Matlab Code

Random text at the beginning hopefully it will work as normal `../"Capstone- MATLAB"/a.m`

```
% INFLATION_INTERVAL is 'monthly','quarterly' or 'annually'
% and is used to computer the regime history

% ------------------------------------------------------------------------
% SYMBOLFILE specifies which file to fetch symbols from. A list of the files
% available are given below

% -Symbols_SP.m has the full S&P500 Stocks. It is important to note that only some
% of these have aviable price history that dates back to 1974.
% -SecondSymbols has a small subset and is used for testing
% -ThirdSYmbols is an even smaller subset, having only 3 stocks.
% -Symbols_NYSE has a full list of all NYSE stocks. However at this time,
%  only a subset of these 3300+ stocks have local csv files avaiable for fetching
% -Symbols_NYSE2 has a list of the NYSE stocks that have local csv files
% -Symbols_NYSE_SP has a list of NYSE stocks AND S&P500 stocks. ALl of these
% are avaiable for fetching. It is the recommended file to use because it has
% the most data avaiable which is important for early years where most stocks
% dont have time series data
% ------------------------------------------------------------------------
% SYEAR/EYEAR is the start/end year of optimization for the regeims
% ------------------------------------------------------------------------
% NUM_OF_TIME_PERIODS is the number of divisions in the time interval
% note that the in sample period to get data is
% (syear-eyear)/num_of_time_periods.

% **ensure that syear-eyear is divisble by num_of_time_periods
% ------------------------------------------------------------------------
% NUM_OF_REOP is the number of times the portfolio is reoptimized.
% note that this is a specified amount and there is no submodel to determine
% whether or not to reoptimize because the transcation cost constraint
% prevents drastic changes in asset allocation
% ------------------------------------------------------------------------
% DESIRED_R IS THE minimum monthly return constraint. Converting this
% to an annual return is (1+desired_R)^12-1
% ------------------------------------------------------------------------
```

```
% DESIRED_TRANSACTION is the the user-specified fixed transaction cost


% inflation_interval='monthly';
% SymbolFile='''Symbols_NYSE_SP.m''';
% startyear=1990;
% endyear=2010;
% num_of_time_divisions=5;
% num_of_reoptimization_periods=2;
% minimum_return=0.005;
% transaction_cost=0.0001;


%  create_inflation_hedged_portfolio...
%       (inflation_interval,SymbolFile,startyear,endyear,
    num_of_time_divisions,...
%          num_of_reoptimization_periods, minimum_return, transaction_cost);

%Commment all the lines above and Uncomment the line below if you want to
% see all the workspace variables

run create_inflation_hedged_portfolio.m;
```

```



% function [] = create_inflation_hedged_portfolio...
%                (inflation_interval,SymbolFile,...
%                 syear,eyear,num_of_time_periods,num_of_reop, ...
%                 desired_R,desired_transaction)

tic

% Uncomment the lines below when making create_inflation_hedged_portfolio a
    non-function
inflation_interval='monthly';
SymbolFile='''Symbols_NYSE_SP.m''';
syear=1971;
eyear=1981;
num_of_time_periods=5;
num_of_reop=1;
desired_R=0.005;
desired_transaction=0.0001;




%
```

```
% STEP 1: RETRIEVING HISTORICAL INFLATION RATES & ALL STOCK DATA AVAILABLE
% NOTE: THIS CAN BE PREPROCESSED

%can be monthly, quarterly or annual
begcol=1;
[inf_file endcol] = return_inflation_file(inflation_interval);

diffyear=eyear-syear;
eachyear=diffyear/num_of_time_periods;
T_reb= (diffyear-eachyear)*12/(num_of_reop+1);

[MLEinf_data, inf_avg] =...
    fetch_regime_inflation_data(1950,syear,begcol,endcol,inf_file);

%THis gets a list of the Stock ticker symbols used as the pool for
% asset allocation along with their corresponding csv files
run(eval(SymbolFile));

%THis fetches a list of ALL the stock/inflation/riskfree data for all dates
% Note that this is used for the optimization model and NOT for creating
% the regime switching model

[month day year price fail_symbols success_symbols]=...
                                                  all_stock_data(SP500_symb_csv,
                                                      SP500_symb);

[infmonth infyear infprice ] = all_inflation_data('inflation_rate_1200.csv');
 [rfmonth rfyear rfprice] = all_riskfree_data('riskfreerate2.csv');
%
    ----------------------------------------------------------------------------------------------------------

% STEP 2: SOLVE FOR THE MLE PARAMETERS OF THE REGIME-SWITCHING MODEL
% AS WELL AS DETERMING THE REGIME THAT EACH PERIOD BELONGS TO
% (WHICH PRODUCES THE REGIME STATE GRAPH)

k=2; %DECLARE THE NUMBER OF REGIMES

[Spec_Out p11 p22 p12 p21 var1 var2 var3 ar1 ar2 ar3 c1 c2 c3]= ...
                                    RegimeSwitching_MLE(k,MLEinf_data);

timelength=length(MLEinf_data);

[whichregime, countregime] =regimecount(k,Spec_Out.smoothProb,timelength);

%Parameter 1 is the number of monthly time periods, 2 is the current inflation
     rate
curr_regime=whichregime(timelength);
% curr_regime=2;
curr_inf_rate=MLEinf_data(timelength);
disp('CHECKPOINT 1');

```

```
% ---------------------------------------------------------------------------------
% TEMP CODE FOR TESTING NEW INFLATION EXPECTED VARIANCE

markov_periods=10; %this is chosen number of periods for markov tree(in months)

[expected_inf, tnodes,tnodeval]=new_exp_inf2(markov_periods,curr_inf_rate,curr_regime...
                        ,c1,c2,ar1,ar2,p11,p12,p21,p22);
[expected_inf_var] = ...
    new_exp_infvar2(markov_periods,1,tnodeval,var1,var2,p11,p12,p21,p22);

% ---------------------------------------------------------------------------------
% STEP 4 : RETRIEVE THE ASSET AND MARKET PRICES FOR THE DESIRED TIME PERIODS

lcase_month='Jan'
ucase_month=upper(lcase_month);

[tcurrprices currpricenames2 market_prices num_assets2 catch_assets2 totalmonths] = ...
        SEC_fetch_stock_data...
        (lcase_month,eyear,lcase_month,syear,month,day,year,price,success_symbols);

%Use the fetch inflation data to compute the inflation rates and the riskfree rates
[inf_prices] = fetch_inflation_data2...
    (ucase_month,eyear,ucase_month,syear,infmonth,infyear,infprice);

[riskfree_prices] = fetch_inflation_data2...
    (ucase_month,eyear,ucase_month,syear,infmonth,infyear,rfprice);

disp('CHECKPOINT 3');


[beg_indices end_indices]=divide_interval(num_of_time_periods,totalmonths);

asset_prices=cell2mat(tcurrprices);

%Parameters are first number of time divisions, then total months


currassetprices=asset_prices(beg_indices(1):end_indices(1)+1,:);
currmarketprices=market_prices(beg_indices(1):end_indices(1)+1);
currinfprices=inf_prices(beg_indices(1):end_indices(1)+1);
currriskfreeprices=riskfree_prices(beg_indices(1):end_indices(1)+1);

```

```
% asset_prices=cell2mat(currprices2);
asset_prices_with_market=[currmarketprices currassetprices];
asset_prices_with_inf=[currinfprices currassetprices];

% ----------------------------------------------------------------------------------------------------
% STEP 5: SOLVE FOR THE MVO PARAMETERS, THE MUS, Q'S AND R'S FOR ALL ASSETS AND
% THE MARKET OVER THE SPECIFIED TIME PERIODS. ALSO SOLVE FOR THE CAPM BETAS OR
% EACH ASSET

[asset_mu,asset_Q,asset_r]= solve_mvo_params(currassetprices,1,size(currassetprices,1));

%Solve for only the market
[Market_mu,Market_Q,Market_r]= solve_mvo_params(currmarketprices,1,size(currmarketprices,1));


% Calculate relevant parameters for testing horizon



disp('CHECKPOINT 4');

% desired_R=0.002;
% desired_transaction=0.0001

[Inf_Beta R2_inf]   =solve_beta3(asset_prices_with_inf,1);
i_Inf_Beta=Inf_Beta;


currinfprices2=currinfprices(1:end-1);
xalloc=zeros(num_assets2,1);

[modelMVO_x(1,:) modelMVO_var MVO_adjret_diagQ nom_ret temp_Q] = ...
    main_MVO(currinfprices2,asset_r,expected_inf/100,expected_inf_var,...
                                    Inf_Beta',desired_R,desired_transaction,xalloc);

i_MVO_adjret_diagQ_InfBeta=MVO_adjret_diagQ;
i_nom_ret=nom_ret;
i_temp_Q=temp_Q;

model_portfolio_beta(1,1) = modelMVO_x(1,:)*Inf_Beta';
% [modelMVO_x modelMVO_var temp_Q adj_ret nom_ret] = main_MVO(currinfprices,asset_r,1.8/100,0.04,...
```

```
%                                                              Inf_Beta
    ',0.000002,0.0005,0.05,xalloc);

%
    ------------------------------------------------------------------------------------------------

futureassetprices=asset_prices(beg_indices(2):end_indices(num_of_time_periods)
    ,:);
futuremarketprices=market_prices(beg_indices(2):end_indices(
    num_of_time_periods));
futureinfprices=inf_prices(beg_indices(2):end_indices(num_of_time_periods));
futureriskfreeprices=riskfree_prices(beg_indices(2):end_indices(
    num_of_time_periods));

[future_asset_mu,future_asset_Q,future_asset_r]= ...
    solve_mvo_params(futureassetprices,1,size(futureassetprices,1));

%Solve for only the market
[future_Market_mu,future_Market_Q,future_Market_r]= solve_mvo_params(
    futuremarketprices,1,size(futuremarketprices,1));




[future_MK1_r] = future_asset_r(:,end);
[future_MK2_r] = future_asset_r(:,end-1);

[benchMVO_x(1,:) benchMVO] = ...
benchmark_MVO(asset_mu', asset_Q, desired_R,desired_transaction, xalloc);

MVO_portfolio_beta(1,1) = benchMVO_x(1,:)*Inf_Beta';

[cumul_benchMVO cumul_modelMVO cumul_SP cumul_MF1 cumul_MF2] = ...
    MVO_comparison(benchMVO_x(1,:)', modelMVO_x(1,:)',future_asset_r,
        future_Market_r,...
        future_MK1_r,future_MK2_r,syear+eachyear,eyear);

obj_vals = [modelMVO_var benchMVO_x(1,:)*temp_Q*benchMVO_x(1,:)'
            modelMVO_x(1,:)*asset_Q*modelMVO_x(1,:)' benchMVO];


%
    ------------------------------------------------------------------------------------------------

%Using our model version of the Sharpe ratio, Preface M

[P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu, asset_Q, temp_Q
    , ...
    modelMVO_x(1,:), benchMVO_x(1,:),modelMVO_var,currriskfreeprices);


```

```
if (num_of_reop >0)

    MVO_returns_reb(1:T_reb)= ...
        future_asset_r(1:T_reb,:)*benchMVO_x(1,:)';
    inf_returns_reb(1:T_reb) = ...
        future_asset_r(1:T_reb,:)*modelMVO_x(1,:)';



      for p = 1:num_of_reop

     %   RECALCULATE RELEVANT PARAMETERS AND REOPTIMIZE

        markov_periods =  10;

        beg_indices(1) = beg_indices(1) + T_reb;
        end_indices(1) = end_indices(1) + T_reb;



        currassetprices=asset_prices(beg_indices(1):end_indices(1),:);
        currmarketprices=market_prices(beg_indices(1):end_indices(1));
        currinfprices=inf_prices(beg_indices(1):end_indices(1));
        currriskfreeprices=riskfree_prices(beg_indices(1):end_indices(1)+1);


        [expected_inf, tnodes,tnodeval]=...
            new_exp_inf2(markov_periods,futureinfprices(T_reb*p+1),...
                curr_regime,c1,c2,ar1,ar2,p11,p12,p21,p22);

        [expected_inf_var] = ...
            new_exp_infvar2(markov_periods,curr_regime,tnodeval, ...
                                var1,var2,p11,p12,p21,p22);

        asset_prices_with_inf=[currinfprices currassetprices];

        [Inf_Beta R2_inf]   =solve_beta3(asset_prices_with_inf,1);

        currinfprices2=currinfprices(1:end-1);

        [asset_mu,asset_Q,asset_r]= solve_mvo_params(currassetprices,1,size(
            currassetprices,1));

        [modelMVO_x(p+1,:) modelMVO_var MVO_adjret_diagQ nom_ret] = ...
            main_MVO(currinfprices2,asset_r,expected_inf/100,expected_inf_var
                ,...
                            Inf_Beta',desired_R,desired_transaction,modelMVO_x(p
                                ,:)');

        model_portfolio_beta(p+1,1) = modelMVO_x(p+1,:)*Inf_Beta';

```

```
        [benchMVO_x(p+1,:) benchMVO] = ...
         benchmark_MVO(asset_mu', asset_Q, desired_R,desired_transaction,
            benchMVO_x(p,:)');
        MVO_portfolio_beta(p+1,1) = benchMVO_x(p+1,:)*Inf_Beta';

        MVO_returns_reb(T_reb*p+1:min((p+1)*T_reb,size((future_asset_r),1)))=
            ...
            future_asset_r((T_reb*p+1):...
                    min((p+1)*T_reb,size((future_asset_r),1)),:)...
                *benchMVO_x(p+1,:)';

        inf_returns_reb(T_reb*p+1:min((p+1)*T_reb,size((future_asset_r),1))) =
            ...
            future_asset_r(T_reb*p+1:min((p+1)*T_reb,size((future_asset_r),1))
                ,:)...
                *modelMVO_x(p+1,:)';

        [P_SRATIOS(:,p+1) S_SRATIOS(:,p+1)]=...
            calculateSharpeRatio(asset_mu, asset_Q, temp_Q, ...
                modelMVO_x(p+1,:), benchMVO_x(p+1,:), modelMVO_var,
                    currriskfreeprices);

    end

        cumul_MVO_reb(1) = MVO_returns_reb(1);
        cumul_inf_reb(1) = inf_returns_reb(1);



      T = length(MVO_returns_reb);

        for i = 2:T
            cumul_MVO_reb(i) = (1+cumul_MVO_reb(i-1))*(1+MVO_returns_reb(i))
                - 1;
            cumul_inf_reb(i) = (1+cumul_inf_reb(i-1))*(1+inf_returns_reb(i)) -
                1;
        end

        figure
        plot(1:T,cumul_MVO_reb*100, '-b');
        hold all
        plot(1:T,cumul_inf_reb*100, '-r');
        hold all
        plot(1:length(cumul_SP),cumul_SP*100, '-g');
        hold all
        plot(1:T,cumul_MF1*100, '-m');
        hold all
        plot(1:T,cumul_MF2*100, '-c');

    h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500',...
            'Vanguard Wellington Inv','CGM Mutual Fund'};
```

```matlab
        h = legend(h);

        grid on;

        title(['Comparing Cumulative Returns of Optimal Rebalanced Portfolios and Market'...
            ,num2str(syear+eachyear),' to ',num2str(eyear)]);

        xlabel('Time (in months)')
        ylabel('Cumlative Returns in %')
end
portfolio_betas = [model_portfolio_beta MVO_portfolio_beta];
clear model_portfolio_beta
clear MVO_portfolio_beta
toc
% ------------------------------------------------------------------------------------------
% STEP 10. AT THE END OF THE LOOP, PLOT THE PORTFOLIO'S PERFORMANCE OVER TIME AND COMPARE IT TO
%   HOW THE STANDARD S&P500 INDEX DID (PLOT BOTH ON SAME GRAPH)
rmpath('m_Files');
rmpath('data_Files');

%Comment/uncomment the end below if you want to see/not see all the workspace variables
% end
```

```matlab
SP500_symb= {'AA'
'ABC'
'ABT'
'LOMMX'
'VWELX'
'^GSPC'
};
SP500_symb=SP500_symb';

SP500_symb_csv={'AA.csv'
     'ABC.csv'
     'ABT.csv'
     'LOMMX.csv'
          'VWELX.csv'
          '^GSPC.csv'
 };
SP500_symb_csv=SP500_symb_csv';
```

```matlab
function [Spec_Out p11 p22 p12 p21 var1 var2 var3 ar1 ar2 ar3 c1 c2 c3] = ...
                    RegimeSwitching_MLE(k,inf_data)

                  disp('Computing Parameters of Regime Switching Model');
                  addpath('../../MS_Regress_FEX_1.08/m_Files'); % add 'm_Files' folder to the search path
```

```
addpath('../../MS_Regress_FEX_1.08/data_Files');

logRet=inf_data;  % load some Data.

% Defining dependent variables in system (solving for two dependent variables)
dep=logRet(:,1);
nLag=1;                              % Number of lags in system
k=k;                                 % Number of States
doIntercept=1;                       % add intercept to eqsuations
%%
advOpt.diagCovMat=0                                          %if this value is 1, then only the elements on the diagonal( the variances) are estimated
%%
advOpt.distrib='Normal';             % The Distribution assumption (only 'Normal' for MS VAR models) )this is the default avlue
advOpt.std_method=1;                 % Defining the method for calculation of standard errors. See pdf file for more details

Spec_Out=MS_VAR_Fit(dep,nLag,k,doIntercept,advOpt);
disp('COMPLETED MS_VAR_FIT function');

% make the transition probabilites easier to call
% where pij is probability of going to regime j from regime i
p11=Spec_Out.Coeff.p(1,1);
p22=Spec_Out.Coeff.p(2,2);
p12=Spec_Out.Coeff.p(2,1);
p21=Spec_Out.Coeff.p(1,2);

% The variances associated with each regime
var1=cell2mat(Spec_Out.Coeff.covMat(1,1));
var2=cell2mat(Spec_Out.Coeff.covMat(1,2));

%autoregressive terms for each regime
%Have to use two lines to properly index values
ar1=Spec_Out.Coeff.S_Param(1,1);
ar1=ar1{1}(2,1);

ar2=Spec_Out.Coeff.S_Param(1,1);
ar2=ar2{1}(2,2);

%intercepts for each regime (These are the mean inflation rates for the regime)
c1=Spec_Out.Coeff.S_Param(1,1);
c1=c1{1}(1,1);
```

```
        c2=Spec_Out.Coeff.S_Param(1,1);
        c2=c2{1}(1,2);

        %Different return values depending on whether there are 2 or 3
            regimes
        if (k==2)
            t=num2cell(zeros(1,8));
            [p13,p23,p33,p32,p31,var3,ar3,c3]=deal(t{:});

        elseif (k==3)
            p13=1-p11-p12;
            p23=1-p22-p21;
            p33=Spec_Out.Coeff.p(3,3);
            p32=Spec_Out.Coeff.p(2,3);
            p31=Spec_Out.Coeff.p(1,3);
            var3=cell2mat(Spec_Out.Coeff.covMat(1,3));
            ar3=Spec_Out.Coeff.S_Param(1,1);
            ar3=ar2{1}(2,3);
            c3=Spec_Out.Coeff.S_Param(1,1);
            c3=c3{1}(1,3);
        end

end
```

```
% This function returns the inflation rates used to compute inflation betas

function [infmonth infyear infprice] = all_inflation_data(filename)

    [infmonth infyear infprice] = textread(filename, '%s %d %f','delimiter
        ',',');

end
```

```
function [rfmonth rfyear rfprice] = all_riskfree_data(filename)

    %[infmonth, infyear, infprice] = deal(cell(1,1));


    [rfmonth rfyear rfprice] = ...
                        textread(filename, '%s %d %f','
                            delimiter',',');

end
```

```
function [month day year price fail_symbols success_symbols] = ...
    all_stock_data(SP500_symb_csv,SP500_symb)




    [month,day,year,price,fail_symbols,success_symbols]=deal(cell(1,1));

```

```
        fc=0;
        sc=1;
        for i=1:length(SP500_symb_csv)
                try
                        curr_csv=char(SP500_symb_csv(i));
                        curr_symb=char(SP500_symb(i));
                        success_symbols{sc}=curr_symb;

                        [month{sc} day{sc} year{sc} price{sc}] = ...
                         textread(curr_csv ,'%s %d %d %f','delimiter',',: ');

                        sc=sc+1;


                catch
                        curr_symb=char(SP500_symb(i));
                        fc=fc+1;
                        fail_symbols{fc}=curr_symb;
                end
        end
        %BY THE END OF THIS LOOP WE HAVE DAILY TIME SERIES DATA
        %NOW ITERATE THROUGH MONTH VECTOR AND DELETE EVERY ELEMENT
        % WHICH HAS THE SAME MONTH

        % price_names=[];

        % for (i=1:length(success_symbols))
        %       price_names=[price_names; success_symbols(i)];
        % end

        % success_symbols=price_names;







end
```

```
%NOTE FORMAT OF INPUT MONTH IS SUPPOSED TO
function [inftimeprices beg_ind end_ind] = fetch_inflation_data2(smonth,syear,
    emonth,eyear,...
                                                                                                    inf

        x=strmatch(smonth,infmonth);
```

```
	for i=1:length(x)
		if infyear(x(i))==syear
			beg_ind=x(i);
		end
	end
	x=strmatch(emonth,infmonth);
	for i=1:length(x)
		if infyear(x(i))==eyear
			end_ind=x(i);
		end
	end

	inftimeprices=flipud(data(beg_ind:end_ind));


end
```

```
%returns inflation data used to compute the RS model. It may be either
%monthly, quarterly or annually depending on the filename passed
function [inf_data inf_data_avg] = ...
		fetch_regime_inflation_data(s_year,e_year,s_month,e_month,
		   filename)

	s_year=s_year-1914;
	e_year=e_year-1914;
	inf_data=csvread(filename,s_year,s_month,[s_year,s_month,e_year,
	    e_month]);

	inf_data=inf_data';
	inf_data=inf_data(:);

	%inf_data_avg=csvread(filename,s_year,13,[s_year,13,e_year,13]);
	inf_data_avg=csvread...
	(filename,s_year,e_month+1,[s_year,e_month+1,e_year,e_month+1]);


end
```

```
%NOTE FORMAT OF INPUT MONTH IS SUPPOSED TO
function [riskfreeprices] = fetch_riskfree_data2(smonth,syear,emonth,eyear,...
																			inf

	x=strmatch(smonth,infmonth);

	for i=1:length(x)
```

```
            if infyear(x(i))==syear
                beg_ind=x(i);
            end
        end
        x=strmatch(emonth,infmonth);
        for i=1:length(x)
            if infyear(x(i))==eyear
                end_ind=x(i);
            end
        end

        inftimeprices=flipud(data(beg_ind:end_ind));


end
```

```
%INPUT PARAMETERS
% month, day, year, price represent all the time series data
% succes_symbols represents the assets that succesfully
% FOR NOW ASSUME ROW 3000 IS THE SAME DATE FOR EVERY ASSET
function [prices price_names2 marketprice num_assets catch_assets ...
                interval_size] = ...
         SEC_fetch_stock_data(smonth,syear,emonth,eyear,month,day,year,data,success_symbols)


%-------CODE TO FIND THE STARTING AND ENDING INDICES ---------------
        x=strmatch(smonth,month{1});
        for i =1:length(x)
            if year{1}(x(i))==syear
                beg_ind=x(i);
            end
        end

        x=strmatch(emonth,month{1});
        % disp(emonth);
        % disp(month{1});
        % disp(x);
        % disp(eyear);
        % disp(year);
        disp(eyear);
        for i =1:length(x)
            disp(year{1}(x(i)));
            if year{1}(x(i))==eyear
                end_ind=x(i);
            end
        end
        interval_size=end_ind-beg_ind;
%---------------AFTER FINDING THE INDICES ITERATE THROUGH ASSETS TO GET PRICE DATA -----
        count=1;
        %beg_ind=beg_ind-1;
```

```
        %end_ind=end_ind-1;
        catch_assets=1;
        %The last element

        for i = 1:(length(success_symbols)-1)
                %symb_matrix{i}=SP500_symb_csv{i};
                try
                        % disp('entered try block')
                        prices{count}=flipud(data{i}(beg_ind:end_ind));
                        price_names{count}=success_symbols(i);
                        count=count+1;
                        %disp(SP500_symb_csv(i))
                catch
                        catch_assets=catch_assets+1;
                end
        end
        num_assets=count-1; % add 1 for the market index

        %retrieve time series price information for the marketprice
        try
                marketprice=data{end}(beg_ind:end_ind);
                marketname=success_symbols(end);
        catch
        end

        marketprice=flipud(marketprice);


        price_names2=[];

        for (i=1:length(price_names))
                price_names2=[price_names2; price_names{i}];
                % disp(price_names2)
        end



end
```

```
% This function generates the markov tree and comptues the expected inflation raet

%curr_regime is 0 or 1
function [inf_rate,tnodes,tnodeval3] = new_exp_inf2(n,y0,curr_regime,c1,c2,ar1,ar2,p11,p12,p21,p22)

        % Compute an associated binary value for each termianl node which represents
        % the unique pathe taken to get to that node
        tnodes=terminal_nodes(n);

```

```
        % compute that actual terminal inflation values conditional on the
            path taken
        tnodeval=terminal_inflation(n,tnodes,c1,c2,ar1,ar2,y0);
        tnodeval2{n}=tnodeval;
        maxelements=2^n;

        % iterate over every time period and compute the values for each node
        for i=n:-1:1
                tnodes2=terminal_nodes(i);
                a=terminal_inflation(i,tnodes2,c1,c2,ar1,ar2,y0);
                % disp(i);
                % disp(a);
                tnodeval3(i,:)=[a zeros(1,maxelements-2^i)];
        end


        % transition probability matrix
        p= [p11 p12
            p21 p22];

        inf_rate=0;
        for i=1:2^n
                currprod=tnodeval(i);
                cnode=tnodes{i};
                %for the n-1 transition probabilities
                for j=n:-1:2
                        ind1=str2num(cnode(j))+1;
                        ind2=str2num(cnode(j-1))+1;
                        currprod=currprod*p(ind2,ind1);
                end
                %for the initial transition probability
                ind3=str2num(cnode(1))+1;
                currprod=currprod*p(curr_regime,ind3);

                inf_rate=inf_rate+currprod;

        end


end
```

```
% THis function is used to compute the expected variance of the inflation rate
function [inf_var inf_var_node_val ] =...
         new_exp_infvar2(n,start_regime,tnodeval,var1,var2,p11,p12,p21,p22)



        p= [p11 p12
                p21 p22];
        % For every terminal node, assign either variance of regime 1 or
            variance of regime 2
        % as its value
```

```
        for i=1:2^n
                if (mod(i,2)==0)
                        initialvarval(i)=var1;
                elseif (mod(i,2)==1)
                        initialvarval(i)=var2;
                end
        end


        infvar_node_val=initialvarval;

        % Iterate over every time period and call the recursive function new_infvar_nodeval
        % until you reach the initial time period
        for i=n:-1:1
                infvar_node_val=...
                        new_infvar_nodeval(i,p,infvar_node_val,tnodeval(i,:),start_regime);
        end
        inf_var=infvar_node_val(1);

end
```

```
% This is the recursive function that commputes the expected inflation variance
% associated with each node
function [nodeval] = new_infvar_nodeval(n,p,varval,muinfval,start_regime)

        n=n-1;

        % Determines the two hcild nodes of a node and uses those values to get
        % the variances used in the expectation formula
        for i=1:2^n
                        var2(i)=varval(2*i-1);
                        var1(i)=varval(2*i);
                        mudiff(i)=(muinfval(2*i-1)-muinfval(2*i))^2;
        end

        % This is the base case and you use either p(1,1) and p(1,2) or
        % p(2,1) and p(2,2) depending on the starting regime
        if n==0
                if (start_regime==1)
                        prob=p(1,1);


                elseif (start_regime==2)
                        prob=p(2,1);
                end
                nodeval(i)=prob*var1(i)+(1-prob)*var2(i)+prob*(1-prob)*mudiff(i);

```

```
        else
                % This is the recursive case but the node value can still be computed
                % at that point.
                for i=1:2^n
                        if mod(i,2) == 0
                                prob=p(1,1);
                        else
                                prob=p(2,1);
                        end
                        nodeval(i)=prob*var1(i)+(1-prob)*var2(i)+prob*(1-prob)*mudiff(i);
                end
                % end
        end
end
```

```
% This function applies the closed form equation to compute the terminal
% value associated with each node in the markov tree for expected inflation
% This equation is simply the result of recursion(due to the autoregressive term)
% and is unique for each node.
function [tnodeval] = terminal_inflation(n,tnodes,c1,c2,ar1,ar2,y0)

        %{
        a0=c1;
        a1=c2;
        b0=ar1;
        b1=ar2;
    beta1 and beta2 represent the betas for regime 1 and 2
        %}
        a=[c1 c2];
        b=[ar1 ar2];

        %iterates over each terminal node
        tnodeval = [];
        for i=1:2^n
                cnode=tnodes{i};
                cnodeval = 0;
                %For a given terminal node, iterates over each time step
                for j=n:-1:1
                        %period=n-j+1;
                         if (j==n)
                        %       disp('Value of a() is');
                        %       a(str2num(cnode(n))+1)
                                cnodeval = cnodeval + a(str2num(cnode(n))+1);
                        else
                                ind_j=str2num(cnode(j))+1;
                                currprod=a(ind_j);

                               % over number of terms for a given time step
                                for k = j+1:n;
```

```
                                        % disp('Value of a() is');
                                        % a(str2num(cnode(n))+1)
                                        ind_k=str2num(cnode(k))+1;
                                        currprod = currprod*b(ind_k);
                                end
                                cnodeval=cnodeval+currprod;
                        end
                        if (j==1)
                                currprod=1;
                                for k= 1:n
                                        ind_k=str2num(cnode(k))+1;
                                        currprod=currprod*b(ind_k);
                                end
                                cnodeval=cnodeval+currprod*y0;
                        end
        end
        tnodeval(i)=cnodeval;
    end

end
```

```
%This function returns a vector of all the terminal nodes
function [S_bin] = terminal_nodes(n)
        for i=0:2^n-1
                S_bin{i+1}=dec2bin(i,n);
        end
end
```

```
% This function takes in the total months and total intervals
% and returns matrices of equally spaced interval(except the last one
% may be slightly longer in the case of numintervals not being a factor
% of total months.
function [beg_ind end_ind] = divide_interval(numintervals,totalmonths)

        beg_ind=[];
        end_ind=[];
        indice_size=floor(totalmonths/numintervals);

        for i=1:numintervals-1
                beg_ind=[beg_ind; indice_size*(i-1)+1];
                end_ind=[end_ind; indice_size*i];
        end
        %if mod(totalmonths,numintervals)>0

                beg_ind=[beg_ind; (indice_size*(numintervals-1)+1)];
                end_ind=[end_ind; (indice_size*numintervals+mod(totalmonths,
                    numintervals))];
        % end


end
```

```
%input parameters are # of states, smoothingprobabilities,and no. of time periods
%THe function returns a matrix of the resulting regime for each period
%and counts the total type of each regime for the historical inflation data

%The regime associated with each time is ultimately the greater of the two
% probabilities of that time period belonging to that regime
function [whichregime, countregime] = ...
                regimecount(k,smoothProb,timelength)



        %NOte that the size of Spec_Out.filtProb is (# of time periods) x (# of states)
        %IMPORTANT: NOTE THAT IT Is likely to change .filtProb to .smoothProb
        for i=1:length(smoothProb(1,:))
                r_prob(:,i)=smoothProb(:,i);
        end


        %initializes countregime matrix of size k(no. of states)
        countregime=zeros(1,k);

        for i=1:timelength
                if (k==3)
                        if (r_prob(i,1)>=r_prob(i,2) && r_prob(i,1)>=r_prob(i,3))
                                whichregime(i)=1;
                                countregime(1)=countregime(1)+1;
                        elseif (r_prob(i,2)>=r_prob(i,1) && r_prob(i,2)>=r_prob(i,3))
                                whichregime(i)=2;
                                countregime(2)=countregime(2)+1;
                        else
                                whichregime(i)=3;
                                countregime(3)=countregime(3)+1;
                        end
                elseif (k==2)
                        if (r_prob(i,1)>=r_prob(i,2))
                                whichregime(i)=1;
                                countregime(1)=countregime(1)+1;
                        else
                                whichregime(i)=2;
                                countregime(2)=countregime(2)+1;
                        end
                end
        end

end
```

```
% This function returns the apprtopriate inflation fil
% for monthly, quarterly or annual data
```

```
function [inf_file endcol] = return_inflation_file(inflation_interval)
        if (strcmp(inflation_interval,'monthly'))
                inf_file='inflation_data_monthly.csv';
                endcol=12;
        elseif (strcmp(inflation_interval,'quarterly'))
                inf_file='inflation_data_quarterly.csv';
                endcol=4;
        elseif (strcmp(inflation_interval,'annually'))
                inf_file='inflation_data_annual.csv';
                endcol=1;
        end
end
```

```
%% STANDARD MARKOWITZ MVO MODEL VIA QUADPROG
% Input the number of assets, the range of desired returns, the mean vector
% for the assets, and the covariance matrix. The function will return the
% optimal portfolio weights in MVO_x, and the corresponding variance
% function value in MVO_var.
%
% [optimal weights, corresponding optimal objective function]
%   = sMVO(# of assets, range of desired returns, expected return vector,
%           covariance matrix)

% Set quadprog parameters
function [ MVO_x MVO_var] = benchmark_MVO( mu, Q, return_range,
    transaction_cost, previous_portfolio)


        n= length(mu);

        c = [zeros(3*n,1);];
        Aeq = [ones(1,n) zeros(1,2*n)
                    eye(n) -eye(n) eye(n)];
        beq = [1; previous_portfolio];
        A=[-mu' transaction_cost*ones(1,2*n)];
        lb=zeros(3*n,1);
        ub=9999*ones(3*n,1);
        R = return_range;

        tempQ=Q;
        Q = [Q zeros(n,2*n); zeros(2*n, 3*n)];

        % Set quadprog options
        options = optimset('Algorithm', 'interior-point-convex', 'TolFun',
            1/10^10, 'MaxIter', 1000, 'TolCon', 1/(10^10));


        %Solve MVO and store SD values for plotting

        %Modify this so that it works for variable length R (length>1)
        for i = 1:length(R);
```

```
            b=[-R(i);  ];

            [MVO_x(i,:), MVO_var(i,1)] = quadprog(Q, c, A, b,Aeq, beq, lb, ub,
                [], options);

            %MVO_std = MVO_var.^.5;
            MVO_x=MVO_x(i,1:n);
        end

        adjvector = [MVO_x' mu diag(tempQ)];
end
```

```
function [ MVO_x MVO_var adjvector  nominal_return temp_Q] = main_MVO(
    historical_inflation, historical_asset_returns,...
        expected_inflation, inflation_variance, infbeta, R, transaction_cost,
            previous_portfolio)
        % expected_inflation, inflation_variance, infbeta, R, transaction_cost
            , max_transactions, previous_portfolio)
    %{
    historical_inflation: time series data for inflation values over the
    relevant period

    **IN DECIMAL FORM**

    historical_asset_returns: time series data for relevant assets over
    relevant periods

    expected_inflation: inflation calculated through markov RS model

    inflation_variance: inflation variance calculated through markov RS

    beta: vector of n beta values corresponding to each asset for the current
    regime

    %}

    r_M = historical_inflation';
    r_it = historical_asset_returns;
    mu_M = expected_inflation;
    del_M = inflation_variance;

    [T, n] = size(r_it);
    c = [zeros(3*n,1);];
    Aeq = [ones(1,n) zeros(1,2*n)
           eye(n) -eye(n) eye(n)];
    beq = [1; previous_portfolio];
    lb=zeros(3*n,1);

    ub=9999*ones(3*n,1);

    nominal_return = prod(1+r_it).^(1/T) - 1;
```

```
    nominal_return = nominal_return';
    % disp(size(nominal_return));
    % disp(size(infbeta));
    % disp(size(mu_M));
    adjusted_return = nominal_return + infbeta*mu_M;
    alpha = nominal_return(1:end)-infbeta*mu_M;
    % disp(nominal_return(1:end))
    % disp(infbeta:mu_M)
    % disp(size(r_it(:,1)));
    % disp(size(alpha(1)));
    % disp(size(infbeta(1)));
    for i=1:n
        epsi(:,i)=r_it(:,i)-(alpha(i)+infbeta(i)*r_M(:,1));
    end
    % infbeta=infbeta.^(-1);
    del_epsi=diag(cov(epsi));
    for i = 1:n;
        for j = 1:n;
            if i==j
                Q_sf(i,i)=infbeta(i)^2*del_M+del_epsi(i);
            else
                Q_sf(i,j)=infbeta(i)*infbeta(j)*del_M;
            end
        end
    end

    temp_Q = Q_sf;

    Q_sf = [Q_sf zeros(n,2*n); zeros(2*n, 3*n)];

    options = optimset('Algorithm', 'interior-point-convex', 'TolFun',
        1/10^10, 'MaxIter', 1000, 'TolCon', 1/(10^10));

    % Solve Single Factor model and store SD values
    disp('--------------------')
    disp(size(adjusted_return'));
    disp(size(zeros(1,2*n)));
    disp(size(zeros(1,n)));
    disp(size(ones(1,2*n)));

    for i = 1:length(R);
        A=[-(adjusted_return') transaction_cost*ones(1,2*n);];
            %zeros(1,n) transaction_cost * ones(1,2*n)];
        b=[-R(i); ]%max_transactions];


        [MVO_x(i,:), fval(i,1)] = quadprog(Q_sf, c, A,b, Aeq,beq,...
                                            lb,ub,[],options);
        MVO_var(i,1)=(MVO_x(i,:)*Q_sf*MVO_x(i,:)');
        MVO_x=MVO_x(i,1:n);

```

```
    end

    adjvector = [MVO_x' adjusted_return diag(temp_Q) infbeta];

end
```

```
% This function takes in the asset allocatiosn of standard MVO and our custom
    inflation
% hedged model. It also tak

%% MVO_COMPARISON
%{
    Take in the two MVO weightings, one for standard MVO and the other for
    inflation hedged MVO, then show their projected returns over the sample
    period, alongside the market returns for that period.

    This function can be modified accodingly if more benchmark models are
    added.

    MVO_x: vector of weights corresponding to standard MVO w/ transactions
    inf_x: vector of weights corresponding to our model
    asset_returns: relevant for the relevant assets over the sample
    period
    market_returns: market returns
%}
function [cumul_MVO cumul_inf cumul_SP cumul_MF1 cumul_MF2] = MVO_comparison(
    MVO_x, inf_x, projected_returns,...
    market_returns,MF_returns1,MF_returns2,startyear,endyear)


    [T n] = size(projected_returns);


    MVO_returns = projected_returns*MVO_x;
    inf_returns = projected_returns*inf_x;

    cumul_MVO(1) = MVO_returns(1);
    cumul_inf(1) = inf_returns(1);
    cumul_SP(1) = market_returns(1);
    cumul_MF1(1) = MF_returns1(1);
    cumul_MF2(1) = MF_returns2(1);


    for i = 2:T
        cumul_MVO(i) = (1+cumul_MVO(i-1))*(1+MVO_returns(i))  - 1;
        cumul_inf(i) = (1+cumul_inf(i-1))*(1+inf_returns(i)) - 1;
        cumul_SP(i) =  (1+cumul_SP(i-1))*(1+market_returns(i)) - 1;
        cumul_MF1(i) =  (1+cumul_MF1(i-1))*(1+MF_returns1(i)) - 1;
        cumul_MF2(i) =  (1+cumul_MF2(i-1))*(1+MF_returns2(i)) - 1;
    end

    portfolio_MVO=1 * (1 + cumul_MVO);
```

```
    portfolio_inf=1 * (1 + cumul_inf);
    portfolio_SP=1 * (1 + cumul_SP);

    figure
    plot(1:T,cumul_MVO*100, '-b');
    hold all
    plot(1:T,cumul_inf*100, '-r');
    hold all
    plot(1:T,cumul_SP*100, '-g');
    hold all
    plot(1:T,cumul_MF1*100, '-m');
    hold all
    plot(1:T,cumul_MF2*100, '-c');

%   plot(1:T,currinfprices,'-m');

    h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500',...
         'Vanguard Wellington Inv','CGM Mutual Fund'};
    h = legend(h);

    grid on;

    title(['Comparing Cumulative Returns of Optimal Portfolios and Market'...
        ,num2str(startyear),' to ',num2str(endyear)])
    xlabel('Time (in months)')
    ylabel('Cumulative Monthly Returns (in %)')


    figure
    plot(1:T,MVO_returns, '-b');
    hold all
    plot(1:T,inf_returns, '-r');
    hold all
    plot(1:T,market_returns,'-g');
    hold all
%   plot(1:T,currinfprices, '-m');

    h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500','inflation rate'};

    title(['Comparing Return Values of Optimal Portfolios and Market'...
        ,num2str(startyear),' to ',num2str(endyear)])
    xlabel('Time (in months)')
    ylabel('Monthly Returns')



    h = legend(h);

    grid on;

    hold all
```

```
end
```

```
%This function is passed in the pricedata for ONE asset
function [ mu, Q, r_it ] = mvo_params(pricedata, end_pred)
%% DETERMINE EXPECTED RETURNS AND COVARIANCES FOR ASSETS
% Input a matrix of time series data for the desired assets, as well as
% the number of days in the estimation horizon starting from day 1, and the
% function will return the expected returns and covariances calculated from
% the time series data from the estimation horizon.
%
% [expected returns, covariance matrix] = param_data(time series,
%                                              estimation horizon)

r_it = (data(2:end_pred,:)./data(1:end_pred-1,:)) - 1;
[T, n] = size(r_it); %time periods, assets
mu = prod(1+r_it).^(1/T) - 1;
Q = cov(r_it);

end
```

```
function [ nominal_return, nominal_Q] = nominal_parameters(asset_data,
    market_data)
% take in the relevant time series data for assets and market
% and return the nominal return and covariances for assets

% Model: Single factor CAPM



        r_it = (asset_data(2:end,:)./asset_data(1:end-1,:)) - 1;
        r_M = (market_data(2:end,:)./market_data(1:end-1,:)) - 1;

        [T, n_assets] = size(r_it);

        nominal_return = prod(1+r_it).^(1/T) - 1;
        mu_M = prod(1+r_M).^(1/T) - 1;

        del_M=sum((r_M(:,1) - mu_M(1)).^2)/T; % Factor variance
        beta=(sum(r_it(:,1:end).*repmat(r_M(:,1),1,n_assets))/T-mean(r_it(:,1:
            end)*...
            mean(r_M(:,1)))/del_M);
        alpha = nominal_return(1:end)-beta*mu_M;

        %Noise vector
        for i=1:n_assets
            epsi(:,i)=r_it(:,i)-(alpha(i)+beta(i)*r_M(:,1));
        end
        del_epsi=diag(cov(epsi));

        % Single factor covariance
```

```
        for i = 1:n_assets;
            for j = 1:n_assets;
                if i==j
                    nominal_Q(i,i)=beta(i)^2*del_M+del_epsi(i);
                else
                    nominal_Q(i,j)=beta(i)*beta(j)*del_M;
                end
            end
        end
end
```

```
% This function is used to compute the inflation Beta, but it can also
% compute the CAPM beta if desired. The second parameter passed in can either be
% inflation rate date or market data
function [solved_beta R_squared] = solve_beta3(tprice,inf_or_market)

    %MARKET - CAPM
        if (inf_or_market==2)
                r_it=tprice(2:end,:)./tprice(1:end-1,:)-1;
    %MARKET - INFLATION
        elseif (inf_or_market==1)
                r_it=tprice(2:end,2:end)./tprice(1:end-1,2:end)-1;
                [T n] = size(r_it)
        disp(size(tprice(:,1)))
                r_it=[tprice(1:T,1) r_it];
        end

        [T, n]=size(r_it);                                  %number of time period T and asset n
        mu=prod(1+r_it).^(1/T)-1;                           %Geometric mean for factor and assets
        del_M=sum((r_it(:,1)-mu(1)).^2)/T;                  %variance of factor under geometric mean

        %Calculate the Beta coefficient

    if inf_or_market==1
        for i = 2:n
        p = polyfit(r_it(:,1)./100,r_it(:,i),1);
        solved_beta(i - 1) = p(1);

        yfit = polyval(p,r_it(:,1)./100);
        y = r_it(:,i);
        yresid = y - yfit;
        SSresid = sum(yresid.^2);
        SStotal = (length(y)-1) * var(y);
        R_squared(i - 1) = 1 - SSresid/SStotal;
    end

    else
    for i = 2:n
```

```
        p = polyfit(r_it(:,1),r_it(:,i),1);
        solved_beta(i - 1) = p(1);

        yfit = polyval(p,r_it(:,1));
        y = r_it(:,i);
        yresid = y - yfit;
        SSresid = sum(yresid.^2);
        SStotal = (length(y)-1) * var(y);
        R_squared(i - 1) = 1 - SSresid/SStotal;
    end


end
```

```
%This function takes in a matrix of asset prices,

function [mu, Q, r_it] = solve_mvo_params(asset_prices,beg_pred,end_pred)

        data=asset_prices;
        r_it = (data((beg_pred+1):end_pred,:)./data(beg_pred:end_pred-1,:)) - 1;
        [T, n] = size(r_it);
        mu = prod(1+r_it).^(1/T) - 1;
        Q = cov(r_it);

end
```

```
% This function computes the sharpe ratio and modified sharpe ratio computed
% for each portfolio
function [P_SRATIOS S_SRATIOS] = ...
                calculateSharpeRatio(asset_mu, asset_Q, temp_Q, modelMVO_x, benchMVO_x, ...
                                                    modelMVO_var, currriskfreeprices)

        %Using our model version of the Sharpe ratio, Preface M

        [P_premodel_sratios] = ...
                (asset_mu*modelMVO_x'-currriskfreeprices(end)/100)/(modelMVO_var^0.5);

        [P_preMVO_sratios P_preMVO_smu P_preMVO_s_sigmap]=...
            sharperatio2(asset_mu,temp_Q,benchMVO_x,currriskfreeprices(end)/100);

        % ------------------------------------------------------------------------------------------

        %Using regular version of the Sharpe ratio, Preface S
        [S_premodel_sratios S_premodel_smu S_premodel_s_sigmap]=...
```

```
            sharperatio2(asset_mu,asset_Q,modelMVO_x,currriskfreeprices(end)/100);

        [S_preMVO_sratios S_preMVO_smu S_postMVO_sigmap]=...
            sharperatio2(asset_mu,asset_Q,benchMVO_x,currriskfreeprices(end)/100);

        P_SRATIOS = [P_premodel_sratios; P_preMVO_sratios];
        S_SRATIOS = [S_premodel_sratios; S_preMVO_sratios];
end

% [P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu, asset_Q, temp_Q, ...
%      modelMVO_x(1,:), benchMVO_x(1,:),modelMVO_var,currriskfreeprices);
```

```
%Takes in the asset prices for the desired periods
function [sratio mu_p sigma_p] = ...
        sharperatio2(asset_mu, asset_cov, xalloc, riskfreeprice)


        mu_p=asset_mu*xalloc';
        sigma_p=(xalloc*asset_cov*xalloc')^0.5;

        sratio=(mu_p-riskfreeprice)/sigma_p;

end

% [P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu, asset_Q, ...
%              modelMVO_x, benchMVO_x,modelMVO_var,currriskfreeprices);
```