TABLE OF CONTENTS

Part I. Full Sample

```
clc
clearvars
```

Downloading data

```
Data = readtimetable('Data_MT.xlsx', 'Sheet', 'Hedging data', 'Range', 'A7:I1044');

variables = {'D','Clc','M','NI','OR','CP','SR','HR'};
Data.Properties.VariableNames = variables;

Data = Data(timerange('2015-11-27','2021-01-02'),:);  % define time period
delta_variables = strcat({'delta '}, variables);

 j    = size(Data,2);           % number of varibles
 data = table2array(Data);

ldata = diff(data,1,1); ldata(1,:) = [];  % first difference of data

[r, v] = size(data(:,2:end));
[m, w] = size(ldata);
```

Statistics

Descriptive Statistics

```
Descriptive_Statistics_level          = table();
Descriptive_Statistics_level.mean     = mean(data,     1)';
Descriptive_Statistics_level.std      = std(data,      1)';
Descriptive_Statistics_level.skewness = skewness(data,[],1)';
Descriptive_Statistics_level.kurtosis = kurtosis(data,[],1)';
Descriptive_Statistics_level.minimum  = min(data)';
Descriptive_Statistics_level.maximum  = max(data)';
Descriptive_Statistics_level.Properties.RowNames = variables;

Descriptive_Statistics_delta          = table();
Descriptive_Statistics_delta.mean     = mean(ldata,     1)';
Descriptive_Statistics_delta.std      = std(ldata,      1)';
Descriptive_Statistics_delta.skewness = skewness(ldata,[],1)';
Descriptive_Statistics_delta.kurtosis = kurtosis(ldata,[],1)';
Descriptive_Statistics_delta.minimum  = min(ldata)';
Descriptive_Statistics_delta.maximum  = max(ldata)';
Descriptive_Statistics_delta.Properties.RowNames = variables;
```

Multicollinearity

```
corr_level = corr(data);
corr_level = table(corr_level);
corr_level = splitvars(corr_level);
corr_level.Properties.RowNames      = variables;
corr_level.Properties.VariableNames = variables

corr_delta = corr(ldata);
corr_delta = table(corr_delta);
corr_delta = splitvars(corr_delta);
corr_delta.Properties.RowNames      = delta_variables;
corr_delta.Properties.VariableNames = delta_variables;

R0 = corrcoef(data(:,2:end));
VIF = array2table(diag(inv(R0))');
VIF.Properties.VariableNames = variables(2:end);

% Belsley Collinearity diagnostic
[sv,conIdx,varDecomp] = collintest(data(:,2:end),'display','off');
Collintest = array2table([sv,conIdx,varDecomp]);
Collintest.Properties.VariableNames = [{'sValue','condIdx'}, variables(2:end)];
```

Jarque Bera test for Normality

Price Levels

```
for i = 1:w
    [~, ~, jbstat_level(:,i), cval_level(:,i)] = jbtest(data(:,i)); % Jarque Bera test
End

jbtest_level = array2table([jbstat_level; cval_level(1,:)]);
jbtest_level.Properties.VariableNames = variables;
jbtest_level.Properties.RowNames = {'Test Statistic','Critical Value'}
```

First Difference

```
for i = 1:w
    [~, ~, jbstat_delta(:,i), cval_delta(:,i)] = jbtest(ldata(:,i)); % Jarque Bera test
end

jbtest_delta = array2table([jbstat_delta; cval_delta(1,:)]);
jbtest_delta.Properties.VariableNames = variables;
jbtest_delta.Properties.RowNames = {'Test Statistic','Critical Value'}
```

Unit root test

Augmented Dickey-Fuller Test

H0: Non-stationary vs. HA: Stationary

```
models  = 1;      % including a constant in alternative model
maxlag  = 52;     % 52 for weekly data (arbitrary number)
ic      = 'BIC';  % BIC as information criteria
alpha   = [ 0.01, 0.05, 0.10 ]';

stat = nan(1,j); pval = nan(1,j);
cval_level = nan(6,j); lags = nan(1,j);
```

First Difference

```
for i = 1:j
    [stat(:,i), pval(:,i), cval_level(:,i), ~, lags(:,i)] ...
        = augdfautolag(ldata(:,i), models, maxlag, ic);
end

delta_stationary_variables = delta_variables(pval < alpha(2)) % stationary variables

ADF_delta = array2table([stat; cval_level(2,:); pval; lags]);
ADF_delta.Properties.VariableNames = delta_variables;
ADF_delta.Properties.RowNames = {'Test Statistics','Critical Value','p-value', 'lags'}
```

Price Levels

```
for i = 1:j
    [stat(:,i), pval(:,i), cval_level(:,i), ~, lags(:,i)] ...
        = augdfautolag(data(:,i), models, maxlag, ic);
end

stationary_variables     = variables(pval < alpha(2))       % Stationary variables
non_stationary_variables = variables(pval > alpha(2))       % non-stationary variables
non_stationary_variable  = non_stationary_variables(2:end); % excluding D

ADF_level = array2table([stat; cval_level(2,:); pval; lags]);
ADF_level.Properties.VariableNames = variables;
ADF_level.Properties.RowNames = ADF_delta.Properties.RowNames
```

Cointegration

Define non-stationary variables

```
non_stationary = pval > alpha(2); % columns of non-stationary variables
non_stationary = non_stationary(2:end); % excluding D

Y_t = data(:,1);
X_t = data(:,2:end);
X_t = X_t(:, non_stationary); % non stationary variables (exl. D)
```

Single Cointegration

Augmented Dickey Fuller test

```
[~, c] = size(X_t);
stat = nan(1, c); pval = nan(1, c);
cval_level = nan(6, c); lags = nan(1, c);

 for i = 1:c
     reg_t = fitlm(X_t(:,i), Y_t);
     [stat(:,i), pval(:,i), cval_level(:,i), ~, lags(:,i)] ...
            = augdfautolag(reg_t.Residuals.Raw, models, maxlag, ic);
 end
cointegrated_values     = non_stationary_variable(pval < alpha(2));
non_cointegrated_values = non_stationary_variable(pval > alpha(2));

coint = array2table([stat; cval_level(2,:); pval; lags]);
coint.Properties.VariableNames = strcat({'D ~ '}, non_stationary_variable);
coint.Properties.RowNames = ADF_delta.Properties.RowNames
```

Information criteria

```
InformationCriteria = varorder([Y_t, X_t], 12);
lags = InformationCriteria.bicor - 1;     % lags in VECM: Information criteria – 1

for i = 1:c
    IC = varorder([Y_t, X_t(:,i)], 12);
    lag(i,:) = IC.bicor - 1;              % lags in ECM: Information criteria - 1
end
```

Multiple Cointegration

Johanssen test

Null hypothesis *H*(*r*) of cointegration rank less than or equal to *r*
H1*: Include intercepts;   Data: no deterministic trends in the levels of the data.

```
[htrace,  pValue1,stat1,cValue1, mles1] = jcitest([Y_t, X_t], 'test', 'trace', 'model',
'H1*', 'lags', lags);
[hmaxeig, pValue2,stat2,cValue2, mles2] = jcitest([Y_t, X_t], 'test', 'maxeig',
'model', 'H1*', 'lags', lags);
```

Part II. Models

In-Sample Variables (27 November 2015 to 31 December 2018)

```
iData = Data(timerange('2015-11-27','2018-12-31'),:);
data  = table2array(iData);
lidata = diff(data, 1, 1);  % Taking first difference

Y_t = data(:, 1);
X_t = data(:, 2:end);

Y_t_min_1 = lagmatrix(Y_t, 1); Y_t_min_1(1, :) = [];
X_t_min_1 = lagmatrix(X_t, 1); X_t_min_1(1, :) = [];

delta_Y_t = lidata(:, 1);
delta_X_t = lidata(:, 2:end);
```

Models

Naïve

```
error_Naive = delta_Y_t - delta_X_t;
```

ECM

```
non_stationary_X_t = X_t(:,non_stationary);
v = size(non_stationary_X_t,2);

for i = 1:v
    mdl = vecm(2, 1, 0);
    ECM = estimate(mdl, [Y_t, non_stationary_X_t(:,i)], 'model', 'H1*');

    ECM_CointegrationConstant(i,1) = ECM.CointegrationConstant;
```

```
    ECM_Cointegration(i,1)         = ECM.Cointegration(1);
    ECM_Constant(i,:)    = ECM.Constant(1);
    ECM_Adjustment(i,:)  = ECM.Adjustment(1);
    ECM_Impact(i,:)      = ECM.Impact(1,1);

    ECM_model            = summarize(ECM);
    ECM_SE(:,i)          = ECM_model.Table.StandardError;
    ECM_pvalue(:,i)      = ECM_model.Table.PValue;

    errors               = infer(ECM, [Y_t, non_stationary_X_t(:,i)]);
    OLS_ECM              = fitlm(errors(:,2), errors(:,1));

    alpha_ECM(:,i)       = OLS_ECM.Coefficients.Estimate(1);
    h_ECM(:,i)           = OLS_ECM.Coefficients.Estimate(2);  % beta, hedge ratio
    SE_ECM(:,i)          = OLS_ECM.Coefficients.SE;
    p_ECM(:,i)           = OLS_ECM.Coefficients.pValue;

    error_ECM(:,i)       = OLS_ECM.Residuals.Raw;
end

ECM_models = array2table([ECM_CointegrationConstant, ECM_Cointegration, ECM_Constant,
ECM_Impact]);
ECM_models.Properties.VariableNames = {'Cointegration
Constant','Cointegration','Constant','ECT'};
ECM_models.Properties.RowNames = strcat({'delta '}, non_stationary_variable)
```

OLS

```
mOLS  = fitlm(delta_X_t, delta_Y_t); % Multivariate OLS
SE_mOLS = mOLS.Coefficients.SE;

f = size(X_t,2);

for i = 1:f
    OLS  = fitlm(delta_X_t(:,i), delta_Y_t);
    SE_OLS(:,i)    = OLS.Coefficients.SE;
    alpha_OLS(1,i) = OLS.Coefficients.Estimate(1)';
    p_OLS(:,i).    = OLS.Coefficients.pValue;

    h_OLS(1,i)     = OLS.Coefficients.Estimate(2)'; % Hedge Ratio
    error_OLS(:,i) = OLS.Residuals.Raw;
end
```

VECM

```
rank       = sum(table2array(hmaxeig));     % Number of cointegration relationship
VECM_data  = [Y_t, X_t(:, non_stationary)];
numseries  = size(VECM_data,2);

mdl        = vecm(numseries, rank, 0);                        % Define VECM
VECM       = estimate(mdl, VECM_data, 'model', 'H1*');  % Estimate VECM
VECM_model = summarize(VECM);

error_VECM = infer(VECM,VECM_data);                           % Infer residuals from VECM
OLS_VECM   = fitlm(error_VECM(:,2:end), error_VECM(:,1)); % OLS of the residuals
SE_VECM    = OLS_VECM.Coefficients.SE;

n = [SE_VECM' nan; SE_mOLS'; SE_ECM(1,:) nan nan; SE_ECM(2,:) nan nan; SE_OLS(1,:) nan;
SE_OLS(2,:) nan]
```

Multicollinearity in Residuals (VECM)

Correlation Matrix

```
corr_VECM = corr(error_VECM);
corr_VECM = table(corr_VECM);
corr_VECM = splitvars(corr_VECM);
corr_VECM.Properties.RowNames      = non_stationary_variables;
corr_VECM.Properties.VariableNames = non_stationary_variables
```

Variance Inflation Factor

```
R0 = corrcoef(error_VECM);
VIF = array2table(diag(inv(R0))');
VIF.Properties.VariableNames = non_stationary_variables
```

Belsley Collinearity diagnotic

```
[sv,conIdx,varDecomp] = collintest(error_VECM,'display','off');
Collintest = array2table([sv,conIdx,varDecomp]);
Collintest.Properties.VariableNames = [{'sValue','condIdx'}, non_stationary_variables]

vecm_coint = array2table([VECM.CointegrationConstant VECM.Cointegration']);
vecm_coint.Properties.VariableNames = [{'intercept'}, non_stationary_variables];
```

Heteroscedasticity, Autocorrelation & Normality

- Ljung Box (LBQ1) ~ H0: no autocorrelation in residuals
- Engle's ARCH effect ~ H0: no conditional heteroscedasticity in the residuals
- Jarque-Bera ~ H0: residuals comes from a normal distribution with an unknown mean & variance

```
names = [{'VECM'}, strcat({'ECM ('}, non_stationary_variable, {')'}), {'OLS'},
strcat({'OLS ('}, variables(2:end), {')'})];

estimated_Residuals = [OLS_VECM.Residuals.Raw, error_ECM, mOLS.Residuals.Raw,
error_OLS];

lags = 4; l = size(estimated_Residuals,2);

LBQ1st = nan(lags,l); LBQ1cv = nan(lags,l);
ARCHst = nan(lags,l); ARCHcv = nan(lags,l);
JBstat = nan(1,l);    JBcval = nan(1,l);

for i = 1:l
   [~, ~, LBQ1st(:,i), LBQ1cv(:,i)] = lbqtest(estimated_Residuals(:,i), 'lags',
[1:lags]);

   [~, ~, ARCHst(:,i), ARCHcv(:,i)] = archtest(estimated_Residuals(:,i), 'lags',
[1:lags]);

   [~, ~, JBstat(:,i), JBcval(:,i)] = jbtest(estimated_Residuals(:,i));
End

LBQ01 = array2table([LBQ1st LBQ1cv(:,1)]);
LBQ01.Properties.VariableNames = [names, {'Critical value'}];

ARCH  = array2table([ARCHst ARCHcv(:,1)]);
ARCH.Properties.VariableNames  = LBQ01.Properties.VariableNames;

JBtest = array2table([JBstat JBcval(1)]);
JBtest.Properties.VariableNames = LBQ01.Properties.VariableNames;
```

Hedge Ratio & Portfolio Variance

```
unhedged  = var(delta_Y_t);
```

Find portfolio variance by using the built-in-function PORTVAR(DATA, WEIGHT), where weight is the hedge ratio

Naïve

```
c = size(X_t,2);

for i = 1:c
    var_Naive(1,i) = portvar([delta_Y_t, delta_X_t(:,i)],[1 -1]);

end
```

ECM

```
delta_non_stationary_X_t = delta_X_t(:, non_stationary);

for i = 1:v
    var_ECM(1,i) = portvar([delta_Y_t, delta_X_t(:,i)],[1 -h_ECM(i)]);
end
```

OLS

```
h_mOLS   = mOLS.Coefficients.Estimate(2:end)';
var_mOLS = portvar([delta_Y_t, delta_X_t],[1 -h_mOLS]);

for i = 1:f
    var_OLS(1,i) = portvar([delta_Y_t, delta_X_t(:,i)],[1 -h_OLS(:,i)]);
end
```

VECM

```
h_VECM   = OLS_VECM.Coefficients.Estimate(2:end)';
var_VECM = portvar( [delta_Y_t, delta_non_stationary_X_t],[1 -h_VECM]);
```

Summary

```
name = [{'VECM'}, strcat({'ECM ('}, non_stationary_variable,{')'}), strcat({'Näive ('},
variables(2:end),{')'}),{'Mulivariate OLS'}, strcat({'OLS ('}, variables(2:end),{')'})];

Hedging_Result = array2table([unhedged, var_VECM, var_ECM, var_Naive, var_mOLS var_OLS;
NaN, 1 - [var_VECM var_ECM, var_Naive var_mOLS var_OLS]./unhedged]');

Hedging_Result.Properties.VariableNames = {'Variance','HE'};
Hedging_Result.Properties.RowNames = [{'unhedged'}, name]
Hedging_Ratios = array2table([h_VECM(1), NaN, h_VECM(2:end); h_ECM(1), NaN,
h_ECM(2:end); h_mOLS; h_OLS ]');
```

```
Hedging_Ratios.Properties.VariableNames = {'VECM','ECM','Multivariate OLS', 'Univariate
OLS'};
Hedging_Ratios.Properties.RowNames = variables(2:end)
```

Out-of-Sample Variables (01 January 2019 to 01 January 2021)

```
oData     = Data(timerange('2019-01-01','2021-01-01'),:);
data19    = table2array(oData);
ldata_19 = diff(data19,1,1);

Y_t_19 = data19(:,1);
X_t_19 = data19(:,2:end);

Y_t_min_1_19 = lagmatrix(Y_t_19,1); Y_t_min_1_19(1,:) = [];
X_t_min_1_19 = lagmatrix(X_t_19,1); X_t_min_1_19(1,:) = [];

delta_Y_t_19 = ldata_19(:,1);
delta_X_t_19 = ldata_19(:,2:end);

delta_non_stationary_X_t_19 = delta_X_t_19(:, non_stationary);
```

Hedging Results

```
unhedged_19   = var(delta_Y_t_19);
```

Find portfolio variance by using the built-in-function PORTVAR(DATA, WEIGHT), where weight is the hedge ratio

Naïve

```
c = size(X_t_19,2);

for i = 1:c
    var_Naive_19(1,i) = portvar([delta_Y_t_19, delta_X_t_19(:,i)],[1 -1]);
end
```

OLS

```
var_mOLS_19 = portvar([delta_Y_t_19, delta_X_t_19],[1 -h_mOLS]);

for i = 1:f
var_OLS_19(1,i) = portvar([delta_Y_t_19, delta_X_t_19(:,i)],[1 -h_OLS(:,i)]);
end
```

ECM

```
for i = 1:v
var_ECM_19(1,i) = portvar([delta_Y_t_19, delta_non_stationary_X_t_19(:,i)],[1 -
h_ECM(i)]);
end
```

VECM

```
var_VECM_19  = portvar( [delta_Y_t_19, delta_non_stationary_X_t_19 ],[1 -h_VECM]);
```

Summary

```
Hedging_Result_19 = array2table(...
    [unhedged_19, var_VECM_19, var_ECM_19, var_Naive_19 var_mOLS_19, var_OLS_19;...
     NaN 1 - [var_VECM_19 var_ECM_19 var_Naive_19 var_mOLS_19
var_OLS_19]./unhedged_19]');

Hedging_Result_19.Properties.VariableNames = {'Variance','HE'};
Hedging_Result_19.Properties.RowNames = [{'unhedged'}, name]
```

Part III. Stability test

Variables

```
Y_t_min_1 = lagmatrix(Y_t,1); Y_t_min_1(1,:) = [];
X_t_min_1 = lagmatrix(X_t,1); X_t_min_1(1,:) = [];

delta_Y_t = lidata(:,1);
delta_X_t = lidata(:,2:end);

delta_Y_t_min_1 = diff(Y_t_min_1);

non_stationary_X_t       = X_t(:,non_stationary);
non_stationary_X_t_min_1 = lagmatrix(non_stationary_X_t,1);
non_stationary_X_t_min_1(1,:) = [];

delta_non_stationary_X_t_min_1 = diff(non_stationary_X_t_min_1);
```

CUSUM Test

OLS

```
cusumtest(delta_X_t(:,3), delta_Y_t,'Intercept',true);
```

ECM

```
[r, v]                   = size(non_stationary_X_t);

for i = 1:v
    mdl         = vecm(2, 1, 0);
    ECM         = estimate(mdl,   [Y_t, non_stationary_X_t(:,i)]);

    ECM_CointegrationConstant(i,1) = ECM.CointegrationConstant;
    ECM_Cointegration(i,1)         = ECM.Cointegration(1);
end




for i = 1:v
    XXX(:,i) = [ECM_Cointegration(i).*non_stationary_X_t_min_1(:,i) +
ECM_CointegrationConstant(i)]';
    XXY(:,i) = Y_t_min_1 - XXX(:,i);

    cusumtest([delta_non_stationary_X_t_min_1(:,i), XXY(2:end,i), delta_Y_t_min_1],
delta_Y_t(2:end),'Intercept',true,'Alpha', 0.1);
end
```

VECM

```
non_stationary_data = [Y_t, non_stationary_X_t]; numseries =
size(non_stationary_data,2);
mdl        = vecm(numseries, rank, 1);
VECM       = estimate(mdl,   non_stationary_data);
VECM_model = summarize(VECM); VECM_model.Table;

XX1 = [VECM.Cointegration(:,1)'.*[Y_t_min_1 non_stationary_X_t_min_1] +
VECM.CointegrationConstant(1)];
XX2 = [VECM.Cointegration(:,2)'.*[Y_t_min_1 non_stationary_X_t_min_1] +
VECM.CointegrationConstant(2)];

XY1 = Y_t_min_1 - sum(XX1,2);
XY2 = Y_t_min_1 - sum(XX2,2);

cusumtest([XY1(2:end), XY2(2:end)], delta_Y_t(2:end), 'Intercept',true)
```

Chow Test

```
bp = size(Y_t,1)/2; % breaking point
```

VECM

```
[h1 p1 stat1 cv1] = chowtest([XY1, XY2],     delta_Y_t, bp-1, 'Intercept', true);
```

OLS

```
[h2 p2 stat2 cv2] = chowtest(delta_X_t(:,3), delta_Y_t, bp-1, 'Intercept', true);
```

ECM

```
for i = 1:v
   [h3(i) p3(:,i) stat3(:,i) cv3(:,i)] = chowtest(XXY(2:end,i), delta_Y_t(2:end), bp-2,
'Intercept',true);
end

ChowTest = array2table([h1 h3 h2; stat1 stat3 stat2; cv1 cv3 cv2; p1 p3 p2]);
ChowTest.Properties.VariableNames = [name(1:7), name(end)];
ChowTest.Properties.RowNames = {'ChowTest','Test Statiatic','Critical Value','p-value'}
```