplet,merit_R_Cfiltert]...
= plot_merits_time...
(num_intervals,R_St,R_Sresamplet,R_Sfiltert,R_Bt,R_Ft,R_Ct,R_Cresamplet,...
R_Cfiltert,interval_list,merits_time);

end

% A plot of the log determinant of each of the dependency matricies (merits) where the x-axis
represents time as an expanding window (effectively a cumulative merit).
if merits_scale == 1 || (summarise_scale == 1 && summarise_data == 1)

[merit_R_SWt,merit_R_SresampleWt,merit_R_SfilterWt,merit_R_BWt,...
merit_R_FWt,merit_R_CWt,merit_R_CresampleWt,merit_R_CfilterWt]...
= plot_merits_scale...
(num_intervals,R_SWt,R_SresampleWt,R_SfilterWt,R_BWt,...
R_FWt,R_CWt,R_CresampleWt,R_CfilterWt,interval_list,merits_time);

end

% A plot of the log determinant of each of the dependency matricies (merits) as a function of
sampling frequency (only for resample and resample and filter corrleations).
if merits_resampling == 1 ||...
   (summarise_resampling == 1 && summarise_data == 1)

[merit_R_SresampleSt,merit_R_SfilterSt,merit_R_CresampleSt,...
merit_R_CfilterSt] ...
= plot_merits_resampling...
(fs_range,num_freq,R_SresampleSt,R_SfilterSt,R_CresampleSt,R_CfilterSt,...
merits_resampling);

end

% A summary of all the results with each dependency method in a seperate figure.
% Each function is for varying: time, time window or resample.
if summarise_time == 1 && summarise_data == 1

[str2] = summarise_results_time...
(largest_R_Bt,largest_R_Ct,largest_R_Sfiltert,largest_R_Ft,...
largest_R_Sresamplet,largest_R_St,largest_R_Cresamplet,largest_R_Cfiltert,...
merit_R_Bt,merit_R_Ct,merit_R_Sfiltert,merit_R_Ft,merit_R_Sresamplet,...
merit_R_St,merit_R_Cresamplet,merit_R_Cfiltert,...
R_Bt,R_Ct,R_Sfiltert,R_Ft,R_Sresamplet,R_St,R_Cresamplet,R_Cfiltert,...
summarise_data,summarise_standard,summarise_standard_resample,...
summarise_standard_filter,summarise_brownian,summarise_fourier,...
summarise_copula,summarise_copula_resample,summarise_copula_filter,...
relate_feature_list,interval_list);

end
```

```
if summarise_scale == 1 && summarise_data == 1

[str2] = summarise_results_scale...
(largest_R_BWt,largest_R_CWt,largest_R_SfilterWt,largest_R_FWt,...
largest_R_SresampleWt,largest_R_SWt,largest_R_CresampleWt,...
largest_R_CfilterWt,...
merit_R_BWt,merit_R_CWt,merit_R_SfilterWt,merit_R_FWt,...
merit_R_SresampleWt,merit_R_SWt,merit_R_CresampleWt,merit_R_CfilterWt,...
R_BWt,R_CWt,R_SfilterWt,R_FWt,R_SresampleWt,R_SWt,R_CresampleWt,...
R_CfilterWt,...
summarise_data,summarise_standard,summarise_standard_resample,...
summarise_standard_filter,summarise_brownian,summarise_fourier,...
summarise_copula,summarise_copula_resample,summarise_copula_filter,...
relate_feature_list,interval_list);

end

if summarise_resampling == 1 && summarise_data == 1

[str2] = summarise_results_resampling...
(largest_R_SfilterSt,largest_R_SresampleSt,largest_R_CresampleSt,...
largest_R_CfilterSt,...
merit_R_SfilterSt,merit_R_SresampleSt,merit_R_CresampleSt,merit_R_CfilterSt,...
R_SfilterSt,R_SresampleSt,R_CresampleSt,R_CfilterSt,...
summarise_data,summarise_standard_resample,summarise_standard_filter,...
summarise_copula_resample,summarise_copula_filter,...
relate_feature_list,fs_range,num_freq);

end

% A figure with a visualisation of each of the dependency matricies.
% The time updates every 3 seconds, so to close the figure, call "ctrl-C"
% in the command window. Each function is for varying: time, time window or resample.
if visualise_matrix == 1 && visualise_time == 1

[str] = visualise_matrices_time...
(R_St,R_Sresamplet,R_Sfiltert,R_Bt,R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,...
num_intervals,selected_features,feature_name_list,relate_feature_list);

end

if visualise_matrix == 1 && visualise_scale == 1

[str] = visualise_matrices_scale...
(R_SWt,R_SresampleWt,R_SfilterWt,R_BWt,R_FWt,R_CWt,R_CresampleWt,...
R_CfilterWt,num_intervals,selected_features,feature_name_list,...
relate_feature_list);

end

if visualise_matrix == 1 && visualise_resampling == 1

[str] = visualise_matrices_resampling...
(R_SresampleSt,R_SfilterSt,R_CresampleSt,R_CfilterSt,selected_features,...
feature_name_list,relate_feature_list);

end
```

```
disp('End of script');

% Matthew Newton
% August - October 2017
```

Data

Import Data

```
% Imports all the data from csv file, putting the values (price and sizes) into a matrix and
then the DateTime values into a seperate matrix.

function
[XOMMarketDepthOct2016_Values,XOMMarketDepthOct2016_DateTime,XOMTransactionsOct2016,XOMTransa
ctionsOct2016_DateTime] = import_data()

% Initialize variables.
filename = 'C:\Users\mnewton\Documents\MATLAB\Final\data\XOM_MarketDepth_Oct2016.csv';
delimiter = ',';
startRow = 3;

% Format for each line of text:
formatSpec =
'%*q%*q%*q%*q%*q%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%
f%f%[^\n\r]';

% Open the text file.
fileID = fopen(filename,'r');

% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',
'EmptyValue', NaN, 'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

% Close the text file.
fclose(fileID);

% Create output variable
XOMMarketDepthOct2016_Values = [dataArray{1:end-1}];

% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;

% Initialize variables.
filename = 'C:\Users\mnewton\Documents\MATLAB\Final\data\XOM_MarketDepth_Oct2016.csv';
delimiter = ',';
startRow = 3;

% Format for each line of text:
formatSpec =
'%*q%*q%*q%q%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s
%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%[^\n\r]';

% Open the text file.
fileID = fopen(filename,'r');
```

```
% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',
'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

% Close the text file.
fclose(fileID);

% Create output variable
XOMMarketDepthOct2016_DateTime = [dataArray{1:end-1}];

% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;

% Initialize variables.
filename = 'C:\Users\mnewton\Documents\MATLAB\Final\data\XOM_Transactions_Oct2016.csv';
delimiter = ',';
startRow = 4;

% Format for each line of text:
formatSpec = '%*q%*q%*q%*q%*q%f%f%[^\n\r]';

% Open the text file.
fileID = fopen(filename,'r');

% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',
'EmptyValue', NaN, 'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

% Close the text file.
fclose(fileID);

% Create output variable
XOMTransactionsOct2016 = [dataArray{1:end-1}];

% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;

% Initialize variables.
filename = 'C:\Users\mnewton\Documents\MATLAB\Final\data\XOM_Transactions_Oct2016.csv';
delimiter = ',';
startRow = 4;

% Format for each line of text:
formatSpec = '%*q%*q%*q%q%*s%*s%*s%[^\n\r]';

% Open the text file.
fileID = fopen(filename,'r');

% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',
'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

% Close the text file.
fclose(fileID);

% Create output variable
XOMTransactionsOct2016_DateTime = [dataArray{1:end-1}];
```

```
% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;

% Remove corrupted data
XOMMarketDepthOct2016_Values(isnan(XOMMarketDepthOct2016_Values)) = 0;
XOMMarketDepthOct2016_Values(abs(XOMMarketDepthOct2016_Values) > 10^20) = 0;

% Replaces zeros with the previous value
for j = 1:size(XOMMarketDepthOct2016_Values,2)

    for i = 1:size(XOMMarketDepthOct2016_Values,1)

        if XOMMarketDepthOct2016_Values(i,j) == 0

            XOMMarketDepthOct2016_Values(i,j) = XOMMarketDepthOct2016_Values(i-1,j);
            XOMMarketDepthOct2016_DateTime(i,1) = XOMMarketDepthOct2016_DateTime(i-1,1);

        end

    end

end

end
```

Estimator Functions

Standard Correlation

```
% Computes the linear correlation using the standard MATLAB function.

function R_S = standard_correlation(Values,selected_features)

% Checks wether there are any values to compare, returns a zero matrix otherwise.
if ~isempty(Values)

    % Return log difference.
    Values_lr = diff(log(Values));

    % Pre-allocate matrix.
    R_S = zeros(2*size(Values,2),2*size(Values,2));

    % Cycles through all features and find the corrcoef of each.
    for i = selected_features

        for j = selected_features

        R_S(2*i-1:2*i,2*j-1:2*j) = corrcoef(Values_lr(:,i),Values_lr(:,j));

        end

    end
```

```
    R_S2 = zeros(2*size(Values,2),2*size(Values,2));

    % Extracts the relevant element of each 2x2 matrix and puts it into a
    % smaller matrix.
    for i = selected_features

        for j = selected_features

            A = R_S(2*i-1:2*i,2*j-1:2*j);
            R_S2(i,j) = A(2,1);

        end
    end

    % Only keeps the releveant terms.
    R_S = zeros(size(selected_features,2),size(selected_features,2));

    n = 1;

    for i = selected_features

        m = 1;

        for j = selected_features

            if isnan(R_S2(i,j))

                R_S(n,m) = 0;

                m = m + 1;

            else

                R_S(n,m) = real(R_S2(i,j));

                m = m + 1;

            end

        end

        n = n + 1;


    end

else

    R_S = zeros(size(selected_features,2),size(selected_features,2));

end


end
```

Brownian Estimator

```
% Uses Takaki Hayashi and Nakahiro Yoshidas method from their paper "On covariance
% estimation of non-sychronously observed diffusion processes" - 2005. Where the
% data is effectively resychonised through the overlapping changes between
% features.

% The funciton is titled "brownian estimator" as the method is based on
% brownian motion

function [R_B] = brownian_estimator(Values,selected_features)

% Takes the log of the Values and find the difference between the previous
% element and the current element (X(2)-X(1)) is the first element.  So the
% total size is reduced by one.
Values_lr = diff(log(Values));

% Matrix to contain the non-zero values of the Values_lr with the start and
% end of their time intevals (index values).
% Will be used to compare the values contained within it.
compare = zeros(size(Values_lr,1),2*size(Values,2));

% Checks wether there is any data to use.
if ~isempty(Values)

for j = 1:size(Values_lr,2)

    % Parameter to count the rows.
    n = 1;

    for i = 1:size(Values_lr,1)

        if Values_lr(i,j) ~= 0 && Values_lr(i,j) ~= -Inf ...
           && Values_lr(i,j) ~= Inf && ~isnan(Values_lr(i,j))

            compare(n,2*j-1) = Values_lr(i,j);  %Value to use

            compare(n,2*j) = i + 1;             %The index of the value

            n = n + 1;

        end

    end

end

% Removes redundent rows from the compare matrix.
compare(all(compare == 0,2),:) = [];

% Add row at top of compare to make next section easier.
compare = [toeplitz(mod(0:0,2),mod(0:(2*size(Values,2)-1),2));compare];

% Preallocate correlation matrix.
R_B = zeros(size(Values,2),size(Values,2));

% Compare each feature to one another by using the time windows between
```

```
% each element relative to the different features.
for g = selected_features

    for h = selected_features

        % Create matrix to hold overlapping terms that are to be summed.
        terms = zeros(10*max(size(compare)),1);

        % Parameter to add terms to "terms".
        n = 1;

        % Cycles through all the time elements of the feature g.
        for  i = 1:(size(compare,1) - 1)

            % Time boundaries on feature g.
            t1g = compare(i,2*g);
            t2g = compare(i+1,2*g);

            % Cycles through all the time elements of the feature h.
            for j = 1:(size(compare,1) - 1)

                % Time boundaries on feature h.
                t1h = compare(j,2*h);
                t2h = compare(j+1,2*h);

                % h1 is between g1 and g2, h2 is outside or equal to g2.
                if t1h >= t1g && t1h < t2g && t2h >= t2g

                    terms(n) = (compare(i+1,2*g-1))*(compare(j+1,2*h-1));
                    n = n + 1;

                % h1 is between g1 and g2, h2 is inside g2.
                elseif t1h >= t1g && t1h < t2g && t2h < t2g

                    terms(n) = (compare(i+1,2*g-1))*(compare(j+1,2*h-1));
                    n = n + 1;

                % g1 is between h1 and h2, g2 is outside or equal to h2.
                elseif t1g >= t1h && t1g < t2h && t2g >= t2h

                    terms(n) = (compare(i+1,2*g-1))*(compare(j+1,2*h-1));
                    n = n + 1;

                % g1 is between h1 and h2, g2 is inside h2.
                elseif t1g >= t1h && t1g < t2h && t2g < t2h

                    terms(n) = (compare(i+1,2*g-1))*(compare(j+1,2*h-1));
                    n = n + 1;

                % Outside the region of interest.
                elseif t1h >= t2g

                    break

                end

            end
```

```
        end

        R_B(h,g) = sum(terms);

    end

end

% The estimator can be standardised to find the dependencies.

% Contains all the sigma terms corresponding to each feature.
sigma = zeros(size(Values,2),1);

for i = selected_features

    X = compare(:,2*i-1);

    sigma(i) = (sum(X.^2))^0.5;

end

% Standardises the dependencies.
for j = selected_features

    for i = selected_features

        R_B(i,j) = (R_B(i,j))./(sigma(i)*sigma(j));

    end

end

% Only keeps the releveant selected features.
R_B2 = zeros(size(selected_features,2),size(selected_features,2));
n = 1;

for i = selected_features

    m = 1;

    for j = selected_features

        if isnan(R_B(i,j))

            R_B2(n,m) = 0;

            m = m + 1;

        else

            R_B2(n,m) = R_B(i,j);

            m = m + 1;

        end

    end
```

```
    n = n + 1;


end

R_B = R_B2;

else

    R_B = zeros(size(selected_features,2),size(selected_features,2));

end


end
```

Fourier Estimator

```
% Uses a Fourier transform method from the paper " A Fourier transform method for
nonparametric estimation of multivariate volatility" - Paul Malliavin and Maria Mancino
(2009).
% The Fourier transform is used to deal with the asychronisity of the time-series data.

% The following papers aided in providing a better understanding of this method:

% "Fourier method for the measurement of univariate and multivariate volatility in the
presence of high frequency data" - Chanel Malherbe (2007).
% "Fourier series method for measurement of multivariate volatilites" - Paul Malliavin and
Maria Elvira Mancino (2002).

function [R_F] = fourier_estimator(Values,DateTime,selected_features)

% Checks wether there is any data to use.
if ~isempty(Values)

% Rescale the timeperiod between 0 and 2pi.
DateTime_rescaled = zeros(size(DateTime,1),size(DateTime,2));

t1 = datenum(DateTime(1));
tn = datenum(DateTime(end));

for i = 1:size(DateTime_rescaled)

    DateTime_rescaled(i) = (2*pi*(datenum(DateTime(i)) - t1))/(tn - t1);

end

% The log values.
Values_l = log(Values);

% Number of Fourier coefficients to be calculated.
% These can be changed but have been kept constant since creating the function.
N = 1000;
nrfc = N/2;

% Number of Fourier coefficients to exclude from the beginning.
n0 = 1;
```

```
R_F = zeros(size(Values,2),size(Values,2));

% Cycles throught different features.
for h = selected_features

    % Matricies to contain all the fourier coefficients.
    fca1 = zeros(round(nrfc),1);
    fcb1 = zeros(round(nrfc),1);

    fca2 = zeros(round(nrfc),1);
    fcb2 = zeros(round(nrfc),1);

    % Cycles through all the values of k, calculated the fourier transfer
    % coefficeints for each k.
    for k = 1:round(nrfc)

        terms_cos = zeros(size(DateTime_rescaled,1),1);
        terms_sin = zeros(size(DateTime_rescaled,1),1);

        for i = 1:(size(DateTime_rescaled,1) - 1)

            terms_cos(i) = (cos(k*DateTime_rescaled(i)) - ...
                cos(k*DateTime_rescaled(i+1)))*Values_l(i,h);
            terms_sin(i) = (sin(k*DateTime_rescaled(i)) - ...
                sin(k*DateTime_rescaled(i+1)))*Values_l(i,h);

        end

        fca1(k,1) = (1/pi)*(sum(terms_cos));
        fcb1(k,1) = (1/pi)*(sum(terms_sin));

    end

    % Sums the fourier coeff, uses formula to find varaince.
    var1 = (pi^2/(nrfc + 1 - n0))*...
        (sum(fca1(n0:round(nrfc)).*fca1(n0:round(nrfc))) + ...
         sum(fcb1(n0:round(nrfc)).*fcb1(n0:round(nrfc))));

    % Cycles through the second features to be compared against the first.
    for g = selected_features

        % Repeats same calculation as above.
        for k = 1:round(nrfc)

            terms_cos = zeros(size(DateTime_rescaled,1),1);
            terms_sin = zeros(size(DateTime_rescaled,1),1);

            for i = 1:(size(DateTime_rescaled,1) - 1)

                terms_cos(i) = (cos(k*DateTime_rescaled(i)) - ...
                    cos(k*DateTime_rescaled(i+1)))*Values_l(i,g);
                terms_sin(i) = (sin(k*DateTime_rescaled(i)) - ...
                    sin(k*DateTime_rescaled(i+1)))*Values_l(i,g);

            end

            fca2(k,1) = (1/pi)*(sum(terms_cos));
```

```
            fcb2(k,1) = (1/pi)*(sum(terms_sin));

        end

        % Calculate the integrated volatility and covolatility over the
        % entire time window.
        covar = (pi^2/(nrfc + 1 - n0))*...
                (sum(fca1(n0:round(nrfc)).*fca2(n0:round(nrfc))) +
sum(fcb1(n0:round(nrfc)).*fcb2(n0:round(nrfc))));

        var2 = (pi^2/(nrfc + 1 - n0))*...
               (sum(fca2(n0:round(nrfc)).*fca2(n0:round(nrfc))) +
sum(fcb2(n0:round(nrfc)).*fcb2(n0:round(nrfc))));

       % Ignores terms that have zero varaince as it will create an error.
        if ((var1 > 0) && (var2 > 0))

            % Calculate the dependency matrix.
            R_F(h,g) = covar/(sqrt(var1*var2));

        else

            R_F(h,g) = 0;

        end

    end

end

% Only keeps the releveant terms.
R_F2 = zeros(size(selected_features,2),size(selected_features,2));
n = 1;

for i = selected_features

    m = 1;

    for j = selected_features

        if isnan(R_F(i,j))

            R_F2(n,m) = 0;

            m = m + 1;

        else

            R_F2(n,m) = R_F(i,j);

            m = m + 1;

        end

    end

    n = n + 1;
```

```
end

R_F = R_F2;


else

    R_F = zeros(size(selected_features,2),size(selected_features,2));

end


end
```

Copula Estimator

```
% Copula estimator to measure the dependency between the different features.
% The inbuild MATLAB function "copulafit" is used to find a student-t copula fit for the
data.
% Features that are too similar are removed when copulafit is run and added back into the
dependency matrix afterwards.

% The understanding of copulas and the reasoning for using them in this context were obtained
from the following papers.

% "Copulas: A persoanl view" - Paul Embrechts 2009.
% "A review of copula models for economic time series" - Andrew Patton 2012.
% "Modelling dependencies: An Overview" - Martyn Dorey & Phil Joubert 2005.

% The options at the top of this function are not intended for standard use, they have been
left in incase the user would like to experiment with them!

function [R_C] = copula_estimator(Values,selected_features)

% Determines whether copulafit includes all features or just the selected features.
copula_all_features = 0;

% Determines whether the removed components are merged or just removed.
copula_merge = 0;

% Runs the pca analysis.
copula_pca = 0;

if copula_pca == 0

% Compute log returns.
Values_lr = diff(log(Values));

% Avoids issues with complex numbers.
Values_lr = real(Values_lr);

if copula_all_features == 0

    % Converts the data into a scale for copulas using a kernel estimator.
    Values_ks = zeros(size(Values_lr,1),size(selected_features,1));
```

```
    n = 1;
    for i = selected_features

        Values_ks(:,n) = ksdensity(Values_lr(:,i),Values_lr(:,i),'function','cdf');
        n = n + 1;

    end

end

if copula_all_features == 1

    selected_features = 1:40;

    % Converts the data into a scale for copulas using a kernel estimator.
    Values_ks = zeros(size(Values_lr,1),size(selected_features,1));

    n = 1;
    for i = selected_features

        Values_ks(:,n) = ksdensity(Values_lr(:,i),Values_lr(:,i),'function','cdf');
        n = n + 1;

    end

end

% Looks for data that is too similar using their mean squared error.
% If this is not done the columns in Values_ks will be too similar and the copulafit function
will be rank defficient.
n = 1;
mark = zeros(size(Values_ks,2));

for i = 1:size(Values_ks,2)
    m = 2;
    for j = i:size(Values_ks,2)

        if j ~= i

            Values_ks_diff = Values_ks(:,i) - Values_ks(:,j);

            % Criteria to determine similarity.
            if mse(Values_ks_diff) < 5e-15

                % Mark matrix to contain reference of columns to be removed
                % for being too similar.
                 mark(n,1) = i;
                 mark(n,m) = j;
                 m = m + 1;


            end

        end

    end
    n = n + 1;
end
```

```
% Removes rows and columns that are all zeros.
mark(all(mark == 0,2),:) = [];
mark(:,all(mark == 0,1)) = [];

% Converts the marked columns from Values_ks to zero.
for i = 1:size(mark,1)

    for j = 2:size(mark,2)

        if mark(i,j) ~= 0

            % Merges the columns if the option is selected.
            if copula_merge == 1

                Values_ks(:,mark(i,1)) = (Values_ks(:,mark(i,1)) + Values_ks(:,mark(i,j)))/2;

            end

            Values_ks(:,mark(i,j)) = 0;

        end

    end

end

% Removes the zero columns (marked columns) from Values_ks.
Values_ks(:,all(Values_ks == 0,1)) = [];

% Parameter to determine if the copula fit has failed or not.
a = 0;
try

    % Fits the data to the specified copula.
    % Could change approximateML to ML (Max liklihood).
    [Rho,~] = copulafit('t',Values_ks,'Method','ApproximateML');

catch

    a = 1;

    try

        % Looks for data that is too similar using a mean squared error.
        % If this is not done the columns in Values_ks will be too similar
        % and the copulafit function will be rank defficient.
        n = 1;
        mark = zeros(size(Values_ks,2));

        for i = 1:size(Values_ks,2)
            m = 2;
            for j = i:size(Values_ks,2)

                if j ~= i

                    Values_ks_diff = Values_ks(:,i) - Values_ks(:,j);
```

```
                if mse(Values_ks_diff) < 5e-14

                    % Mark matrix to contain reference of columns to be removed
                    % for being too similar.
                     mark(n,1) = i;
                     mark(n,m) = j;
                     m = m + 1;


                end

            end

        end
        n = n + 1;
    end

    % Removes rows and columns that are all zeros.
    mark(all(mark == 0,2),:) = [];
    mark(:,all(mark == 0,1)) = [];

    % Converts the marked columns from Values_ks to zero.
    for i = 1:size(mark,1)

        for j = 2:size(mark,2)

            if mark(i,j) ~= 0

                % Merges the columns if the option is selected.
                if copula_merge == 1

                    Values_ks(:,mark(i,1)) = (Values_ks(:,mark(i,1)) ...
                    + Values_ks(:,mark(i,j)))/2;

                end

                Values_ks(:,mark(i,j)) = 0;

            end

        end

    end

    % Removes the zero columns (marked columns) from Values_ks.
    Values_ks(:,all(Values_ks == 0,1)) = [];

    % Fits the data to the specified copula.
    % Could change approximateML to ML (Max liklihood).
    [Rho,~] = copulafit('t',Values_ks,'Method','ApproximateML');

catch

    a = 2;

end

end
```

```
if a == 1

    % Looks for data that is too similar using a mean squared error.
    % If this is not done the columns in Values_ks will be too similar and the
    % copulafit function will be rank defficient.
    n = 1;
    mark = zeros(size(Values_ks,2));

    for i = 1:size(Values_ks,2)
        m = 2;
        for j = i:size(Values_ks,2)

            if j ~= i

                Values_ks_diff = Values_ks(:,i) - Values_ks(:,j);

                if mse(Values_ks_diff) < 5e-14

                    % Mark matrix to contain reference of columns to be removed
                    % for being too similar.
                     mark(n,1) = i;
                     mark(n,m) = j;
                     m = m + 1;


                end

            end

        end
        n = n + 1;
    end

    % Removes rows and columns that are all zeros.
    mark(all(mark == 0,2),:) = [];
    mark(:,all(mark == 0,1)) = [];

    % Converts the marked columns from Values_ks to zero.
    for i = 1:size(mark,1)

        for j = 2:size(mark,2)

            if mark(i,j) ~= 0

                % Merges the columns if the option is selected.
                if copula_merge == 1

                    Values_ks(:,mark(i,1)) = (Values_ks(:,mark(i,1))...
                    + Values_ks(:,mark(i,j)))/2;

                end

                Values_ks(:,mark(i,j)) = 0;

            end

        end
```

```
    end

    % Removes the zero columns (marked columns) from Values_ks.
    Values_ks(:,all(Values_ks == 0,1)) = [];

    % Fits the data to the specified copula.
    % Could change approximateML to ML (Max liklihood).
    [Rho,~] = copulafit('t',Values_ks,'Method','ApproximateML');

    % Dependency matrix calculated from copulas.
    R_C = 2.*asin(Rho)./pi;
    R_C2 = R_C;

    mark2 = mark(:,2:end);
    mark_unique = unique(mark2);
    mark_unique(mark_unique == 0) = [];

    % Add relvant vectors into the repeated gaps.
    for i = mark_unique.'

        [row,~] = find(mark == i,1,'last');
        R_C = [R_C(:,1:i-1) R_C(:,mark(row,1)) R_C(:,i:end)];
        R_C = [R_C(1:i-1,:); R_C(mark(row,1),:); R_C(i:end,:)];

    end

elseif a == 2

    R_C = ones(length(selected_features));

else

    % Dependency matrix calculated from copulas.
    R_C = 2.*asin(Rho)./pi;
    R_C2 = R_C;

    mark2 = mark(:,2:end);
    mark_unique = unique(mark2);
    mark_unique(mark_unique == 0) = [];

    % Add relvant vectors into the repeated gaps.
    for i = mark_unique.'

        [row,~] = find(mark == i,1,'last');
        R_C = [R_C(:,1:i-1) R_C(:,mark(row,1)) R_C(:,i:end)];
        R_C = [R_C(1:i-1,:); R_C(mark(row,1),:); R_C(i:end,:)];

    end

end

% This section is incomplete, the idea is to use pca analysis to determine whether the
copula_fit will work.
% The code has been commented out for potential future improvements.
elseif copula_pca == 1

    Values_lr = diff(log(Values));
```

```
    [COEFF, SCORE, LATENT, TSQUARED, EXPLAINED, MU] = pca(Values_lr);


%     m = 1;
%     for i = 1:length(EXPLAINED)
%
%         sum_exp = sum(EXPLAINED(1:i,1));
%
%         if sum_exp < 99
%
%             m = m + 1;
%
%         end
%
%     end
%
%     Values_lr_recon = SCORE(:,1:m)*COEFF(:,1:m)';
%
%     Values_ks = zeros(size(Values_lr_recon,1),size(selected_features,1));
%
%     % Converts the data into a scale for copulas using a kernel estimator
%     n = 1;
%     for i = selected_features
%
%         Values_ks(:,n) =
ksdensity(Values_lr_recon(:,i),Values_lr_recon(:,i),'function','cdf');
%         n = n + 1;
%
%     end
%
%     [Rho,~] = copulafit('t',Values_ks,'Method','ApproximateML');  %ApproximateML or ML
%
%     % Correlation matrix calculated from copulas
%     R_C = 2.*asin(Rho)./pi;


end

end
```

Global Functions

Select Features

```
% Selects the relevenet features subject to the inputs in the main script.

function [selected_features] = select_features(select_ask_prices, select_bid_prices,
select_ask_sizes, select_bid_sizes)

if any(select_ask_prices < 1) || any(select_ask_prices > 10) ||...
   any(select_bid_prices < 1) || any(select_bid_prices > 10) || ...
   any(select_ask_sizes < 1) || any(select_ask_sizes > 10) || ...
   any(select_bid_sizes < 1) || any(select_bid_sizes > 10)

    error('Features to select must be between 1 and 10');
```

```
else

    selected_features = [4*(select_bid_prices)-3, 4*(select_bid_sizes)-2,
4*(select_ask_prices)-1, 4*(select_ask_sizes)];

    selected_features = sort(selected_features);

end

end
```

Calculate All Dependencies Vary Time

```
% Calculates all dependecies along with a list of their most significant values, uses
seperate intervals to compute dependencies.
% Also creates a list of intervals to be used in the plots.

function [R_St,R_Sresamplet,R_Sfiltert,R_Bt,R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,...
    largest_R_St,largest_R_Sresamplet,largest_R_Sfiltert,largest_R_Bt,largest_R_Ft,...
    largest_R_Ct,largest_R_Cresamplet,largest_R_Cfiltert] =
calculate_all_dependencies_vary_time(selected_features,num_intervals,num_max,Set_Date_1,
Set_Date_2, XOMMarketDepthOct2016_Values,DateTime_Con,fs,type,freq_fraction,filter_order)


% Pre-define matrices.
% Each array conatains each matrix, with the third dimension representing the different
times.
R_St = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Sresamplet = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Sfiltert = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Bt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Ft = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Ct = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Cresamplet = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_Cfiltert = zeros(size(selected_features,2),size(selected_features,2),num_intervals);

largest_R_St = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Sresamplet = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Sfiltert = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Bt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Ft = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Ct = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Cresamplet = zeros(round(0.5*num_max),3,num_intervals);
largest_R_Cfiltert = zeros(round(0.5*num_max),3,num_intervals);

% Runs each interval as a seperate loop.
for i = 1:num_intervals

X = ['Finding dependencies for interval number ', num2str(i)];
disp(X);

% Sorts the useful data into two matrices, Values containing all the feature numerical data,
DateTime containing all the dates corresponding to the numerical data.
[Values,DateTime] = find_datetime_time_interval(Set_Date_1,Set_Date_2,
XOMMarketDepthOct2016_Values, DateTime_Con,i,num_intervals);
```

```
% Resamples the data.
% Type = 0 ('event') takes the most recent event in the data.
% Type = 1 ('inter') linear interpolates the data.
[Values_resample,~] = resample_data(Values,DateTime,fs,type,selected_features);

% Filters the resampled data using a butterworth filter.
[Values_filter] = filter_data(filter_order,freq_fraction,Values_resample);

% Computes all dependency methods for the required data.
R_S = standard_correlation(Values,selected_features);
R_Sresample = standard_correlation(Values_resample,selected_features);
R_Sfilter = standard_correlation(Values_filter,selected_features);
R_B = brownian_estimator(Values,selected_features);
R_F = fourier_estimator(Values,DateTime,selected_features);
R_C = copula_estimator(Values,selected_features);
R_Cresample = copula_estimator(Values_resample,selected_features);
R_Cfilter = copula_estimator(Values_filter,selected_features);

% Puts the matricies in the 3D matrix with time.
R_St(:,:,i) = R_S;
R_Sresamplet(:,:,i) = R_Sresample;
R_Sfiltert(:,:,i) = R_Sfilter;
R_Bt(:,:,i) = R_B;
R_Ft(:,:,i) = R_F;
R_Ct(:,:,i) = R_C;
R_Cresamplet(:,:,i) = R_Cresample;
R_Cfiltert(:,:,i) = R_Cfilter;

% Finds the most signicant dependicies for each time interval.
largest_R_St(:,:,i) = find_largest_R(R_S,num_max);
largest_R_Sresamplet(:,:,i) = find_largest_R(R_Sresample,num_max);
largest_R_Sfiltert(:,:,i) = find_largest_R(R_Sfilter,num_max);
largest_R_Bt(:,:,i) = find_largest_R(R_B,num_max);
largest_R_Ft(:,:,i) = find_largest_R(R_F,num_max);
largest_R_Ct(:,:,i) = find_largest_R(R_C,num_max);
largest_R_Cresamplet(:,:,i) = find_largest_R(R_Cresample,num_max);
largest_R_Cfiltert(:,:,i) = find_largest_R(R_Cfilter,num_max);

X = ['Dependencies for intveral number ', num2str(i), ' found'];
disp(X);

end

% Lists the date and time of the intervals.
interval_list = find_intervals(Set_Date_1,Set_Date_2,DateTime_Con,num_intervals);

end
```

Calculate All Dependencies Vary Window

```
% Calculates all dependecies along with a list of their most significant values, uses
seperate intervals to compute dependencies.

function [R_SWt,R_SresampleWt,R_SfilterWt,R_BWt,R_FWt,R_CWt,R_CresampleWt,R_CfilterWt,...
    largest_R_SWt,largest_R_SresampleWt,largest_R_SfilterWt,largest_R_BWt,largest_R_FWt,...
    largest_R_CWt,largest_R_CresampleWt,largest_R_CfilterWt] ...
    = calculate_all_dependencies_vary_window(selected_features,num_intervals,num_max,
```

```
Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,fs,type,
freq_fraction, filter_order)

% Pre-define matrices.
R_SWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_SresampleWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_SfilterWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_BWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_FWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_CWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_CresampleWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);
R_CfilterWt = zeros(size(selected_features,2),size(selected_features,2),num_intervals);

largest_R_SWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_SresampleWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_SfilterWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_BWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_FWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_CWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_CresampleWt = zeros(round(0.5*num_max),3,num_intervals);
largest_R_CfilterWt = zeros(round(0.5*num_max),3,num_intervals);

for i = 1:num_intervals

X = ['Finding dependencies for window number ', num2str(i)];
disp(X);

% Puts data into two matrices, Values containing all the numerical data and DateTime
containing all the dates.
[Values,DateTime] =
find_datetime_time_window(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,i,n
um_intervals);

% Resamples the data.
% Type = 0 ('event') takes the most recent event in the data.
% Type = 1 ('inter') linear interpolates the data.
[Values_resample,~] = resample_data(Values,DateTime,fs,type,selected_features);

% Filters the data using a butterworth filter.
[Values_filter] = filter_data(filter_order,freq_fraction,Values_resample);

% Computes all dependency methods for the required data.
R_SW = standard_correlation(Values,selected_features);
R_SresampleW = standard_correlation(Values_resample,selected_features);
R_SfilterW = standard_correlation(Values_filter,selected_features);
R_BW = brownian_estimator(Values,selected_features);
R_FW = fourier_estimator(Values,DateTime,selected_features);
R_CW = copula_estimator(Values,selected_features);
R_CresampleW = copula_estimator(Values_resample,selected_features);
R_CfilterW = copula_estimator(Values_filter,selected_features);

% Puts the matricies in the 3D matrix with time.
R_SWt(:,:,i) = R_SW;
R_SresampleWt(:,:,i) = R_SresampleW;
R_SfilterWt(:,:,i) = R_SfilterW;
R_BWt(:,:,i) = R_BW;
R_FWt(:,:,i) = R_FW;
R_CWt(:,:,i) = R_CW;
```

```
R_CresampleWt(:,:,i) = R_CresampleW;
R_CfilterWt(:,:,i) = R_CfilterW;

% Finds the most signicant dependicies for each time window.
largest_R_SWt(:,:,i) = find_largest_R(R_SW,num_max);
largest_R_SresampleWt(:,:,i) = find_largest_R(R_SresampleW,num_max);
largest_R_SfilterWt(:,:,i) = find_largest_R(R_SfilterW,num_max);
largest_R_BWt(:,:,i) = find_largest_R(R_BW,num_max);
largest_R_FWt(:,:,i) = find_largest_R(R_FW,num_max);
largest_R_CWt(:,:,i) = find_largest_R(R_CW,num_max);
largest_R_CresampleWt(:,:,i) = find_largest_R(R_CresampleW,num_max);
largest_R_CfilterWt(:,:,i) = find_largest_R(R_CfilterW,num_max);

X = ['Dependencies for window number ', num2str(i), ' found'];
disp(X);

end

end
```

Calculate All Dependencies Vary Resampling

```
% Calculates all dependecies along with a list of their most significant values, uses
different resample rates to compute dependencies.

function [R_SresampleSt,R_SfilterSt,R_CresampleSt,R_CfilterSt,...
    largest_R_SresampleSt,largest_R_SfilterSt,largest_R_CresampleSt,largest_R_CfilterSt] = .
calculate_all_dependencies_vary_resampling(selected_features,num_max,Set_Date_1,Set_Date_2,..
XOMMarketDepthOct2016_Values,DateTime_Con,type,freq_fraction,filter_order,fs_range,num_freq)

% Predefine matrices
R_SresampleSt = zeros(size(selected_features,2),size(selected_features,2),num_freq);
R_SfilterSt = zeros(size(selected_features,2),size(selected_features,2),num_freq);
R_CresampleSt = zeros(size(selected_features,2),size(selected_features,2),num_freq);
R_CfilterSt = zeros(size(selected_features,2),size(selected_features,2),num_freq);

largest_R_SresampleSt = zeros(round(0.5*num_max),3,num_freq);
largest_R_SfilterSt = zeros(round(0.5*num_max),3,num_freq);
largest_R_CresampleSt = zeros(round(0.5*num_max),3,num_freq);
largest_R_CfilterSt = zeros(round(0.5*num_max),3,num_freq);

% Calls data to be used.
[Values,DateTime] =
find_datetime_no_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con);

% Vector of all frequencies to be computed.
fs = linspace(fs_range(1),fs_range(2),num_freq);

for i = 1:length(fs)

    X = ['Finding dependencies for frequency ', num2str(fs(i))];
    disp(X);

    % Resamples the data at each frequency.
    [Values_resample,~] = resample_data(Values,DateTime,fs(i),type,selected_features);

    % Filters the data using a butterworth filter.
```

```
    [Values_filter] = filter_data(filter_order,freq_fraction,Values_resample);

    % Computes all dependency methods for the required data.
    R_SresampleS = standard_correlation(Values_resample,selected_features);
    R_SfilterS = standard_correlation(Values_filter,selected_features);
    R_CresampleS = copula_estimator(Values_resample,selected_features);
    R_CfilterS = copula_estimator(Values_filter,selected_features);

    % Puts the matricies in the 3D matrix with time.
    R_SresampleSt(:,:,i) = standard_correlation(Values_resample,selected_features);
    R_SfilterSt(:,:,i) = standard_correlation(Values_filter,selected_features);
    R_CresampleSt(:,:,i) = copula_estimator(Values_resample,selected_features);
    R_CfilterSt(:,:,i) = copula_estimator(Values_filter,selected_features);

    % Finds the most signicant dependicies for each time interval.
    largest_R_SresampleSt(:,:,i) = find_largest_R(R_SresampleS,num_max);
    largest_R_SfilterSt(:,:,i) = find_largest_R(R_SfilterS,num_max);
    largest_R_CresampleSt(:,:,i) = find_largest_R(R_CresampleS,num_max);
    largest_R_CfilterSt(:,:,i) = find_largest_R(R_CfilterS,num_max);

    X = ['Dependencies for frequency ', num2str(fs(i)), ' found'];
    disp(X);

end

end
```

Find DateTime Time Interval

```
% Sorts the useful data into two matrices, Values containing all the feature numerical data,
DateTime containing all the dates corresponding to the numerical data.
% Finds rows in datetime corresonding to the two date and time inputs.
% Reduces the data to only contain data between this time window.

function [Values,DateTime] =
find_datetime_time_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,i
,num_intervals)

% Criteria to reject incorrect data.
if datenum(Set_Date_2) - datenum(Set_Date_1) < 0

    error('End date must be later than start date')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

        error('Start time must be later than 07:30:00')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

        error('End time must be earlier than 16:30:00')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

        error('Start time must be later than 07:30:00 and end time must be earlier than
```

```
16:30:00')

% If on the same day.
elseif day(datenum(Set_Date_2)) == day(datenum(Set_Date_1))

    % If inside the times of the relevant data '03-Oct-2016 07:30:00' and
    % '03-Oct-2016 16:30:00'.
    if  hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

        % Work out interval.
        time_interval = (datenum(Set_Date_2) - datenum(Set_Date_1))/num_intervals;

        % Finds start and end of each time interval.
        ti_start = datenum(Set_Date_1) + (i-1)*time_interval;
        ti_end = datenum(Set_Date_1) + (i)*time_interval;

        % Finds the closed DateTime to each start and end interval.
        [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
        [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));

        % Reducing the time range and makes easier to sort.
        Values = XOMMarketDepthOct2016_Values(t1:t2,:);
        DateTime = DateTime_Con(t1:t2);


    end

% If the end date is after the start date.
elseif day(datenum(Set_Date_2)) > day(datenum(Set_Date_1))

    % Finds the number of days between the start and the end.
    day_diff = day(datenum(Set_Date_2)) - day(datenum(Set_Date_1));

    % Random variables to determine the number of weekends.
    x = Set_Date_1;
    num_weekend = 0;

    % Finds the number of weekends between the two intervals.
    while day(datenum(Set_Date_2)) > day(datenum(x))

        if weekday(x) == 1

            num_weekend = num_weekend + 1;

            x = addtodate(datenum(x),1,'day');
            x = datestr(x);

        elseif weekday(x) == 7

            num_weekend = num_weekend + 2;

           x = addtodate(datenum(x),2,'day');
           x = datestr(x);

        else

            x = addtodate(datenum(x),1,'day');
```

```
        x = datestr(x);

    end

end

% The total time between the two dates.
elapsed_time = datenum(Set_Date_2) - datenum(Set_Date_1);

% Removes the deadtime, (when the market is closed).
relevant_time = elapsed_time - (day_diff - num_weekend)*(15/24) - num_weekend;

% The time interval, (how long each interval should be).
time_interval = (relevant_time)/num_intervals;

% Create vector to contain all the time intervals.
time_vec = zeros(i,1);
time_vec(1) = datenum(Set_Date_1);

% Inserts all the intervals into the vector.
for j = 1:i

    time_vec(j+1) = datenum(Set_Date_1) + (j)*time_interval;

end

% Add a hours such that none of the components of time_vec fall in the deadtime.
for j = 1:(i+1)

    if hour(time_vec(j)) + minute(time_vec(j))/60 > 16.5

        for k = j:(i+1)

            time_vec(k) = time_vec(k) + 15/24;

        end

    end

end

% Removes the weekend components of the deadtime.
for j = 1:(i+1)

    if weekday(datestr(time_vec(j))) == 7

        for k = j:(i+1)

            time_vec(k) = time_vec(k) + 2;

        end

    end

end

% Set ti_start and ti_end as the last two components of the time_vec.
ti_start = time_vec(length(time_vec) - 1);
```

```
    ti_end = time_vec(length(time_vec));

    % Finds the closed DateTime to each start and end interval.
    [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
    [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));

    % Reducing the time range and makes easier to sort.
    Values = XOMMarketDepthOct2016_Values(t1:t2,:);
    DateTime = DateTime_Con(t1:t2);


end

end
```

Find DateTime Time Window

```
% Sorts the useful data into two matrices, Values containing all the feature numerical data,
DateTime containing all the dates corresponding to the numerical data.
% Finds rows in datetime corresonding to the two date and time inputs.
% Reduces the data to only contain data between this time window.

function [Values,DateTime] =
find_datetime_time_window(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,i,n
um_intervals)

% Criteria to reject incorrect data.
if datenum(Set_Date_2) - datenum(Set_Date_1) < 0

    error('End date must be later than start date')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

        error('Start time must be later than 07:30:00')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

        error('End time must be earlier than 16:30:00')

elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

        error('Start time must be later than 07:30:00 and end time must be earlier than
16:30:00')

% If on the same day.
elseif day(datenum(Set_Date_2)) == day(datenum(Set_Date_1))

    % If inside the times of the relevant data '03-Oct-2016 07:30:00' and
    % '03-Oct-2016 16:30:00'.
    if  hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

        % Work out interval.
        time_interval = (datenum(Set_Date_2) - datenum(Set_Date_1))/num_intervals;
```

```
        % Finds start and end of each time interval.
        ti_start = datenum(Set_Date_1) + (i-1)*time_interval;
        ti_end = datenum(Set_Date_1) + (i)*time_interval;

        % Finds the closed DateTime to each start and end interval.
        [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
        [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));

        % Reducing the time range and makes easier to sort.
        Values = XOMMarketDepthOct2016_Values(t1:t2,:);
        DateTime = DateTime_Con(t1:t2);


    end

% If the end date is after the start date.
elseif day(datenum(Set_Date_2)) > day(datenum(Set_Date_1))

    % Finds the number of days between the start and the end.
    day_diff = day(datenum(Set_Date_2)) - day(datenum(Set_Date_1));

    % Random variables to determine the number of weekends.
    x = Set_Date_1;
    num_weekend = 0;

    % Finds the number of weekends between the two intervals.
    while day(datenum(Set_Date_2)) > day(datenum(x))

        if weekday(x) == 1

            num_weekend = num_weekend + 1;

            x = addtodate(datenum(x),1,'day');
            x = datestr(x);

        elseif weekday(x) == 7

            num_weekend = num_weekend + 2;

           x = addtodate(datenum(x),2,'day');
           x = datestr(x);

        else

            x = addtodate(datenum(x),1,'day');
            x = datestr(x);

        end

    end

    % The total time between the two dates.
    elapsed_time = datenum(Set_Date_2) - datenum(Set_Date_1);

    % Removes the deadtime, (when the market is closed).
    relevant_time = elapsed_time - (day_diff - num_weekend)*(15/24) - num_weekend;
```

```
    % The time interval, (how long each interval should be.
    time_interval = (relevant_time)/num_intervals;

    % Create vector to contain all the time intervals.
    time_vec = zeros(i,1);
    time_vec(1) = datenum(Set_Date_1);

    % Inserts all the intervals into the vector
    for j = 1:i

        time_vec(j+1) = datenum(Set_Date_1) + (j)*time_interval;

    end

    % Add a hours such that none of the components of time_vec fall in the deadtime.
    for j = 1:(i+1)

        if hour(time_vec(j)) + minute(time_vec(j))/60 > 16.5

            for k = j:(i+1)

                time_vec(k) = time_vec(k) + 15/24;

            end

        end

    end

   % Removes the weekend components of the deadtime.
    for j = 1:(i+1)

        if weekday(datestr(time_vec(j))) == 7

            for k = j:(i+1)

                time_vec(k) = time_vec(k) + 2;

            end

        end

    end

    % Set ti_start and ti_end as the last two components of the time_vec.
    ti_start = time_vec(1);
    ti_end = time_vec(length(time_vec));

    % Finds the closed DateTime to each start and end interval.
    [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
    [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));

    % Reducing the time range and makes easier to sort.
    Values = XOMMarketDepthOct2016_Values(t1:t2,:);
    DateTime = DateTime_Con(t1:t2);


end
```

```
end
```

Find DateTime No Interval

```
% Finds columns corresonding to the two date and time inputs.
% Reduces the data to only contain data between this time window.

function [Values,DateTime] =
find_datetime_no_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con)

[~,t1] = min(abs(datenum(Set_Date_1) - datenum(DateTime_Con)));
[~,t2] = min(abs(datenum(Set_Date_2) - datenum(DateTime_Con)));

% Reducing the time range and makes easier to sort.
Values = XOMMarketDepthOct2016_Values(t1:t2,:);
DateTime = DateTime_Con(t1:t2);

end
```

Resample Data

```
% Resamples the data in different ways depending on the input of type.
% Type = '0' takes the most recent event.
% Type = '1' linear interpolates the data.

function [Values_resample,DateTime_resample] =
resample_data(Values,DateTime,fs,type,selected_features)

if type == 1

    % Resample using built in resample function.
    [Values_resample,DateTime_resample] = resample(Values_selected,DateTime,fs);

end

% Resample by taking the previous event from the data.
% Includes all terms from values.
if type == 0

    % Finds the elapsed time between the start and end date.
    Time_diff = abs(etime(datevec(DateTime(1)),datevec(DateTime(end))));

    % Creates a vector of all the time increments.
    % Conceptualy, creates a grid to map the data onto.
    Time_series = seconds(0:(1/fs):Time_diff);

    % Creates a grid containing each resample point.
    grid_time = datenum(DateTime(1)) + datenum(Time_series);

    for i = 1:length(grid_time)

        if (hour(grid_time(i)) + (minute(grid_time(i)))/60) > 16.5 || ...
           (hour(grid_time(i)) + (minute(grid_time(i)))/60) < 7.5 || ...
            weekday(grid_time(i)) == 1 || weekday(grid_time(i)) == 7
```

```
            grid_time(i) = 0;

        end

    end

    grid_time(grid_time == 0) = [];

    Values_resample = zeros(length(grid_time),size(Values,2));
    DateTime_resample = zeros(length(grid_time),1);

    % Resamples the data for each feature.
    for h = selected_features

        k = 1;

        % Goes through each grid component.
        for i = 1:length(grid_time)

            for j = k:length(DateTime)

                if datenum(grid_time(i)) >= datenum(DateTime(j))

                    Values_resample(i,h) = Values(j,h);

                    DateTime_resample(i) = grid_time(i);

                else

                    if j == 1

                        % k term drastical reduce comutational time.
                        k = 1;

                    else

                        k = j - 1;

                    end

                    break

                end

            end

        end

    end

    % Removes all zero values from resampled data.
    Values_resample(all(Values_resample == 0,2),:) = [];
    DateTime_resample(all(DateTime_resample == 0,2),:) = [];

end
```

Filter Data

```
% Filters the resampled data using a butterworth filter.

function [Values_filter] = filter_data(filter_order,freq_fraction,Values_resample)

% Butter function finds numerator and denomator of low pass Butterworth filter.
[butter_num, butter_den] = butter(filter_order,freq_fraction,'low');

% Checks whether resamples has any terms.
if ~isempty(Values_resample)

    % Add rows to set up inital conditions for filter.
    Values_resample = [zeros(100,size(Values_resample,2));Values_resample];

    for i = 1:100

        Values_resample(i,:) = Values_resample(101,:);

    end

    % Applies the Butterworth filter.
    Values_filter = filter(butter_num,butter_den,Values_resample,[],1);

    % Remove the rows that were to fix initial conditions.
    Values_filter(1:100,:) = [];

else

    Values_filter = [];

end

end
```

Largest R

```
% Find the largest values of the correlation matrix to hence find the parameters with the
greatest dependency.
% Largest_R has 1st column containing corresponding row and 2nd column contatins
corresponding column. 3rd column containing largest values.

function largest_R = find_largest_R(R,num_max)

% Remove diagonal as will always be of value one.
for i = 1:size(R,2)

    R(i,i) = R(i,i) - 1;

end

% Pre-define matrices.
largest_R = zeros(num_max,3);

for i = 1:num_max

    % Finds max values.
```

```
    [~,I] = max(abs(R(:)));

    % Converts to row and column indecies.
    [I_row,I_col] = ind2sub(size(R),I);

    largest_R(i,1) = I_row;
    largest_R(i,2) = I_col;
    largest_R(i,3) = R(I_row,I_col);

    % Removes term that was used.
    R(I_row,I_col) = 0;

end

% Remove the repeated terms.
largest_R = largest_R .* toeplitz(mod(1:num_max,2),mod(1:1,2));

largest_R(~any(largest_R,2),:) = [];

end
```

Find Intervals

```
% Contains a list of all the intervals, function works in the same way as find_datetime but
instead stores the values in a vector.

function [interval_list] = find_intervals(Set_Date_1,Set_Date_2,DateTime_Con,num_intervals)

for i = 1:num_intervals

    % Criteria to reject incorrect data.
    if datenum(Set_Date_2) - datenum(Set_Date_1) < 0

        error('End date must be later than start date')

    elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

            error('Start time must be later than 07:30:00')

    elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

            error('End time must be earlier than 16:30:00')

    elseif hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 < 7.4 &&
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 > 16.5

            error('Start time must be later than 07:30:00 and end time must be earlier than
16:30:00')

    % If on the same day.
    elseif day(datenum(Set_Date_2)) == day(datenum(Set_Date_1))

        % If inside the times of the relevant data '03-Oct-2016 07:30:00'
        % and '03-Oct-2016 16:30:00'.
        if  hour(datenum(Set_Date_1)) + minute(datenum(Set_Date_1))/60 > 7.4 &&
```

```
hour(datenum(Set_Date_2)) + minute(datenum(Set_Date_2))/60 < 16.5

            % Work out interval.
            time_interval = (datenum(Set_Date_2) - datenum(Set_Date_1))/num_intervals;

            % Finds start and end of each time interval.
            ti_start = datenum(Set_Date_1) + (i-1)*time_interval;
            ti_end = datenum(Set_Date_1) + (i)*time_interval;

            % Finds the closed DateTime to each start and end interval.
            [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
            [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));

            % Looks up the actual datetime of the intervals.
            t11 = DateTime_Con(t1,:);
            t22 = DateTime_Con(t2,:);

            % Places the times in the interval lists.
            interval_list(i) = t11;
            interval_list(i+1) = t22;

        end

    % If the end date is after the start date.
    elseif day(datenum(Set_Date_2)) > day(datenum(Set_Date_1))

        % Finds the number of days between the start and the end.
        day_diff = day(datenum(Set_Date_2)) - day(datenum(Set_Date_1));

        % Random variables to determine the number of weekends.
        x = Set_Date_1;
        num_weekend = 0;

        % Finds the number of weekends between the two intervals.
        while day(datenum(Set_Date_2)) > day(datenum(x))

            if weekday(x) == 1

                num_weekend = num_weekend + 1;

                x = addtodate(datenum(x),1,'day');
                x = datestr(x);

            elseif weekday(x) == 7

                num_weekend = num_weekend + 2;

               x = addtodate(datenum(x),2,'day');
               x = datestr(x);

            else

                x = addtodate(datenum(x),1,'day');
                x = datestr(x);

            end

        end
```

```
    % The total time between the two dates.
    elapsed_time = datenum(Set_Date_2) - datenum(Set_Date_1);

    % Removes the deadtime, (when the market is closed).
    relevant_time = elapsed_time - (day_diff - num_weekend)*(15/24) - num_weekend;

    % The time interval, (how long each interval should be.
    time_interval = (relevant_time)/num_intervals;

    % Create vector to contain all the time intervals.
    time_vec = zeros(i,1);
    time_vec(1) = datenum(Set_Date_1);

    % Inserts all the intervals into the vector.
    for j = 1:i

        time_vec(j+1) = datenum(Set_Date_1) + (j)*time_interval;

    end

    % Add a hours such that none of the components of time_vec fall in the deadtime.
    for j = 1:(i+1)

        if hour(time_vec(j)) + minute(time_vec(j))/60 > 16.5

            for k = j:(i+1)

                time_vec(k) = time_vec(k) + 15/24;

            end

        end

    end

   % Removes the weekend components of the deadtime.
    for j = 1:(i+1)

        if weekday(datestr(time_vec(j))) == 7

            for k = j:(i+1)

                time_vec(k) = time_vec(k) + 2;

            end

        end

    end

    % Set ti_start and ti_end as the last two components of the time_vec.
    ti_start = time_vec(length(time_vec) - 1);
    ti_end = time_vec(length(time_vec));

    % Finds the closed DateTime to each start and end interval.
    [~,t1] = min(abs(ti_start - datenum(DateTime_Con)));
    [~,t2] = min(abs(ti_end - datenum(DateTime_Con)));
```

```
        % Looks up the actual datetime of the intervals.
        t11 = DateTime_Con(t1,:);
        t22 = DateTime_Con(t2,:);

        % Places the times in the interval lists.
        interval_list(i) = t11;
        interval_list(i+1) = t22;

    end

end

end
```

Feature Name List

```
% Lists the names of all the features, (keeps main script more tidy).

function [feature_name_list] = feature_name_list()

feature_name_list = ...
["Level 1 Bid Price", "Level 1 Bid Size", "Level 1 Ask Price", "Level 1 Ask Size", ...
"Level 2 Bid Price", "Level 2 Bid Size", "Level 2 Ask Price", "Level 2 Ask Size", ...
"Level 3 Bid Price", "Level 3 Bid Size", "Level 3 Ask Price", "Level 3 Ask Size", ...
"Level 4 Bid Price", "Level 4 Bid Size", "Level 4 Ask Price", "Level 4 Ask Size", ...
"Level 5 Bid Price", "Level 5 Bid Size", "Level 5 Ask Price", "Level 5 Ask Size", ...
"Level 6 Bid Price", "Level 6 Bid Size", "Level 6 Ask Price", "Level 6 Ask Size", ...
"Level 7 Bid Price", "Level 7 Bid Size", "Level 7 Ask Price", "Level 7 Ask Size", ...
"Level 8 Bid Price", "Level 8 Bid Size", "Level 8 Ask Price", "Level 8 Ask Size", ...
"Level 9 Bid Price", "Level 9 Bid Size", "Level 9 Ask Price", "Level 9 Ask Size", ...
"Level 10 Bid Price", "Level 10 Bid Size", "Level 10 Ask Price", "Level 10 Ask Size"];

end
```

Relate Feature List

```
% Function to reorganise the features and relate them so they can be displayed in the
matricies.
% Each row contains the corresponding feature related to the selected feature.
% The row index is the number it will appear in the matricies.

function [relate_feature_list] = relate_feature_list(selected_features,feature_name_list)

for i = 1:length(selected_features)

    relate_feature_list(i,1) = feature_name_list(selected_features(i));

end

end
```

Plotting Functions

Plot Levels Scatter

```
% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are super-imposed on the same plot.
% Dotted line represents the interval spacing.

function
plot_levels_scatter(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,circle_si
ze,selected_features,feature_name_list,interval_list)

% Finds the date and time but does not consider the intervals and the time when the market is
closed.
[Values,DateTime] =
find_datetime_no_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con);

f1 = figure('Name','Scatter of Selected Levels','NumberTitle','off');
figure(f1)

for j = selected_features

    % Only takes the odd terms.
    if mod(j,2) ~= 0

        scatter(DateTime,Values(:,j),Values(:,j+1).^circle_size, 'filled')
        hold on

    end

end

hold on

% Plots the intervals on the graph.
for i = 1:size(interval_list,2)

    x = interval_list(i);
    plot([x x],ylim,':k')
    hold on

end

title('Price of selected levels against time where the line width represents size of the
level')
xlabel('Date and time')
ylabel('Price of each level')

% Creates a vector to contain the inputs to the legend.
n = 1;
for i = 1:size(selected_features,2)

    if mod(selected_features(i),2) ~= 0
```

```
        selected_features2(n) = selected_features(i);
        n = n + 1;

    end

end

str = feature_name_list(selected_features2);
legend(str)

end
```

Plot Levels Scatter Sub

```
% A scatter plot showing the price of the selected levels against time, with the size of each
line representing the size of each level.
% All features are ploted on seperate subplots. Dotted line represents the interval spacing.

% Subplots are created automatically.

function [] =
plot_levels_scatter_sub(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,circl
e_size,selected_features,feature_name_list,interval_list)

% Imports data to plot.
[Values,DateTime] =
find_datetime_no_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con);

% Calculate number of features to use for subplots.
k = 0;

for j = selected_features

    if mod(j,2) ~= 0

        k = k + 1;

    end

end

% Conditions used to determine the positions of the subplots.
if k <= 2

    l = 1;
    m = 2;

elseif k > 2 && k <= 4

    l = 2;
    m = 2;

elseif k > 4 && k <= 6

    l = 2;
    m = 3;
```

```
elseif k > 6 && k <= 9

    l = 3;
    m = 3;

elseif k > 9 && k <= 12

    l = 3;
    m = 4;

elseif k > 12 && k <= 16

    l = 4;
    m = 4;

elseif k > 16 && k <= 20

    l = 4;
    m = 5;

end

% Vector to link the axes together.
ax = zeros(k,1);

f2 = figure('Name','Scatter of Selected Levels on Seperate Plots','NumberTitle','off');
figure(f2)

n = 1;

% Plots only the selected features.
for j = selected_features

    if mod(j,2) ~= 0

        % All subplots linked to the same axis.
        ax(n) = subplot(l,m,n);

        n = n + 1;

        scatter(DateTime,Values(:,j),Values(:,j+1).^circle_size, 'filled')

        hold on

        for i = 1:size(interval_list,2)

            x = interval_list(i);
            plot([x x],ylim,':k')
            hold on

        end

        title(feature_name_list(j))
        xlabel('Date and time')
        ylabel('Price of level')

    end
```

```
end

% Links the axes together so the values are easier to compare.
linkaxes(ax,'y');

end
```

Plot Compare Sampling

```
% A scatter plot showing the price of the selected levels against time, with the size of each line representing the size of each level.
% Each subplot compares the effect sampling and filtering to raw data.

function [] =
plot_compare_sampling(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con,circle_size,selected_features,feature_name_list,interval_list,fs,type,filter_order,freq_fraction)

[Values,DateTime] =
find_datetime_no_interval(Set_Date_1,Set_Date_2,XOMMarketDepthOct2016_Values,DateTime_Con);

% Resamples the data.
% Type = 0 ('event') takes the most recent event in the data.
% Type = 1 ('inter') linear interpolates the data.
[Values_resample,DateTime_resample] =
resample_data(Values,DateTime,fs,type,selected_features);

% Filters the data using a butterworth filter.
[Values_filter] = filter_data(filter_order,freq_fraction,Values_resample);

% Add size values to the resampled and filtered matricies.
selected_features2 = zeros(size(selected_features,1));
n = 1;

for j = selected_features

    if mod(j,2) ~= 0

        selected_features2(n) = j + 1;
        n = n + 1;

    end

end

% Creates feature list for legend.
n = 1;
for i = 1:size(selected_features,2)

    if mod(selected_features(i),2) ~= 0

        selected_features3(n) = selected_features(i);
        n = n + 1;

    end

end
```

```
% Resamples the extra required data.
[Values_resample2,~] = resample_data(Values,DateTime,fs,type,selected_features2);
[Values_filter2] = filter_data(filter_order,freq_fraction,Values_resample2);

% Superimposes the two data vectors together.
Values_resample = Values_resample + Values_resample2;
Values_filter = Values_filter + Values_filter2;

% Remove negative terms from filtered data.
Values_filter(Values_filter < 0) = 0.0001;

ax = zeros(3,1);

f3 = figure('Name','Scatter Comparing the Sampling of the Data','NumberTitle','off');
figure(f3)

% Plots the raw data.
ax(1) = subplot(3,1,1);

for j = selected_features

    if mod(j,2) ~= 0

        scatter(DateTime,Values(:,j),Values(:,j+1).^circle_size, 'filled')
        hold on

    end

end

hold on

% Overlays the interval list on the plot.
for i = 1:size(interval_list,2)

    x = interval_list(i);
    plot([x x],ylim,':k')
    hold on

end

title('Prices of raw data')
xlabel('Date and time')
ylabel('Price of each level')

str = feature_name_list(selected_features3);
legend(str)

hold off

% Plots resampled data.
ax(2) = subplot(3,1,2);

for j = selected_features

    if mod(j,2) ~= 0
```

```
scatter(datetime(DateTime_resample,'ConvertFrom','datenum'),Values_resample(:,j),Values_resam
ple(:,j+1).^circle_size, 'filled')
        hold on

    end

end

hold on

% Converts interval list from datetime to datenum then back so that the time zones on the
graph match.
interval_list = datenum(interval_list);
interval_list = datetime(interval_list,'ConvertFrom','datenum');

for i = 1:size(interval_list,2)

    y = interval_list(i);
    plot([y y],ylim,':k')
    hold on

end

title('Prices of resampled data')
xlabel('Date and time')
ylabel('Price of each level')

str = feature_name_list(selected_features3);
legend(str)

hold off

ax(3) = subplot(3,1,3);

for j = selected_features

    if mod(j,2) ~= 0


scatter(datetime(DateTime_resample,'ConvertFrom','datenum'),Values_filter(:,j),Values_filter(
:,j+1).^circle_size, 'filled')
        hold on

    end

end

hold on

for i = 1:size(interval_list,2)

    z = interval_list(i);
    plot([z z],ylim,':k')
    hold on

end
```

```
title('Prices of resampled and filtered data')
xlabel('Date and time')
ylabel('Price of each level')

str = feature_name_list(selected_features3);
legend(str)

% Links the axes together so the values are easier to compare.
linkaxes(ax,'y');

end
```

Plot Merits Time

```
% A plots of the log determinant of each of the dependency matricies (merits) as a function
of time.
% Used to compare the dependency matricies.
% Merit is effectively a measure of off-diagonal mass.

function [merit_R_St,merit_R_Sresamplet,merit_R_Sfiltert,merit_R_Bt,...
    merit_R_Ft,merit_R_Ct,merit_R_Cresamplet,merit_R_Cfiltert]...
    = plot_merits_time(num_intervals,R_St,R_Sresamplet,R_Sfiltert,R_Bt,...
    R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,interval_list,merits_time)

% Pre-allocate the merit vectors.
merit_R_St = zeros(num_intervals,1);
merit_R_Sresamplet = zeros(num_intervals,1);
merit_R_Sfiltert = zeros(num_intervals,1);
merit_R_Bt = zeros(num_intervals,1);
merit_R_Ft = zeros(num_intervals,1);
merit_R_Ct = zeros(num_intervals,1);
merit_R_Cresamplet = zeros(num_intervals,1);
merit_R_Cfiltert = zeros(num_intervals,1);

% Finds the merit for each time interval seperately.
for i = 1:num_intervals

    % Standard data.
    % Gets rid of stupid values
    if abs(-log(det(R_St(:,:,i)))) < 10^3 && abs(-log(det(R_St(:,:,i)))) > 0

        merit_R_St(i,1) = -log(det(R_St(:,:,i)));

    else

        merit_R_St(i,1) = 0;

    end

    % Resampled data.
    if abs(-log(det(R_Sresamplet(:,:,i)))) < 10^3 && abs(-log(det(R_Sresamplet(:,:,i)))) > 0

        merit_R_Sresamplet(i,1) = -log(det(R_Sresamplet(:,:,i)));

    else

        merit_R_Sresamplet(i,1) = 0;
```

```
    end

    % Filtered data.
    if abs(-log(det(R_Sfiltert(:,:,i)))) < 10^3 && abs(-log(det(R_Sfiltert(:,:,i)))) > 0

        merit_R_Sfiltert(i,1) = -log(det(R_Sfiltert(:,:,i)));

    else

        merit_R_Sfiltert(i,1) = 0;

    end

    % Brownian data.
    if abs(-log(det(R_Bt(:,:,i)))) < 10^3 && abs(-log(det(R_Bt(:,:,i)))) > 0

        merit_R_Bt(i,1) = -log(det(R_Bt(:,:,i)));

    else

        merit_R_Bt(i,1) = 0;

    end

    % Fourier data.
     if abs(-log(det(R_Ft(:,:,i)))) < 10^3 && abs(-log(det(R_Ft(:,:,i)))) > 0

        merit_R_Ft(i,1) = -log(det(R_Ft(:,:,i)));

    else

        merit_R_Ft(i,1) = 0;

     end

     % Copula data.
     if abs(-log(det(R_Ct(:,:,i)))) < 10^3 && abs(-log(det(R_Ct(:,:,i)))) > 0

        merit_R_Ct(i,1) = -log(det(R_Ct(:,:,i)));

     else

        merit_R_Ct(i,1) = 0;

     end

     % Copula resampled data.
     if abs(-log(det(R_Cresamplet(:,:,i)))) < 10^3 && abs(-log(det(R_Cresamplet(:,:,i)))) > 0

        merit_R_Cresamplet(i,1) = -log(det(R_Cresamplet(:,:,i)));

     else

        merit_R_Cresamplet(i,1) = 0;

     end
```

```
     % Copula resampled and filtered data.
     if abs(-log(det(R_Cfiltert(:,:,i)))) < 10^3 && abs(-log(det(R_Cfiltert(:,:,i)))) > 0

        merit_R_Cfiltert(i,1) = -log(det(R_Cfiltert(:,:,i)));

     else

        merit_R_Cfiltert(i,1) = 0;

     end

end

% Duplicates the last term of the merit vector.
merit_R_St = [merit_R_St;merit_R_St(end)];
merit_R_resamplet = [merit_R_Sresamplet;merit_R_Sresamplet(end)];
merit_R_filtert = [merit_R_Sfiltert;merit_R_Sfiltert(end)];
merit_R_Bt = [merit_R_Bt;merit_R_Bt(end)];
merit_R_Ft = [merit_R_Ft;merit_R_Ft(end)];
merit_R_Ct = [merit_R_Ct;merit_R_Ct(end)];
merit_R_Cresamplet = [merit_R_Cresamplet;merit_R_Cresamplet(end)];
merit_R_Cfiltert = [merit_R_Cfiltert;merit_R_Cfiltert(end)];

% Fixes issues with merits equaling zero that should be infinite by setting them to 1000.
merit_R_St(merit_R_St == 0) = 1000;
merit_R_Sresamplet(merit_R_Sresamplet == 0) = 1000;
merit_R_Sfiltert(merit_R_Sfiltert == 0) = 1000;
merit_R_Bt(merit_R_Bt == 0) = 1000;
merit_R_Ft(merit_R_Ft == 0) = 1000;
merit_R_Ct(merit_R_Ct == 0) = 1000;
merit_R_Cresamplet(merit_R_Cresamplet == 0) = 1000;
merit_R_Cfiltert(merit_R_Cfiltert == 0) = 1000;

if merits_time == 1

x =
[merit_R_St,merit_R_resamplet,merit_R_filtert,merit_R_Bt,merit_R_Ft,merit_R_Ct,merit_R_Cresam
plet,merit_R_Cfiltert];

% Plot the merits over time.
f4 = figure('Name','Differential Entropy as a Function of Time','NumberTitle','off');
figure(f4)

stairs(interval_list,x)

% Uncomment the below, to change between a stairs plot and a normal plot
%plot(interval_list,real(merit_R_St),real(merit_R_resamplet),real(merit_R_filtert),real(merit
_R_Bt),real(merit_R_Ft),real(merit_R_Ct),real(merit_R_Cresamplet),real(merit_R_Cfiltert))

title('Differential entropy as a function of time')
xlabel('Date and Time')
ylabel('Differential entropy')
legend('Standard','Standard with resampling','Standard with resampling and
filtering','Brownian','Fourier','Copula','Copula with resampling','Copula with resampling and
filtering')

end
```

```
end
```

Plot Merits Scale

```
% A plot of the log determinant of each of the dependency matricies (merits) where the x-axis
represents time as an expanding window.
% Effectively a cumulative merit.

function [merit_R_St,merit_R_Sresamplet,merit_R_Sfiltert,merit_R_Bt,...
    merit_R_Ft,merit_R_Ct,merit_R_Cresamplet,merit_R_Cfiltert]...
    = plot_merits_scale(num_intervals,R_St,R_Sresamplet,R_Sfiltert,R_Bt,...
    R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,interval_list,merits_time)

% Pre-allocate the merit vectors.
merit_R_St = zeros(num_intervals,1);
merit_R_Sresamplet = zeros(num_intervals,1);
merit_R_Sfiltert = zeros(num_intervals,1);
merit_R_Bt = zeros(num_intervals,1);
merit_R_Ft = zeros(num_intervals,1);
merit_R_Ct = zeros(num_intervals,1);
merit_R_Cresamplet = zeros(num_intervals,1);
merit_R_Cfiltert = zeros(num_intervals,1);

% Finds the merit for each time interval seperately.
for i = 1:num_intervals

    % Standard data.
    % Gets rid of stupid values.
    if abs(-log(det(R_St(:,:,i)))) < 10^3 && abs(-log(det(R_St(:,:,i)))) > 0

        merit_R_St(i,1) = -log(det(R_St(:,:,i)));

    else

        merit_R_St(i,1) = 0;

    end

    % Resampled data.
    if abs(-log(det(R_Sresamplet(:,:,i)))) < 10^3 && abs(-log(det(R_Sresamplet(:,:,i)))) > 0

        merit_R_Sresamplet(i,1) = -log(det(R_Sresamplet(:,:,i)));

    else

        merit_R_Sresamplet(i,1) = 0;

    end

    % Filtered data.
    if abs(-log(det(R_Sfiltert(:,:,i)))) < 10^3 && abs(-log(det(R_Sfiltert(:,:,i)))) > 0

        merit_R_Sfiltert(i,1) = -log(det(R_Sfiltert(:,:,i)));

    else
```

```
        merit_R_Sfiltert(i,1) = 0;

    end

    % Brownian data.
    if abs(-log(det(R_Bt(:,:,i)))) < 10^3 && abs(-log(det(R_Bt(:,:,i)))) > 0

        merit_R_Bt(i,1) = -log(det(R_Bt(:,:,i)));

    else

        merit_R_Bt(i,1) = 0;

    end

    % Fourier data.
     if abs(-log(det(R_Ft(:,:,i)))) < 10^3 && abs(-log(det(R_Ft(:,:,i)))) > 0

        merit_R_Ft(i,1) = -log(det(R_Ft(:,:,i)));

    else

        merit_R_Ft(i,1) = 0;

     end

     % Copula data.
     if abs(-log(det(R_Ct(:,:,i)))) < 10^3 && abs(-log(det(R_Ct(:,:,i)))) > 0

        merit_R_Ct(i,1) = -log(det(R_Ct(:,:,i)));

     else

        merit_R_Ct(i,1) = 0;

     end

     % Copula resampled data.
     if abs(-log(det(R_Cresamplet(:,:,i)))) < 10^3 && abs(-log(det(R_Cresamplet(:,:,i)))) > 0

        merit_R_Cresamplet(i,1) = -log(det(R_Cresamplet(:,:,i)));

     else

        merit_R_Cresamplet(i,1) = 0;

     end

     % Copula resampled and filtered data.
     if abs(-log(det(R_Cfiltert(:,:,i)))) < 10^3 && abs(-log(det(R_Cfiltert(:,:,i)))) > 0

        merit_R_Cfiltert(i,1) = -log(det(R_Cfiltert(:,:,i)));

     else

        merit_R_Cfiltert(i,1) = 0;

     end
```

```
end

% Duplicates the last term of the merit vector.
merit_R_St = [merit_R_St;merit_R_St(end)];
merit_R_resamplet = [merit_R_Sresamplet;merit_R_Sresamplet(end)];
merit_R_filtert = [merit_R_Sfiltert;merit_R_Sfiltert(end)];
merit_R_Bt = [merit_R_Bt;merit_R_Bt(end)];
merit_R_Ft = [merit_R_Ft;merit_R_Ft(end)];
merit_R_Ct = [merit_R_Ct;merit_R_Ct(end)];
merit_R_Cresamplet = [merit_R_Cresamplet;merit_R_Cresamplet(end)];
merit_R_Cfiltert = [merit_R_Cfiltert;merit_R_Cfiltert(end)];

% Fixes issues with merits equaling zero that should be infinite but setting them to 1000.
merit_R_St(merit_R_St == 0) = 1000;
merit_R_Sresamplet(merit_R_Sresamplet == 0) = 1000;
merit_R_Sfiltert(merit_R_Sfiltert == 0) = 1000;
merit_R_Bt(merit_R_Bt == 0) = 1000;
merit_R_Ft(merit_R_Ft == 0) = 1000;
merit_R_Ct(merit_R_Ct == 0) = 1000;
merit_R_Cresamplet(merit_R_Cresamplet == 0) = 1000;
merit_R_Cfiltert(merit_R_Cfiltert == 0) = 1000;

if merits_time == 1

x =
[merit_R_St,merit_R_resamplet,merit_R_filtert,merit_R_Bt,merit_R_Ft,merit_R_Ct,merit_R_Cresam
plet,merit_R_Cfiltert];

% Plot the merits over time.
f5 = figure('Name','Differential Entropy as a Function of Scale','NumberTitle','off');
figure(f5)

stairs(interval_list,x)

% Plot the merits over time.
%figure
% Uncomment the below, to change between a stairs plot and a normal plot
%plot(interval_list,real(merit_R_St),real(merit_R_resamplet),real(merit_R_filtert),real(merit
_R_Bt),real(merit_R_Ft),real(merit_R_Ct),real(merit_R_Cresamplet),real(merit_R_Cfiltert))

title('Differential entropy as a function of scale')
xlabel('Date and Time')
ylabel('Differential entropy')
legend('Standard','Standard with resampling','Standard with resampling and
filtering','Brownian','Fourier','Copula','Copula with resampling','Copula with resampling and
filtering')

end

end
```

Plot Merits Resampling

```
% A plots of the log determinant of each of the dependency matricies (merits) as a function
of sampling frequency (only for resample and resample and filter corrleations).
```

```
function [merit_R_SresampleSt,merit_R_SfilterSt,merit_R_CresampleSt,merit_R_CfilterSt] ...
    = plot_merits_resampling(fs_range,num_freq,R_SresampleSt,R_SfilterSt,R_CresampleSt,
R_CfilterSt,merits_resampling)

% Vector of all the frequencies used.
fs = linspace(fs_range(1),fs_range(2),num_freq);

% Pre-allocate the merit vectors.
merit_R_SresampleSt = zeros(num_freq,1);
merit_R_SfilterSt = zeros(num_freq,1);
merit_R_CresampleSt = zeros(num_freq,1);
merit_R_CfilterSt = zeros(num_freq,1);

% Finds the merits for different resampling frequencies seperately.
for i = 1:num_freq

    % Resampled data.
    if abs(-log(det(R_SresampleSt(:,:,i)))) < 10^3 && abs(-log(det(R_SresampleSt(:,:,i)))) >
0

        merit_R_SresampleSt(i,1) = -log(det(R_SresampleSt(:,:,i)));

    else

        merit_R_SresampleSt(i,1) = 0;

    end

    % Filtered data.
    if abs(-log(det(R_SfilterSt(:,:,i)))) < 10^3 && abs(-log(det(R_SfilterSt(:,:,i)))) > 0

        merit_R_SfilterSt(i,1) = -log(det(R_SfilterSt(:,:,i)));

    else

        merit_R_SfilterSt(i,1) = 0;

    end

    % Copula resample data.
    if abs(-log(det(R_CresampleSt(:,:,i)))) < 10^3 && abs(-log(det(R_CresampleSt(:,:,i)))) >
0

        merit_R_CresampleSt(i,1) = -log(det(R_CresampleSt(:,:,i)));

    else

        merit_R_CresampleSt(i,1) = 0;

    end

    % Copula filtered data.
    if abs(-log(det(R_CfilterSt(:,:,i)))) < 10^3 && abs(-log(det(R_CfilterSt(:,:,i)))) > 0

        merit_R_CfilterSt(i,1) = -log(det(R_CfilterSt(:,:,i)));

    else
```

```
        merit_R_CfilterSt(i,1) = 0;

    end


end

% Plot the merits over time.
if merits_resampling == 1

f6 = figure('Name','Differential Entropy as a Function of Sampling
Freqeuncy','NumberTitle','off');
figure(f6)

% Adds an extra term on the end of the vectors to help with stairs plot.
merit_R_SresampleSt = [merit_R_SresampleSt; merit_R_SresampleSt(end)];
merit_R_SfilterSt = [merit_R_SfilterSt; merit_R_SfilterSt(end)];
merit_R_CresampleSt = [merit_R_CresampleSt; merit_R_CresampleSt(end)];
merit_R_CfilterSt = [merit_R_CfilterSt; merit_R_CfilterSt(end)];

% Fixes issues with merits equaling zero that should be infinite but setting them to 1000.
merit_R_SresampleSt(merit_R_SresampleSt == 0) = 1000;
merit_R_SfilterSt(merit_R_SfilterSt == 0) = 1000;
merit_R_CresampleSt(merit_R_CresampleSt == 0) = 1000;
merit_R_CfilterSt(merit_R_CfilterSt == 0) = 1000;

% Adds an extra term and shifts the values so that the plot makes more sense.
fs_step = fs(2) - fs(1);
fs = fs.';
fs = [fs; (fs(end) + fs_step)];
fs = fs - (fs_step/2);
fs(fs < 0) = 0;

x = [merit_R_SresampleSt,merit_R_SfilterSt,merit_R_CresampleSt,merit_R_CfilterSt];

stairs(fs,x);

%plot(fs,real(merit_R_SresampleSt),fs,real(merit_R_SfilterSt),fs,real(merit_R_CresampleSt),fs
,real(merit_R_CfilterSt))

title('Differential entropy as a function of sampling frequency')
xlabel('Sampling frequency')
ylabel('Differential entropy')
legend('Standard with resampling','Standard with resampling and filtering','Copula with
resampling','Copula with resampling and filtering')

end

end
```

Summarise Results Time

```
% A summary of all the results for varying time, with each dependency method in a seperate
figure.

function [str2] = summarise_results_time(largest_R_Bt,largest_R_Ct,...
    largest_R_Sfiltert,largest_R_Ft,largest_R_Sresamplet,largest_R_St,...
```

```
    largest_R_Cresamplet,largest_R_Cfiltert,...
    merit_R_Bt,merit_R_Ct,merit_R_Sfiltert,merit_R_Ft,merit_R_Sresamplet,...
    merit_R_St,merit_R_Cresamplet,merit_R_Cfiltert,...
    R_Bt,R_Ct,R_Sfiltert,R_Ft,R_Sresamplet,R_St,R_Cresamplet,R_Cfiltert,...
    summarise_data,summarise_standard,summarise_standard_resample,...
    summarise_standard_filter,summarise_brownian,summarise_fourier,...
    summarise_copula,summarise_copula_resample,summarise_copula_filter,...
    relate_feature_list,interval_list)

% Dummy variable to make code work.
str2 = 'done';

% Conditional statements to plot the required figures.
if summarise_data == 1

   if summarise_standard == 1

       % Creates the releveant figure.
       f_standard = figure('Name','Standard Correlation Results
Summary','NumberTitle','off');
       figure(f_standard);

       % Goes through each matrix (time interval) and plots the data on the
       % figure at different locations.
       % The sizes and locations of each component will be automatically
       % assigned to fit the figure
       for i = 1:size(R_St,3)

           % Displays the dependency matrix in a table, takes up 60% of
           % the figure.
           table_standard(i) = uitable(f_standard,'Data',R_St(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_St,3)) 0.6
1/size(R_St,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           % Displays the time intervals for each matrix, takes up 10% of
           % the figure.
           table_standard2(i) = uitable(f_standard,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_St,3)) 0.1
1/size(R_St,3)],...
               'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
               'ColumnWidth','auto');

           % Displays a list of the most significant values along with
           % their relevant features, takes up 20% of the figure.
           table_standard3(i) = uitable(f_standard,'Data',largest_R_St(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_St,3)) 0.2
1/size(R_St,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');

           % Displays the value of differential entropy, takes up 10% of
           % the figure.
           table_standard4(i) = uitable(f_standard,'Data',real(merit_R_St(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_St,3)) 0.1
1/size(R_St,3)],...
```

```
               'ColumnName',{'Entropy'},...
               'ColumnWidth','auto');

       end

   end

   if summarise_standard_resample == 1

       % Comments are not repeated as they are the same.
       f_standard_resample = figure('Name','Standard Correlation Resampled Results
Summary','NumberTitle','off');
       figure(f_standard_resample);

       for i = 1:size(R_Sresamplet,3)

           table_standard_resample(i) =
uitable(f_standard_resample,'Data',R_Sresamplet(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_Sresamplet,3)) 0.6
1/size(R_Sresamplet,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           table_standard_resample2(i) = uitable(f_standard_resample,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
               'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
               'ColumnWidth','auto');

           table_standard_resample3(i) =
uitable(f_standard_resample,'Data',largest_R_Sresamplet(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sresamplet,3)) 0.2
1/size(R_Sresamplet,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');

           table_standard_resample4(i) =
uitable(f_standard_resample,'Data',real(merit_R_Sresamplet(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
               'ColumnName',{'Entropy'},...
               'ColumnWidth','auto');

       end

   end

   if summarise_standard_filter == 1

       f_standard_filter = figure('Name','Standard Correlation Resampled and Filtered Results
Summary','NumberTitle','off');
       figure(f_standard_filter);

       for i = 1:size(R_Sfiltert,3)

           table_standard_filter(i) = uitable(f_standard_filter,'Data',R_Sfiltert(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_Sfiltert,3)) 0.6
```

```
1/size(R_Sfiltert,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           table_standard_filter2(i) = uitable(f_standard_filter,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sfiltert,3)) 0.1
1/size(R_Sfiltert,3)],...
               'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
               'ColumnWidth','auto');

           table_standard_filter3(i) =
uitable(f_standard_filter,'Data',largest_R_Sfiltert(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sfiltert,3)) 0.2
1/size(R_Sfiltert,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');

           table_standard_filter4(i) =
uitable(f_standard_filter,'Data',real(merit_R_Sfiltert(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sfiltert,3)) 0.1
1/size(R_Sfiltert,3)],...
               'ColumnName',{'Entropy'},...
               'ColumnWidth','auto');

       end

   end

   if summarise_brownian == 1

       f_brownian = figure('Name','Brownian Estimator Results Summary','NumberTitle','off');
       figure(f_brownian);

       for i = 1:size(R_Bt,3)

           table_brownian(i) = uitable(f_brownian,'Data',R_Bt(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_Bt,3)) 0.6
1/size(R_Bt,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           table_brownian2(i) = uitable(f_brownian,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Bt,3)) 0.1
1/size(R_Bt,3)],...
               'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
               'ColumnWidth','auto');

           table_brownian3(i) = uitable(f_brownian,'Data',largest_R_Bt(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Bt,3)) 0.2
1/size(R_Bt,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');


           table_brownian4(i) = uitable(f_brownian,'Data',real(merit_R_Bt(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Bt,3)) 0.1
```

```
1/size(R_Bt,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_fourier == 1

        f_fourier = figure('Name','Fourier Estimator Results Summary','NumberTitle','off');
        figure(f_fourier);

        for i = 1:size(R_Ft,3)

            table_fourier(i) = uitable(f_fourier,'Data',R_Ft(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Ft,3)) 0.6
1/size(R_Ft,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_fourier2(i) = uitable(f_fourier,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Ft,3)) 0.1
1/size(R_Ft,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_fourier3(i) = uitable(f_fourier,'Data',largest_R_Ft(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Bt,3)) 0.2
1/size(R_Bt,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_fourier4(i) = uitable(f_fourier,'Data',real(merit_R_Ft(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Ft,3)) 0.1
1/size(R_Ft,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula == 1

        f_copula = figure('Name','Copula Estimator Results Summary','NumberTitle','off');
        figure(f_copula);

        for i = 1:size(R_Ct,3)

            table_copula(i) = uitable(f_copula,'Data',R_Ct(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Ct,3)) 0.6
1/size(R_Ct,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula2(i) = uitable(f_copula,'Data',[],...
```

```
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Ct,3)) 0.1
1/size(R_Ct,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copula3(i) = uitable(f_copula,'Data',largest_R_Ct(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Ct,3)) 0.2
1/size(R_Ct,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula4(i) = uitable(f_copula,'Data',real(merit_R_Ct(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Ct,3)) 0.1
1/size(R_Ct,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula_resample == 1

        f_copula_resample = figure('Name','Copula Estimator Resampled Results
Summary','NumberTitle','off');
        figure(f_copula_resample);

        for i = 1:size(R_Cresamplet,3)

            table_copula_resample(i) =
uitable(f_copula_resample,'Data',R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cresamplet,3)) 0.6
1/size(R_Cresamplet,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_resample2(i) = uitable(f_copula_resample,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copula_resample3(i) =
uitable(f_copula_resample,'Data',largest_R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cresamplet,3)) 0.2
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_resample4(i) =
uitable(f_copula_resample,'Data',real(merit_R_Cresamplet(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');
```

```
        end

    end

    if summarise_copula_filter == 1

        f_copula_filter = figure('Name','Copula Estimator Resampled and Filtered Results
Summary','NumberTitle','off');
        figure(f_copula_filter);

        for i = 1:size(R_Cfiltert,3)

            table_copula_filter(i) = uitable(f_copula_filter,'Data',R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cfiltert,3)) 0.6
1/size(R_Cfiltert,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_filter2(i) = uitable(f_copula_filter,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copula_filter3(i) =
uitable(f_copula_filter,'Data',largest_R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cfiltert,3)) 0.2
1/size(R_Cfiltert,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_filter4(i) =
uitable(f_copula_filter,'Data',real(merit_R_Cfiltert(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

end

end
```

Summarise Results Scale

```
% A summary of all the results for varying time, with each dependency method in a seperate
figure.

function [str2] = summarise_results_scale(largest_R_Bt,largest_R_Ct,...
    largest_R_Sfiltert,largest_R_Ft,largest_R_Sresamplet,largest_R_St,...
    largest_R_Cresamplet,largest_R_Cfiltert,...
    merit_R_Bt,merit_R_Ct,merit_R_Sfiltert,merit_R_Ft,merit_R_Sresamplet,...
    merit_R_St,merit_R_Cresamplet,merit_R_Cfiltert,...
```

```
    R_Bt,R_Ct,R_Sfiltert,R_Ft,R_Sresamplet,R_St,R_Cresamplet,R_Cfiltert,...
    summarise_data,summarise_standard,summarise_standard_resample,...
    summarise_standard_filter,summarise_brownian,summarise_fourier,...
    summarise_copula,summarise_copula_resample,summarise_copula_filter,...
    relate_feature_list,interval_list)

% Dummy variable to make code work.
str2 = 'done';

% Conditional statements to plot the required figures.
if summarise_data == 1

   if summarise_standard == 1

       % Creates the releveant figure.
       f_standardW = figure('Name','Standard Correlation Varying with Scale Results
Summary','NumberTitle','off');
       figure(f_standardW);

       % Goes through each matrix (time window) and plots the data on the
       % figure at different locations.
       % The sizes and locations of each component will be automatically
       % assigned to fit the figure.
       for i = 1:size(R_St,3)

           % Displays the dependency matrix in a table, takes up 60% of
           % the figure.
           table_standardW(i) = uitable(f_standardW,'Data',R_St(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_St,3)) 0.6
1/size(R_St,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           % Displays the time windows for each matrix, takes up 10% of
           % the figure.
           table_standardW2(i) = uitable(f_standardW,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_St,3)) 0.1
1/size(R_St,3)],...
               'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
               'ColumnWidth','auto');

           % Displays a list of the most significant values along with
           % their relevant features, takes up 20% of the figure.
           table_standardW3(i) = uitable(f_standardW,'Data',largest_R_St(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_St,3)) 0.2
1/size(R_St,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');

           % Displays the value of differential entropy, takes up 10% of
           % the figure.
           table_standardW4(i) = uitable(f_standardW,'Data',real(merit_R_St(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_St,3)) 0.1
1/size(R_St,3)],...
               'ColumnName',{'Entropy'},...
               'ColumnWidth','auto');
```

```
        end

    end

    if summarise_standard_resample == 1

        % Comments are not repeated as they are the same.
        f_standard_resampleW = figure('Name','Standard Correlation Resampled Varying with
Scale Results Summary','NumberTitle','off');
        figure(f_standard_resampleW);


        for i = 1:size(R_Sresamplet,3)

            table_standard_resampleW(i) =
uitable(f_standard_resampleW,'Data',R_Sresamplet(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Sresamplet,3)) 0.6
1/size(R_Sresamplet,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_standard_resampleW2(i) = uitable(f_standard_resampleW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_standard_resampleW3(i) =
uitable(f_standard_resampleW,'Data',largest_R_Sresamplet(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sresamplet,3)) 0.2
1/size(R_Sresamplet,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_standard_resampleW4(i) =
uitable(f_standard_resampleW,'Data',real(merit_R_Sresamplet(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_standard_filter == 1

        f_standard_filterW = figure('Name','Standard Correlation Resampled and Filtered
Varying with Scale Results Summary','NumberTitle','off');
        figure(f_standard_filterW);

        for i = 1:size(R_Sfiltert,3)

            table_standard_filterW(i) =
uitable(f_standard_filterW,'Data',R_Sfiltert(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Sfiltert,3)) 0.6
1/size(R_Sfiltert,3)],...
```

```
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_standard_filterW2(i) = uitable(f_standard_filterW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sfiltert,3)) 0.1 
1/size(R_Sfiltert,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End 
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_standard_filterW3(i) = 
uitable(f_standard_filterW,'Data',largest_R_Sfiltert(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sfiltert,3)) 0.2 
1/size(R_Sfiltert,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_standard_filterW4(i) = 
uitable(f_standard_filterW,'Data',real(merit_R_Sfiltert(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sfiltert,3)) 0.1 
1/size(R_Sfiltert,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_brownian == 1

        f_brownianW = figure('Name','Brownian Estimator Varying with Scale Results 
Summary','NumberTitle','off');
        figure(f_brownianW);

        for i = 1:size(R_Bt,3)

            table_brownianW(i) = uitable(f_brownianW,'Data',R_Bt(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Bt,3)) 0.6 
1/size(R_Bt,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_brownianW2(i) = uitable(f_brownianW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Bt,3)) 0.1 
1/size(R_Bt,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End 
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_brownianW3(i) = uitable(f_brownianW,'Data',largest_R_Bt(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Bt,3)) 0.2 
1/size(R_Bt,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_brownianW4(i) = uitable(f_brownianW,'Data',real(merit_R_Bt(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Bt,3)) 0.1 
1/size(R_Bt,3)],...
```

```
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_fourier == 1

        f_fourierW = figure('Name','Fourier Estimator Varying with Scale Results
Summary','NumberTitle','off');
        figure(f_fourierW);

        for i = 1:size(R_Ft,3)

            table_fourierW(i) = uitable(f_fourierW,'Data',R_Ft(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Ft,3)) 0.6
1/size(R_Ft,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_fourierW2(i) = uitable(f_fourierW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Ft,3)) 0.1
1/size(R_Ft,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_fourierW3(i) = uitable(f_fourierW,'Data',largest_R_Ft(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Bt,3)) 0.2
1/size(R_Bt,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_fourierW4(i) = uitable(f_fourierW,'Data',real(merit_R_Ft(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Ft,3)) 0.1
1/size(R_Ft,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula == 1

        f_copulaW = figure('Name','Copula Estimator Varying with Scale Results
Summary','NumberTitle','off');
        figure(f_copulaW);

        for i = 1:size(R_Ct,3)

            table_copulaW(i) = uitable(f_copulaW,'Data',R_Ct(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Ct,3)) 0.6
1/size(R_Ct,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');
```

```
            table_copulaW2(i) = uitable(f_copulaW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Ct,3)) 0.1
1/size(R_Ct,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copulaW3(i) = uitable(f_copulaW,'Data',largest_R_Ct(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Ct,3)) 0.2
1/size(R_Ct,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copulaW4(i) = uitable(f_copulaW,'Data',real(merit_R_Ct(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Ct,3)) 0.1
1/size(R_Ct,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula_resample == 1

        f_copula_resampleW = figure('Name','Copula Estimator Resampled Varying with Scale
Results Summary','NumberTitle','off');
        figure(f_copula_resampleW);

        for i = 1:size(R_Cresamplet,3)

            table_copula_resampleW(i) =
uitable(f_copula_resampleW,'Data',R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cresamplet,3)) 0.6
1/size(R_Cresamplet,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_resampleW2(i) = uitable(f_copula_resampleW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copula_resampleW3(i) =
uitable(f_copula_resampleW,'Data',largest_R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cresamplet,3)) 0.2
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_resampleW4(i) =
uitable(f_copula_resampleW,'Data',real(merit_R_Cresamplet(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');
```

```
        end

    end

    if summarise_copula_filter == 1

        f_copula_filterW = figure('Name','Copula Estimator Resampled and Filtered Varying with
Scale Results Summary','NumberTitle','off');
        figure(f_copula_filterW);

        for i = 1:size(R_Cfiltert,3)

            table_copula_filterW(i) = uitable(f_copula_filterW,'Data',R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cfiltert,3)) 0.6
1/size(R_Cfiltert,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_filterW2(i) = uitable(f_copula_filterW,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
                'RowName',{'Start Time';datestr(interval_list(i));'';'End
Time';datestr(interval_list(i+1))},...
                'ColumnWidth','auto');

            table_copula_filterW3(i) =
uitable(f_copula_filterW,'Data',largest_R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cfiltert,3)) 0.2
1/size(R_Cfiltert,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_filterW4(i) =
uitable(f_copula_filterW,'Data',real(merit_R_Cfiltert(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

end

end
```

Summarise Results Resampling

```
% A summary of all the results for varying sampling frequency, with each dependency method in
a seperate figure.

function [str2] = summarise_results_resampling(largest_R_Sfiltert,...
    largest_R_Sresamplet,largest_R_Cresamplet,largest_R_Cfiltert,...
    merit_R_Sfiltert,merit_R_Sresamplet,merit_R_Cresamplet,merit_R_Cfiltert,...
    R_Sfiltert,R_Sresamplet,R_Cresamplet,R_Cfiltert,...
```

```
    summarise_data,summarise_standard_resample,summarise_standard_filter,...
    summarise_copula_resample,summarise_copula_filter,...
    relate_feature_list,fs_range,num_freq)

% Vector containing the various frequencies.
fs = linspace(fs_range(1),fs_range(2),num_freq);

% Dummy variable to make code work.
str2 = 'done';

% Conditional statements to plot the required figures.
if summarise_data == 1

   if summarise_standard_resample == 1

       % Creates the releveant figure.
       f_standard_resampleS = figure('Name','Standard Correlation with Varying Resampling
Results Summary','NumberTitle','off');
       figure(f_standard_resampleS);

       % Goes through each matrix (frequency) and plots the data on the
       % figure at different locations.
       % The sizes and locations of each component will be automatically
       % assigned to fit the figure.
       for i = 1:size(R_Sresamplet,3)

           % Displays the dependency matrix in a table, takes up 60% of
           % the figure.
           table_standard_resampleS(i) = uitable(f_standard_resampleS,'Data',
R_Sresamplet(:,:,i),...
               'Units','Normalized','Position',[0 (1 - (i)/size(R_Sresamplet,3)) 0.6
1/size(R_Sresamplet,3)],...
               'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
               'ColumnWidth','auto');

           % Displays the time windows for each matrix, takes up 10% of
           % the figure.
           table_standard_resampleS2(i) = uitable(f_standard_resampleS,'Data',[],...
               'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
               'RowName',{'Frequency';fs(i)},...
               'ColumnWidth','auto');

           % Displays a list of the most significant values along with
           % their relevant features, takes up 20% of the figure.
           table_standard_resampleS3(i) =
uitable(f_standard_resampleS,'Data',largest_R_Sresamplet(:,:,i),...
               'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sresamplet,3)) 0.2
1/size(R_Sresamplet,3)],...
               'ColumnName',{'Feature 1';'Feature 2';'Value'},...
               'ColumnWidth','auto');
           % Displays the value of differential entropy, takes up 10% of
           % the figure.
           table_standard_resampleS4(i) =
uitable(f_standard_resampleS,'Data',real(merit_R_Sresamplet(i)),...
               'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sresamplet,3)) 0.1
1/size(R_Sresamplet,3)],...
               'ColumnName',{'Entropy'},...
```

```
                'ColumnWidth','auto');

        end

    end

    if summarise_standard_filter == 1

        % Comments are not repeated as they are the same.
        f_standard_filterS = figure('Name','Standard Correlation Resampled and Filtered 
Results Summary','NumberTitle','off');
        figure(f_standard_filterS);

        for i = 1:size(R_Sfiltert,3)

            table_standard_filterS(i) = 
uitable(f_standard_filterS,'Data',R_Sfiltert(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Sfiltert,3)) 0.6 
1/size(R_Sfiltert,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_standard_filterS2(i) = uitable(f_standard_filterS,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Sfiltert,3)) 0.1 
1/size(R_Sfiltert,3)],...
                'RowName',{'Frequency';fs(i)},...
                'ColumnWidth','auto');

            table_standard_filterS3(i) = 
uitable(f_standard_filterS,'Data',largest_R_Sfiltert(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Sfiltert,3)) 0.2 
1/size(R_Sfiltert,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_standard_filterS4(i) = 
uitable(f_standard_filterS,'Data',real(merit_R_Sfiltert(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Sfiltert,3)) 0.1 
1/size(R_Sfiltert,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula_resample == 1

        f_copula_resampleS = figure('Name','Copula Estimator Resampled with Varying Frequency 
Results Summary','NumberTitle','off');
        figure(f_copula_resampleS);

        for i = 1:size(R_Cresamplet,3)

            table_copula_resampleS(i) = 
uitable(f_copula_resampleS,'Data',R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cresamplet,3)) 0.6 
1/size(R_Cresamplet,3)],...
```

```
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_resampleS2(i) = uitable(f_copula_resampleS,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'RowName',{'Frequency';fs(i)},...
                'ColumnWidth','auto');

            table_copula_resampleS3(i) =
uitable(f_copula_resampleS,'Data',largest_R_Cresamplet(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cresamplet,3)) 0.2
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_resampleS4(i) =
uitable(f_copula_resampleS,'Data',real(merit_R_Cresamplet(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cresamplet,3)) 0.1
1/size(R_Cresamplet,3)],...
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

    if summarise_copula_filter == 1

        f_copula_filterS = figure('Name','Copula Estimator Resampled and Filtered with Varying
Frequency Results Summary','NumberTitle','off');
        figure(f_copula_filterS);

        for i = 1:size(R_Cfiltert,3)

            table_copula_filterS(i) = uitable(f_copula_filterS,'Data',R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0 (1 - (i)/size(R_Cfiltert,3)) 0.6
1/size(R_Cfiltert,3)],...
                'RowName',relate_feature_list,'ColumnName',relate_feature_list,...
                'ColumnWidth','auto');

            table_copula_filterS2(i) = uitable(f_copula_filterS,'Data',[],...
                'Units','Normalized','Position',[0.6 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
                'RowName',{'Frequency';fs(i)},...
                'ColumnWidth','auto');

            table_copula_filterS3(i) =
uitable(f_copula_filterS,'Data',largest_R_Cfiltert(:,:,i),...
                'Units','Normalized','Position',[0.7 (1 - (i)/size(R_Cfiltert,3)) 0.2
1/size(R_Cfiltert,3)],...
                'ColumnName',{'Feature 1';'Feature 2';'Value'},...
                'ColumnWidth','auto');

            table_copula_filterS4(i) =
uitable(f_copula_filterS,'Data',real(merit_R_Cfiltert(i)),...
                'Units','Normalized','Position',[0.9 (1 - (i)/size(R_Cfiltert,3)) 0.1
1/size(R_Cfiltert,3)],...
```

```
                'ColumnName',{'Entropy'},...
                'ColumnWidth','auto');

        end

    end

end

end
```

Visualise Matrices Time

```
% A figure with a visualisation of each of the dependency matricies with varying time.
% The time updates every 3 seconds, so to close the figure, call "ctrl-C" in the command
window.

function [str] = visualise_matrices_time(R_St,R_Sresamplet,R_Sfiltert,...
    R_Bt,R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,num_intervals,...
    selected_features,feature_name_list,relate_feature_list)

% Creates a list of all the selected features.
n = 1;
for i = 1:size(selected_features,2)

    if mod(selected_features(i),2) ~= 0

        selected_features2(n) = selected_features(i);
        n = n + 1;

    end

end

% Creates a string vector of all the selected feature names.
str = feature_name_list(selected_features2);

% Inserts gaps into the vector so that they can be ploted on the figure.
n = 2;
m = 1;
for i = 1:length(relate_feature_list)

    relate_feature_list2(n) = relate_feature_list(i);
    relate_feature_list2(m) = '';
    n = n + 2;
    m = m + 2;

end

f7 = figure('Name','Visualisation of Varying Time','NumberTitle','off');

% Variable to keep code running.
z = 1;

% The code for each matrix is essentially the same but must be repeated to plot all of them.
while z == 1
```

```matlab
for i = 1:num_intervals

    % Creates new figure.
    figure(f7)

    % Each matrix to be ploted on a new subplot.
    subplot(2,4,1)

    % Plots the matrix as an image, with each value representing a
    % pixel.
    imagesc(abs(R_St(:,:,i)));

    % Sets the axis of the matrix and relates it to the feature names.
    set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    % Required due to notation issues.
    R_S = R_St(:,:,i);

    % Converts the numerical values into text, that are to be plotted
    % on the figure.
    textstrings = num2str(R_S(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    % Creates a x-y grid equal to the size of the matrix.
    [x,y] = meshgrid(1:size(R_St,1));

    % Alligns the grid with textstrings.
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');

    % Assigns the colours each grid square in accordance to the value
    % in the original matrix.
    midvalue = mean(get(gca,'CLim'));
    textcolors = repmat(R_S(:) > midvalue,1,3);
    set(hstrings,{'Color'},num2cell(textcolors,2));

    title('Standard')

    % Comments are not repeated as are exactly the same.
    subplot(2,4,2)

    imagesc(abs(R_Sresamplet(:,:,i)));
    set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    R_Sresample = R_Sresamplet(:,:,i);

    textstrings = num2str(R_Sresample(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    [x,y] = meshgrid(1:size(R_St,1));
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
```

```
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Sresample(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Standard with resampling')

subplot(2,4,3)

imagesc(abs(R_Sfiltert(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_Sfilter = R_Sfiltert(:,:,i);

textstrings = num2str(R_Sfilter(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_Sfiltert,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Sfilter(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Standard with resampling and filtering')

subplot(2,4,4)

imagesc(abs(R_Bt(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_B = R_Bt(:,:,i);

textstrings = num2str(R_B(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_St,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_B(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Brownian')

subplot(2,4,8)

imagesc(abs(R_Ft(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
```

```
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    R_F = R_Ft(:,:,i);

    textstrings = num2str(R_F(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    [x,y] = meshgrid(1:size(R_St,1));
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
    midvalue = mean(get(gca,'CLim'));
    textcolors = repmat(R_F(:) > midvalue,1,3);

    set(hstrings,{'Color'},num2cell(textcolors,2));

    title('Fourier')

    subplot(2,4,5)

    imagesc(abs(R_Ct(:,:,i)));
    set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    R_C = R_Ct(:,:,i);

    textstrings = num2str(R_C(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    [x,y] = meshgrid(1:size(R_Ct,1));
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
    midvalue = mean(get(gca,'CLim'));
    textcolors = repmat(R_C(:) > midvalue,1,3);

    set(hstrings,{'Color'},num2cell(textcolors,2));

    title('Copula')

    subplot(2,4,6)

    imagesc(abs(R_Cresamplet(:,:,i)));
    set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    R_Cresample = R_Cresamplet(:,:,i);

    textstrings = num2str(R_Cresample(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    [x,y] = meshgrid(1:size(R_Cresamplet,1));
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
    midvalue = mean(get(gca,'CLim'));
```

```
        textcolors = repmat(R_Cresample(:) > midvalue,1,3);

        set(hstrings,{'Color'},num2cell(textcolors,2));

        title('Copula with resampling')

        subplot(2,4,7)

        imagesc(abs(R_Cfiltert(:,:,i)));
        set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
        set(gca,'xticklabel',relate_feature_list2);
        xtickangle(90);
        set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
        set(gca,'yticklabel',relate_feature_list2);

        R_Cfilter = R_Cfiltert(:,:,i);

        textstrings = num2str(R_Cfilter(:),'%0.2f');
        textstrings = strtrim(cellstr(textstrings));

        [x,y] = meshgrid(1:size(R_Cfiltert,1));
        hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
        midvalue = mean(get(gca,'CLim'));
        textcolors = repmat(R_Cfilter(:) > midvalue,1,3);

        set(hstrings,{'Color'},num2cell(textcolors,2));

        title('Copula with resampling and filtering')

        pause(3);

    end

end

% Dummy variable to make function work
str = 'done';

end
```

Visualise Matrices Scale

```
% A figure with a visualisation of each of the dependency matricies with varying time window.
% The time updates every 3 seconds, so to close the figure, call "ctrl-C" in the command
window.

function [str] = visualise_matrices_scale(R_St,R_Sresamplet,R_Sfiltert,...
    R_Bt,R_Ft,R_Ct,R_Cresamplet,R_Cfiltert,...
    num_intervals,selected_features,feature_name_list,relate_feature_list)

% Creates a list of all the selected features.
n = 1;
for i = 1:size(selected_features,2)

    if mod(selected_features(i),2) ~= 0

        selected_features2(n) = selected_features(i);
```

```
        n = n + 1;

    end

end

% Creates a string vector of all the selected feature names.
str = feature_name_list(selected_features2);

% Inserts gaps into the vector so that they can be ploted on the figure.
n = 2;
m = 1;
for i = 1:length(relate_feature_list)

    relate_feature_list2(n) = relate_feature_list(i);
    relate_feature_list2(m) = '';
    n = n + 2;
    m = m + 2;

end

f8 = figure('Name','Visualisation of Varying Scale','NumberTitle','off');

% Variable to keep code running.
z = 1;

% The code for each matrix is essentially the same but must be repeated to plot all of them.
while z == 1

    for i = 1:num_intervals

        % Creates new figure.
        figure(f8)

        % Each matrix to be ploted on a new subplot.
        subplot(2,4,1)

        % Plots the matrix as an image, with each value representing a
        % pixel.
        imagesc(abs(R_St(:,:,i)));

        % Sets the axis of the matrix and relates it to the feature names.
        set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
        set(gca,'xticklabel',relate_feature_list2);
        xtickangle(90);
        set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
        set(gca,'yticklabel',relate_feature_list2);

        % Required due to notation issues.
        R_S = R_St(:,:,i);

        % Converts the numerical values into text, that are to be plotted
        % on the figure.
        textstrings = num2str(R_S(:),'%0.2f');
        textstrings = strtrim(cellstr(textstrings));

        % Creates a x-y grid equal to the size of the matrix.
        [x,y] = meshgrid(1:size(R_St,1));
```

```
% Alligns the grid with textstrings.
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');

% Assigns the colours each grid square in accordance to the value
% in the original matrix.
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_S(:) > midvalue,1,3);
set(hstrings,{'Color'},num2cell(textcolors,2));

title('Standard')

% Comments are not repeated as are exactly the same.
subplot(2,4,2)

imagesc(abs(R_Sresamplet(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_Sresample = R_Sresamplet(:,:,i);

textstrings = num2str(R_Sresample(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_St,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Sresample(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Standard with resampling')

subplot(2,4,3)

imagesc(abs(R_Sfiltert(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_Sfilter = R_Sfiltert(:,:,i);

textstrings = num2str(R_Sfilter(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_Sfiltert,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Sfilter(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Standard with resampling and filtering')
```

```
subplot(2,4,4)

imagesc(abs(R_Bt(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_B = R_Bt(:,:,i);

textstrings = num2str(R_B(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_St,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_B(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Brownian')

subplot(2,4,8)

imagesc(abs(R_Ft(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_F = R_Ft(:,:,i);

textstrings = num2str(R_F(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_St,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_F(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Fourier')

subplot(2,4,5)

imagesc(abs(R_Ct(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_C = R_Ct(:,:,i);
```

```
textstrings = num2str(R_C(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_Ct,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_C(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Copula')

subplot(2,4,6)

imagesc(abs(R_Cresamplet(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_Cresample = R_Cresamplet(:,:,i);

textstrings = num2str(R_Cresample(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_Cresamplet,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Cresample(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Copula with resampling')

subplot(2,4,7)

imagesc(abs(R_Cfiltert(:,:,i)));
set(gca,'xtick',[0.5:0.5:size(R_St,1)]);
set(gca,'xticklabel',relate_feature_list2);
xtickangle(90);
set(gca,'ytick',[0.5:0.5:size(R_St,1)]);
set(gca,'yticklabel',relate_feature_list2);

R_Cfilter = R_Cfiltert(:,:,i);

textstrings = num2str(R_Cfilter(:),'%0.2f');
textstrings = strtrim(cellstr(textstrings));

[x,y] = meshgrid(1:size(R_Cfiltert,1));
hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
midvalue = mean(get(gca,'CLim'));
textcolors = repmat(R_Cfilter(:) > midvalue,1,3);

set(hstrings,{'Color'},num2cell(textcolors,2));

title('Copula with resampling and filtering')
```

```
        pause(3);

    end

end

% Dummy variable to make function work
str = 'done';

end
```

Visualise Matrices Resampling

```
% A figure with a visualisation of each of the dependency matricies with varying frequency.
% The time updates every 3 seconds, so to close the figure, call "ctrl-C"
% in the command window.

function [str] = visualise_matrices_resampling(R_Sresamplet,R_Sfiltert,...
    R_Cresamplet,R_Cfiltert,...
    selected_features,feature_name_list,relate_feature_list)

% Creates a list of all the selected features.
n = 1;
for i = 1:size(selected_features,2)

    if mod(selected_features(i),2) ~= 0

        selected_features2(n) = selected_features(i);
        n = n + 1;

    end

end

% Creates a string vector of all the selected feature names.
str = feature_name_list(selected_features2);

% Inserts gaps into the vector so that they can be ploted on the figure.
n = 2;
m = 1;
for i = 1:length(relate_feature_list)

    relate_feature_list2(n) = relate_feature_list(i);
    relate_feature_list2(m) = '';
    n = n + 2;
    m = m + 2;

end

f9 = figure('Name','Visualisation of Varying Resampling','NumberTitle','off');

% Variable to keep code running.
z = 1;

% The code for each matrix is essentially the same but must be repeated to plot all of them.
while z == 1
```

```
for i = 1:size(R_Sresamplet,3)

    % Creates new figure.
    figure(f9)

    % Each matrix to be ploted on a new subplot.
    subplot(2,2,1)

    % Plots the matrix as an image, with each value representing a
    % pixel.
    imagesc(abs(R_Sresamplet(:,:,i)));

    % Sets the axis of the matrix and relates it to the feature names.
    set(gca,'xtick',[0.5:0.5:size(R_Sresamplet,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_Sresamplet,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    % Required due to notation issues.
    R_Sresample = R_Sresamplet(:,:,i);

    % Converts the numerical values into text, that are to be plotted.
    % on the figure.
    textstrings = num2str(R_Sresample(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    % Creates a x-y grid equal to the size of the matrix.
    [x,y] = meshgrid(1:size(R_Sresamplet,1));

    % Alligns the grid with textstrings.
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');

    % Assigns the colours each grid square in accordance to the value
    % in the original matrix.
    midvalue = mean(get(gca,'CLim'));
    textcolors = repmat(R_Sresample(:) > midvalue,1,3);
    set(hstrings,{'Color'},num2cell(textcolors,2));

    title('Standard with resampling')

    % Comments are not repeated as are exactly the same.
    subplot(2,2,2)

    imagesc(abs(R_Sfiltert(:,:,i)));
    set(gca,'xtick',[0.5:0.5:size(R_Sfiltert,1)]);
    set(gca,'xticklabel',relate_feature_list2);
    xtickangle(90);
    set(gca,'ytick',[0.5:0.5:size(R_Sfiltert,1)]);
    set(gca,'yticklabel',relate_feature_list2);

    R_Sfilter = R_Sfiltert(:,:,i);

    textstrings = num2str(R_Sfilter(:),'%0.2f');
    textstrings = strtrim(cellstr(textstrings));

    [x,y] = meshgrid(1:size(R_Sfiltert,1));
    hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
```

```
        midvalue = mean(get(gca,'CLim'));
        textcolors = repmat(R_Sfilter(:) > midvalue,1,3);

        set(hstrings,{'Color'},num2cell(textcolors,2));

        title('Standard with resampling and filtering')

        subplot(2,2,3)

        imagesc(abs(R_Cresamplet(:,:,i)));
        set(gca,'xtick',[0.5:0.5:size(R_Cresamplet,1)]);
        set(gca,'xticklabel',relate_feature_list2);
        xtickangle(90);
        set(gca,'ytick',[0.5:0.5:size(R_Cresamplet,1)]);
        set(gca,'yticklabel',relate_feature_list2);

        R_Cresample = R_Cresamplet(:,:,i);

        textstrings = num2str(R_Cresample(:),'%0.2f');
        textstrings = strtrim(cellstr(textstrings));

        [x,y] = meshgrid(1:size(R_Cresamplet,1));
        hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
        midvalue = mean(get(gca,'CLim'));
        textcolors = repmat(R_Cresample(:) > midvalue,1,3);

        set(hstrings,{'Color'},num2cell(textcolors,2));

        title('Copula with resampling')

        subplot(2,2,4)

        imagesc(abs(R_Cfiltert(:,:,i)));
        set(gca,'xtick',[0.5:0.5:size(R_Cfiltert,1)]);
        set(gca,'xticklabel',relate_feature_list2);
        xtickangle(90);
        set(gca,'ytick',[0.5:0.5:size(R_Cfiltert,1)]);
        set(gca,'yticklabel',relate_feature_list2);

        R_Cfilter = R_Cfiltert(:,:,i);

        textstrings = num2str(R_Cfilter(:),'%0.2f');
        textstrings = strtrim(cellstr(textstrings));

        [x,y] = meshgrid(1:size(R_Cfiltert,1));
        hstrings = text(x(:),y(:),textstrings(:),'HorizontalAlignment','center');
        midvalue = mean(get(gca,'CLim'));
        textcolors = repmat(R_Cfilter(:) > midvalue,1,3);

        set(hstrings,{'Color'},num2cell(textcolors,2));

        title('Copula with resampling and filtering')

        pause(3);

    end

end
```

```
% Dummy variable to make function work.
str = 'done';

end
```