December 10th 2018

Krishnachaitanya Pullakandam
Student ID - 200280960

OR 506 Project

Problem statement: Best way to select a portfolio in order to:

1. a) Maximize Returns &
2. b) Minimize Risk (or) Variance

Step1: Import the data (returns on different instruments A,B,C,D for a period of 10 years)

A	B	C	D
0.63%	0.66%	1.07%	2.34%
0.15%	0.76%	-6.84%	-7.53%
1.86%	-2.48%	2.88%	8.76%
3.56%	-5.51%	3.20%	3.15%
10.11%	1.12%	11.81%	10.39%
9.11%	0.19%	-1.12%	-10.09%
9.81%	78.91%	-0.12%	-1.21%
10.09%	9.12%	1.23%	9.78%
6.70%	7.81%	7.91%	7.82%
18.19%	9.11%	8.12%	10.12%

Step 2: For the above returns data calculated the Arithmetic mean, Geometric mean and Standard deviation of the returns.

Arithmetic Mean	7.02%	9.97%	2.81%	3.35%
Standard Deviation	5.60%	24.72%	5.35%	7.50%
Geometric Mean	6.89%	8.08%	2.69%	3.10%

Step 3: Based on the mean values, calculated in Step 2 and the actual values of Step 1, calculated the excess returns (x-μ):

Excess Returns	A	B	C	D
	-6.39%	-9.31%	-1.74%	-1.01%
	-6.87%	-9.21%	-9.65%	-10.88%
	-5.16%	-12.45%	0.06%	5.41%
	-3.46%	-15.48%	0.39%	-0.20%
	3.09%	-8.85%	9.00%	7.04%
	2.09%	-9.78%	-3.94%	-13.44%
	2.79%	68.94%	-2.93%	-4.56%

3.07%	-0.85%	-1.58%	6.43%
-0.32%	-2.16%	5.10%	4.47%
11.17%	-0.86%	5.31%	6.77%

Step 4: Based on the Excess Returns calculated in Step 3, computed the Variance-Covariance matrix:

Variance-Covariance	A	B	C	D
A	0.00282612	0.00373572	0.0013997	0.00128848
B	0.00373572	0.05498918	-0.0015929	-0.0022085
C	0.0013997	-0.0015929	0.00054508	0.00285206
D	0.00128848	-0.0022085	0.00285206	0.00506861

Step 5: Based on the variance-covariance matrix generated the returns and risk functions making alpha1 and alpha2 as variables (which signifies the risk appetite of an investor) and varied them ranging from 0 to 1 for each of the alpha values.

Step 6: Generated the penalty function, using the constraints as mentioned below:

Constraint1: x1+x2+x3+x4-1 = 0 (sum of all the weights should equal to one)

Constraint2: x1 >= 0, x2 >= 0, x3 >= 0, x4 >= 0 (each weight should individually be >= 0)

Portfolio Returns function (f1) = transpose(weights)*geoMean;

Portfolio Variance function =
0.5 * transpose(weights) * varianceCovarianceMatrix * weights;

Final penaltyFunction = alpha1*f1 + alpha2*f2 + mu*(x1+x2+x3+x4-1)^2 + 0.001*mu*(x1^2 + x2^2 + x3^2 + x4^2); where the weigts alpha1 and alpha2 represents the weights which signifies the risk appetite of an investor.

Step 7: Once the penalty function is generated, used Cauchy's steepest Descent Method to minimize the function.

Note: The coefficient of the penalty functions are taken as (mu, mu/1000) for the constraints 1 & 2, keeping in mind, that the first constraint needs to be more tightly enforced as compared to the second constraint. (In real life, negative weights represent the possibility that an investor can short sell an instrument, without investing in it directly)

Results:

Run Time: Total elapsed time is 36.929123 seconds.

(Alpha1, Alpha2) -> (0,1)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44765	0.34713	-0.052122	-0.052126
2	100	0.5465	0.44571	0.046926	0.046929
3	1000	0.51873	0.41761	0.019365	0.019371
4	10000	0.52649	0.42505	0.027333	0.027343
5	1e+05	0.52428	0.42252	0.02533	0.025345
6	1e+06	0.52487	0.42279	0.026126	0.026144
7	1e+07	0.52389	0.42021	0.026175	0.026214
8	1e+08	0.52479	0.4208	0.027286	0.027329

(Alpha1, Alpha2) -> (0.1,0.9)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44777	0.34732	-0.052077	-0.052073
2	100	0.5466	0.44594	0.046883	0.046899
3	1000	0.51887	0.41792	0.019284	0.019309
4	10000	0.52664	0.42544	0.027197	0.027232
5	1e+05	0.52446	0.42298	0.025145	0.025191
6	1e+06	0.52507	0.42333	0.025889	0.025945
7	1e+07	0.52419	0.42112	0.025685	0.025792
8	1e+08	0.52511	0.42178	0.026744	0.026862

(Alpha1, Alpha2) -> (0.2,0.8)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44789	0.34752	-0.052031	-0.052021
2	100	0.5467	0.44616	0.04684	0.046868
3	1000	0.519	0.41824	0.019203	0.019248
4	10000	0.5268	0.42582	0.02706	0.027121
5	1e+05	0.52463	0.42344	0.024959	0.025036
6	1e+06	0.52526	0.42386	0.025651	0.025745
7	1e+07	0.52511	0.4235	0.02556	0.02567
8	1e+08	0.52543	0.42277	0.026202	0.026394

(Alpha1, Alpha2) -> (0.3,0.7)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44801	0.34771	-0.051985	-0.051968
2	100	0.54681	0.44639	0.046796	0.046838
3	1000	0.51914	0.41855	0.019123	0.019186
4	10000	0.52695	0.42621	0.026924	0.02701
5	1e+05	0.52481	0.42391	0.024773	0.024882
6	1e+06	0.52546	0.4244	0.025414	0.025545
7	1e+07	0.52532	0.42411	0.025272	0.025426
8	1e+08	0.52576	0.42375	0.025659	0.025926

(Alpha1, Alpha2) -> (0.4,0.6)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44813	0.34791	-0.05194	-0.051916
2	100	0.54691	0.44661	0.046753	0.046807
3	1000	0.51928	0.41887	0.019042	0.019124
4	10000	0.5271	0.42659	0.026787	0.026899
5	1e+05	0.52498	0.42437	0.024587	0.024727
6	1e+06	0.52565	0.42493	0.025177	0.025346
7	1e+07	0.52554	0.42472	0.024984	0.025182
8	1e+08	0.52608	0.42473	0.025117	0.025458

(Alpha1, Alpha2) -> (0.5,0.5)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44825	0.3481	-0.051894	-0.051863
2	100	0.54701	0.44684	0.04671	0.046777
3	1000	0.51941	0.41918	0.018961	0.019063
4	10000	0.52725	0.42698	0.026651	0.026788
5	1e+05	0.52516	0.42483	0.024401	0.024573
6	1e+06	0.52585	0.42547	0.024939	0.025146
7	1e+07	0.52576	0.42533	0.024696	0.024937
8	1e+08	0.5264	0.42572	0.024574	0.02499

(Alpha1, Alpha2) -> (0.6,0.4)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44837	0.34829	-0.051848	-0.05181
2	100	0.54711	0.44706	0.046666	0.046746
3	1000	0.51955	0.4195	0.01888	0.019001
4	10000	0.52741	0.42736	0.026514	0.026676
5	1e+05	0.52533	0.42529	0.024215	0.024418
6	1e+06	0.52604	0.42601	0.024702	0.024946
7	1e+07	0.52597	0.42594	0.024407	0.024693
8	1e+08	0.52672	0.42671	0.024031	0.024521

(Alpha1, Alpha2) -> (0.7,0.3)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44849	0.34849	-0.051803	-0.051758
2	100	0.54721	0.44729	0.046623	0.046716
3	1000	0.51968	0.41981	0.0188	0.018939
4	10000	0.52756	0.42775	0.026378	0.026565
5	1e+05	0.52551	0.42575	0.024029	0.024264
6	1e+06	0.52624	0.42654	0.024464	0.024746
7	1e+07	0.52599	0.42659	0.022738	0.023255
8	1e+08	0.52704	0.42769	0.023487	0.024052

(Alpha1, Alpha2) -> (0.8,0.2)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44861	0.34868	-0.051757	-0.051705
2	100	0.54731	0.44751	0.04658	0.046685
3	1000	0.51982	0.42013	0.018719	0.018878
4	10000	0.52771	0.42813	0.026241	0.026454
5	1e+05	0.52568	0.42622	0.023843	0.024109
6	1e+06	0.52643	0.42708	0.024227	0.024546
7	1e+07	0.52629	0.4275	0.022246	0.022832
8	1e+08	0.52736	0.42868	0.022943	0.023583

(Alpha1, Alpha2) -> (0.9,0.1)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44873	0.34887	-0.051712	-0.051653
2	100	0.54742	0.44774	0.046536	0.046655
3	1000	0.51996	0.42044	0.018638	0.018816
4	10000	0.52786	0.42852	0.026105	0.026343
5	1e+05	0.52586	0.42668	0.023657	0.023954
6	1e+06	0.52663	0.42762	0.023989	0.024346
7	1e+07	0.52659	0.42842	0.021754	0.022408
8	1e+08	0.52768	0.42967	0.022399	0.023113

(Alpha1, Alpha2) -> (1,0)

Iterations	Mu_Value	x_1	x_2	x_3	x_4
1	10	0.44885	0.34907	-0.051666	-0.0516
2	100	0.54752	0.44796	0.046493	0.046624
3	1000	0.52009	0.42076	0.018557	0.018754
4	10000	0.52802	0.4289	0.025968	0.026231
5	1e+05	0.52603	0.42714	0.023471	0.0238
6	1e+06	0.52682	0.42815	0.023752	0.024146
7	1e+07	0.52689	0.42933	0.021261	0.021984
8	1e+08	0.528	0.43066	0.021855	0.022643

Matlab Code:

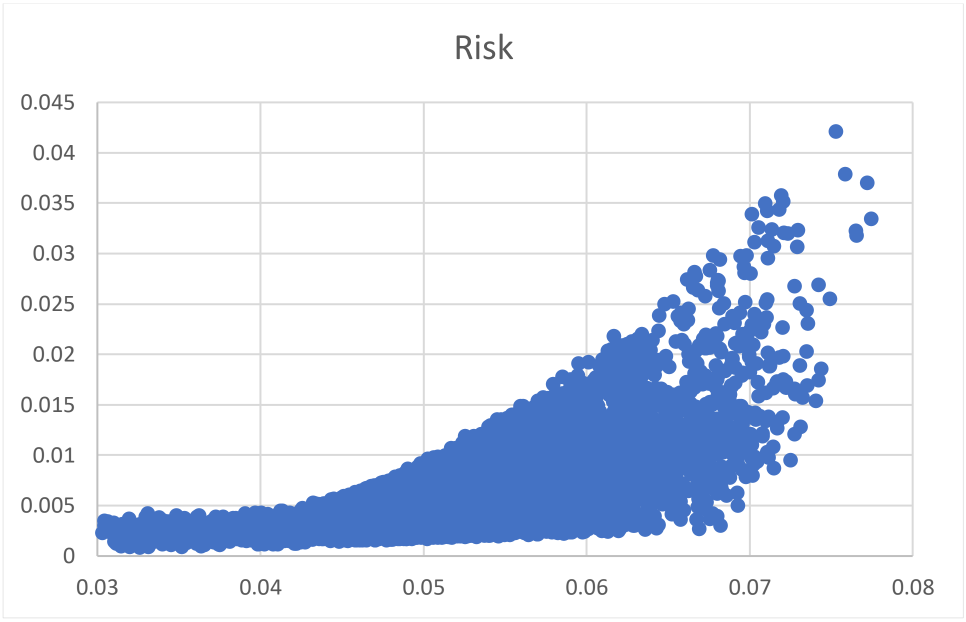
```
% Input of the returns of different stocks over a period of 10 years
stockA_returns = [0.0063 0.0015 0.01861 0.0356 0.1011 0.0911 0.0981 0.1009
0.0670 0.1819];
stockB_returns = [0.0066 0.00762 -0.0248 -0.0551 0.0112 0.0019 0.7891
0.0912 0.0781 0.0911];
stockC_returns = [0.0107 -0.0684 0.02876 0.0320 0.1181 -0.01123 -0.00121
0.01231 0.0791 0.0812];
stockD_returns = [0.0234 -0.0753 0.08761 0.0315 0.1039 -0.1009 -0.0121
0.0978 0.0782 0.1012];

stock_returns = [stockA_returns; stockB_returns; stockC_returns;
stockD_returns];

% geometricMean of the returns of the stock of a company over years
geoMean = zeros(4,1);
for i = 1:4
    geoMean(i) = findGeoMean(stock_returns(i,:));
end

% arithmeticMean of the returns of the stock of a company over years
arithmeticMean = zeros(4,1);
for i = 1:4
    arithmeticMean(i) = mean(stock_returns(i,:));
end

stdDevA = std(stockA_returns)
stdDevB = std(stockB_returns)
stdDevC = std(stockC_returns)
stdDevD = std(stockD_returns)

excessReturnsA = findExcessReturns(stockA_returns, arithmeticMean(1));
excessReturnsB = findExcessReturns(stockB_returns, arithmeticMean(2));
excessReturnsC = findExcessReturns(stockC_returns, arithmeticMean(3));
excessReturnsD = findExcessReturns(stockD_returns, arithmeticMean(4));

excessReturns = [excessReturnsA; excessReturnsB; excessReturnsC;
excessReturnsD];

varianceCovarianceMatrix = excessReturns*excessReturns';
varianceCovarianceMatrix = 0.1*varianceCovarianceMatrix; % Divinding the
elements of the variance-covariance matrix with 10 the number of
observations

syms x1 x2 x3 x4 mu % these four variables represent the weights associated
to the portfolio

weights = [x1; x2; x3; x4]; % this is a matrix which holds the portfolio
weights

portfolioReturn = transpose(weights)*geoMean;
portfolioVariance =
0.5*transpose(weights)*varianceCovarianceMatrix*weights;

%defined the objective functions:
f1 = -portfolioReturn;
f2 = portfolioVariance;
```

```
% alpha is the weight which tells us the relative importance of
% risk/reward we are targeting at
% (eg; some one might think that they have to give 80% relative importance
% to risk over returns; someone might think they have to give 50-50
% importance to both risk and returns).

for alphas = 0:.1:1

%     alphaFirst = alpha1(weightCombination);
%     alphaSecond = alpha2(weightCombination);

    alphaFirst = alphas;
    alphaSecond = 1-alphas;

    X = ['Alpha1 is -> ', num2str(alphaFirst),' Alpha2 is -> ',
num2str(alphaSecond)];
    disp(X);

    % constraints: x1+x2+x3+x4 = 1
    % generating the penalty function which would be the input to the
exterior penalty method
    objectiveFunction = alphaFirst*f1 + alphaSecond*f2 + mu*(x1+x2+x3+x4-
1)^2 + 0.001*mu*(x1^2 + x2^2 + x3^2 + x4^2);

    x0 = [0.80;0.70;0.30;0.30];
    n = 1;
    epsilon = 10^-6;
    x_new = x0;
    x_old = x0;

    mu_value = 10; % Initialization value for mu
    T = table;
    while mu_value < 10^8

        mu_value = 10^n;
        for counter = 1:100
            if counter ~= 1 && (findSmallEnough(x_old,x_new,epsilon))
                break;
            end

            objectiveFunction = subs(objectiveFunction, mu, mu_value);
            grad_f = gradient(objectiveFunction);
            dk = -
subs(grad_f,{x1,x2,x3,x4},{x_old(1),x_old(2),x_old(3),x_old(4)});

            alpha = 10^-4;
            neta = 0.9;
            lambda = 1/5;

            x_new = x_old + lambda*dk;

            f_new =
subs(objectiveFunction,{x1,x2,x3,x4},{x_new(1),x_new(2),x_new(3),x_new(4)})
;
            g_new =
subs(grad_f,{x1,x2,x3,x4},{x_new(1),x_new(2),x_new(3),x_new(4)});
```

```
            f_old = 
subs(objectiveFunction,{x1,x2,x3,x4},{x_old(1),x_old(2),x_old(3),x_old(4)})
;
            g_old = 
subs(grad_f,{x1,x2,x3,x4},{x_old(1),x_old(2),x_old(3),x_old(4)});

            while (f_new - f_old) > (alpha*lambda*dk'*g_old) || (dk'*g_new)
< (neta*dk'*g_old)
                lambda = lambda/5;
                x_new = x_old + lambda*dk;
                f_new = 
subs(objectiveFunction,{x1,x2,x3,x4},{x_new(1),x_new(2),x_new(3),x_new(4)})
;
                g_new = 
subs(grad_f,{x1,x2,x3,x4},{x_new(1),x_new(2),x_new(3),x_new(4)});
            end

            x_new = x_new + lambda*dk; %-- this is a redundant step as we
already have the value we want
            x_old = x_new;

            x_new(1) = vpa(x_new(1),6);
            x_new(2) = vpa(x_new(2),6);
            x_new(3) = vpa(x_new(3),6);
            x_new(4) = vpa(x_new(4),6);
            f_new = subs(objectiveFunction,{x1,x2,x3,x4},{x_new(1),
x_new(2), x_new(3), x_new(4)});
            x_t = table(n, mu_value,
double(vpa(x_new(1),6)),double(vpa(x_new(2),6)), ...
                    double(vpa(x_new(3),6)),double(vpa(x_new(4),6))); %
,sum
            T = [T; x_t];
        end
        n = n + 1;
    end

    T.Properties.VariableNames = {'Iterations' 'Mu_Value' 'x_1' 'x_2' 'x_3'
'x_4'}; %  'Constraint_Equals_1'
    disp(T);
end

% this function is used to find the geometric mean of the rate of return
% for a particular stock over the years
function geometricMean = findGeoMean(stockA_returns)
    geometricMean = 1;
    for n = 1 : length(stockA_returns)
       geometricMean = geometricMean*(stockA_returns(n)+1);
    end
    geometricMean = geometricMean^(1/length(stockA_returns)) - 1;
end

% this function is used to find the excess returns matrix which is
% basically the value of (return - geometricMean of return)
function excessReturns = findExcessReturns(stockA_returns, geoMeanA)
    excessReturns = zeros(1,10);
    for n = 1 : length(stockA_returns)
        excessReturns(n) = stockA_returns(n) - geoMeanA;
    end
end
```

```
function smallEnough = findSmallEnough(x_old,x_new, epsilon)

    smallEnough = true;
    for i = 1:4
        if (abs((x_new(i)-x_old(i))/(x_old(i)))) < epsilon
            smallEnough = smallEnough*true;
        else
            smallEnough = false;
        end
    end
end
```

Pareto Front



Pareto Front / Efficient frontier

Note:

Pareto front (created using 10,000 combinations of weights generated using the rand() -> random number generation function in excel) and then finding the corresponding returns and std. deviation using the variance-covariance matrix which was computed in the above steps and then plotted them (x axis -> Standard Deviation; y axis -> Returns)