Table of Contents

Housekeeping

```
clear all;
close all;
clc;
```

Import interest rates

```
projectdir_ir    = '/Users/sd/Desktop/Study/master/data/
interest_rates.xlsx';
opts               = detectImportOptions(projectdir_ir);
opts.Sheet         = 'InterestRates1M';
opts.VariableTypes =
 {'string', 'double', 'double', 'double', 'double', 'double', 'double', 'double',
opts.VariableNamesRange = 'A3';
opts.DataRange     = 'A4';

ir               = readtable(projectdir_ir, opts);

ir.Date          = datetime(ir.Date); % change date's format
ir               = table2timetable(ir);
ir.Date          = eomdate(ir.Date);
```

Import currencies

```
projectdir_curr    = '/Users/sd/Desktop/Study/master/data/
curr_base.xlsx';
opts               = detectImportOptions(projectdir_curr);
```

```
opts.Sheet          = 'XR_NB';
opts.VariableTypes =
 {'string', 'double', 'double', 'double', 'double', 'double', 'double', 'double'};
opts.VariableNamesRange = 'A4';
opts.DataRange      = 'A7';

curr                = readtable(projectdir_curr, opts);

curr.Date           = datetime(curr.QUOTE_CUR,'InputFormat', 'yyyy-
MM', 'Format', 'yyyy-MM-dd'); % change date's format
curr                = table2timetable(curr);
curr.Date           = eomdate(curr.Date);
curr                = curr(:, 2:end);

% Merge currencies and interest rates
curr  = innerjoin(curr, ir);

% Curr returns
curr.CHF_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.CHF_r(t) = (curr.CHF(t) *
 (1+(curr.Switzerland_ir(t)/12)/100)) ./ curr.CHF(t-1) -
 (1+(curr.Norway_ir(t)/12)/100);
end

curr.JPY_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.JPY_r(t) = (curr.JPY(t) * (1+(curr.Japan_ir(t)/12)/100))./
 curr.JPY(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end

curr.CAD_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.CAD_r(t) = (curr.CAD(t) * (1+(curr.CAD_ir(t)/12)/100))./
 curr.CAD(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end

curr.EUR_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.EUR_r(t) = (curr.EUR(t) * (1+(curr.EUR_ir(t)/12)/100))./
 curr.EUR(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end

curr.GBP_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.GBP_r(t) = (curr.GBP(t) * (1+(curr.UK_ir(t)/12)/100))./
 curr.GBP(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end

curr.SEK_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.SEK_r(t) = (curr.SEK(t) * (1+(curr.Sweden_ir(t)/12)/100))./
 curr.SEK(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end
```

```
curr.USD_r = nan(size(curr, 1),1);
for t = 2 : size(curr, 1)
    curr.USD_r(t) = (curr.USD(t) * (1+(curr.USA_ir(t)/12)/100))./
 curr.USD(t-1) - (1+(curr.Norway_ir(t)/12)/100);
end

% Exclude prices
curr_ret = curr(2:end, 16:end);
```

Import indices

```
projectdir_indices    = '/Users/sd/Desktop/Study/master/data/
indices.xlsx';
opts                  = detectImportOptions(projectdir_indices );
opts.Sheet            = 'Indices';
opts.VariableTypes =
 {'string', 'double', 'double', 'double', 'double', 'double', 'double', 'double',
opts.VariableNamesRange = 'A2';
opts.DataRange        = 'A3';

indices               = readtable(projectdir_indices, opts);

indices.Date          = datetime(indices.Name); % change date's format
indices               = table2timetable(indices);
indices.Date          = eomdate(indices.Date);

% Merge currencies and interest rates
indices   = innerjoin(indices , curr);

% indices returns
indices.MSCI_NORWAY_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_NORWAY_r(t) = indices.MSCI_NORWAY(t) ./
 indices.MSCI_NORWAY(t-1) - (indices.Norway_ir(t)/12)/100 - 1;
end

indices.MSCI_CANADA_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_CANADA_r(t) = (indices.MSCI_CANADA(t)./
indices.MSCI_CANADA(t-1) -
 (indices.CAD_ir(t)/12)/100)*(indices.CAD(t)./ indices.CAD(t-1)) - 1;
end

indices.MSCI_FRANCE_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_FRANCE_r(t) = (indices.MSCI_FRANCE(t)./
indices.MSCI_FRANCE(t-1) -
 (indices.EUR_ir(t)/12)/100)*(indices.EUR(t)./ indices.EUR(t-1)) - 1;
end

indices.MSCI_GERMANY_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
```

```
    indices.MSCI_GERMANY_r(t) = (indices.MSCI_GERMANY(t)./
indices.MSCI_GERMANY(t-1) -
 (indices.EUR_ir(t)/12)/100)*(indices.EUR(t)./ indices.EUR(t-1)) - 1;
end

indices.MSCI_JAPAN_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_JAPAN_r(t) = (indices.MSCI_JAPAN(t)./
indices.MSCI_JAPAN(t-1) -
 (indices.Japan_ir(t)/12)/100)*(indices.JPY(t)./ indices.JPY(t-1)) -
 1;
end

indices.MSCI_NETHERLANDS_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_NETHERLANDS_r(t) = (indices.MSCI_NETHERLANDS(t)./
indices.MSCI_NETHERLANDS(t-1) -
 (indices.EUR_ir(t)/12)/100)*(indices.EUR(t)./ indices.EUR(t-1)) - 1;
end

indices.MSCI_ITALY_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_ITALY_r(t) = (indices.MSCI_ITALY(t)./
indices.MSCI_ITALY(t-1) -
 (indices.EUR_ir(t)/12)/100)*(indices.EUR(t)./ indices.EUR(t-1)) - 1;
end

indices. MSCI_BELGIUM_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices. MSCI_BELGIUM_r(t) = (indices.MSCI_BELGIUM(t)./
indices.MSCI_BELGIUM(t-1) -
 (indices.EUR_ir(t)/12)/100)*(indices.EUR(t)./ indices.EUR(t-1)) - 1;
end

indices.MSCI_SWITZERLAND_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_SWITZERLAND_r(t) = (indices.MSCI_SWITZERLAND(t)./
indices.MSCI_SWITZERLAND(t-1) -
 (indices.Switzerland_ir(t)/12)/100)*(indices.CHF(t)./
 indices.CHF(t-1)) - 1;
end

indices.MSCI_UK_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_UK_r(t) = (indices.MSCI_UK(t)./indices.MSCI_UK(t-1) -
 (indices.UK_ir(t)/12)/100)*(indices.GBP(t)./ indices.GBP(t-1)) - 1;
end

indices.MSCI_USA_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_USA_r(t) = (indices.MSCI_USA(t)./
indices.MSCI_USA(t-1) - (indices.USA_ir(t)/12)/100)*(indices.USD(t)./
 indices.USD(t-1)) - 1;
end
```

```
indices.MSCI_SWEDEN_r = nan(size(indices, 1),1);
for t = 2 : size(indices, 1)
    indices.MSCI_SWEDEN_r(t) = (indices.MSCI_SWEDEN(t)./
indices.MSCI_SWEDEN(t-1) -
 (indices.Sweden_ir(t)/12)/100)*(indices.SEK(t)./ indices.SEK(t-1)) -
 1;
end

% Exclude prices
indices_ret = indices(2:end, 36:end);
```

Data separation and merging

```
data_all  = innerjoin(indices_ret, curr_ret); % Merge
data = table2array(data_all(:, 1:end));
L = 135;
```

Descriptive Statistics

```
Mu       = mean(data); % monthly

s_dev    = std(data); % Calculate st. dev.
median_d = median(data);
min_d = min(data);
max_d = max(data);

res = array2table([Mu' s_dev' median_d' min_d' max_d'], 'RowNames',
 data_all.Properties.VariableNames, 'VariableNames', {'mean', 'st.
 dev.', 'median', 'min','max'});
writetable(res,'/Users/sd/Desktop/Study/master/data/export/
d_stat.xlsx','WriteRowNames',true)
```

Correlations: to do

```
corr_all = array2table(corr(data));
writetable(corr_all,'/Users/sd/Desktop/Study/master/data/export/
corr_all.xlsx','WriteRowNames',true)
```

Equity weights

```
% Min-var:
[W_eq, W_eq_ns, eq_ret, eq_ret_ns] = equity_weights(data);
% Eq. weighted
W_eq_ew = ones(size(indices_ret,2), L)/size(indices_ret,2); % L could
 be needed to change!!!
```

Retruns

```
T = 300; % choose 10 years to build initial model
i = 1; % count
```

```
eq_ret_ew = zeros(1, size(data, 1) - T);
 while T < size(data, 1) % do not need for the last month
        eq_ret_ew(1,i)    = data(T + 1,1:12)*W_eq_ew(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end

% avg
m_eq_ret_ns = mean(eq_ret_ns);
m_eq_ret = mean(eq_ret);
m_eq_ret_ew = mean(eq_ret_ew);

% St. dev.
eq_std_nL_ns = std(eq_ret);
eq_std_nL = std(eq_ret_ns);
eq_std_nL_ew = std(eq_ret_ew);

% SR
SR_eq_ns = m_eq_ret_ns/eq_std_nL_ns;
SR_eq = m_eq_ret/eq_std_nL;
SR_eq_ew = m_eq_ret_ew/eq_std_nL_ew;

% Kurtosis
eq_kurtosis_nL_ns = kurtosis(eq_ret');
eq_kurtosis_nL = kurtosis(eq_ret_ns');
eq_kurtosis_nL_ew = kurtosis(eq_ret_ew');

% Skewness
eq_skewness_nL_ns = skewness(eq_ret');
eq_skewness_nL = skewness(eq_ret_ns');
eq_skewness_nL_ew = skewness(eq_ret_ew');
```

Unitary hedge

ns MV

```
W_currency_gh_ns = zeros(7,L);
for i = 1:1:L
    W_currency_gh_ns(:, i) = [-W_eq_ns(9,i) -W_eq_ns(5,i) -
W_eq_ns(2,i) -(W_eq_ns(3,i) + W_eq_ns(4,i) + W_eq_ns(6,i) +
 W_eq_ns(7,i) + W_eq_ns(8,i)) -W_eq_ns(10,i) -W_eq_ns(12,i) -
W_eq_ns(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_gh_ns = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
        curr_ret_gh_ns(1,i)    = data(T +
 1,13:end)*W_currency_gh_ns(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end
```

```
% MV
W_currency_gh = zeros(7,L);
for i = 1:1:L
    W_currency_gh(:, i) = [-W_eq(9,i) -W_eq(5,i) -W_eq(2,i) -
(W_eq(3,i) + W_eq(4,i) + W_eq(6,i) + W_eq(7,i) + W_eq(8,i)) -
W_eq(10,i) -W_eq(12,i) -W_eq(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_gh = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
        curr_ret_gh(1,i)    = data(T + 1,13:end)*W_currency_gh(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end

% EW
W_currency_gh_ew = zeros(7,L);
for i = 1:1:L
    W_currency_gh_ew(:, i) = [-W_eq_ew(9,i) -W_eq_ew(5,i) -
W_eq_ew(2,i) -(W_eq_ew(3,i) + W_eq_ew(4,i) + W_eq_ew(6,i) +
 W_eq_ew(7,i) + W_eq_ew(8,i)) -W_eq_ew(10,i) -W_eq_ew(12,i) -
W_eq_ew(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_gh_ew = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
        curr_ret_gh_ew(1,i)    = data(T +
 1,13:end)*W_currency_gh_ew(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end


% Final weights
W_gh_ns = [W_eq_ns', W_currency_gh_ns']';
W_gh = [W_eq', W_currency_gh']';
W_gh_ew = [W_eq_ew', W_currency_gh_ew']';

% weights sum
W_currency_gh_ns_s = sum(W_currency_gh_ns);
W_currency_gh_s = sum(W_currency_gh);
W_currency_gh_ew_s = sum(W_currency_gh_ew);

% Retruns
[port_ret_gh_ns, port_ret_gh_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_gh_ns);
[port_ret_gh, port_ret_gh_cum] = ptf_return(eq_ret, curr_ret_gh);
[port_ret_gh_ew, port_ret_gh_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_gh_ew);
```

```
% avg
m_port_ret_gh_ns = mean(port_ret_gh_ns);
m_port_ret_gh = mean(port_ret_gh);
m_port_ret_gh_ew = mean(port_ret_gh_ew);

m_curr_ret_gh_ns = mean(curr_ret_gh_ns);
m_curr_ret_gh = mean(curr_ret_gh);
m_curr_ret_gh_ew = mean(curr_ret_gh_ew);

% St. dev.
port_std_gh_ns = std(port_ret_gh_ns);
port_std_gh = std(port_ret_gh);
port_std_gh_ew = std(port_ret_gh_ew);

curr_std_gh_ns = std(curr_ret_gh_ns);
curr_std_gh = std(curr_ret_gh);
curr_std_gh_ew = std(curr_ret_gh_ew);

% SR
SR_ptf_gh_ns = m_port_ret_gh_ns/port_std_gh_ns;
SR_ptf_gh = m_port_ret_gh/port_std_gh;
SR_ptf_gh_ew = m_port_ret_gh_ew/port_std_gh_ew;

SR_curr_gh_ns = m_curr_ret_gh_ns/curr_std_gh_ns;
SR_curr_gh = m_curr_ret_gh/curr_std_gh;
SR_curr_gh_ew = m_curr_ret_gh_ew/curr_std_gh_ew;

% Kurtosis
port_kurtosis_gh_ns = kurtosis(port_ret_gh_ns');
port_kurtosis_gh = kurtosis(port_ret_gh');
port_kurtosis_gh_ew = kurtosis(port_ret_gh_ew');

curr_kurtosis_gh_ns = kurtosis(curr_ret_gh_ns');
curr_kurtosis_gh = kurtosis(curr_ret_gh');
curr_kurtosis_gh_ew = kurtosis(curr_ret_gh_ew');

% Skewness
port_skewness_gh_ns = skewness(port_ret_gh_ns');
port_skewness_gh = skewness(port_ret_gh');
port_skewness_gh_ew = skewness(port_ret_gh_ew');

curr_skewness_gh_ns = skewness(curr_ret_gh_ns');
curr_skewness_gh = skewness(curr_ret_gh');
curr_skewness_gh_ew = skewness(curr_ret_gh_ew');
```

Universal hedge

```
k = 1;
for i = 13:1:19
    for j = 13:1:19
        ER_var(i-12, k) =  mean(var([data(1:300, i) data(1:300,
 j)],0,2));
```

```
        k = k + 1;
        if j == 19
            i = i+1;
            k = 1;
        end
    end
end

ER_var_avg = 0;
for r = 1:size(ER_var)
    for c = 1:size(ER_var)
        if c >= r
            ER_var_avg = ER_var(r,c) + ER_var_avg;
        end
    end
end


universal_hedge_ratio = (mean(mean(data(1:300, :))) -
 mean(var(data(1:300, :))))/(mean(mean(data(1:300, :))) -
 0.5*mean(ER_var_avg));

% ns MV
W_currency_universal_ns = zeros(7,L);
for i = 1:1:L
    W_currency_universal_ns(:, i) = universal_hedge_ratio*[-
W_eq_ns(9,i) -W_eq_ns(5,i) -W_eq_ns(2,i) -(W_eq_ns(3,i) + W_eq_ns(4,i)
 + W_eq_ns(6,i) + W_eq_ns(7,i) + W_eq_ns(8,i)) -W_eq_ns(10,i) -
W_eq_ns(12,i) -W_eq_ns(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_universal_ns = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
        curr_ret_universal_ns(1,i)    = data(T +
 1,13:end)*W_currency_universal_ns(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end

% MV
W_currency_universal = zeros(7,L);
for i = 1:1:L
    W_currency_universal(:, i) = universal_hedge_ratio*[-W_eq(9,i) -
W_eq(5,i) -W_eq(2,i) -(W_eq(3,i) + W_eq(4,i) + W_eq(6,i) + W_eq(7,i) +
 W_eq(8,i)) -W_eq(10,i) -W_eq(12,i) -W_eq(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_universal = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
```

```
        curr_ret_universal(1,i)    = data(T +
 1,13:end)*W_currency_universal(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end

% EW
W_currency_universal_ew = zeros(7,L);
for i = 1:1:L
    W_currency_universal_ew(:, i) = universal_hedge_ratio*[-
W_eq_ew(9,i) -W_eq_ew(5,i) -W_eq_ew(2,i) -(W_eq_ew(3,i) + W_eq_ew(4,i)
 + W_eq_ew(6,i) + W_eq_ew(7,i) + W_eq_ew(8,i)) -W_eq_ew(10,i) -
W_eq_ew(12,i) -W_eq_ew(11,i)]';
end

T = 300; % choose 10 years to build initial model
i = 1; % count
curr_ret_universal_ew = zeros(1,L);
 while T < size(data, 1) % do not need for the last month
        curr_ret_universal_ew(1,i)    = data(T +
 1,13:end)*W_currency_universal_ew(:,i);
        T = T + 1; % rebalancing each month
        i = i + 1;
 end


% Final weights
W_universal_ns = [W_eq_ns', W_currency_universal_ns']';
W_universal = [W_eq', W_currency_universal']';
W_universal_ew = [W_eq_ew', W_currency_universal_ew']';

% weights sum
W_currency_universal_ns_s = sum(W_currency_universal_ns);
W_currency_universal_s = sum(W_currency_universal);
W_currency_universal_ew_s = sum(W_currency_universal_ew);

% Retruns
[port_ret_universal_ns, port_ret_universal_ns_cum] =
 ptf_return(eq_ret_ns, curr_ret_universal_ns);
[port_ret_universal, port_ret_universal_cum] = ptf_return(eq_ret,
 curr_ret_universal);
[port_ret_universal_ew, port_ret_universal_ew_cum] =
 ptf_return(eq_ret_ew, curr_ret_universal_ew);

% avg
m_port_ret_universal_ns = mean(port_ret_universal_ns);
m_port_ret_universal = mean(port_ret_universal);
m_port_ret_universal_ew = mean(port_ret_universal_ew);

m_curr_ret_universal_ns = mean(curr_ret_universal_ns);
m_curr_ret_universal = mean(curr_ret_universal);
m_curr_ret_universal_ew = mean(curr_ret_universal_ew);

% St. dev.
```

```
port_std_universal_ns = std(port_ret_universal_ns);
port_std_universal = std(port_ret_universal);
port_std_universal_ew = std(port_ret_universal_ew);

curr_std_universal_ns = std(curr_ret_universal_ns);
curr_std_universal = std(curr_ret_universal);
curr_std_universal_ew = std(curr_ret_universal_ew);

% SR
SR_ptf_universal_ns = m_port_ret_universal_ns/port_std_universal_ns;
SR_ptf_universal = m_port_ret_universal/port_std_universal;
SR_ptf_universal_ew = m_port_ret_universal_ew/port_std_universal_ew;

SR_curr_universal_ns = m_curr_ret_universal_ns/curr_std_universal_ns;
SR_curr_universal = m_curr_ret_universal/curr_std_universal;
SR_curr_universal_ew = m_curr_ret_universal_ew/curr_std_universal_ew;

% Kurtosis
port_kurtosis_universal_ns = kurtosis(port_ret_universal_ns');
port_kurtosis_universal = kurtosis(port_ret_universal');
port_kurtosis_universal_ew = kurtosis(port_ret_universal_ew');

curr_kurtosis_universal_ns = kurtosis(curr_ret_universal_ns');
curr_kurtosis_universal = kurtosis(curr_ret_universal');
curr_kurtosis_universal_ew = kurtosis(curr_ret_universal_ew');

% Skewness
port_skewness_universal_ns = skewness(port_ret_universal_ns');
port_skewness_universal = skewness(port_ret_universal');
port_skewness_universal_ew = skewness(port_ret_universal_ew');

curr_skewness_universal_ns = skewness(curr_ret_universal_ns');
curr_skewness_universal = skewness(curr_ret_universal');
curr_skewness_universal_ew = skewness(curr_ret_universal_ew');
```

No Lasso: Currency hedging position

```
[W_currency_nL_ns, curr_ret_nL_ns] = hedge_no_Lasso(data, W_eq_ns); %
 Constrained optimization
[W_currency_nL, curr_ret_nL]    = hedge_no_Lasso(data, W_eq); %
 Unconstrained optimization
[W_currency_nL_ew, curr_ret_nL_ew, det_curr_inv_nL_ew,
 trace_curr_eq_nL_ew]    = hedge_no_Lasso(data, W_eq_ew); % eq.
 weighted

% Final weights
W_nL_ns = [W_eq_ns', W_currency_nL_ns']';
W_nL = [W_eq', W_currency_nL']';
W_nL_ew = [W_eq_ew', W_currency_nL_ew']';

% weights sum
W_currency_nL_ns_s = sum(W_currency_nL_ns);
W_currency_nL_s = sum(W_currency_nL);
```

```
W_currency_nL_ew_s = sum(W_currency_nL_ew);

% Retruns
[port_ret_nL_ns, port_ret_nL_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_nL_ns);
[port_ret_nL, port_ret_nL_cum] = ptf_return(eq_ret, curr_ret_nL);
[port_ret_nL_ew, port_ret_nL_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_nL_ew);

% avg
m_port_ret_nL_ns = mean(port_ret_nL_ns);
m_port_ret_nL = mean(port_ret_nL);
m_port_ret_nL_ew = mean(port_ret_nL_ew);

m_curr_ret_nL_ns = mean(curr_ret_nL_ns);
m_curr_ret_nL = mean(curr_ret_nL);
m_curr_ret_nL_ew = mean(curr_ret_nL_ew);

% St. dev.
port_std_nL_ns = std(port_ret_nL_ns);
port_std_nL = std(port_ret_nL);
port_std_nL_ew = std(port_ret_nL_ew);

curr_std_nL_ns = std(curr_ret_nL_ns);
curr_std_nL = std(curr_ret_nL);
curr_std_nL_ew = std(curr_ret_nL_ew);

% SR
SR_ptf_nL_ns = m_port_ret_nL_ns/port_std_nL_ns;
SR_ptf_nL = m_port_ret_nL/port_std_nL;
SR_ptf_nL_ew = m_port_ret_nL_ew/port_std_nL_ew;

SR_curr_nL_ns = m_curr_ret_nL_ns/curr_std_nL_ns;
SR_curr_nL = m_curr_ret_nL/curr_std_nL;
SR_curr_nL_ew = m_curr_ret_nL_ew/curr_std_nL_ew;

% Kurtosis
port_kurtosis_nL_ns = kurtosis(port_ret_nL_ns');
port_kurtosis_nL = kurtosis(port_ret_nL');
port_kurtosis_nL_ew = kurtosis(port_ret_nL_ew');

curr_kurtosis_nL_ns = kurtosis(curr_ret_nL_ns');
curr_kurtosis_nL = kurtosis(curr_ret_nL');
curr_kurtosis_nL_ew = kurtosis(curr_ret_nL_ew');

% Skewness
port_skewness_nL_ns = skewness(port_ret_nL_ns');
port_skewness_nL = skewness(port_ret_nL');
port_skewness_nL_ew = skewness(port_ret_nL_ew');

curr_skewness_nL_ns = skewness(curr_ret_nL_ns');
curr_skewness_nL = skewness(curr_ret_nL');
curr_skewness_nL_ew = skewness(curr_ret_nL_ew');
```

Lasso: Currency hedging position

```
[W_currency_L_ns, curr_ret_L_ns] = hedge_Lasso(data, W_eq_ns,
 1, 'sqp'); % Constrained optimization
[W_currency_L, curr_ret_L]    = hedge_Lasso(data, W_eq, 1, 'sqp'); %
 Unconstrained optimization
[W_currency_L_ew, curr_ret_L_ew, Sigma, estimator, p, bic,
 det_curr_inv_L_ew, trace_curr_eq_L_ew]    = hedge_Lasso(data,
 W_eq_ew, 1, 'sqp');

% Final weights
W_L_ns = [W_eq_ns', W_currency_L_ns'];
W_L = [W_eq', W_currency_L']';
W_L_ew = [W_eq_ew', W_currency_L_ew']';

% weights sum
W_currency_L_ns_s = sum(W_currency_L_ns);
W_currency_L_s = sum(W_currency_L);
W_currency_L_ew_s = sum(W_currency_L_ew);

% Retruns
[port_ret_L_ns, port_ret_L_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_L_ns);
[port_ret_L, port_ret_L_cum] = ptf_return(eq_ret, curr_ret_L);
[port_ret_L_ew, port_ret_L_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_L_ew);

% avg
m_port_ret_L_ns = mean(port_ret_L_ns);
m_port_ret_L = mean(port_ret_L);
m_port_ret_L_ew = mean(port_ret_L_ew);

m_curr_ret_L_ns = mean(curr_ret_L_ns);
m_curr_ret_L = mean(curr_ret_L);
m_curr_ret_L_ew = mean(curr_ret_L_ew);

% St. dev.
port_std_L_ns = std(port_ret_L_ns);
port_std_L = std(port_ret_L);
port_std_L_ew = std(port_ret_L_ew);

curr_std_L_ns = std(curr_ret_L_ns);
curr_std_L = std(curr_ret_L);
curr_std_L_ew = std(curr_ret_L_ew);

% SR
SR_ptf_L_ns = m_port_ret_L_ns/port_std_L_ns;
SR_ptf_L = m_port_ret_L/port_std_L;
SR_ptf_L_ew = m_port_ret_L_ew/port_std_L_ew;

SR_curr_L_ns = m_curr_ret_L_ns/curr_std_L_ns;
SR_curr_L = m_curr_ret_L/curr_std_L;
SR_curr_L_ew = m_curr_ret_L_ew/curr_std_L_ew;
```

```
% Kurtosis
port_kurtosis_L_ns = kurtosis(port_ret_L_ns');
port_kurtosis_L = kurtosis(port_ret_L');
port_kurtosis_L_ew = kurtosis(port_ret_L_ew');

curr_kurtosis_L_ns = kurtosis(curr_ret_L_ns');
curr_kurtosis_L = kurtosis(curr_ret_L');
curr_kurtosis_L_ew = kurtosis(curr_ret_L_ew');

% Skewness
port_skewness_L_ns = skewness(port_ret_L_ns');
port_skewness_L = skewness(port_ret_L');
port_skewness_L_ew = skewness(port_ret_L_ew');

curr_skewness_L_ns = skewness(curr_ret_L_ns');
curr_skewness_L = skewness(curr_ret_L');
curr_skewness_L_ew = skewness(curr_ret_L_ew');
```

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 4.900000e+03.

gLasso: Currency hedging position

```
[W_currency_gl_ns, curr_ret_gl_ns] = hedge_gLasso(data, W_eq_ns); %
 Constrained optimization
[W_currency_gl, curr_ret_gl]    = hedge_gLasso(data, W_eq); %
 Unconstrained optimization
[W_currency_gl_ew, curr_ret_gl_ew, Sigma_inv_gl, estimator_gl, S_gl,
 det_curr_inv_gl_ew, trace_curr_eq_gl_ew]    = hedge_gLasso(data,
 W_eq_ew);

% Final weights
W_gl_ns = [W_eq_ns', W_currency_gl_ns'];
W_gl = [W_eq', W_currency_gl']';
W_gl_ew = [W_eq_ew', W_currency_gl_ew']';

% Retruns
[port_ret_gl_ns, port_ret_gl_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_gl_ns);
[port_ret_gl, port_ret_gl_cum] = ptf_return(eq_ret, curr_ret_gl);
[port_ret_gl_ew, port_ret_gl_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_gl_ew);

% avg
m_port_ret_gl_ns = mean(port_ret_gl_ns);
m_port_ret_gl = mean(port_ret_gl);
m_port_ret_gl_ew = mean(port_ret_gl_ew);

m_curr_ret_gl_ns = mean(curr_ret_gl_ns);
m_curr_ret_gl = mean(curr_ret_gl);
m_curr_ret_gl_ew = mean(curr_ret_gl_ew);

% St. dev.
port_std_gl_ns = std(port_ret_gl_ns);
port_std_gl = std(port_ret_gl);
port_std_gl_ew = std(port_ret_gl_ew);

curr_std_gl_ns = std(curr_ret_gl_ns);
curr_std_gl = std(curr_ret_gl);
curr_std_gl_ew = std(curr_ret_gl_ew);

% SR
SR_ptf_gl_ns = m_port_ret_gl_ns/port_std_gl_ns;
SR_ptf_gl = m_port_ret_gl/port_std_gl;
SR_ptf_gl_ew = m_port_ret_gl_ew/port_std_gl_ew;

SR_curr_gl_ns = m_curr_ret_gl_ns/curr_std_gl_ns;
SR_curr_gl = m_curr_ret_gl/curr_std_gl;
SR_curr_gl_ew = m_curr_ret_gl_ew/curr_std_gl_ew;

% Kurtosis
port_kurtosis_gl_ns = kurtosis(port_ret_gl_ns');
```

```
port_kurtosis_gl = kurtosis(port_ret_gl');
port_kurtosis_gl_ew = kurtosis(port_ret_gl_ew');

curr_kurtosis_gl_ns = kurtosis(curr_ret_gl_ns');
curr_kurtosis_gl = kurtosis(curr_ret_gl');
curr_kurtosis_gl_ew = kurtosis(curr_ret_gl_ew');

% Skewness
port_skewness_gl_ns = skewness(port_ret_gl_ns');
port_skewness_gl = skewness(port_ret_gl');
port_skewness_gl_ew = skewness(port_ret_gl_ew');

curr_skewness_gl_ns = skewness(curr_ret_gl_ns');
curr_skewness_gl = skewness(curr_ret_gl');
curr_skewness_gl_ew = skewness(curr_ret_gl_ew');
```

Ledoit: Currency hedging position

```
[W_currency_ledoit_ns, curr_ret_ledoit_ns]  = hedge_Ledoit(data,
 W_eq_ns);
[W_currency_ledoit, curr_ret_ledoit]  = hedge_Ledoit(data, W_eq);
[W_currency_ledoit_ew, curr_ret_ledoit_ew, Sigma_ledoit,
 det_curr_inv_ledoit_ew, trace_curr_eq_ledoit_ew]  =
 hedge_Ledoit(data, W_eq_ew);

% Final weights
W_ledoit_ns = [W_eq_ns', W_currency_ledoit_ns'];
W_ledoit = [W_eq', W_currency_ledoit']';
W_ledoit_ew = [W_eq_ew', W_currency_ledoit_ew']';

% Retruns
[port_ret_ledoit_ns, port_ret_ledoit_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_ledoit_ns);
[port_ret_ledoit, port_ret_ledoit_cum] = ptf_return(eq_ret,
 curr_ret_ledoit);
[port_ret_ledoit_ew, port_ret_ledoit_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_ledoit_ew);

% avg
m_port_ret_ledoit_ns = mean(port_ret_ledoit_ns);
m_port_ret_ledoit = mean(port_ret_ledoit);
m_port_ret_ledoit_ew = mean(port_ret_ledoit_ew);

m_curr_ret_ledoit_ns = mean(curr_ret_ledoit_ns);
m_curr_ret_ledoit = mean(curr_ret_ledoit);
m_curr_ret_ledoit_ew = mean(curr_ret_ledoit_ew);

% St. dev.
port_std_ledoit_ns = std(port_ret_ledoit_ns);
port_std_ledoit = std(port_ret_ledoit);
port_std_ledoit_ew = std(port_ret_ledoit_ew);

curr_std_ledoit_ns = std(curr_ret_ledoit_ns);
```

```
curr_std_ledoit = std(curr_ret_ledoit);
curr_std_ledoit_ew = std(curr_ret_ledoit_ew);

% SR
SR_ptf_ledoit_ns = m_port_ret_ledoit_ns/port_std_ledoit_ns;
SR_ptf_ledoit = m_port_ret_ledoit/port_std_ledoit;
SR_ptf_ledoit_ew = m_port_ret_ledoit_ew/port_std_ledoit_ew;

SR_curr_ledoit_ns = m_curr_ret_ledoit_ns/curr_std_ledoit_ns;
SR_curr_ledoit = m_curr_ret_ledoit/curr_std_ledoit;
SR_curr_ledoit_ew = m_curr_ret_ledoit_ew/curr_std_ledoit_ew;

% Kurtosis
port_kurtosis_ledoit_ns = kurtosis(port_ret_ledoit_ns');
port_kurtosis_ledoit = kurtosis(port_ret_ledoit');
port_kurtosis_ledoit_ew = kurtosis(port_ret_ledoit_ew');

curr_kurtosis_ledoit_ns = kurtosis(curr_ret_ledoit_ns');
curr_kurtosis_ledoit = kurtosis(curr_ret_ledoit');
curr_kurtosis_ledoit_ew = kurtosis(curr_ret_ledoit_ew');

% Skewness
port_skewness_ledoit_ns = skewness(port_ret_ledoit_ns');
port_skewness_ledoit = skewness(port_ret_ledoit');
port_skewness_ledoit_ew = skewness(port_ret_ledoit_ew');

curr_skewness_ledoit_ns = skewness(curr_ret_ledoit_ns');
curr_skewness_ledoit = skewness(curr_ret_ledoit');
curr_skewness_ledoit_ew = skewness(curr_ret_ledoit_ew');

% %% Glasso cov: Currency hedging position
% [W_currency_glcov_ns, curr_ret_glcov_ns]  = hedge_glcov(data,
 W_eq_ns);
% [W_currency_glcov, curr_ret_glcov]  = hedge_glcov(data, W_eq);
% [W_currency_glcov_ew, curr_ret_glcov_ew, Sigma_glcov]  =
 hedge_glcov(data, W_eq_ew);
%
% % Final weights
% W_glcov_ns = [W_eq_ns', W_currency_glcov_ns'];
% W_glcov = [W_eq', W_currency_glcov']';
% W_glcov_ew = [W_eq_ew', W_currency_glcov_ew']';
%
% % Retruns
% [port_ret_glcov_ns, port_ret_glcov_ns_cum] = ptf_return(eq_ret_ns,
 curr_ret_glcov_ns);
% [port_ret_glcov, port_ret_glcov_cum] = ptf_return(eq_ret,
 curr_ret_glcov);
% [port_ret_glcov_ew, port_ret_glcov_ew_cum] = ptf_return(eq_ret_ew,
 curr_ret_glcov_ew);
%
% % avg
% m_port_ret_glcov_ns = mean(port_ret_glcov_ns);
% m_port_ret_glcov = mean(port_ret_glcov);
% m_port_ret_glcov_ew = mean(port_ret_glcov_ew);
```

```
%
% m_curr_ret_glcov_ns = mean(curr_ret_glcov_ns);
% m_curr_ret_glcov = mean(curr_ret_glcov);
% m_curr_ret_glcov_ew = mean(curr_ret_glcov_ew);
%
% % St. dev.
% port_std_glcov_ns = std(port_ret_glcov_ns);
% port_std_glcov = std(port_ret_glcov);
% port_std_glcov_ew = std(port_ret_glcov_ew);
%
% curr_std_glcov_ns = std(curr_ret_glcov_ns);
% curr_std_glcov = std(curr_ret_glcov);
% curr_std_glcov_ew = std(curr_ret_glcov_ew);
%
% % SR
% SR_ptf_glcov_ns = m_port_ret_glcov_ns/port_std_glcov_ns;
% SR_ptf_glcov = m_port_ret_glcov/port_std_glcov;
% SR_ptf_glcov_ew = m_port_ret_glcov_ew/port_std_glcov_ew;
%
% SR_curr_glcov_ns = m_curr_ret_glcov_ns/curr_std_glcov_ns;
% SR_curr_glcov = m_curr_ret_glcov/curr_std_glcov;
% SR_curr_glcov_ew = m_curr_ret_glcov_ew/curr_std_glcov_ew;
%
% % Kurtosis
% port_kurtosis_glcov_ns = kurtosis(port_ret_glcov_ns');
% port_kurtosis_glcov = kurtosis(port_ret_glcov');
% port_kurtosis_glcov_ew = kurtosis(port_ret_glcov_ew');
%
% curr_kurtosis_glcov_ns = kurtosis(curr_ret_glcov_ns');
% curr_kurtosis_glcov = kurtosis(curr_ret_glcov');
% curr_kurtosis_glcov_ew = kurtosis(curr_ret_glcov_ew');
%
% % Skewness
% port_skewness_glcov_ns = skewness(port_ret_glcov_ns');
% port_skewness_glcov = skewness(port_ret_glcov');
% port_skewness_glcov_ew = skewness(port_ret_glcov_ew');
%
% curr_skewness_glcov_ns = skewness(curr_ret_glcov_ns');
% curr_skewness_glcov = skewness(curr_ret_glcov');
% curr_skewness_glcov_ew = skewness(curr_ret_glcov_ew');
```

Results

Currency

```
curr_ret_stat = array2table([[m_curr_ret_nL_ew m_curr_ret_gh_ew
 m_curr_ret_L_ew m_curr_ret_gl_ew m_curr_ret_ledoit_ew
 m_curr_ret_universal_ew]; ...
    [curr_std_nL_ew curr_std_gh_ew curr_std_L_ew curr_std_gl_ew
 curr_std_ledoit_ew curr_std_universal_ew]; ...
    [SR_curr_nL_ew SR_curr_gh_ew SR_curr_L_ew SR_curr_gl_ew
 SR_curr_ledoit_ew SR_curr_universal_ew]; ...
```

```
    [curr_skewness_nL_ew curr_skewness_gh_ew curr_skewness_L_ew
 curr_skewness_gl_ew curr_skewness_ledoit_ew
 curr_skewness_universal_ew]; ...
    [curr_kurtosis_nL_ew curr_kurtosis_gh_ew
 curr_kurtosis_L_ew curr_kurtosis_gl_ew curr_kurtosis_ledoit_ew
 curr_kurtosis_universal_ew]], ...
    'RowNames', {'Return', 'Standard deviation', 'Sharpe
 ratio', 'Skewness', 'Kurtosis'}, ...
    'VariableNames', {'Optimal Hedge', 'Unitary
 hedge', 'SQP', 'Glasso', 'Ledoit-Wolf', 'Universal hedge'});
writetable(curr_ret_stat,'/Users/sd/Desktop/Study/master/data/export/
curr_ret.xlsx','WriteRowNames',true)

% PTF
port_ret_stat = array2table([[m_eq_ret_ew m_port_ret_nL_ew
 m_port_ret_gh_ew m_port_ret_L_ew m_port_ret_gl_ew
 m_port_ret_ledoit_ew m_port_ret_universal_ew]; ...
    [eq_std_nL_ew port_std_nL_ew port_std_gh_ew port_std_L_ew
 port_std_gl_ew port_std_ledoit_ew port_std_universal_ew]; ...
    [SR_eq_ew SR_ptf_nL_ew SR_ptf_gh_ew SR_ptf_L_ew SR_ptf_gl_ew
 SR_ptf_ledoit_ew SR_ptf_universal_ew]; ...
    [eq_skewness_nL_ew port_skewness_nL_ew port_skewness_gh_ew
 port_skewness_L_ew port_skewness_gl_ew port_skewness_ledoit_ew
 port_skewness_universal_ew]; ...
    [eq_kurtosis_nL_ew port_kurtosis_nL_ew port_kurtosis_gh_ew
 port_kurtosis_L_ew port_kurtosis_gl_ew port_kurtosis_ledoit_ew
 port_kurtosis_universal_ew]], ...
    'RowNames', {'Return', 'Standard deviation', 'Sharpe
 ratio', 'Skewness', 'Kurtosis'}, ...
    'VariableNames', {'Equity', 'Optimal Hedge', 'Unitary
 hedge', 'SQP', 'Glasso', 'Ledoit-Wolf', 'Universal hedge'});
writetable(port_ret_stat,'/Users/sd/Desktop/Study/master/data/export/
port_ret.xlsx','WriteRowNames',true)
```

Det and Trace

```
% Det
Mu_det = [mean(det_curr_inv_nL_ew) mean(det_curr_inv_L_ew)
 mean(det_curr_inv_gl_ew) mean(det_curr_inv_ledoit_ew)];
s_dev_det = [std(det_curr_inv_nL_ew) std(det_curr_inv_L_ew)
 std(det_curr_inv_gl_ew) std(det_curr_inv_ledoit_ew)];
median_det = [median(det_curr_inv_nL_ew) median(det_curr_inv_L_ew)
 median(det_curr_inv_gl_ew) median(det_curr_inv_ledoit_ew)];
min_det = [min(det_curr_inv_nL_ew) min(det_curr_inv_L_ew)
 min(det_curr_inv_gl_ew) min(det_curr_inv_ledoit_ew)];
max_det = [max(det_curr_inv_nL_ew) max(det_curr_inv_L_ew)
 max(det_curr_inv_gl_ew) max(det_curr_inv_ledoit_ew)];

res_det = array2table([Mu_det' s_dev_det' median_det' min_det'
 max_det'], 'RowNames', data_all.Properties.VariableNames, ...
    'RowNames', {'Optimal Hedge', 'SQP', 'Glasso', 'Ledoit-Wolf'}, ...
    'VariableNames', {'mean', 'st. dev.', 'median', 'min','max'});
```

```
writetable(res_det,'/Users/sd/Desktop/Study/master/data/export/
d_stat_det.xlsx','WriteRowNames',true)

% trace
Mu_trace = [mean(trace_curr_eq_nL_ew) mean(trace_curr_eq_L_ew)
 mean(trace_curr_eq_gl_ew) mean(trace_curr_eq_ledoit_ew)];
s_dev_trace = [std(trace_curr_eq_nL_ew) std(trace_curr_eq_L_ew)
 std(trace_curr_eq_gl_ew) std(trace_curr_eq_ledoit_ew)];
median_trace = [median(trace_curr_eq_nL_ew) median(trace_curr_eq_L_ew)
 median(trace_curr_eq_gl_ew) median(trace_curr_eq_ledoit_ew)];
min_trace = [min(trace_curr_eq_nL_ew) min(trace_curr_eq_L_ew)
 min(trace_curr_eq_gl_ew) min(trace_curr_eq_ledoit_ew)];
max_trace = [max(trace_curr_eq_nL_ew) max(trace_curr_eq_L_ew)
 max(trace_curr_eq_gl_ew) max(trace_curr_eq_ledoit_ew)];

res_trace = array2table([Mu_trace' s_dev_trace'
 median_trace' min_trace' max_trace'], 'RowNames',
 data_all.Properties.VariableNames, ...
    'RowNames', {'Optimal Hedge', 'SQP', 'Glasso', 'Ledoit-Wolf'}, ...
    'VariableNames', {'mean', 'st. dev.', 'median', 'min','max'});
writetable(res_trace,'/Users/sd/Desktop/Study/master/data/export/
d_stat_trace.xlsx','WriteRowNames',true)
```

Functions

```
function [port_ret, port_ret_cum] = ptf_return(eq_ret, curr_ret)

    port_ret  = eq_ret + curr_ret;
    port_ret_cum = 1;
    for j = 1:size(port_ret, 2)
        port_ret_cum = port_ret_cum*(1+port_ret(j));
    end
    port_ret_cum;

end

function [W_currency, curr_ret, det_curr_inv, trace_curr_eq] =
 hedge_no_Lasso(data, W_equity)

    T = 300; % choose 10 years to build initial model
    i = 1; % count

    while T < size(data, 1) % do not need for the last month
        Sigma         = cov(data(1:T,:)); % Covariance matrix
        mv_hedge      = inv(Sigma(13:end, 13:end))*Sigma(13:end,
 1:12); % Min Var currency hedges for equity indices
        W_currency(:,i) =  -mv_hedge*W_equity(:,i);

        curr_ret(:,i)      = data(T + 1,13:end)*W_currency(:,i);

        det_curr_inv(:,i)      = det(inv(Sigma(13:end, 13:end)));
        trace_curr_eq(:,i)      = trace(Sigma(13:end,
 13:end)*inv(Sigma(13:end, 13:end)));
```

```
        T = T + 1; % rebalancing each month
        i = i + 1;
    end

end

function [W_currency, curr_ret, Sigma, estimator, p, bic,
 det_curr_inv, trace_curr_eq] = hedge_Lasso(data, W_equity, model,
 method)

    T = 300; % choose 10 years to build initial model
    i = 1; % count

    while T < size(data, 1) % do not need for the last month
        Sigma = cov(data(1:T,:)); % Covariance matrix

        % penalization search
        if T == 300
            j = 1;
            step = 0.1;
            for pv = 0.1:step:2
                x_tmp = inv(Sigma(13:end, 13:end));

                if model == 1
                    fun_min = @(x) -(T/2 * log(det(x)) -
 T/2 * trace(Sigma(13:end, 13:end)*x) - pv*sum(sum(abs(x -
 diag(diag(x)))))); % 3: sum of estimator's elements w/o diagonal(i !
= j)
                elseif model == 2
                    fun_min = @(x) -(log(det(x)) - trace(Sigma(13:end,
 13:end)*x) - pv*sum(sum(abs(x - diag(diag(x)))))); % 3: sum of
 estimator's elements w/o diagonal(i != j)
                else
                    fun_min = @(x) -(T/2 * log(det(x)) -
 T/2 * trace(Sigma(13:end, 13:end)*x) + pv*sum(sum(abs(x -
 diag(diag(x)))))); % 3: sum of estimator's elements w/o diagonal(i !
= j)
                end

                if method == 'sqp'
                    OPTIONS =
 optimoptions('fmincon','Algorithm','sqp');
                    estimator = fmincon(fun_min, x_tmp, [],[],[],[],
[],[],[], OPTIONS);
%                     estimator = BFGS(fun_min, x_tmp,20, 0.0001,
 0,1 ,20)
                elseif method == 'active-set'
                    OPTIONS =
 optimoptions('fmincon','Algorithm','active-set');
                    estimator = fmincon(fun_min, x_tmp, [],[],[],[],
[],[],[], OPTIONS);
                else
```

```
                    OPTIONS =
 optimoptions('fmincon','Algorithm','sqp');
                    estimator = fmincon(fun_min, x_tmp, [],[],[],[],
[],[],[], OPTIONS);
                end

                [aic(j), bic(j)] = aicbic(-fun_min(estimator), 1, T);
                j = j + 1;
            end
        end

        [min_BIC, min_BIC_indx] = min(bic);
        p = step*min_BIC_indx; % regularization parameter
%         p = 0.001;

        x_tmp = inv(Sigma(13:end, 13:end));

        % Negate the value because -f(x) was the function worked upon
 (min --> max)

        % inverse cov
        if model == 1
            fun_min = @(x) -(T/2 * log(det(x)) - T/2
 * trace(Sigma(13:end, 13:end)*x) - p*sum(sum(abs(x -
 diag(diag(x))))); % 3: sum of estimator's elements w/o diagonal(i !
= j)
        elseif model == 2
            fun_min = @(x) -(log(det(x)) - trace(Sigma(13:end,
 13:end)*x) - p*sum(sum(abs(x - diag(diag(x)))))); % 3: sum of
 estimator's elements w/o diagonal(i != j)
        else
            fun_min = @(x) -(T/2 * log(det(x)) - T/2
 * trace(Sigma(13:end, 13:end)*x) - p*sum(sum(abs(x -
 diag(diag(x))))); % 3: sum of estimator's elements w/o diagonal(i !
= j)
        end

        if method == 'sqp'
            OPTIONS = optimoptions('fmincon','Algorithm','sqp');
            estimator = fmincon(fun_min, x_tmp, [],[],[],[],[],[],[],
 OPTIONS);
        elseif method == 'active-set'
            OPTIONS = optimoptions('fmincon','Algorithm','active-
set');
            estimator = fmincon(fun_min, x_tmp, [],[],[],[],[],[],[],
 OPTIONS);
        else
            OPTIONS = optimoptions('fmincon','Algorithm','sqp');
            estimator = fmincon(fun_min, x_tmp, [],[],[],[],[],[],[],
 OPTIONS);
        end
```

```
        mv_hedge = estimator*Sigma(13:end, 1:12); % Min Var currency
 hedges for equity indices
        W_currency(:,i) = - mv_hedge*W_equity(:,i);

        curr_ret(:,i)   = data(T + 1,13:end)*W_currency(:,i);

        det_curr_inv(:,i)      = det(estimator);
        trace_curr_eq(:,i)      = trace(Sigma(13:end,
 13:end)*estimator);

        T = T + 1; % rebalancing each month
        i = i + 1;

    end

end

function [W_currency, curr_ret, Sigma_inv, estimator, S, det_curr_inv,
 trace_curr_eq] = hedge_gLasso(data, W_equity)

    T = 300; % choose 10 years to build initial model
    i = 1; % count

    while T < size(data, 1) % do not need for the last month
        Sigma = cov(data(1:T,:)); % Covariance matrix
        Sigma_inv = inv(Sigma(13:end, 13:end));
%         p = 0.00017;
        p = 0.0001;

        [estimator S] = graphicalLasso(Sigma(13:end, 13:end), p);

        mv_hedge = estimator*Sigma(13:end, 1:12); % Min Var currency
 hedges for equity indices
        W_currency(:,i) = - mv_hedge*W_equity(:,i);

        curr_ret(:,i)   = data(T + 1,13:end)*W_currency(:,i);

        det_curr_inv(:,i)      = det(estimator);
        trace_curr_eq(:,i)     = trace(Sigma(13:end,
 13:end)*estimator);

        T = T + 1; % rebalancing each month
        i = i + 1;

    end

end

function [W_currency, curr_ret, Sigma, det_curr_inv, trace_curr_eq] =
 hedge_Ledoit(data, W_equity)

    T = 300; % choose 10 years to build initial model
    i = 1; % count
```

```
    while T < size(data, 1) % do not need for the last month
        Sigma         = covDiag(data(1:T,:)); % Covariance matrix
        mv_hedge      = inv(Sigma(13:end, 13:end))*Sigma(13:end,
 1:12); % Min Var currency hedges for equity indices
        W_currency(:,i) =  -mv_hedge*W_equity(:,i);

        curr_ret(:,i)      = data(T + 1,13:end)*W_currency(:,i);

        det_curr_inv(:,i)      = det(inv(Sigma(13:end, 13:end)));
        trace_curr_eq(:,i)      = trace(Sigma(13:end,
 13:end)*inv(Sigma(13:end, 13:end)));

        T = T + 1; % rebalancing each month
        i = i + 1;
    end

end

function [W_currency, curr_ret, Sigma] = hedge_glcov(data, W_equity)

    T = 300; % choose 10 years to build initial model
    i = 1; % count

    while T < size(data, 1) % do not need for the last month
        Sigma = cov(data(1:T,:)); % Covariance matrix
        p = 0.0001;

        [e S] = graphicalLasso(Sigma, p);
        mv_hedge      = inv(S(13:end, 13:end))*S(13:end, 1:12); % Min
 Var currency hedges for equity indices
        W_currency(:,i) =  -mv_hedge*W_equity(:,i);

        curr_ret(:,i)      = data(T + 1,13:end)*W_currency(:,i);

        T = T + 1; % rebalancing each month
        i = i + 1;
    end

end

function [W_eq, W_eq_ns, eq_ret, eq_ret_ns] = equity_weights(data)

    T = 300; % choose 10 years to build initial model
    i = 1; % count

    eq_ret = zeros(1, size(data, 1) - T);
    eq_ret_ns = zeros(1, size(data, 1) - T);
    W_eq = zeros(12, size(data, 1) - T);
    W_eq_ns = zeros(12, size(data, 1) - T);

    while T < size(data, 1) % do not need for the last month
        Sigma        = cov(data(1:T, :)); % Covariance matrix

        % Unconstrained optimization
```

```
        W_eq(:,i)   = port_minvar(Sigma(1:12, 1:12)); % for index with
 shorting
        eq_ret(:,i) = data(T + 1,1:12)*W_eq(:,i);

        % Constrained optimization
        W_eq_ns(:,i)   = port_minvar(Sigma(1:12, 1:12), 0); % for
 index with no shorting
        eq_ret_ns(:,i) = data(T + 1,1:12)*W_eq_ns(:,i);

        T = T + 1; % rebalancing each month
        i = i + 1;
    end

end

function [W] = port_minvar(Sigma, lb, ub)
    % [W, port_std, port_ret] = port_minvar(Mu, Sigma, lb, ub)
    %
    %   Minimum-variance portfolio optimization
    %        Sigma: Covariance matrix (n x n)
    %
    %   The following are optional inputs. Can set to [].
    %           lb: Lower bound for portfolio weights (scalar or
 vector).
    %           ub: Upper bound for portfolio weights (scalar or
 vector).
    %
    %   Output:
    %            W: Portfolio weights (n x 1)
    %     port_std: The Portfolio standard deviation

    %inputs
    n = size(Sigma,1);
    I = ones(n,1);         %vector of ones

    if nargin < 2 % was 3
        %Unconstrained optimization
        %=========================
        a = I' * (Sigma\I);
        W = (Sigma \ I)/a;
    else
        %Constrained optimization
        %=======================
        %vector of ones and initial weight vector
        Z = I/n;

        %set upper and lower portfolio weight bounds
        if isempty(lb)
            lb = [];
        else
            if length(lb)==1
                lb = lb*I;
            end
        end
```

```
        if nargin < 4
            ub =  [];
        else
            if length(ub)==1
                ub = I*ub;
            end
        end

        %optimization options
        options =
 optimoptions('fmincon','Display','notify','StepTolerance',1.0e-8,'FunctionToleran
            'ConstraintTolerance',1.0e-8);

        %Constraints: Weights sum to one, and weight bounds.
        W = fmincon(@(x) (x' * Sigma * x), Z, [], [], I', 1, lb, ub,
 [], options);
    end

    % %Portfolio standard deviation and mean
    % port_std = sqrt(W' * Sigma * W);
    % port_ret = sum(Mu'.*W);

end

% Graphical Lasso function
% Author: Xiaohui Chen (xiaohuic@ece.ubc.ca)
% Version: 2012-Feb

function [Theta W] = graphicalLasso(S, rho, maxIt, tol)

% Solve the graphical Lasso
% minimize_{Theta > 0} tr(S*Theta) - logdet(Theta) + rho * ||Theta||_1
% Ref: Friedman et al. (2007) Sparse inverse covariance estimation
 with the
% graphical lasso. Biostatistics.
% Note: This function needs to call an algorithm that solves the Lasso
% problem. Here, we choose to use to the function *lassoShooting*
 (shooting
% algorithm) for this purpose. However, any Lasso algorithm in the
% penelized form will work.
%
% Input:
% S -- sample covariance matrix
% rho --  regularization parameter
% maxIt -- maximum number of iterations
% tol -- convergence tolerance level
%
% Output:
% Theta -- inverse covariance matrix estimate
% W -- regularized covariance matrix estimate, W = Theta^-1

    p = size(S,1);

    if nargin < 4, tol = 1e-6; end
```

```
    if nargin < 3, maxIt = 1e2; end

    % Initialization
%     W = S + rho * eye(p);   % diagonal of W remains unchanged
    W = S;
    W_old = W;
    i = 0;

    % Graphical Lasso loop
    while i < maxIt
        i = i+1;
        for j = p:-1:1
            jminus = setdiff(1:p,j);
            [V D] = eig(W(jminus,jminus));
            d = diag(D);
            X = V * diag(sqrt(d)) * V'; % W_11^(1/2)
            Y = V * diag(1./sqrt(d)) * V' * S(jminus,j);    %
 W_11^(-1/2) * s_12
            b = lassoShooting(X, Y, rho, maxIt, tol);
            W(jminus,j) = W(jminus,jminus) * b;
            W(j,jminus) = W(jminus,j)';
        end
        % Stop criterion
        if norm(W-W_old,1) < tol
            break;
        end
        W_old = W;
    end
    if i == maxIt
        fprintf('%s\n', 'Maximum number of iteration reached, glasso
 may not converge.');
    end

    Theta = W^-1;
end

% Shooting algorithm for Lasso (unstandardized version)
function b = lassoShooting(X, Y, lambda, maxIt, tol)

    if nargin < 4, tol = 1e-6; end
    if nargin < 3, maxIt = 1e2; end

    % Initialization
    [n, p] = size(X);
    if p > n
        b = zeros(p,1); % From the null model, if p > n
    else
        b = X \ Y;  % From the OLS estimate, if p <= n
    end
    b_old = b;
    i = 0;

    % Precompute X'X and X'Y
    XTX = X'*X;
```

```
    XTY = X'*Y;

    % Shooting loop
    while i < maxIt
        i = i+1;
        for j = 1:p
            jminus = setdiff(1:p,j);
            S0 = XTX(j,jminus)*b(jminus) - XTY(j);  % S0 =
 X(:,j)'*(X(:,jminus)*b(jminus)-Y)
            if S0 > lambda
                b(j) = (lambda-S0) / norm(X(:,j),2)^2;
            elseif S0 < -lambda
                b(j) = -(lambda+S0) / norm(X(:,j),2)^2;
            else
                b(j) = 0;
            end
        end
        delta = norm(b-b_old,1);    % Norm change during successive
 iterations
        if delta < tol, break; end
        b_old = b;
    end
    if i == maxIt
        fprintf('%s\n', 'Maximum number of iteration reached, shooting
 may not converge.');
    end
end

function sigmahat = covDiag(Y,k)

    % function sigmahat=covDiag(Y,k)
    %
    % Y (N*p): raw data matrix of N iid observations on p random
 variables
    % sigmahat (p*p): invertible covariance matrix estimator
    %
    % Preserves sample variances but shrinks all sample covariances
 towards
    %    zero, as in Appendix B.2 of Ledoit (1995, MIT PhD Thesis)
    %
    % If the second (optional) parameter k is absent, not-a-number, or
 empty,
    % then the algorithm demeans the data by default, and adjusts the
 effective
    % sample size accordingly. If the user inputs k = 0, then no
 demeaning
    % takes place; if (s)he inputs k = 1, then it signifies that the
 data Y has
    % already been demeaned.
    %
    % This version: 01/2021, based on the 04/2014 version

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
```

```
   % This file is released under the BSD 2-clause license.

   % Copyright (c) 2014-2021, Olivier Ledoit and Michael Wolf
   % All rights reserved.
   %
   % Redistribution and use in source and binary forms, with or
without
   % modification, are permitted provided that the following
conditions are
   % met:
   %
   % 1. Redistributions of source code must retain the above
copyright notice,
   % this list of conditions and the following disclaimer.
   %
   % 2. Redistributions in binary form must reproduce the above
copyright
   % notice, this list of conditions and the following disclaimer in
the
   % documentation and/or other materials provided with the
distribution.
   %
   % THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS
   % IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO,
   % THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
   % PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
OR
   % CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL,
   % EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO,
   % PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
OR
   % PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF
   % LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING
   % NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS
   % SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

   % de-mean returns if required
   [N,p]=size(Y);                      % sample size and matrix
dimension
   if (nargin<2)||isnan(k)||isempty(k) % default setting
      Y=Y-repmat(mean(Y),[N 1]);       % demean the raw data matrix
      k=1;                             % subtract one degree of
freedom
   end
```

```
    n=N-k;                                  % adjust effective sample size

    % compute sample covariance matrix
    sample=(Y'*Y)./n;

    % compute shrinkage target
    target=diag(diag(sample));

    % estimate the parameter that we call pi in Ledoit and Wolf (2003,
 JEF)
    Y2=Y.^2;
    sample2=(Y2'*Y2)./n; % sample covariance matrix of squared returns
    piMat=sample2-sample.^2;
    pihat=sum(sum(piMat));

    % estimate the parameter that we call gamma in Ledoit and Wolf
 (2003, JEF)
    gammahat=norm(sample-target,'fro')^2;

    % diagonal part of the parameter that we call rho
    rho_diag=sum(diag(piMat));

    % off-diagonal part of the parameter that we call rho
    rho_off=0;

    % compute shrinkage intensity
    rhohat=rho_diag+rho_off;
    kappahat=(pihat-rhohat)/gammahat;
    shrinkage=max(0,min(1,kappahat/n));

    % compute shrinkage estimator
    sigmahat=shrinkage*target+(1-shrinkage)*sample;
end
```

Published with MATLAB® R2020b