

# Matlab Code

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

```
1 % INFLATION_INTERVAL is 'monthly','quarterly' or 'annually'
2 % and is used to computer the regime history
3
4 %


---


5 % SYMBOLFILE specifies which file to fetch symbols from. A list of the files
6 % available are given below
7
8 % -Symbols_SP.m has the full S&P500 Stocks. It is important to note that only
9 % some
10 % of these have aviable price history that dates back to 1974.
11 % -SecondSymbols has a small subset and is used for testing
12 % -ThirdSymbols is an even smaller subset, having only 3 stocks.
13 % -Symbols_NYSE has a full list of all NYSE stocks. However at this time,
14 % only a subset of these 3300+ stocks have local csv files avaiable for
15 % fetching
16 % -Symbols_NYSE2 has a list of the NYSE stocks that have local csv files
17 % -Symbols_NYSE_SP has a list of NYSE stocks AND S&P500 stocks. All of these
18 % are avaiable for fetching. It is the recommended file to use because it has
19 % the most data avaiable which is important for early years where most stocks
20 % dont have time series data
21 %


---


22 % SYEAR/EYEAR is the start/end year of optimization for the regeims
23 %


---


24 % NUM_OF_TIME_PERIODS is the number of divisions in the time interval
25 % note that the in sample period to get data is
26 % (syear-eyear)/num_of_time_periods.
27 %


---


28 % **ensure that syear-eyear is divisble by num_of_time_periods
29 %


---


30 % NUMOFREOP is the number of times the portfolio is reoptimized.
31 % note that this is a specified amount and there is no submodel to determine
32 % whether or not to reoptimize because the transcation cost constraint
33 % prevents drastic changes in asset allocation
34 %


---


35 % DESIRED_R IS THE minimum monthly return constraint. Converting this
36 % to an annual return is (1+desired_R)^12-1
37 %


---


```

```

36 % DESIRED_TRANSACTION is the the user-specified fixed transaction cost
37
38
39 % inflation_interval='monthly';
40 % SymbolFile='''Symbols_NYSE_SP.m''' ;
41 % startyear=1990;
42 % endyear=2010;
43 % num_of_time_divisions=5;
44 % num_of_reoptimization_periods=2;
45 % minimum_return=0.005;
46 % transaction_cost=0.0001;
47
48
49 % create_inflation_hedged_portfolio ...
50 %     (inflation_interval,SymbolFile,startyear,endyear,
51 %     num_of_time_divisions,...
52 %     num_of_reoptimization_periods, minimum_return, transaction_cost);
53
54 %Comment all the lines above and Uncomment the line below if you want to
55 % see all the workspace variables
56
57 run create_inflation_hedged_portfolio.m;
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

26 % STEP 1: RETRIEVING HISTORICAL INFLATION RATES & ALL STOCK DATA AVAILABLE
27 % NOTE: THIS CAN BE PREPROCESSED
28
29 %can be monthly, quarterly or annual
30 begcol=1;
31 [inf_file endcol] = return_inflation_file(inflation_interval);
32
33 diffyear=eyear-syear;
34 eachyear=diffyear/num_of_time_periods;
35 T_reb= (diffyear-eachyear)*12/(num_of_reop+1);
36
37 [MLEinf_data, inf_avg] =...
38     fetch_regime_inflation_data(1950,syear,begcol,endcol,inf_file);
39
40 %THIS gets a list of the Stock ticker symbols used as the pool for
41 % asset allocation along with their corresponding csv files
42 run(eval(SymbolFile));
43
44 %THIS fetches a list of ALL the stock/inflation/riskfree data for all dates
45 % Note that this is used for the optimization model and NOT for creating
46 % the regime switching model
47
48 [month day year price fail_symbols success_symbols]=...
49                                     all_stock_data(SP500_symb_csv ,
50                                                     SP500_symb);
51
52 [infmonth infyear infprice ] = all_inflation_data('inflation_rate_1200.csv');
53 [rfmonth rfyear rfprice] = all_riskfree_data('riskfreerate2.csv');
54 %

```

---

```

54 % STEP 2: SOLVE FOR THE MLE PARAMETERS OF THE REGIME-SWITCHING MODEL
55 % AS WELL AS DETERMING THE REGIME THAT EACH PERIOD BELONGS TO
56 % (WHICH PRODUCES THE REGIME STATE GRAPH)
57
58 k=2; %DECLARE THE NUMBER OF REGIMES
59
60 [Spec_Out p11 p22 p12 p21 var1 var2 var3 ar1 ar2 ar3 c1 c2 c3]= ...
61                                     RegimeSwitching_MLE(k,MLEinf_data);
62
63 timelength=length(MLEinf_data);
64
65 [whichregime, countregime] =regimecount(k,Spec_Out.smoothProb,timelength);
66
67 %Parameter 1 is the number of monthly time periods, 2 is the current inflation
68     rate
69 curr_regime=whichregime(timelength);
70 % curr_regime=2;
71 curr_inf_rate=MLEinf_data(timelength);
72 disp('CHECKPOINT 1');

```

```

73 %


---


74 % TEMP CODE FOR TESTING NEW INFLATION EXPECTED VARIANCE
75
76 markov_periods=10; %this is chosen number of periods for markov tree(in months
    )
77
78 [expected_inf , tnodes ,tnodeval]=new_exp_inf2(markov_periods , curr_inf_rate ,
    curr_regime ...
79             ,c1 ,c2 ,ar1 ,ar2 ,p11 ,p12 ,p21 ,p22);
80 [expected_inf_var] = ...
81     new_exp_infvar2(markov_periods ,1 ,tnodeval , var1 , var2 ,p11 ,p12 ,p21 ,p22);
82
83 %


---


84 % STEP 4 : RETRIEVE THE ASSET AND MARKET PRICES FOR THE DESIRED TIME PERIODS
85
86 lcase_month='Jan'
87 ucase_month=upper(lcase_month);
88
89 [tcurrprices currpricenames2 market_prices num_assets2 catch_assets2
    totalmonths] = ...
90     SEC_fetch_stock_data ...
91     (lcase_month ,eyear ,lcase_month ,syear ,month ,day ,year , price ,
        success_symbols);
92
93 %Use the fetch inflation data to compute the inflation rates and the riskfree
    rates
94 [inf_prices] = fetch_inflation_data2 ...
95     (ucase_month ,eyear ,ucase_month ,syear ,infmonth ,infyear , infprice);
96
97 [riskfree_prices] = fetch_inflation_data2 ...
98     (ucase_month ,eyear ,ucase_month ,syear ,infmonth ,infyear , rfprice);
99
100 disp('CHECKPOINT 3');
101
102
103 [beg_indices end_indices]=divide_interval(num_of_time_periods ,totalmonths);
104
105 asset_prices=cell2mat(tcurrprices);
106
107 %Parameters are first number of time divisions , then total months
108
109
110 currassetprices=asset_prices(beg_indices(1):end_indices(1)+1,:);
111 currmarketprices=market_prices(beg_indices(1):end_indices(1)+1);
112 currinfprices=inf_prices(beg_indices(1):end_indices(1)+1);
113 currriskfreeprices=riskfree_prices(beg_indices(1):end_indices(1)+1);
114

```

```

115
116
117 % asset_prices=cell2mat(currprices2);
118 asset_prices_with_market=[currmarketprices currassetprices];
119 asset_prices_with_inf=[currinfprices currassetprices];
120
121 %

```

---

```

122 % STEP 5: SOLVE FOR THE MVO PARAMETERS, THE MUS, Q'S AND R'S FOR ALL ASSETS
      AND
123 % THE MARKET OVER THE SPECIFIED TIME PERIODS. ALSO SOLVE FOR THE CAPM BETAS OR
124 % EACH ASSET
125
126 [asset_mu,asset_Q,asset_r]= solve_mvo_params(currassetprices,1,size(
      currassetprices,1));
127
128 %Solve for only the market
129 [Market_mu,Market_Q,Market_r]= solve_mvo_params(currmarketprices,1,size(
      currmarketprices,1));
130
131
132 % Calculate relevant parameters for testing horizon
133
134
135
136 disp('CHECKPOINT 4');
137
138 % desired_R=0.002;
139 % desired_transaction=0.0001
140
141 [Inf_Beta R2_inf] =solve_beta3(asset_prices_with_inf,1);
142 i_Inf_Beta=Inf_Beta;
143
144
145 currinfprices2=currinfprices(1:end-1);
146 xalloc=zeros(num_assets2,1);
147
148 [modelMVO_x(1,:) modelMVO_var MVO_adjret_diagQ nom_ret temp_Q] = ...
149     main_MVO(currinfprices2,asset_r,expected_inf/100,expected_inf_var,...
150             Inf_Beta',desired_R,desired_transaction,xalloc
151             );
152 i_MVO_adjret_diagQ_InfBeta=MVO_adjret_diagQ;
153 i_nom_ret=nom_ret;
154 i_temp_Q=temp_Q;
155
156 model_portfolio_beta(1,1) = modelMVO_x(1,:)*Inf_Beta';
157 % [modelMVO_x modelMVO_var temp_Q adj_ret nom_ret] = main_MVO(currinfprices,
      asset_r,1.8/100,0.04,...

```

```

158 %                                     Inf_Beta
    ',0.000002,0.0005,0.05,xalloc);
159
160 %

```

---

```

161 futureassetprices=asset_prices(beg_indices(2):end_indices(num_of_time_periods)
    ,:);
162 futuremarketprices=market_prices(beg_indices(2):end_indices(
    num_of_time_periods));
163 futureinfprices=inf_prices(beg_indices(2):end_indices(num_of_time_periods));
164 futureriskfreeprices=riskfree_prices(beg_indices(2):end_indices(
    num_of_time_periods));
165
166 [future_asset_mu , future_asset_Q , future_asset_r]= ...
167     solve_mvo_params(futureassetprices ,1 ,size(futureassetprices ,1));
168
169 %Solve for only the market
170 [future_Market_mu , future_Market_Q , future_Market_r]= solve_mvo_params(
    futuremarketprices ,1 ,size(futuremarketprices ,1));
171
172
173
174
175 [future_MK1_r] = future_asset_r(:,end);
176 [future_MK2_r] = future_asset_r(:,end-1);
177
178 [benchMVO_x(1,:) benchMVO] = ...
179 benchmark_MVO(asset_mu ', asset_Q , desired_R ,desired_transaction , xalloc);
180
181 MVO_portfolio_beta(1,1) = benchMVO_x(1,:)*Inf_Beta ';
182
183 [cumul_benchMVO cumul_modelMVO cumul_SP cumul_MF1 cumul_MF2] = ...
184     MVO_comparison(benchMVO_x(1,:) ', modelMVO_x(1,:) ', future_asset_r ,
    future_Market_r ,...
185     future_MK1_r ,future_MK2_r ,syear+eachyear ,eyear);
186
187 obj_vals = [modelMVO_var benchMVO_x(1,:)*temp_Q*benchMVO_x(1,:) '
188     modelMVO_x(1,:)*asset_Q*modelMVO_x(1,:) ' benchMVO];
189
190
191 %

```

---

```

192 %Using our model version of the Sharpe ratio , Preface M
193
194 [P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu , asset_Q , temp_Q
    , ...
195     modelMVO_x(1,:) , benchMVO_x(1,:) ,modelMVO_var ,currriskfreeprices);
196
197

```

```

198 if (num_of_reop>0)
199
200     MVO_returns_reb(1:T_reb)= ...
201         future_asset_r(1:T_reb,:)*benchMVO_x(1,:)';
202     inf_returns_reb(1:T_reb) = ...
203         future_asset_r(1:T_reb,:)*modelMVO_x(1,:)';
204
205
206
207     for p = 1:num_of_reop
208
209         % RECALCULATE RELEVANT PARAMETERS AND REOPTIMIZE
210
211         markov_periods = 10;
212
213         beg_indices(1) = beg_indices(1) + T_reb;
214         end_indices(1) = end_indices(1) + T_reb;
215
216
217
218         currassetprices=asset_prices(beg_indices(1):end_indices(1),:);
219         currmarketprices=market_prices(beg_indices(1):end_indices(1));
220         currinfprices=inf_prices(beg_indices(1):end_indices(1));
221         currriskfreeprices=riskfree_prices(beg_indices(1):end_indices(1)+1);
222
223
224         [expected_inf, tnodes,tnodeval]=...
225             new_exp_inf2(markov_periods,futureinfprices(T_reb*p+1),...
226                 curr_regime,c1,c2,ar1,ar2,p11,p12,p21,p22);
227
228         [expected_inf_var] = ...
229             new_exp_infvar2(markov_periods,curr_regime,tnodeval,...
230                 var1,var2,p11,p12,p21,p22);
231
232         asset_prices_with_inf=[currinfprices currassetprices];
233
234         [Inf_Beta R2_inf] =solve_beta3(asset_prices_with_inf,1);
235
236         currinfprices2=currinfprices(1:end-1);
237
238         [asset_mu,asset_Q,asset_r]= solve_mvo_params(currassetprices,1,size(
239             currassetprices,1));
240
241         [modelMVO_x(p+1,:) modelMVO_var MVO_adjret_diagQ nom_ret] = ...
242             main_MVO(currinfprices2,asset_r,expected_inf/100,expected_inf_var
243                 ,..., Inf_Beta',desired_R,desired_transaction,modelMVO_x(p
244                     ,:)');
245
246         model_portfolio_beta(p+1,1) = modelMVO_x(p+1,:)*Inf_Beta';

```

```

246 [benchMVO_x(p+1,:) benchMVO] = ...
247     benchmark_MVO(asset_mu', asset_Q, desired_R, desired_transaction,
248         benchMVO_x(p,:) ');
249 MVO_portfolio_beta(p+1,1) = benchMVO_x(p+1,:)*Inf_Beta';
250 MVO_returns_reb(T_reb*p+1:min((p+1)*T_reb, size((future_asset_r),1)))=
251     ...
252     future_asset_r((T_reb*p+1):...
253         min((p+1)*T_reb, size((future_asset_r),1)), :) ...
254         *benchMVO_x(p+1,:) ');
255 inf_returns_reb(T_reb*p+1:min((p+1)*T_reb, size((future_asset_r),1))) =
256     ...
257     future_asset_r(T_reb*p+1:min((p+1)*T_reb, size((future_asset_r),1))
258         ,:) ...
259         *modelMVO_x(p+1,:) ');
260 [P_SRATIOS(:,p+1) S_SRATIOS(:,p+1)]=...
261     calculateSharpeRatio(asset_mu, asset_Q, temp_Q, ...
262         modelMVO_x(p+1,:), benchMVO_x(p+1,:), modelMVO_var,
263         curriskfreeprices);
264
265 end
266
267 cumul_MVO_reb(1) = MVO_returns_reb(1);
268 cumul_inf_reb(1) = inf_returns_reb(1);
269
270 T = length(MVO_returns_reb);
271
272 for i = 2:T
273     cumul_MVO_reb(i) = (1+cumul_MVO_reb(i-1))*(1+MVO_returns_reb(i))
274         - 1;
275     cumul_inf_reb(i) = (1+cumul_inf_reb(i-1))*(1+inf_returns_reb(i)) -
276         1;
277 end
278
279 figure
280 plot(1:T,cumul_MVO_reb*100, '-b');
281 hold all
282 plot(1:T,cumul_inf_reb*100, '-r');
283 hold all
284 plot(1:length(cumul_SP),cumul_SP*100, '-g');
285 hold all
286 plot(1:T,cumul_MF1*100, '-m');
287 hold all
288 plot(1:T,cumul_MF2*100, '-c');
289
290 h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500', ...
291     'Vanguard Wellington Inv', 'CGM Mutual Fund'};

```



```

290         h = legend(h);
291
292         grid on;
293
294         title(['Comparing Cumulative Returns of Optimal Rebalanced Portfolios
                and Market'...
                ,num2str(syear+eachyear), ' to ', num2str(eyear)]);
295
296
297         xlabel('Time (in months)')
298         ylabel('Cumulative Returns in %')
299     end
300     portfolio_betas = [model_portfolio_beta MVO_portfolio_beta];
301     clear model_portfolio_beta
302     clear MVO_portfolio_beta
303     toc
304     %

```

---

```

305 % STEP 10. AT THE END OF THE LOOP, PLOT THE PORTFOLIO'S PERFORMANCE OVER TIME
        AND COMPARE IT TO
306 %     HOW THE STANDARD S&P500 INDEX DID (PLOT BOTH ON SAME GRAPH)
307 rmpath('m_Files');
308 rmpath('data_Files');
309
310 %Comment/uncomment the end below if you want to see/not see all the workspace
        variables
311 % end

1  SP500_symb= {'AA'
2  'ABC'
3  'ABT'
4  'LOMMX'
5  'VWELX'
6  '^GSPC'
7  };
8  SP500_symb=SP500_symb';
9
10 SP500_symb_csv={'AA.csv'
11 'ABC.csv'
12 'ABT.csv'
13 'LOMMX.csv'
14 'VWELX.csv'
15 '^GSPC.csv'
16 };
17 SP500_symb_csv=SP500_symb_csv';

1  function [Spec_Out p11 p22 p12 p21 var1 var2 var3 ar1 ar2 ar3 c1 c2 c3] = ...
2      RegimeSwitching_MLE(k, inf_data)
3
4      disp('Computing Parameters of Regime Switching Model');
5      addpath(' ../../MS_Regress_FEX_1.08/m_Files'); % add 'm_Files'
        folder to the search path

```

```

6      addpath( '..../MS_Regress_FEX_1.08/data_Files' );
7
8      logRet=inf_data; % load some Data.
9
10     % Defining dependent variables in system (solving for two
        dependent variables)
11     dep=logRet(:,1);
12     nLag=1; % Number of lags in system
13     k=k; % Number of States
14     doIntercept=1; % add intercept to
        eqsuations
15     %%
16     advOpt.diagCovMat=0 %if
        this value is 1, then only the elements on the diagonal(
        the variances) are estimated
17     %%
18     advOpt.distrib='Normal'; % The Distribution
        assumption (only 'Normal' for MS VAR models) )this is the
        default avlue
19     advOpt.std_method=1; % Defining the method for
        calculation of standard errors. See pdf file for more
        details
20
21     Spec_Out=MS_VAR_Fit(dep,nLag,k,doIntercept,advOpt);
22     disp('COMPLETED MS_VAR_FIT function');
23
24     % make the transition probabilities easier to call
25     % where pij is probability of going to regime j from regime i
26     p11=Spec_Out.Coeff.p(1,1);
27     p22=Spec_Out.Coeff.p(2,2);
28     p12=Spec_Out.Coeff.p(2,1);
29     p21=Spec_Out.Coeff.p(1,2);
30
31     % The variances associated with each regime
32     var1=cell2mat(Spec_Out.Coeff.covMat(1,1));
33     var2=cell2mat(Spec_Out.Coeff.covMat(1,2));
34
35     %autoregressive terms for each regime
36     %Have to use two lines to properly index values
37     ar1=Spec_Out.Coeff.S_Param(1,1);
38     ar1=ar1{1}(2,1);
39
40     ar2=Spec_Out.Coeff.S_Param(1,1);
41     ar2=ar2{1}(2,2);
42
43     %intercepts for each regime (These are the mean inflation
        rates for the regime)
44     c1=Spec_Out.Coeff.S_Param(1,1);
45     c1=c1{1}(1,1);
46

```

```

47         c2=Spec_Out.Coeff.S_Param(1,1);
48         c2=c2{1}(1,2);
49
50         %Different return values depending on whether there are 2 or 3
           regimes
51         if (k==2)
52             t=num2cell(zeros(1,8));
53             [p13,p23,p33,p32,p31,var3,ar3,c3]=deal(t{:});
54
55         elseif (k==3)
56             p13=1-p11-p12;
57             p23=1-p22-p21;
58             p33=Spec_Out.Coeff.p(3,3);
59             p32=Spec_Out.Coeff.p(2,3);
60             p31=Spec_Out.Coeff.p(1,3);
61             var3=cell2mat(Spec_Out.Coeff.covMat(1,3));
62             ar3=Spec_Out.Coeff.S_Param(1,1);
63             ar3=ar2{1}(2,3);
64             c3=Spec_Out.Coeff.S_Param(1,1);
65             c3=c3{1}(1,3);
66         end
67
68     end

1  % This function returns the inflation rates used to compute inflation betas
2
3  function [infmmonth infyear infprice] = all_inflation_data(filename)
4
5      [infmmonth infyear infprice] = textread(filename, '%s %d %f', 'delimiter
           ',',',');
6
7  end

1  function [rfmmonth rfyear rfprice] = all_riskfree_data(filename)
2
3      %[infmmonth, infyear, infprice] = deal(cell(1,1));
4
5
6      [rfmmonth rfyear rfprice] = ...
7          textread(filename, '%s %d %f', '
           delimiter',',',');
8
9  end

1  function [month day year price fail_symbols success_symbols] = ...
2      all_stock_data(SP500_symb_csv, SP500_symb)
3
4
5
6
7      [month, day, year, price, fail_symbols, success_symbols]=deal( cell(1,1));
8

```

```

9         fc=0;
10        sc=1;
11        for i=1:length(SP500_symb_csv)
12            try
13                curr_csv=char(SP500_symb_csv(i));
14                curr_symb=char(SP500_symb(i));
15                success_symbols{sc}=curr_symb;
16
17                [month{sc} day{sc} year{sc} price{sc}] = ...
18                textread(curr_csv, '%s %d %d %f', 'delimiter', ',,: ');
19
20                sc=sc+1;
21
22            catch
23                curr_symb=char(SP500_symb(i));
24                fc=fc+1;
25                fail_symbols{fc}=curr_symb;
26            end
27        end
28
29        %BY THE END OF THIS LOOP WE HAVE DAILY TIME SERIES DATA
30        %NOW ITERATE THROUGH MONTH VECTOR AND DELETE EVERY ELEMENT
31        % WHICH HAS THE SAME MONTH
32
33        % price_names=[];
34
35        % for (i=1:length(success_symbols))
36        %     price_names=[price_names; success_symbols(i)];
37        % end
38
39        % success_symbols=price_names;
40
41
42
43
44
45
46
47 end

```

1 %NOTE FORMAT OF INPUT MONTH IS SUPPOSED TO

```

2 function [infprices beg_ind end_ind] = fetch_inflation_data2(smonth, year,
    emonth, year, ...
3

```

inf

```

4
5     x=strmatch(smonth, infmonth);

```

```

6
7     for i=1:length(x)
8         if infyear(x(i))==syear
9             beg_ind=x(i);
10        end
11    end
12    x=strmatch(emonth,infmonth);
13    for i=1:length(x)
14        if infyear(x(i))==eyear
15            end_ind=x(i);
16        end
17    end
18
19    inftimeprices=flipud(data(beg_ind:end_ind));
20
21
22 end

1 %returns inflation data used to compute the RS model. It may be either
2 %monthly, quarterly or annually depending on the filename passed
3 function [inf_data inf_data_avg] = ...
4         fetch_regime_inflation_data(s_year,e_year,s_month,e_month,
5                                     filename)
6
7     s_year=s_year-1914;
8     e_year=e_year-1914;
9     inf_data=csvread(filename,s_year,s_month,[s_year,s_month,e_year,
10         e_month]);
11
12     inf_data=inf_data';
13     inf_data=inf_data(:);
14
15     %inf_data_avg=csvread(filename,s_year,13,[s_year,13,e_year,13]);
16     inf_data_avg=csvread...
17     (filename,s_year,e_month+1,[s_year,e_month+1,e_year,e_month+1]);
18 end

1 %NOTE FORMAT OF INPUT MONTH IS SUPPOSED TO
2 function [riskfreeprices] = fetch_riskfree_data2(smonth,syear,emonth,eyear,...
3
4
5     x=strmatch(smonth,infmonth);
6
7     for i=1:length(x)

```

```

8         if infyear(x(i))==syear
9             beg_ind=x(i);
10        end
11    end
12    x=strmatch(emonth,infmonth);
13    for i=1:length(x)
14        if infyear(x(i))==eyear
15            end_ind=x(i);
16        end
17    end
18
19    inftimeprices=flipud(data(beg_ind:end_ind));
20
21
22 end

1 %INPUT PARAMETERS
2 % month, day, year, price represent all the time series data
3 % succes_symbols represents the assets that succesfully
4 % FOR NOW ASSUME ROW 3000 IS THE SAME DATE FOR EVERY ASSET
5 function [prices price_names2 marketprice num_assets catch_assets ...
6           interval_size] = ...
7           SEC_fetch_stock_data(smonth,syear,emonth,eyear,month,day,year,data,
8                               success_symbols)
9
10 %————CODE TO FIND THE STARTING AND ENDING INDICES —————
11     x=strmatch(smonth,month{1});
12     for i =1:length(x)
13         if year{1}(x(i))==syear
14             beg_ind=x(i);
15         end
16     end
17
18     x=strmatch(emonth,month{1});
19     % disp(emonth);
20     % disp(month{1});
21     % disp(x);
22     % disp(eyear);
23     % disp(year);
24     disp(eyear);
25     for i =1:length(x)
26         disp(year{1}(x(i)));
27         if year{1}(x(i))==eyear
28             end_ind=x(i);
29         end
30     end
31     interval_size=end_ind-beg_ind;
32 %————AFTER FINDING THE INDICES ITERATE THROUGH ASSETS TO GET PRICE
33     DATA ———
34     count=1;
35     %beg_ind=beg_ind-1;

```

```

35     %end_ind=end_ind-1;
36     catch_assets=1;
37     %The last element
38
39     for i = 1:(length(success_symbols)-1)
40         %symb_matrix{i}=SP500_symb_csv{i};
41         try
42             % disp('entered try block')
43             prices{count}=flipud(data{i}(beg_ind:end_ind));
44             price_names{count}=success_symbols(i);
45             count=count+1;
46             %disp(SP500_symb_csv(i))
47         catch
48             catch_assets=catch_assets+1;
49         end
50     end
51     num_assets=count-1; % add 1 for the market index
52
53     %retrieve time series price information for the marketprice
54     try
55         marketprice=data{end}(beg_ind:end_ind);
56         marketname=success_symbols(end);
57     catch
58     end
59
60     marketprice=flipud(marketprice);
61
62
63     price_names2=[];
64
65     for (i=1:length(price_names))
66         price_names2=[price_names2; price_names{i}];
67         % disp(price_names2)
68     end
69
70
71
72 end

1 % This function generates the markov tree and comptues the expected inflation
   raet
2
3 %curr_regime is 0 or 1
4 function [inf_rate ,tnodes ,tnodeval3] = new_exp_inf2(n,y0,curr_regime ,c1 ,c2 ,ar1
   ,ar2 ,p11 ,p12 ,p21 ,p22)
5
6     % Compute an associated binary value for each termianl node which
       represents
7     % the unique pathe taken to get to that node
8     tnodes=terminal_nodes(n);
9

```

```

10      % compute that actual terminal inflation values conditional on the
      path taken
11      tnodeval=terminal_inflation(n,tnodes,c1,c2,ar1,ar2,y0);
12      tnodeval2{n}=tnodeval;
13      maxelements=2^n;
14
15      % iterate over every time period and compute the values for each node
16      for i=n:-1:1
17          tnodelist=terminal_nodes(i);
18          a=terminal_inflation(i,tnodes2,c1,c2,ar1,ar2,y0);
19          % disp(i);
20          % disp(a);
21          tnodeval3(i,:)=[a zeros(1,maxelements-2^i)];
22      end
23
24
25      % transition probability matrix
26      p= [p11 p12
27          p21 p22];
28
29      inf_rate=0;
30      for i=1:2^n
31          currprod=tnodeval(i);
32          cnode=tnodes{i};
33          %for the n-1 transition probabilities
34          for j=n:-1:2
35              ind1=str2num(cnode(j))+1;
36              ind2=str2num(cnode(j-1))+1;
37              currprod=currprod*p(ind2,ind1);
38          end
39          %for the initial transition probability
40          ind3=str2num(cnode(1))+1;
41          currprod=currprod*p(curr_regime,ind3);
42
43          inf_rate=inf_rate+currprod;
44
45      end
46
47
48  end
49
50  % This function is used to compute the expected variance of the inflation rate
51  function [inf_var inf_var_node_val] =...
52      new_exp_infvar2(n,start_regime,tnodeval,var1,var2,p11,p12,p21,p22)
53
54
55
56
57
58
59
60
61      p= [p11 p12
62          p21 p22];
63
64      % For every terminal node, assign either variance of regime 1 or
65      variance of regime 2
66      % as its value

```



```

11     for i=1:2^n
12         if (mod(i,2)==0)
13             initialvarval(i)=var1;
14         elseif (mod(i,2)==1)
15             initialvarval(i)=var2;
16         end
17     end
18
19
20     infvar_node_val=initialvarval;
21
22     % Iterate over every time period and call the recursive function
23     new_infvar_nodeval
24     % until you reach the initial time period
25     for i=n:-1:1
26         infvar_node_val=...
27             new_infvar_nodeval(i,p,infvar_node_val,tnodeval(i,:),
28                 start_regime);
29     end
30     inf_var=infvar_node_val(1);
31 end
32
33 % This is the recursive function that computes the expected inflation
34 % variance
35 % associated with each node
36 function [nodeval] = new_infvar_nodeval(n,p,varval,muinfval,start_regime)
37
38     n=n-1;
39
40     % Determines the two child nodes of a node and uses those values to
41     % get
42     % the variances used in the expectation formula
43     for i=1:2^n
44         var2(i)=varval(2*i-1);
45         var1(i)=varval(2*i);
46         mudiff(i)=(muinfval(2*i-1)-muinfval(2*i))^2;
47     end
48
49     % This is the base case and you use either p(1,1) and p(1,2) or
50     % p(2,1) and p(2,2) depending on the starting regime
51     if n==0
52         if (start_regime==1)
53             prob=p(1,1);
54
55         elseif (start_regime==2)
56             prob=p(2,1);
57
58         end
59         nodeval(i)=prob*var1(i)+(1-prob)*var2(i)+prob*(1-prob)*mudiff(
60             i);
61     end

```

```

27         else
28             % This is the recursive case but the node value can still be
                computed
29             % at that point.
30             for i=1:2^n
31                 if mod(i,2) == 0
32                     prob=p(1,1);
33                 else
34                     prob=p(2,1);
35                 end
36                 nodeval(i)=prob*var1(i)+(1-prob)*var2(i)+prob*(1-prob)
                    *mudiff(i);
37             end
38             % end
39         end
40     end

1  % This function applies the closed form equation to compute the terminal
2  % value associated with each node in the markov tree for expected inflation
3  % This equation is simply the result of recursion(due to the autoregressive
    term)
4  % and is unique for each node.
5  function [tnodeval] = terminal_inflation(n,tnodes,c1,c2,ar1,ar2,y0)
6
7      %{
8      a0=c1;
9      a1=c2;
10     b0=ar1;
11     b1=ar2;
12     beta1 and beta2 represent the betas for regime 1 and 2
13     %}
14     a=[c1 c2];
15     b=[ar1 ar2];
16
17     %iterates over each terminal node
18     tnodeval = [];
19     for i=1:2^n
20         cnode=tnodes{i};
21         cnodeval = 0;
22         %For a given terminal node, iterates over each time step
23         for j=n:-1:1
24             %period=n-j+1;
25             if (j==n)
26                 % disp('Value of a() is ');
27                 % a(str2num(cnode(n))+1)
28                 cnodeval = cnodeval + a(str2num(cnode(n))+1);
29             else
30                 ind_j=str2num(cnode(j))+1;
31                 currprod=a(ind_j);
32
33                 % over number of terms for a given time step
34                 for k = j+1:n;

```

```

35         % disp('Value of a() is ');
36         % a(str2num(cnode(n))+1)
37         ind_k=str2num(cnode(k))+1;
38         currprod = currprod*b(ind_k);
39     end
40     cnodeval=cnodeval+currprod;
41 end
42 if (j==1)
43     currprod=1;
44     for k= 1:n
45         ind_k=str2num(cnode(k))+1;
46         currprod=currprod*b(ind_k);
47     end
48     cnodeval=cnodeval+currprod*y0;
49 end
50 end
51 tnodeval(i)=cnodeval;
52 end
53
54 end

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

1 % This function takes in the total months and total intervals
2 % and returns matrices of equally spaced interval(except the last one
3 % may be slightly longer in the case of numintervals not being a factor
4 % of total months.
5 function [beg_ind end_ind] = divide_interval(numintervals,totalmonths)
6
7     beg_ind=[];
8     end_ind=[];
9     indice_size=floor(totalmonths/numintervals);
10
11     for i=1:numintervals-1
12         beg_ind=[beg_ind; indice_size*(i-1)+1];
13         end_ind=[end_ind; indice_size*i];
14     end
15     %if mod(totalmonths,numintervals)>0
16
17         beg_ind=[beg_ind; (indice_size*(numintervals-1)+1)];
18         end_ind=[end_ind; (indice_size*numintervals+mod(totalmonths,
19             numintervals))];
20
21     % end
22 end

```

```

1 %input parameters are # of states , smoothingprobabilities ,and no. of time
  periods
2 %The function returns a matrix of the resulting regime for each period
3 %and counts the total type of each regime for the historical inflation data
4
5 %The regime associated with each time is ultimately the greater of the two
6 % probabilities of that time period belonging to that regime
7 function [whichregime , countregime] = ...
8         regimecount(k,smoothProb,timelength)
9
10
11
12 %Note that the size of Spec_Out.filtProb is (# of time periods) x (#
  of states)
13 %IMPORTANT: NOTE THAT IT Is likely to change .filtProb to .smoothProb
14 for i=1:length(smoothProb(1,:))
15     r_prob(:,i)=smoothProb(:,i);
16 end
17
18
19 %initializes countregime matrix of size k(no. of states)
20 countregime=zeros(1,k);
21
22 for i=1:timelength
23     if (k==3)
24         if (r_prob(i,1)>=r_prob(i,2) && r_prob(i,1)>=r_prob(i
25             ,3))
26             whichregime(i)=1;
27             countregime(1)=countregime(1)+1;
28         elseif (r_prob(i,2)>=r_prob(i,1) && r_prob(i,2)>=
29             r_prob(i,3))
30             whichregime(i)=2;
31             countregime(2)=countregime(2)+1;
32         else
33             whichregime(i)=3;
34             countregime(3)=countregime(3)+1;
35         end
36     elseif (k==2)
37         if (r_prob(i,1)>=r_prob(i,2))
38             whichregime(i)=1;
39             countregime(1)=countregime(1)+1;
40         else
41             whichregime(i)=2;
42             countregime(2)=countregime(2)+1;
43         end
44     end
45 end
46
47 % This function returns the appropriate inflation fil
48 % for monthly, quarterly or annual data

```

```

3
4 function [inf_file endcol] = return_inflation_file(inflation_interval)
5     if (strcmp(inflation_interval, 'monthly'))
6         inf_file='inflation_data_monthly.csv';
7         endcol=12;
8     elseif (strcmp(inflation_interval, 'quarterly'))
9         inf_file='inflation_data_quarterly.csv';
10        endcol=4;
11    elseif (strcmp(inflation_interval, 'annually'))
12        inf_file='inflation_data_annual.csv';
13        endcol=1;
14    end
15 end

1 %% STANDARD MARKOWITZ MVO MODEL VIA QUADPROG
2 % Input the number of assets, the range of desired returns, the mean vector
3 % for the assets, and the covariance matrix. The function will return the
4 % optimal portfolio weights in MVO_x, and the corresponding variance
5 % function value in MVO_var.
6 %
7 % [optimal weights, corresponding optimal objective function]
8 % = sMVO(# of assets, range of desired returns, expected return vector,
9 %        covariance matrix)
10
11 % Set quadprog parameters
12 function [ MVO_x MVO_var] = benchmark_MVO( mu, Q, return_range,
13     transaction_cost, previous_portfolio)
14
15     n= length(mu);
16
17     c = [ zeros(3*n,1) ];
18     Aeq = [ ones(1,n) zeros(1,2*n)
19            eye(n) -eye(n) eye(n) ];
20     beq = [1; previous_portfolio];
21     A=[-mu' transaction_cost*ones(1,2*n)];
22     lb=zeros(3*n,1);
23     ub=9999*ones(3*n,1);
24     R = return_range;
25
26     tempQ=Q;
27     Q = [Q zeros(n,2*n); zeros(2*n, 3*n)];
28
29     % Set quadprog options
30     options = optimset('Algorithm', 'interior-point-convex', 'TolFun',
31         1/10^10, 'MaxIter', 1000, 'TolCon', 1/(10^10));
32
33     %Solve MVO and store SD values for plotting
34
35     %Modify this so that it works for variable length R (length>1)
36     for i = 1:length(R);

```

```

37
38         b=[-R(i); ];
39
40         [MVO_x(i,:), MVO_var(i,1)] = quadprog(Q, c, A, b,Aeq, beq, lb, ub,
41         [], options);
42
43         %MVO_std = MVO_var.^.5;
44         MVO_x=MVO_x(i,1:n);
45     end
46
47     adjvector = [MVO_x' mu diag(tempQ)];
48 end
49
50 function [ MVO_x MVO_var adjvector nominal_return temp_Q] = main_MVO(
51     historical_inflation, historical_asset_returns,...
52     expected_inflation, inflation_variance, infbeta, R, transaction_cost,
53     previous_portfolio)
54     % expected_inflation, inflation_variance, infbeta, R, transaction_cost
55     % , max_transactions, previous_portfolio)
56
57     %{
58     historical_inflation: time series data for inflation values over the
59     relevant period
60
61     **IN DECIMAL FORM**
62
63     historical_asset_returns: time series data for relevant assets over
64     relevant periods
65
66     expected_inflation: inflation calculated through markov RS model
67
68     inflation_variance: inflation variance calculated through markov RS
69
70     beta: vector of n beta values corresponding to each asset for the current
71     regime
72
73     %}
74
75     r_M = historical_inflation';
76     r_it = historical_asset_returns;
77     mu_M = expected_inflation;
78     del_M = inflation_variance;
79
80     [T, n] = size(r_it);
81     c = [zeros(3*n,1)];
82     Aeq = [ones(1,n) zeros(1,2*n)
83            eye(n) -eye(n) eye(n)];
84     beq = [1; previous_portfolio];
85     lb=zeros(3*n,1);
86
87     ub=9999*ones(3*n,1);
88
89     nominal_return = prod(1+r_it).^(1/T) - 1;

```

```

37     nominal_return = nominal_return';
38     % disp(size(nominal_return));
39     % disp(size(infbeta));
40     % disp(size(muM));
41     adjusted_return = nominal_return + infbeta*muM;
42     alpha = nominal_return(1:end)-infbeta*muM;
43     % disp(nominal_return(1:end))
44     % disp(infbeta:muM)
45     % disp(size(r_it(:,1)));
46     % disp(size(alpha(1)));
47     % disp(size(infbeta(1)));
48     for i=1:n
49         epsi(:,i)=r_it(:,i)-(alpha(i)+infbeta(i)*r_M(:,1));
50     end
51     % infbeta=infbeta.^(-1);
52     del_epsilon=diag(cov(epsi));
53     for i = 1:n;
54         for j = 1:n;
55             if i==j
56                 Q_sf(i,i)=infbeta(i)^2*del_M+del_epsilon(i);
57             else
58                 Q_sf(i,j)=infbeta(i)*infbeta(j)*del_M;
59             end
60         end
61     end
62
63     temp_Q = Q_sf;
64
65     Q_sf = [Q_sf zeros(n,2*n); zeros(2*n, 3*n)];
66
67     options = optimset('Algorithm', 'interior-point-convex', 'TolFun',
        1/10^10, 'MaxIter', 1000, 'TolCon', 1/(10^10));
68
69     % Solve Single Factor model and store SD values
70     disp('-----')
71     disp(size(adjusted_return'));
72     disp(size(zeros(1,2*n)));
73     disp(size(zeros(1,n)));
74     disp(size(ones(1,2*n)));
75
76     for i = 1:length(R);
77         A=[-(adjusted_return') transaction_cost*ones(1,2*n)];
78         %zeros(1,n) transaction_cost * ones(1,2*n)];
79         b=[-R(i); ]%max_transactions];
80
81
82         [MVOx(i,:), fval(i,1)] = quadprog(Q_sf, c, A,b, Aeq, beq, ...
83             lb,ub,[], options);
84         MVO_var(i,1)=(MVOx(i,:) * Q_sf * MVOx(i,:)');
85         MVO_x=MVO_x(i,1:n);
86

```

```

87     end
88
89     adjvector = [MVO_x' adjusted_return diag(temp_Q) infbeta];
90
91 end

1  % This function takes in the asset allocation of standard MVO and our custom
   inflation
2  % hedged model. It also tak
3
4  %% MVO_COMPARISON
5  %{
6      Take in the two MVO weightings, one for standard MVO and the other for
7      inflation hedged MVO, then show their projected returns over the sample
8      period, alongside the market returns for that period.
9
10     This function can be modified accordingly if more benchmark models are
11     added.
12
13     MVO_x: vector of weights corresponding to standard MVO w/ transactions
14     inf_x: vector of weights corresponding to our model
15     asset_returns: relevant for the relevant assets over the sample
16     period
17     market_returns: market returns
18  %}
19  function [cumul_MVO cumul_inf cumul_SP cumul_MF1 cumul_MF2] = MVO_comparison(
   MVO_x, inf_x, projected_returns, ...
20     market_returns, MF_returns1, MF_returns2, startyear, endyear)
21
22
23     [T n] = size(projected_returns);
24
25
26     MVO_returns = projected_returns*MVO_x;
27     inf_returns = projected_returns*inf_x;
28
29     cumul_MVO(1) = MVO_returns(1);
30     cumul_inf(1) = inf_returns(1);
31     cumul_SP(1) = market_returns(1);
32     cumul_MF1(1) = MF_returns1(1);
33     cumul_MF2(1) = MF_returns2(1);
34
35
36     for i = 2:T
37         cumul_MVO(i) = (1+cumul_MVO(i-1))*(1+MVO_returns(i)) - 1;
38         cumul_inf(i) = (1+cumul_inf(i-1))*(1+inf_returns(i)) - 1;
39         cumul_SP(i) = (1+cumul_SP(i-1))*(1+market_returns(i)) - 1;
40         cumul_MF1(i) = (1+cumul_MF1(i-1))*(1+MF_returns1(i)) - 1;
41         cumul_MF2(i) = (1+cumul_MF2(i-1))*(1+MF_returns2(i)) - 1;
42     end
43
44     portfolio_MVO=1 * (1 + cumul_MVO);

```



```

45     portfolio_inf=1 * (1 + cumul_inf);
46     portfolio_SP=1 * (1 + cumul_SP);
47
48     figure
49     plot(1:T,cumul_MVO*100, '-b');
50     hold all
51     plot(1:T,cumul_inf*100, '-r');
52     hold all
53     plot(1:T,cumul_SP*100, '-g');
54     hold all
55     plot(1:T,cumul_MF1*100, '-m');
56     hold all
57     plot(1:T,cumul_MF2*100, '-c');
58
59 % plot(1:T,currinfprices, '-m');
60
61 h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500', ...
62     'Vanguard Wellington Inv', 'CGM Mutual Fund'};
63 h = legend(h);
64
65 grid on;
66
67 title(['Comparing Cumulative Returns of Optimal Portfolios and Market'...
68     ,num2str(startyear), ' to ', num2str(endyear)])
69 xlabel('Time (in months)')
70 ylabel('Cumulative Monthly Returns (in %)')
71
72
73 figure
74 plot(1:T,MVO_returns, '-b');
75 hold all
76 plot(1:T,inf_returns, '-r');
77 hold all
78 plot(1:T,market_returns, '-g');
79 hold all
80 % plot(1:T,currinfprices, '-m');
81
82 h = {'Standard MVO', 'Inflation Hedged SF', 'S&P500', 'inflation rate'};
83
84 title(['Comparing Return Values of Optimal Portfolios and Market'...
85     ,num2str(startyear), ' to ', num2str(endyear)])
86 xlabel('Time (in months)')
87 ylabel('Monthly Returns')
88
89
90
91 h = legend(h);
92
93 grid on;
94
95 hold all

```

```

96
97
98 end

1 %This function is passed in the pricedata for ONE asset
2 function [ mu, Q, r_it ] = mvo_params(pricedata, end_pred)
3 %% DETERMINE EXPECTED RETURNS AND COVARIANCES FOR ASSETS
4 % Input a matrix of time series data for the desired assets, as well as
5 % the number of days in the estimation horizon starting from day 1, and the
6 % function will return the expected returns and covariances calculated from
7 % the time series data from the estimation horizon.
8 %
9 % [expected returns, covariance matrix] = param_data(time series,
10 %                                                    estimation horizon)
11
12 r_it = (data(2:end_pred,:) ./ data(1:end_pred-1,:)) - 1;
13 [T, n] = size(r_it); %time periods, assets
14 mu = prod(1+r_it).^(1/T) - 1;
15 Q = cov(r_it);
16
17 end

1 function [ nominal_return, nominal_Q] = nominal_parameters(asset_data,
    market_data)
2 % take in the relevant time series data for assets and market
3 % and return the nominal return and covariances for assets
4
5 % Model: Single factor CAPM
6
7
8
9     r_it = (asset_data(2:end,:) ./ asset_data(1:end-1,:)) - 1;
10    r_M = (market_data(2:end,:) ./ market_data(1:end-1,:)) - 1;
11
12    [T, n_assets] = size(r_it);
13
14    nominal_return = prod(1+r_it).^(1/T) - 1;
15    mu_M = prod(1+r_M).^(1/T) - 1;
16
17    del_M=sum((r_M(:,1) - mu_M(1)).^2)/T; % Factor variance
18    beta=(sum(r_it(:,1:end).*repmat(r_M(:,1),1,n_assets)))/T-mean(r_it(:,1:
    end))*...
19        mean(r_M(:,1)))/del_M);
20    alpha = nominal_return(1:end)-beta*mu_M;
21
22    %Noise vector
23    for i=1:n_assets
24        epsi(:,i)=r_it(:,i)-(alpha(i)+beta(i)*r_M(:,1));
25    end
26    del_epsi=diag(cov(epsi));
27
28    % Single factor covariance

```

```

29     for i = 1:n_assets;
30         for j = 1:n_assets;
31             if i==j
32                 nominal_Q(i,i)=beta(i)^2*del_M+del_epsilon(i);
33             else
34                 nominal_Q(i,j)=beta(i)*beta(j)*del_M;
35             end
36         end
37     end
38 end

1 % This function is used to compute the inflation Beta, but it can also
2 % compute the CAPM beta if desired. The second parameter passed in can either
3 % be
4 % inflation rate date or market data
5 function [solved_beta R_squared] = solve_beta3(tprice,inf_or_market)
6
7 %MARKET - CAPM
8 if (inf_or_market==2)
9     r_it=tprice(2:end,:) ./ tprice(1:end-1,:)-1;
10 %MARKET - INFLATION
11 elseif (inf_or_market==1)
12     r_it=tprice(2:end,2:end) ./ tprice(1:end-1,2:end)-1;
13     [T n] = size(r_it)
14     disp(size(tprice(:,1)))
15     r_it=[tprice(1:T,1) r_it];
16 end
17
18 [T, n]=size(r_it); %number of time
19 period T and asset n
20 mu=prod(1+r_it).^(1/T)-1; %Geometric mean
21 for factor and assets
22 del_M=sum((r_it(:,1)-mu(1)).^2)/T; %variance of
23 factor under geometric mean
24
25 %Calculate the Beta coefficient
26
27 if inf_or_market==1
28     for i = 2:n
29         p = polyfit(r_it(:,1)./100,r_it(:,i),1);
30         solved_beta(i-1) = p(1);
31
32         yfit = polyval(p,r_it(:,1)./100);
33         y = r_it(:,i);
34         yresid = y - yfit;
35         SSresid = sum(yresid.^2);
36         SStotal = (length(y)-1) * var(y);
37         R_squared(i-1) = 1 - SSresid/SStotal;
38     end
39
40 else
41     for i = 2:n

```

```

38     p = polyfit(r_it(:,1), r_it(:,i), 1);
39     solved_beta(i - 1) = p(1);
40
41     yfit = polyval(p, r_it(:,1));
42     y = r_it(:,i);
43     yresid = y - yfit;
44     SSresid = sum(yresid.^2);
45     SStotal = (length(y)-1) * var(y);
46     R_squared(i - 1) = 1 - SSresid/SStotal;
47 end
48
49
50 end

1 %This function takes in a matrix of asset prices ,
2
3 function [mu, Q, r_it] = solve_mvo_params(asset_prices , beg_pred , end_pred)
4
5     data=asset_prices;
6     r_it = (data((beg_pred+1):end_pred ,:) ./ data(beg_pred:end_pred-1,:)) -
7         1;
8     [T, n] = size(r_it);
9     mu = prod(1+r_it).^(1/T) - 1;
10    Q = cov(r_it);
11 end

1 % This function computes the sharpe ratio and modified sharpe ratio computed
2 % for each portfolio
3 function [P_SRATIOS S_SRATIOS] = ...
4     calculateSharpeRatio(asset_mu , asset_Q , temp_Q , modelMVO_x ,
5     benchMVO_x , ...
6     modelMVO_var ,
7     currriskfreeprices
8     )
9
10 %Using our model version of the Sharpe ratio , Preface M
11
12 [P_premodel_sratis] = ...
13     (asset_mu*modelMVO_x' - currriskfreeprices(end)/100)/(
14     modelMVO_var^0.5);
15
16
17 [P_preMVO_sratis P_preMVO_smu P_preMVO_s_sigmap]=...
18     sharperatio2(asset_mu , temp_Q , benchMVO_x , currriskfreeprices(end)
19     /100);
20
21 %
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

---

```

16 %Using regular version of the Sharpe ratio , Preface S
17 [S_premodel_sratis S_premodel_smu S_premodel_s_sigmap]=...

```

```

18         sharperatio2(asset_mu, asset_Q, modelMVO_x, currriskfreeprices(end)
           /100);
19
20     [S_preMVO_sratisos S_preMVO_smu S_postMVO_sigmap]=...
21         sharperatio2(asset_mu, asset_Q, benchMVO_x, currriskfreeprices(end)
           /100);
22
23     P_SRATIOS = [P_premodel_sratisos; P_preMVO_sratisos];
24     S_SRATIOS = [S_premodel_sratisos; S_preMVO_sratisos];
25 end
26
27 % [P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu, asset_Q,
           temp_Q, ...
28 %     modelMVO_x(1,:), benchMVO_x(1,:), modelMVO_var, currriskfreeprices);

1 %Takes in the asset prices for the desired periods
2 function [sratio mu_p sigma_p] = ...
3     sharperatio2(asset_mu, asset_cov, xalloc, riskfreeprice)
4
5
6     mu_p=asset_mu*xalloc';
7     sigma_p=(xalloc*asset_cov*xalloc')^0.5;
8
9     sratio=(mu_p-riskfreeprice)/sigma_p;
10
11 end
12
13 % [P_SRATIOS(:,1) S_SRATIOS(:,1)]=calculateSharpeRatio(asset_mu, asset_Q, ...
14 %     modelMVO_x, benchMVO_x, modelMVO_var, currriskfreeprices);

```