621 Final Exam

Submitted by:

Saeed Rahman

QuestionA

A) Analytical Price of an Geometric Asian Call Option

```
Geometric_Asian_Analytical<-function(S0, K, t, r,div=0, sigma){
  N=252*t
  sigma.new=sigma*sqrt((2*N+1)/(6*(N+1)))
  rho=1/2.0*(r-div-sigma*sigma/2.0+sigma.new*sigma.new)
  d1=1/(sqrt(t)*sigma.new)*(log(S0/K)+(rho+sigma.new*sigma.new/2.0)*t)
  d2=1/(sqrt(t)*sigma.new)*(log(S0/K)+(rho-sigma.new*sigma.new/2.0)*t)
  price=exp(-r*t)*( S0*exp(rho*t)*pnorm(d1) -K*pnorm(d2))
  return(price)
}
```

```
"Geometric Asian Call Option Price= 15.1711296805879"
```

Monte Carlo function to price Arithmetic and Geometric Asian call option

```
Asian_MC<-function(S0, K, t, r,div=0, sigma,n.sims=1000)
{
  N=252*t
  dt=1/252
  spot.steps={}
  sim.prices.arthimetic={}
  sim.prices.geometric={}

  mu=(r-div-(0.5*(sigma^2)))*dt
  sigma=sigma*sqrt(dt)

  for(i in 1:n.sims)
  {
    spot.steps[1]=S0
    for(j in 2:N)
    {
      spot.steps[j]=spot.steps[j-1]*exp(mu+sigma*rnorm(1))
    }
    arthimetic.price=mean(spot.steps)
    geometric.price=spot.steps^(1/N)
    geometric.price=prod(geometric.price)


    sim.prices.arthimetic[i]=max(arthimetic.price-K,0)
    sim.prices.geometric[i]=max(geometric.price-K,0)

  }
  arthimetic.price=mean(sim.prices.arthimetic)*exp(-r*t)
  geometric.price=mean(sim.prices.geometric)*exp(-r*t)
  print(paste("Arthimetic Asian Call Price=",arthimetic.price))
  print(paste("Geometric Asian Call Price=",geometric.price))
```

```
std.dev.arthimetic=sqrt((sum(sim.prices.arthimetic^2)-(sum(sim.prices.arthimetic)*mean(sim.prices.arthimetic)))
                        *(exp(-2*r*t)/(n.sims-1)))
std.error.arthimetic=std.dev.arthimetic/sqrt(n.sims)

std.dev.geometric=sqrt((sum(sim.prices.geometric^2)-(sum(sim.prices.geometric)*mean(sim.prices.geometric)))
                        *(exp(-2*r*t)/(n.sims-1)))

std.error.geometric=std.dev.geometric/sqrt(n.sims)


print(paste("Standard Deviation of Arthimetic Option=",std.dev.arthimetic))
print(paste("Standard Error of Arthimetic Option=",std.error.arthimetic))
print(paste("Standard Deviation of Geometric Option=",std.dev.geometric))
print(paste("Standard Error of Geometric Option=",std.error.geometric))

list(arthimetic.price=arthimetic.price,geometric.price=geometric.price,
     std.dev.arthimetic=std.dev.arthimetic, std.error.arthimetic=std.error.arthimetic,
     std.dev.geometric=std.dev.geometric,std.error.geometric=std.error.geometric,
     sim.prices.arthimetic=sim.prices.arthimetic, sim.prices.geometric=sim.prices.geometric)
}
```

```
asian.call.price=Asian_MC(S0 = 100,K = 100, r = .03, sigma = .3, t = 5, n.sims = 100000 )
```

Number of simulations used: 100,000 due to time constraint

B) & C) Monte Carlo Price of Arithmetic and Geometric Asian Option

```
start.time <- Sys.time()
asian.call.price=Asian_MC(S0 = 100,K = 100, r = .03, sigma = .3, t = 5, n.sims = 100000 )
end.time <- Sys.time()
time.taken <- end.time - start.time
print(paste("Time Taken=",time.taken,"Minutes"))
```

```
[1] "Arthimetic Asian Call Price= 17.3804073829305"
[1] "Geometric Asian Call Price= 15.0879474721449"
[1] "Standard Deviation of Arthimetic Option= 30.6788426950251"
[1] "Standard Error of Arthimetic Option= 0.097015018894297 8"
[1] "Standard Deviation of Geometric Option= 26.5415483797396"
[1] "Standard Error of Geometric Option= 0.0839317455075287"
[1] "Time Taken= 18.5628065307935 Minutes"
```

D) Calculating slop/coefficient “b”

```
X=asian.call.price$sim.prices.geometric*exp(-r*t)
Y=asian.call.price$sim.prices.arthimetic*exp(-r*t)

b=sum((X-mean(X))*(Y-mean(Y)))/(sum(X-mean(X)^2))
print(paste("Slope=",b))
```

```
[1] "Slope= -3.81263162181082"
```

E) Calculating the error

~~~
error=mean(X)-asian.geometric.call.price

print(paste("Error=",error))

```
~~~

```
[1] "Error= -0.0614626576526529"
```

F) Calculating the modified arithmetic option price

~~~
modified.arthimetic.price=mean(Y)-b*error

print(paste("Modified Arthimetic Price=",modified.arthimetic.price))
```
~~~

```
[1] "Modified Arthimetic Price= 17.1613746679464"
```

We can see that as M (number of replication) is increased the error starts to decrease and converge to a stable value.

G) Applying the function to an external (IBM US EQUITY) Asian Option price calculation

```r
library(quantmod)
getSymbols("IBM",from="2016-05-12", to="2017-05-12")
S0=coredata(IBM["2017-05-12",6])
df=read.csv("IBM_Melted.csv")
sigma=sd(periodReturn(IBM,period='daily',subset='2016-05-12::'))*sqrt(252)
df$Days_till_expirty=as.integer(as.Date(df[,1],format="%m/%d/%Y")-as.Date("05/12/17",format="%m/%d/%y"))
df$T=as.double((as.Date(df[,1],format="%m/%d/%Y")-as.Date("05/12/17",format="%m/%d/%y"))/252)
strike={}
time.to.maturity={}
geometric.price={}
arthemetic.price={}
slope={}
analytical.price={}
modified.arthimetic={}
error={}
for (i in 1:nrow(df)){
  price.details=Asian_MC_Market(S0=S0,K = df$Strike[i],t = df$T[i],r = 1.182/100,sigma = sigma)
  analytical.price[i]=price.details$analytical.price
  geometric.price[i]=price.details$geometric.price
  arthemetic.price[i]=price.details$arthimetic.price
  slope[i]=price.details$slope
  error[i]=price.details$error
  modified.arthimetic[i]=price.details$modified.arthimetic.price}
df$Analytical_Price=round(analytical.price,2)
df$Geometric_Price=round(geometric.price,2)
df$Arthemetic_Price=round(arthemetic.price,2)
df$Slope=round(slope,3)
df$Modified_Arthimetic_Price=round(modified.arthimetic,3)
df$Error=round(error,3)
write.csv(df, file = "ProblemA_Bonus_Result.csv")
```

ProblemA_Bonus_Result - Excel

File Home Insert Page Layout Formulas Data Review View Tell me what you want to do

N16

	A	B	C	D	E	F	G	H	I	J	K	L
1		Expiry	Strike	Original.Pr	Days_till_expirty	T	Analytical_Price	Geometric_Pric	Arthemetic_Price	Slope	Modified_Arthim	Error
2	1	6/12/2017	150.325	1.51	31	0.123015873	1.96	1.95	1.97	-4.331	1.897	-0.016
3	2	7/12/2017	150.325	2.18	61	0.242063492	2.77	2.69	2.73	-3.478	2.466	-0.077
4	3	8/12/2017	150.325	2.3	92	0.365079365	3.42	3.46	3.52	-3.083	3.647	0.041
5	4	9/12/2017	150.325	2.92	123	0.488095238	3.96	4.08	4.16	-2.749	4.489	0.118
6	5	10/12/2017	150.325	3.4	153	0.607142857	4.43	4.09	4.19	-3.066	3.154	-0.338
7	6	11/12/2017	150.325	3.37	184	0.73015873	4.87	4.82	4.95	-2.824	4.81	-0.05
8	7	6/12/2017	157.8413	0.05	31	0.123015873	0.14	0.17	0.18	6.143	-0.003	0.029
9	8	7/12/2017	157.8413	0.21	61	0.242063492	0.51	0.55	0.57	14.414	-0.028	0.041
10	9	8/12/2017	157.8413	0.31	92	0.365079365	0.92	0.95	0.99	166.681	-5.013	0.036
11	10	9/12/2017	157.8413	0.62	123	0.488095238	1.32	1.38	1.43	-24.3	2.837	0.058
12	11	10/12/2017	157.8413	0.9	153	0.607142857	1.69	1.82	1.89	-12.808	3.525	0.127
13	12	11/12/2017	157.8413	0.96	184	0.73015873	2.05	1.83	1.92	-13.199	-0.974	-0.219
14	13	6/12/2017	165.3575	0	31	0.123015873	0	0	0	NA	NA	-0.002
15	14	7/12/2017	165.3575	0.02	61	0.242063492	0.05	0.03	0.04	2.777	0.076	-0.015
16	15	8/12/2017	165.3575	0.03	92	0.365079365	0.16	0.22	0.23	9.758	-0.344	0.059
17	16	9/12/2017	165.3575	0.09	123	0.488095238	0.32	0.31	0.34	13.308	0.448	-0.008
18	17	10/12/2017	165.3575	0.17	153	0.607142857	0.5	0.42	0.45	16.97	1.838	-0.082
19	18	11/12/2017	165.3575	0.2	184	0.73015873	0.71	0.79	0.85	54.558	-3.581	0.081
20	19	6/12/2017	172.8737	0	31	0.123015873	0	0	0	NA	NA	0
21	20	7/12/2017	172.8737	0	61	0.242063492	0	0	0	0.683	0.002	-0.002
22	21	8/12/2017	172.8737	0	92	0.365079365	0.02	0.01	0.01	4.169	0.042	-0.007
23	22	9/12/2017	172.8737	0.01	123	0.488095238	0.06	0.03	0.04	6.5	0.187	-0.023
24	23	10/12/2017	172.8737	0.03	153	0.607142857	0.12	0.1	0.12	7.835	0.248	-0.017
25	24	11/12/2017	172.8737	0.04	184	0.73015873	0.2	0.18	0.2	14.984	0.54	-0.023

QuestionB

1) PCA

Collecting data from 2012 for XLF ETF

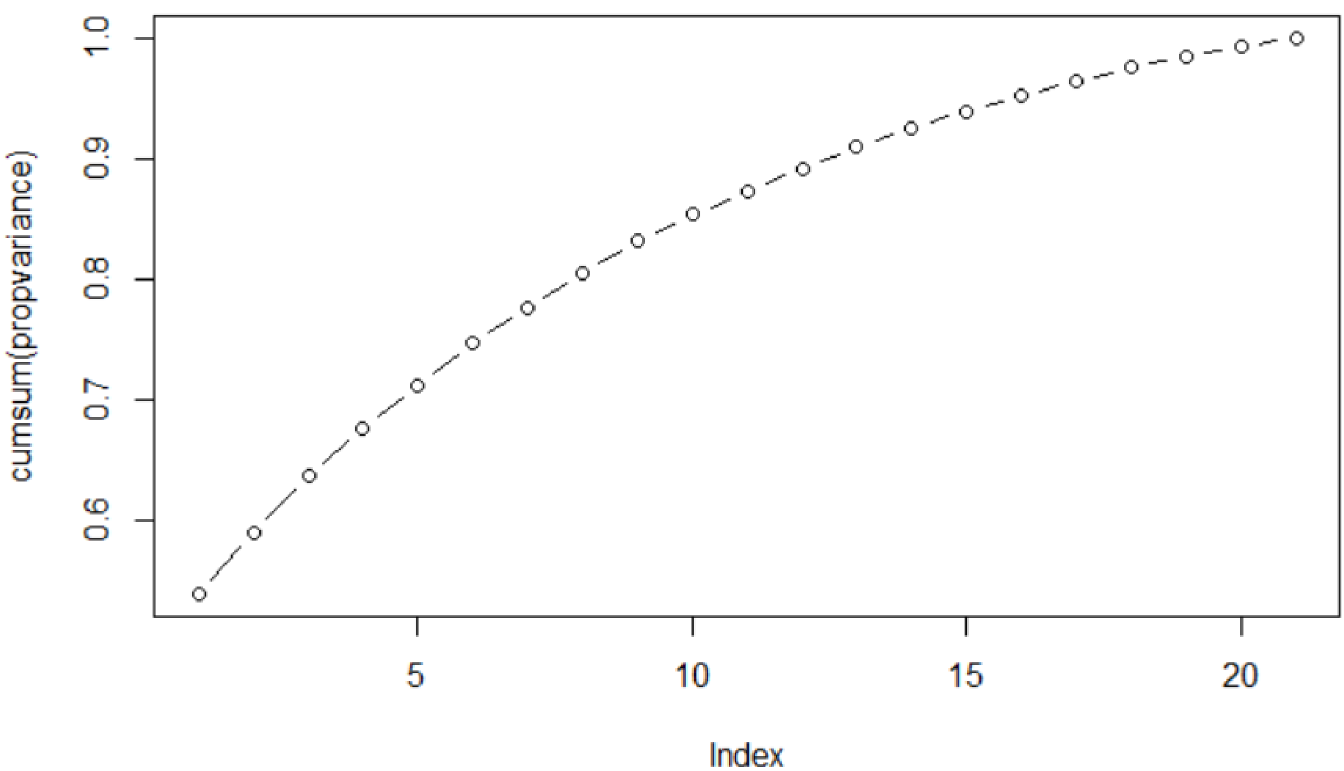
```
                    JPM          BRK          BAC          WFC            C             GS          USB
2012-01-03  0.0270111266  0.018691790  0.008695652  0.017537544  0.044231517  0.0257072171 -0.001086599
2012-01-04  0.0148084209  0.018348818  0.017513135  0.007762844  0.004636198  0.0001055104  0.002545455
2012-01-05  0.0279458665 -0.044643317  0.097391304  0.018245614  0.030730296  0.0067057480  0.019679263
2012-01-06 -0.0092462317  0.000000000 -0.004830918  0.003467441 -0.003838137 -0.0040512259 -0.011047720
2012-01-09 -0.0039503387 -0.009174409  0.001597444  0.005145763  0.012534854  0.0128356405  0.019862730
2012-01-10 -0.0005545051  0.091744090  0.029503106 -0.011096167  0.008403361  0.0099630340 -0.009116410
                     CB          MS         PNC          AXP           MET          AIG        SCHW           BK
2012-01-03 -0.0123577730  0.02030457 0.001696912  0.001863354 -0.0009354225  0.012620951 0.017376195  0.005392157
2012-01-04 -0.0042880933  0.01206349 0.007656985  0.004581466  0.0106682770 -0.001669407 0.006003431  0.009318343
2012-01-05  0.0083718535  0.04425920 0.013900644  0.020066869  0.0299719959  0.005044094 0.024935512  0.017156863
2012-01-06  0.0005715245 -0.01119403 0.005370029 -0.012277451 -0.0066424217 -0.021205739 0.030821918 -0.016826828
2012-01-09  0.0049999714  0.01250006 0.006331256  0.001448655 -0.0056802091  0.013941698 0.007481297  0.015564203
2012-01-10  0.0026908653  0.01319730 0.005717069 -0.005923223  0.0185731438  0.025777372 0.001627339  0.004716887
                    BLK          PRU          CME          COF           MMC          SPGI
2012-01-03 -0.0129593797 -0.003302934 -0.007234698  0.008296866 -0.0084138361  0.0008765505
2012-01-04  0.0022835087  0.013380559 -0.013022679  0.022988506 -0.0081967216 -0.0070360598
2012-01-05 -0.0009470641  0.028985507 -0.005501375  0.022171923 -0.0025616394  0.0202087502
2012-01-06 -0.0030609307  0.001900038 -0.005905094 -0.003090486 -0.0157000966  0.0045851311
2012-01-09 -0.0002788790 -0.010602083 -0.017773383  0.020745950  0.0032583902  0.0054336016
2012-01-10  0.0036728320  0.004135282 -0.007502679  0.016414411 -0.0009658081  0.0360164895
```

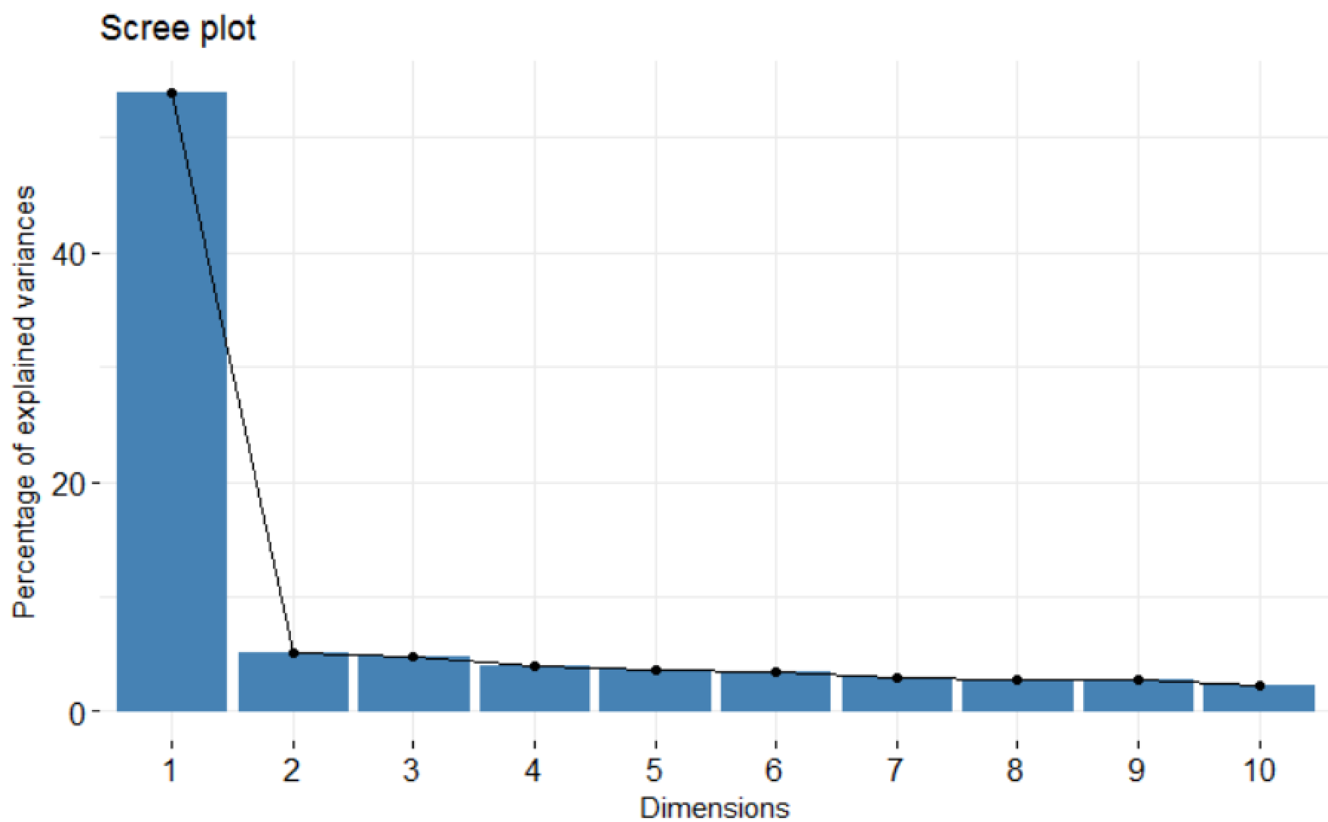
PCA Result

```
eigenvalues <- res.pca$eig
head(eigenvalues)
```

	eigenvalue <dbl>	percentage of variance <dbl>	cumulative percentage of variance <dbl>
comp 1	11.3431260	54.014886	54.01489
comp 2	1.0596860	5.046124	59.06101
comp 3	0.9977062	4.750982	63.81199
comp 4	0.8162306	3.886812	67.69880
comp 5	0.7466916	3.555675	71.25448
comp 6	0.7251213	3.452959	74.70744

Number of Components to account for 80% of the variability






```r
Prin.Comp = prcomp(ReturnMatrix, scale = T) #by default R centers the variables. Scale also makes then sd=1
summary(Prin.Comp)
propvariance = Prin.Comp$sdev^2/sum(Prin.Comp$sdev^2)
plot(cumsum(propvariance), type="b")
number_best_pca= which(cumsum(propvariance)>.8)[1]
print(paste("Number of Components by which 80% of the variance is captured=",number_best_pca))
```

```
> summary(Prin.Comp)
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6     PC7    PC8     PC9    PC10    PC11    PC12
Standard deviation     3.3680 1.02945 0.99884 0.90345 0.86411 0.85154 0.79303 0.7682 0.74838 0.67277 0.64687 0.62798
Proportion of Variance 0.5402 0.05047 0.04751 0.03887 0.03556 0.03453 0.02995 0.0281 0.02667 0.02155 0.01993 0.01878
Cumulative Proportion  0.5402 0.59061 0.63812 0.67699 0.71255 0.74708 0.77702 0.8051 0.83179 0.85335 0.87327 0.89205
                          PC13    PC14    PC15    PC16    PC17    PC18    PC19    PC20    PC21
Standard deviation     0.60195 0.57192 0.55236 0.51589 0.50044 0.48697 0.44286 0.41086 0.39212
Proportion of Variance 0.01725 0.01558 0.01453 0.01267 0.01193 0.01129 0.00934 0.00804 0.00732
Cumulative Proportion  0.90930 0.92488 0.93941 0.95208 0.96401 0.97530 0.98464 0.99268 1.00000
```

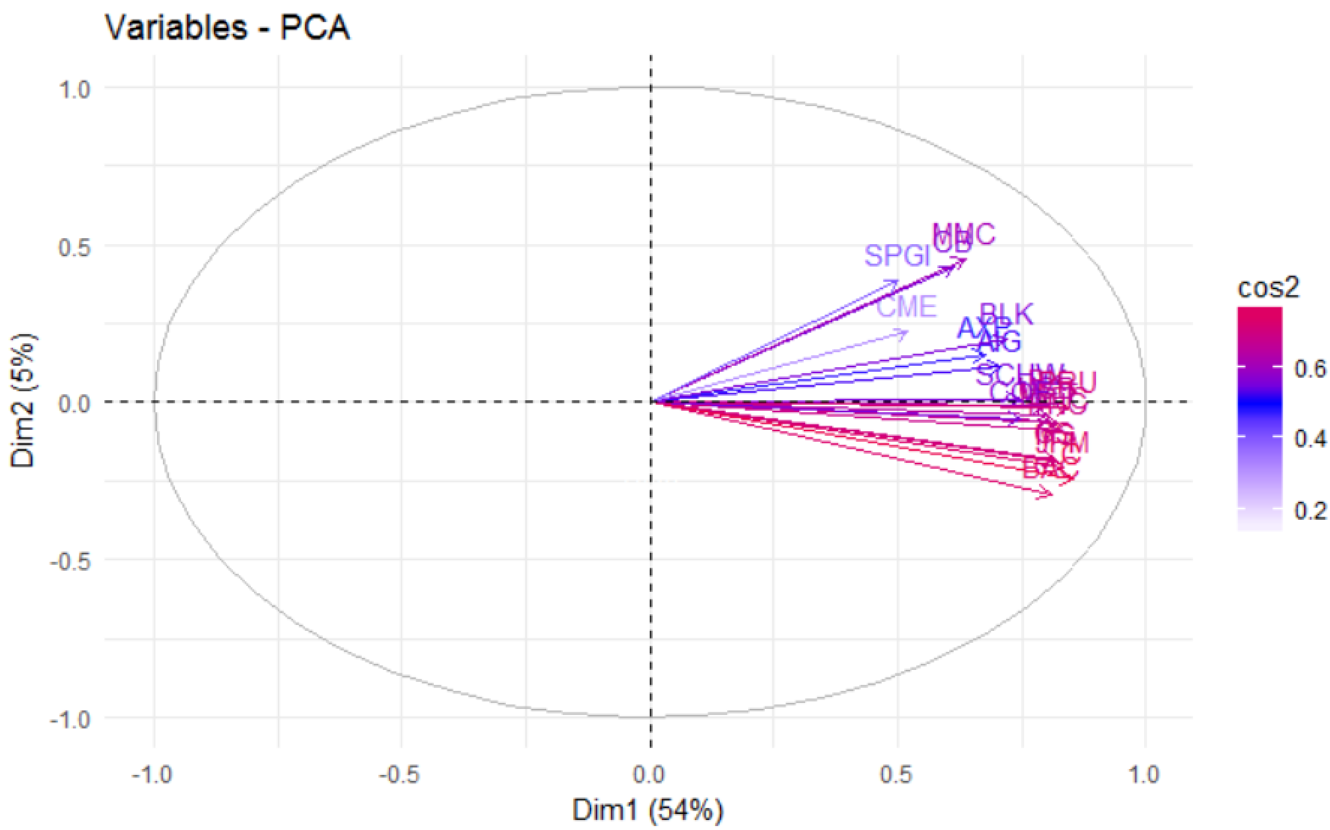
```
> print(paste("Number of Components by which 80% of the variance is captured=",number_best_pca))
[1] "Number of Components by which 80% of the variance is captured= 8"
```

Loadings of different Equities on PCA components

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
JPM	0.834564522	-0.20774654	-0.05803618	-0.08264403	-0.073583891
BRK	0.007969487	-0.35021616	0.93192386	0.07068780	-0.022986050
BAC	0.808761378	-0.29228451	-0.09460527	-0.06794028	0.021790044
WFC	0.821821827	-0.06684379	-0.03668620	-0.14463696	-0.046629284
C	0.854553497	-0.24602975	-0.04197345	-0.05993472	-0.010161952
GS	0.818825126	-0.18165027	-0.04019349	0.04931661	-0.001271263

The squared loading on the PCA components

	Dim.1	Dim.2	Dim.3	Dim.4	Dim.5
JPM	6.964979e-01	0.043158624	0.003368198	0.006830036	5.414589e-03
BRK	6.351273e-05	0.122651358	0.868482074	0.004996766	5.283585e-04
BAC	6.540950e-01	0.085430237	0.008950157	0.004615882	4.748060e-04
WFC	6.753911e-01	0.004468093	0.001345877	0.020919849	2.174290e-03
C	7.302617e-01	0.060530638	0.001761771	0.003592171	1.032653e-04
GS	6.704746e-01	0.032996822	0.001615517	0.002432128	1.616110e-06

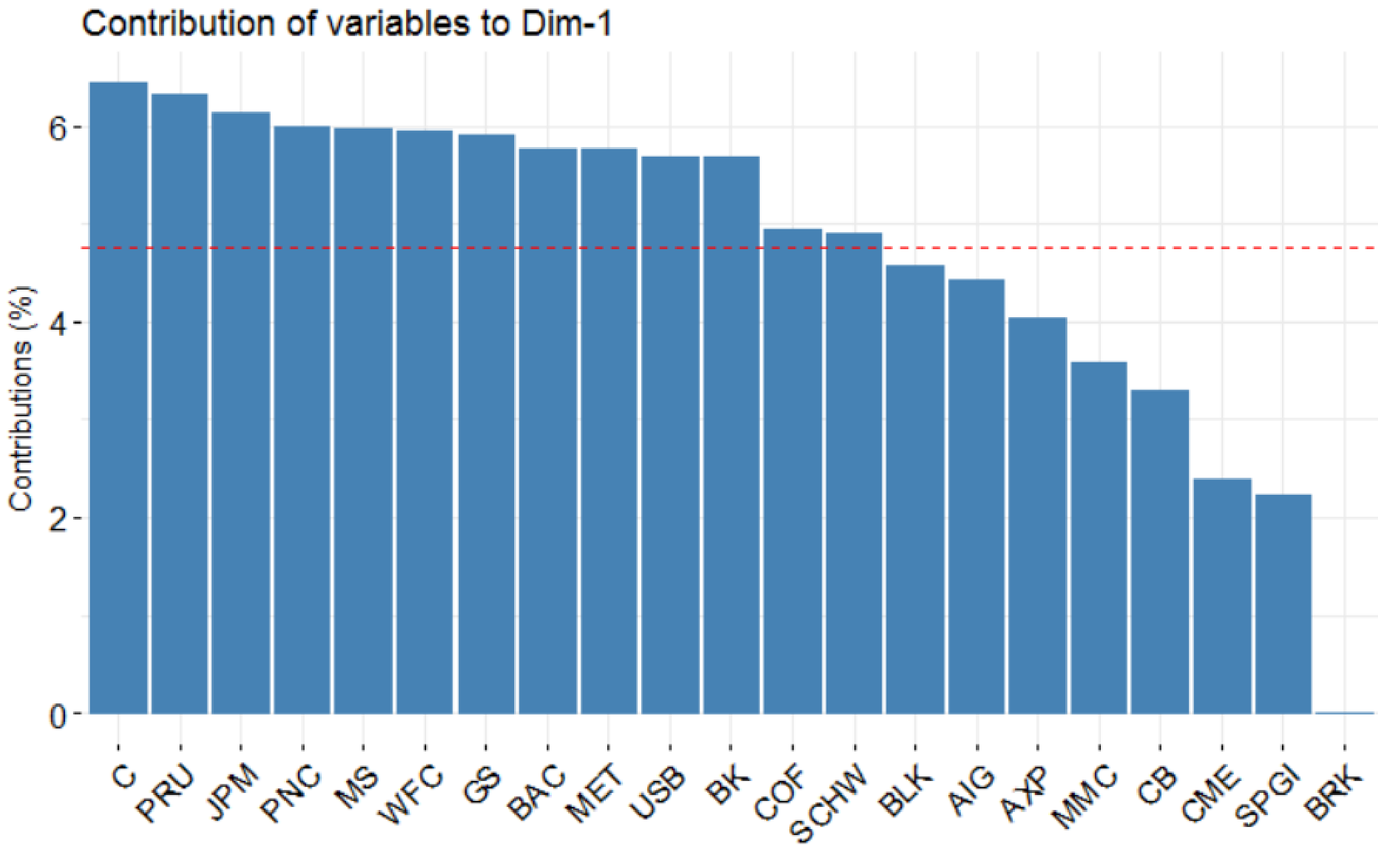


The sum of the cos2 for variables on the principal components is equal to one.

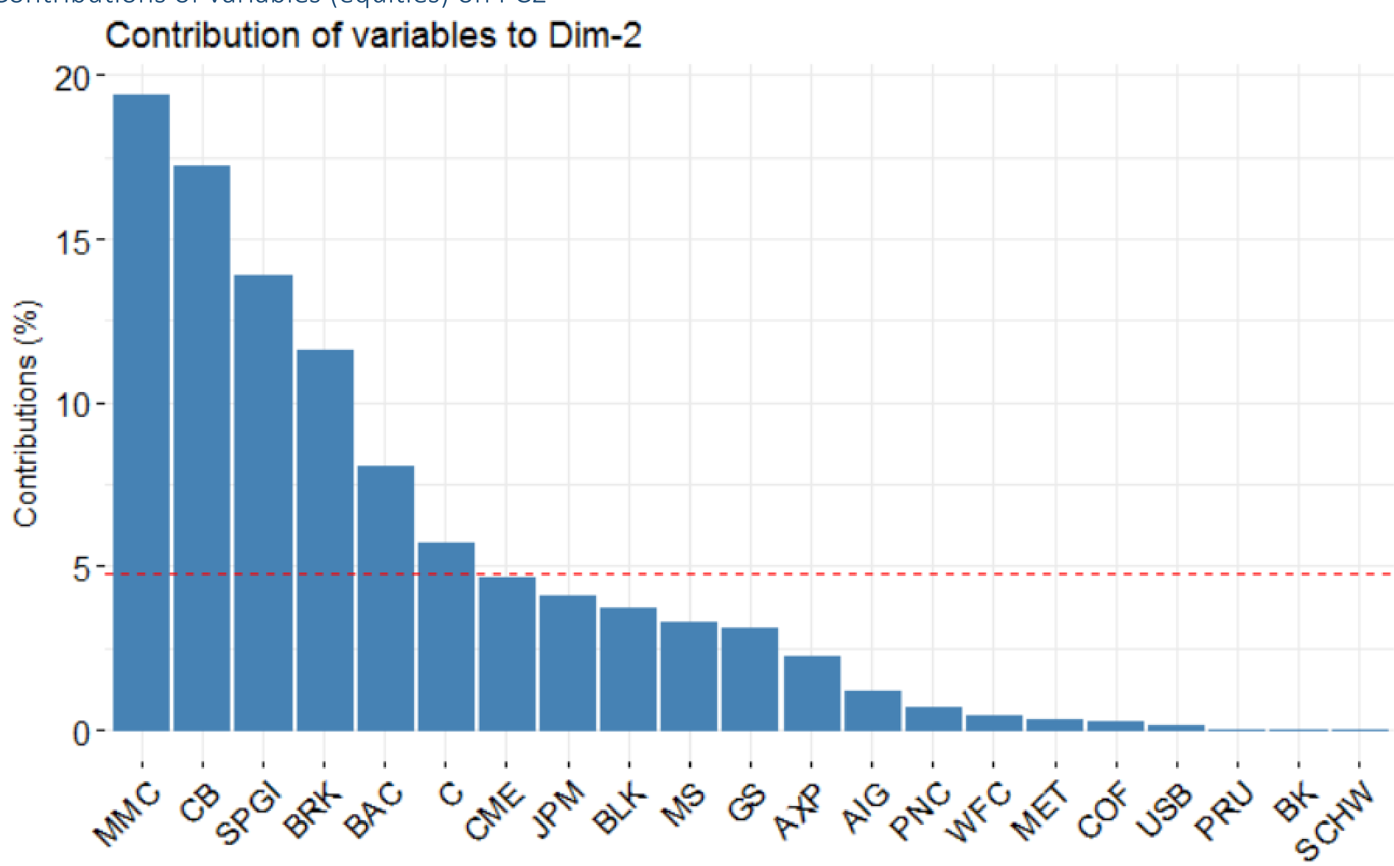
If a variable is perfectly represented by only two components, the sum of the cos2 is equal to one. In this case the variables will be positioned on the circle of correlations.

For some of the variables, more than 2 components are required to perfectly represent the data. In this case the variables are positioned inside the circle of correlations and therefore more than two components are required to represent the variance of each equities.

Contributions of variables (equities) on PC1

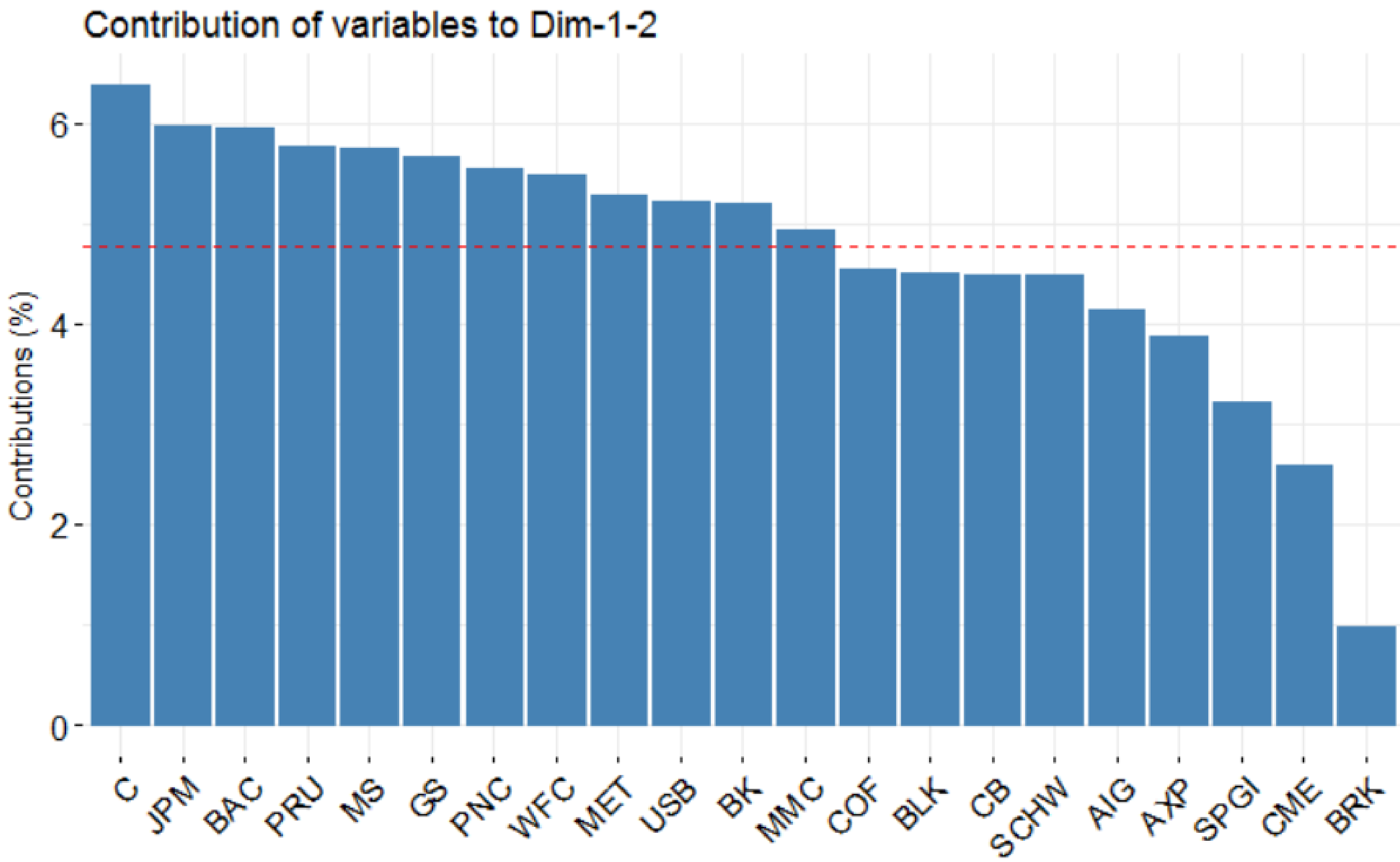


Contributions of variables (equities) on PC2



Contributions of variables (equities) on PC1 and PC2

```
fviz_pca_contrib(res.pca, choice = "var", axes = 1:2)
```



2) Selecting 4 top equities and fitting SDE's to find the right model

Model 1 $dS_t = \theta_1 S_t dt + \theta_2 S_t dW_t$ (Black-Scholes)

Model 2 $dS_t = (\theta_1 + \theta_2 S_t)dt + \theta_3 S_t^{\theta_4} dW_t$ (mean reverting CEV)

Model 3 $dS_t = \theta_1 S_t dt + (\theta_2 + \theta_3 S_t^{\theta_4} dWt)$ (Strange 1)

Model 4 $dS_t = \theta_1 S_t dt + \theta_2 S_t^{\frac{3}{2}} dWt$ (particular CEV)

Model 5 $dS_t = (\theta_1 + \theta_2 S_t)dt + (\theta_3 + \theta_4 \ln S_t) S_t dWt$ (Strange 2)

We will select the symbols **"C","JPM","BAC","PRU"**

```
[1] "For Stock Symbol - C"
[1] "Best model = model  1"
[1] "For Stock Symbol - JPM"
[1] "Best model = model  1"
[1] "For Stock Symbol - BAC"
[1] "Best model = model  1"
[1] "For Stock Symbol - PRU"
[1] "Best model = model  1"
```

3) Correlation Matrix of top 4 stocks

	C	JPM	BAC	PRU
C	1.0000000	0.7973853	0.8108211	0.6864205
JPM	0.7973853	1.0000000	0.7389027	0.6658779
BAC	0.8108211	0.7389027	1.0000000	0.6574654
PRU	0.6864205	0.6658779	0.6574654	1.0000000

4) Monte Carlo for the 4 stocks

```
chol_upper=chol(cor_matrix)

n_iterations=1000
n_steps=252
stocks_sim=matrix(0,n_iterations,4)
stocks_sim[,1]=stock1[1]
stocks_sim[,2]=stock2[1]
stocks_sim[,3]=stock3[1]
stocks_sim[,4]=stock4[1]


dt=1/n_steps
```

```
for(i in 1:n_iterations)
{
  for(j in 2:n_steps)
  {
    w=as.vector(matrix( rnorm(1*4,mean=0,sd=1), 1, 4))
    cor_w=chol_upper%*%w
    for(k in 1:4)
    {

      theta1=params[[k]][1]
      theta2=params[[k]][2]
      theta3=params[[k]][3]
      theta4=params[[k]][4]
      s=stocks_sim[i,k]

      if(best_model[k]==1)
      {
        stocks_sim[i,k]=s+(theta1*dt*s)+(theta2*s*w[1])
      }
      else if(best_model[k]==2)
      {
        stocks_sim[i,k]=s+(theta1+theta2*s)*dt+(theta3*(s^theta4)*w[i])
      }
      else if(best_model[k]==3)
      {
        stocks_sim[i,k]=s+(theta1*s*dt)+(theta2+(theta3*(s^theta4)*w[i]))
      }
      else if(best_model[k]==4)
      {
        stocks_sim[i,k]=s+(theta1*s*dt)+(theta2*(s^(3/2))*w[i])
      }
      else if(best_model[k]==5)
      {
        stocks_sim[i,k]=s+(theta1+theta2*s)*dt+(theta3+theta4*log(s))*s*w[i]
      }
    }

  }
}
head(stocks_sim)
```

```
       Stock1   Stock2   Stock3   Stock4
[1,] 31.41732 33.52855 6.306483 36.82910
[2,] 17.09931 20.25264 3.312467 84.68226
[3,] 33.23515 35.09854 6.695838 34.76679
[4,] 31.21204 33.37474 6.260604 36.44667
[5,] 18.33215 21.45603 3.565731 76.97195
[6,] 26.13043 28.77629 5.189378 47.54572
```

Index represents each simulation

Basic Statistics of the simulations

	Stock1 <dbl>	Stock2 <dbl>	Stock3 <dbl>	Stock4 <dbl>
nobs	1000.000000	1000.000000	1000.000000	1000.000000
NAs	0.000000	0.000000	0.000000	0.000000
Minimum	12.016320	15.119673	2.280231	13.206980
Maximum	65.981884	62.042423	13.828773	136.988616
1. Quartile	22.705635	25.606762	4.471054	35.574725
3. Quartile	32.266018	34.279238	6.487086	57.506691
Mean	27.920200	30.247960	5.578563	48.564836
Median	26.755933	29.348709	5.320606	45.871291
Sum	27920.199677	30247.959760	5578.563181	48564.836017
SE Mean	0.244906	0.218154	0.051944	0.567576

	Stock1 <dbl>	Stock2 <dbl>	Stock3 <dbl>	Stock4 <dbl>
LCL Mean	27.439611	29.819868	5.476631	47.451058
UCL Mean	28.400789	30.676052	5.680495	49.678614
Variance	59.978936	47.590988	2.698188	322.142716
Stdev	7.744607	6.898622	1.642616	17.948335
Skewness	0.918907	0.771828	0.970117	0.932262
Kurtosis	4.262494	3.871262	4.414221	4.176919

5) Fitting the ETF (XLF) data

```
getSymbols("XLF",from="2012-01-01")

etf_price=ts(XLF[,6])

print("The parameter estimates are:")
ls_etf=Diff.mle(fx=fx[1],gx=gx[1],data = etf_price)
ls_etf=ls_etf$Coef[,pmle_type]
print(paste("mu=",round(ls_etf[1],6),"sigma=",round(ls_etf[2],4)))

```
```

```
[1] "mu= 0.000705 sigma= 0.0103"
```
```

6) Multivariate Regression on the 4 stocks and the ETF

```
xlf_returns=periodReturn(XLF,
            period='daily',
            subset=NULL,
            type='arithmetic',
            leading=TRUE)


regressor=lm(formula=xlf_returns~.,
            data=ReturnMatrix)

summary(regressor)

coeff=regressor$coefficients
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.0001207  0.0002257   0.535  0.59289
C            0.1150837  0.0353645   3.254  0.00117 **
JPM         -0.0128873  0.0393131  -0.328  0.74311
BAC          0.0006087  0.0281908   0.022  0.98278
PRU          0.1982564  0.0297942   6.654 4.14e-11 ***
XLF          0.5769561  0.0713477   8.087 1.36e-15 ***
```

7) Basket Security

```
exotic1={}
exotic2={}
etf_price=coredata(tail(XLF[,6],n=1));
for(i in 1:1000)
  exotic1[i]=((stocks_sim[i,1]*coeff[1])+(stocks_sim[i,2]*coeff[2])+(stocks_sim[i,3]*coeff[3])+(stocks_sim[i,4]*coeff[4]))
             -etf_price
  exotic1[i]=max(exotic1[i],0)

  exotic2[i]=etf_price-((stocks_sim[i,1]*coeff[1])+(stocks_sim[i,2]*coeff[2])+(stocks_sim[i,3]*coeff[3])+
                          (stocks_sim[i,4]*coeff[4]))

  exotic2[i]=max(exotic1[i],0)

print(paste("Option Price Today=",mean(exotic[i])))

print(paste("Option Price Today if the buyer exchanged has the option to exchanfe etf for weighted average of
stocks=",mean(exotic2[i])))


```

```
      XLF.Adjusted
[1,]        -23.56
[1] "Option Price Today= 4.1982720340639"
[1] "Option Price Today if the buyer exchanged has the option to exchanfe etf for weighted average of stocks=
4.1982720340639"
```

QuestionC

A) Computing the Implied Volatility

Used Newton-Raphson Method

```
def find_vol(target_value, call_put, S, K, T, r):
    MAX_ITERATIONS = 100
    PRECISION = 1.0e-5

    sigma = 0.5
    for i in range(0, MAX_ITERATIONS):
        price = bs_price(call_put, S, K, T, r, sigma)
        vega = bs_vega(call_put, S, K, T, r, sigma)
        price = price
        diff = target_value - price  # our root
        if (abs(diff) < PRECISION):
            return sigma
        sigma = sigma + diff/vega # f(x) / f'(x)

    return sigma
```

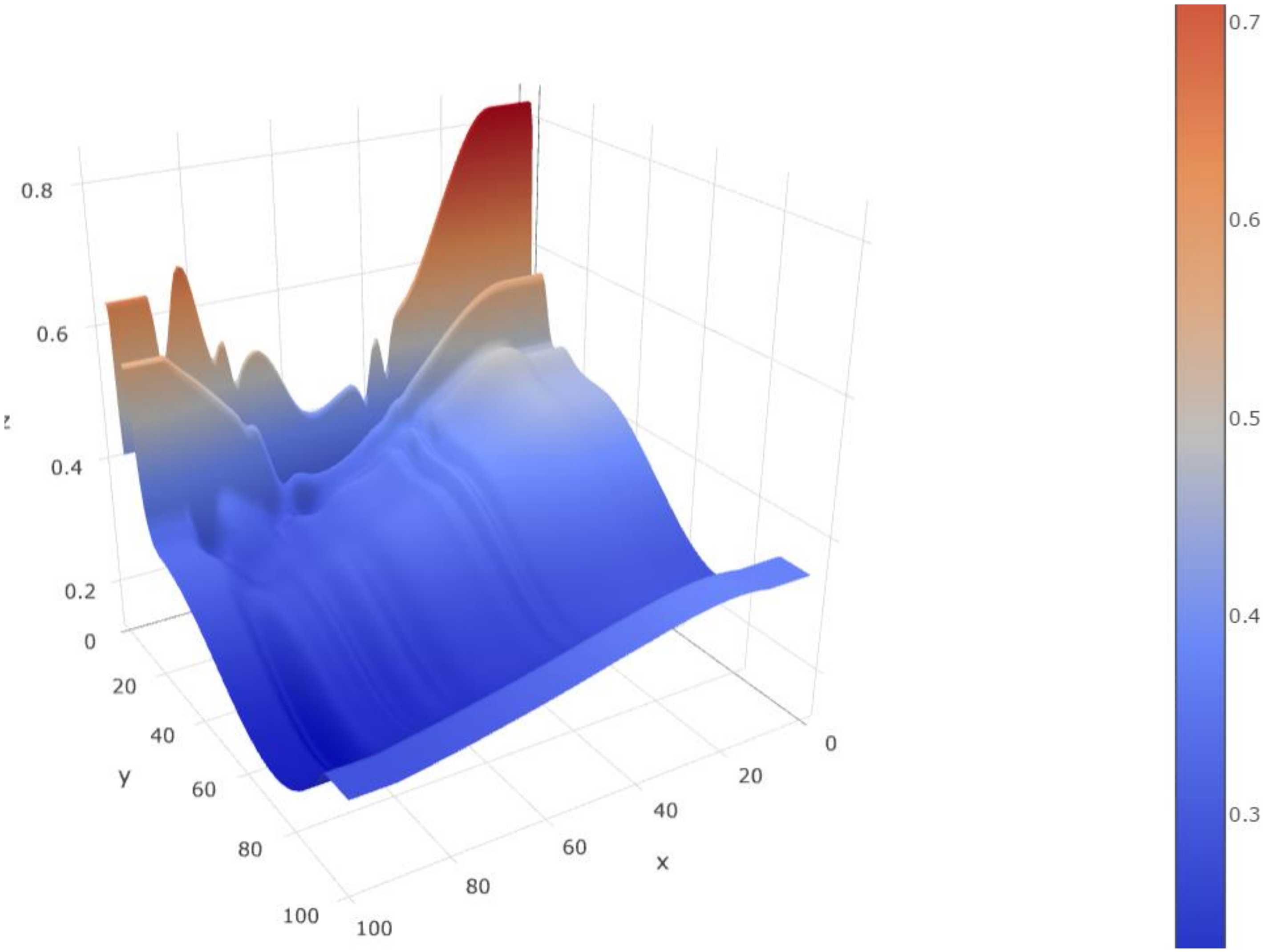
	Time	Strike	Price	Imp_Vol
0	0.071233	850	0.45	0.192977
1	0.071233	875	0.60	0.250505
2	0.071233	900	0.15	0.243389
12	0.071233	475	295.75	0.790720
13	0.071233	490	280.95	0.779920
14	0.071233	500	271.10	0.770282
15	0.071233	525	246.55	0.741374
16	0.071233	530	241.65	0.734557
17	0.071233	540	231.85	0.719254
19	0.071233	550	222.15	0.709292

B) Interpolating the Implied volatility Surface

Package Used: Scipy: Interpolate - interpolate.interp2d

Method: Cubic Spline

	500	600	650	700	725	750	775	800	825	850	...	1000	1025	1050
0.071233	0.829397	0.716864	0.654169	0.592786	0.489546	0.536135	0.439802	0.485424	0.395635	0.44693	...	0.443295	0.47854	0.512539
0.147945	0.568578	0.534447	0.501391	0.464382	0.44587	0.425724	0.409491	0.394652	0.374183	0.365101	...	0.325446	0.341903	0.321896
0.224658	0.500804	0.493046	0.468659	0.440587	0.426297	0.411279	0.396772	0.384726	0.371612	0.357542	...	0.298103	0.284382	0.294967
0.320548	0.470086	0.479653	0.460234	0.437232	0.409959	0.41269	0.384818	0.38758	0.362695	0.365102	...	0.313127	0.292727	0.298689
0.569863	0.442178	0.435714	0.421476	0.405547	0.384925	0.388556	0.366415	0.370409	0.348279	0.352275	...	0.305856	0.300454	0.293569



C)

The no-arbitrage condition is not holding for all the points on the surface. As we can see, there are considerable dips and negative slopes in localized strike regions which could be potential arbitrage areas.

D) Local Volatility

$$\Sigma(K,T) = \sqrt{\frac{\frac{\partial C}{\partial T} + (r-q)K\frac{\partial C}{\partial K} + qC}{\frac{1}{2}K^2\frac{\partial^2 C}{\partial K^2}}},$$

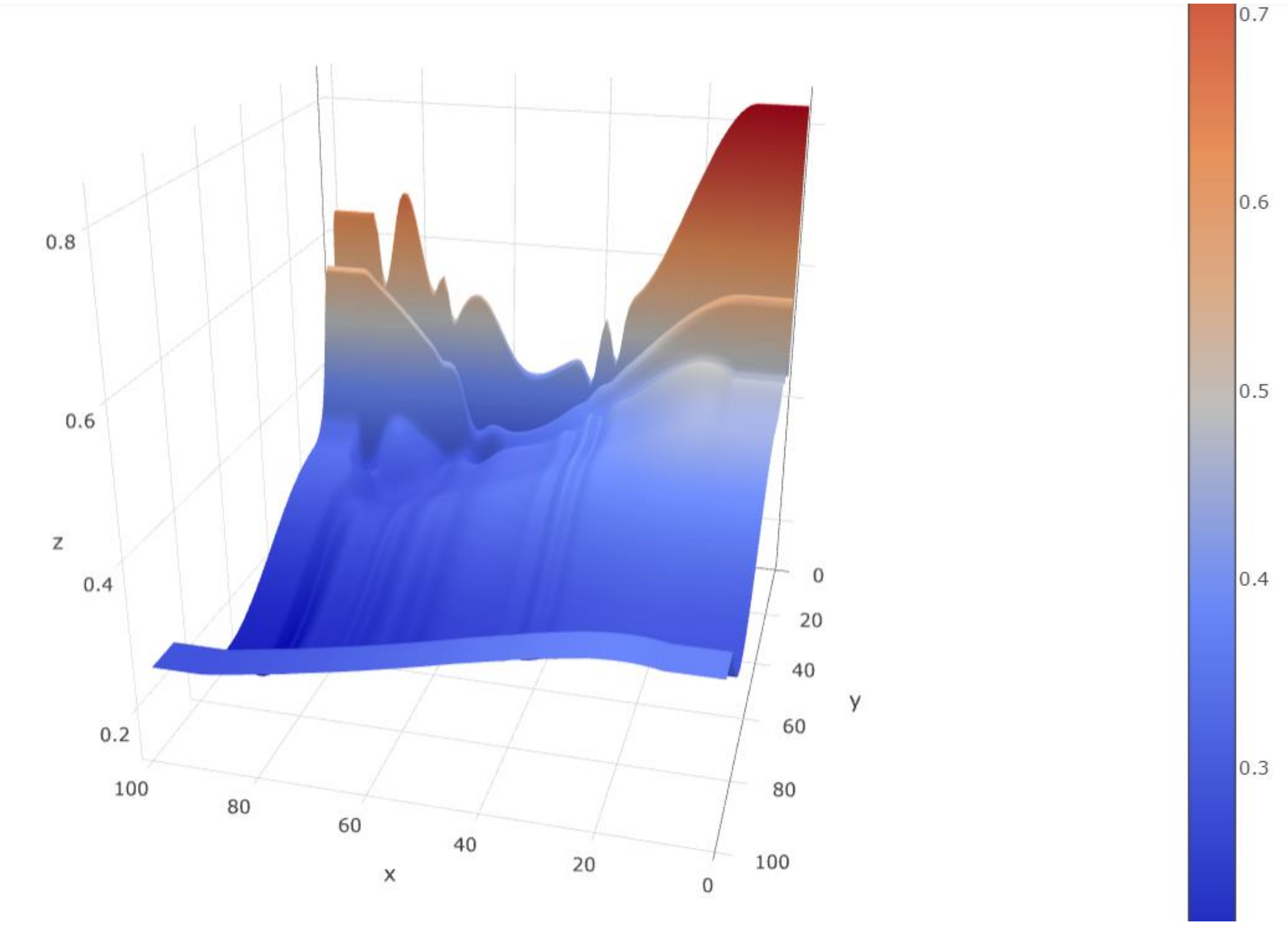
```
def DC_by_DT(K,t,cp_flag,q,r,S0,sigma):
    time1=t
    time2=t+.01
    price1=bs_price(cp_flag='c',K=K,q=q,r=r,S=S0,T=time1,v=sigma)
    price2=bs_price(cp_flag='c',K=K,q=q,r=r,S=S0,T=time2,v=sigma)
    return((price1-price2)/(time1-time2))

def DC_by_DK(K,t,cp_flag,q,r,S0,sigma):
    strike1=K
    strike2=K*1.10
    price1=bs_price(cp_flag='c',K=strike1,q=q,r=r,S=S0,T=t,v=sigma)
    price2=bs_price(cp_flag='c',K=strike2,q=q,r=r,S=S0,T=t,v=sigma)
    return((price1-price2)/(strike1-strike2))

def DDC_by_DDT(K,t,cp_flag,q,r,S0,sigma):
    time1=t
    time2=t+.04
    dcdt1=DC_by_DT(cp_flag='c',K=K,q=q,r=r,S0=S0,t=time1,sigma=sigma)
    dcdt2=DC_by_DT(cp_flag='c',K=K,q=q,r=r,S0=S0,t=time2,sigma=sigma)
    return((dcdt1-dcdt2)/(time1-time2))
```

```
def Dupiare_One(K,t,sigma):
    dC_dT=DC_by_DT(K=K,t=t,cp_flag='c',q=0,r=r,S0=S0,sigma=sigma)
    dC_dK=DC_by_DK(K=K,t=t,cp_flag='c',q=0,r=r,S0=S0,sigma=sigma)
    ddC_ddT=DDC_by_DDT(K=K,t=t,cp_flag='c',q=0,r=r,S0=S0,sigma=sigma)


    C=bs_price(cp_flag='c',K=K,q=q,r=r,S=S0,T=t,v=sigma)
    numerator=dC_dT+(r-q)+(K*dC_dK)+(q*C)
    denominator=.5*(K*K)*ddC_ddT

    return(np.sqrt(numerator/denominator))
```

Interpolated Dupire Local Volatility Grid

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
0	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	...	0.637896	0.637896	0.637896	0.637896
1	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	...	0.637896	0.637896	0.637896	0.637896
2	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	0.828195	...	0.637896	0.637896	0.637896	0.637896
3	0.549802	0.549802	0.549802	0.549802	0.549802	0.549802	0.549802	0.549802	0.549802	0.549802	...	0.461161	0.461161	0.461161	0.461161
4	0.407344	0.407344	0.407344	0.407344	0.407344	0.407344	0.407344	0.407344	0.407344	0.407344	...	0.403566	0.403566	0.403566	0.403566

Plot of Interpolated Dupire Volatility



E) Price of Option with Local Volatility

	Time	Strike	Price	Imp_Vol	Dupiare1_IV	Dupire_Price
33	0.071233	645	132.4	0.586575	0.007014	125.35
34	0.071233	650	127.9	0.580357	0.004121	120.36
35	0.071233	655	123.4	0.573387	0.003157	115.36
36	0.071233	660	119.0	0.567945	0.002574	110.36
37	0.071233	665	114.6	0.561664	0.002206	105.36

We can see that the Dupire price is less than the market price most of the times because we are taking the local volatility and local volatility is one that treats volatility as a function of both the current asset level and of time. As such, a local volatility model is a generalization of the Black-Scholes model, where the volatility is a constant. Because the only source of randomness is the stock price, local volatility models are easy to calibrate. Also, they lead to complete markets where hedging can be based only on the underlying asset. Since in local volatility models the volatility is a deterministic function of the random stock price, local volatility models are not very well used to price options whose values depend specifically on the random nature of volatility itself.

Therefore, Dupire option price is based on the underlying price volatility which is much lesser than the implied volatility and therefore the price calculated using Dupire volatility is less than the real price which is dependent on the implied volatility.

F) Compiling all data into one table

	Time to maturity	Strike price	Option market price	Implied Volatility	Local Volatility	Dupire_Price
33	0.071233	645	132.4	0.586575	0.007014	125.35
34	0.071233	650	127.9	0.580357	0.004121	120.36
35	0.071233	655	123.4	0.573387	0.003157	115.36
36	0.071233	660	119.0	0.567945	0.002574	110.36
37	0.071233	665	114.6	0.561664	0.002206	105.36

"SPXvolatility.csv"

SPXvolatility - Excel

File Home Insert Page Layout Formulas Data Review View Tell me what you want to do

A23 | f_x 1

	A	B	C	D	E	F	G	H	I
1		Time to maturity	Strike price	Option market price	Implied Volatility	Local Volatility	Dupire_Price		
2	0	0.071232877	850	0.45	0.192977046	0.000430298	0		
3	1	0.071232877	875	0.6	0.250504914	0.000421562	0		
4	2	0.071232877	900	0.15	0.243389254	0.000283479	0		
5	3	0.071232877	625	150.75	0.613365993				
6	4	0.071232877	630	146	0.602989971				
7	5	0.071232877	635	141.4	0.596492831				
8	6	0.071232877	640	136.9	0.591978749				
9	7	0.071232877	645	132.4	0.586574662	0.007014195	125.35		
10	8	0.071232877	650	127.9	0.580356696	0.004121123	120.36		
11	9	0.071232877	655	123.4	0.573386901	0.003157087	115.36		
12	10	0.071232877	660	119	0.56794498	0.002574092	110.36		
13	11	0.071232877	665	114.6	0.561663914	0.002206313	105.36		
14	12	0.071232877	670	110.3	0.556644101	0.001919991	100.36		
15	13	0.071232877	675	106	0.550729429	0.001708911	95.37		
16	14	0.071232877	680	101.8	0.545859685	0.001528914	90.37		
17	15	0.071232877	685	97.6	0.540061687	0.001386089	85.37		
18	16	0.071232877	690	93.4	0.533391176	0.001268937	80.37		
19	17	0.071232877	695	89.4	0.529256862	0.001152547	75.38		
20	18	0.071232877	700	85.3	0.522482597	0.001062626	70.38		

G) Creating a custom function and Using External Data

```python
def Local_Vol_Pricing(S0,option_df,r):
    iv=[]
    for index, row in option_df.iterrows():
        try:
            iv.append(find_vol(row.Price, 'c', S0, row.K, row.t, r))
        except ValueError:
            iv.append(0)

    option_df['Imp_Vol'] = pd.Series(iv, index=option_df.index)
    option_df = option_df[np.isfinite(option_df['Imp_Vol'])]

    dupiare_iv=[]
    for index, row in option_df.iterrows():
        dupiare_iv.append(Dupiare_One(row.K,row.t,row.Imp_Vol))
    option_df['Dupiare1_IV']=dupiare_iv

    dupiare_price=[]
    for index,row in option_df.iterrows():
        dupiare_price.append(bs_price(cp_flag='c',K=row.K,r=r,S=S0,v=row.Dupiare1_IV,T=row.t))
    option_df['Dupiar_Price']=np.round(dupiare_price,2)

    final_option_df=option_df
    final_option_df.columns=['Expiry','Time to maturity', 'Strike price', 'Option market price', 'Implied Volatility', \
    'Local Volatility', 'Dupire-Price']

    final_option_df.to_csv("New_DATA_volatility.csv")
    return final_option_df
```

	Expiry	Time to maturity	Strike price	Option market price	Implied Volatility	Local Volatility	Dupire-Price
0	42874	0.019178	148.0	8.15	0.173414	1.443104	16.60
1	42874	0.019178	149.0	7.29	0.231178	1.433419	15.96
2	42874	0.019178	150.0	6.31	0.211508	1.423863	15.33
3	42874	0.019178	152.5	4.04	0.189001	1.400521	13.82
4	42874	0.019178	155.0	2.11	0.171472	1.377933	12.40

Implementation Guide

File Name: **QuestionC_PartG.py**

Code design: Python3.6

```
option_df=pd.read_csv('BloomBerg_Option_Data_Question_C.csv')

option_df.columns=['Expiry','t','K','Price']
option_df['t']=option_df['t']/365

data_frame=Local_Vol_Pricing(S0=156.1,r=1.182/100,option_df=option_df)
```

Appendix:

The code for these question (including the rmdb, jupyter notebook and HTML files) are in the corresponding folders in the submission file.