# CISC 5352 FINANCIAL PROGRAMMING AND DATA ANALYTICS LECTURE NOTE (3)

Henry Han Ph.D.
Department of Computer and Information Science
Fordham University, New York NY 10023

## Last class review

# Last class review

① **Three BS models in option pricing**
   ① **The original BS model**
   ② **BSM model**
   ③ **Generalized BSM model**

**Use one sentence to summarize BS model:**

# Last class review

① **Three BS models in option pricing**
   ① **The original BS model**
   ② **BSM model**
   ③ **Generalized BSM model**

**Use one sentence to summarize BS model:**

All weighted changes w.r.t. t and S is the product of r and f

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2 - rf = 0$$

① Weights: $[1, rS, \frac{1}{2}\sigma S^2]$

② Changes: $[\frac{\partial f}{\partial t}, \frac{\partial f}{\partial S}, \frac{\partial^2 f}{\partial S^2}]$

# Last class review

① **Three BS models in option pricing**
   ① **The original BS model**
   ② **BSM model**
   ③ **Generalized BSM model**

**Use one sentence to summarize BS model:**

All weighted changes w.r.t. t and S is the product of r and f

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2 - rf = 0$$

$$C = SN(d_1) - Ke^{-rT}N(d_2)$$

$$P = Ke^{-rT}N(-d_2) - SN(-d_1)$$

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

① Weights: $[1,\ rS,\ \frac{1}{2}\sigma^2 S^2]$

② Changes: $[\frac{\partial f}{\partial t}, \frac{\partial f}{\partial S}, \frac{\partial^2 f}{\partial S^2}]$

# Last class review: BSM model

use r-q to replace r in the original BS: q=0 means the original BS.

$$\frac{\partial f}{\partial t} + (r-q)S\frac{\partial f}{\partial S} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2 - rf = 0$$

① Weights: $[1,\ (r-q)S,\ \frac{1}{2}\sigma^2 S^2]$

② Changes: $[\frac{\partial f}{\partial t}, \frac{\partial f}{\partial S}, \frac{\partial^2 f}{\partial S^2}]$

$$C = Se^{-qT}N(d_1) - Ke^{-rT}N(d_2)$$

$$P = Ke^{-rT}N(-d_2) - Se^{-qT}N(-d_1)$$

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

# Last class review: generalized-BSM model

Use b to replace r-q in the original BSM model

① Weights: $[1, bS, \frac{1}{2}\sigma^2 S^2]$

② Changes: $[\frac{\partial f}{\partial t}, \frac{\partial f}{\partial S}, \frac{\partial^2 f}{\partial S^2}]$

□

$$C = Se^{-(b-r)T}N(d_1) - Ke^{-rT}N(d_2)$$

$$P = Ke^{-rT}N(-d_2) - Se^{-(b-r)T}N(-d_1)$$

$$d_1 = \frac{\ln(S/K) + (b + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

# Last class review

① **Three BS models in option pricing**
   ① **The original BS model**
   ② **BSM model**
   ③ **Generalized BSM model**

② **Normal CDF calculation**
   ① **Hart algorithm**
   ② norm.cdf(x, mu, sigma) (from Scipy)

③ **Monte Carlo simulation in option pricing**
   ① **Generate normal distribution data:** random.standard_normal(N) (from Numpy)

## Q1: Why do we need multi-threading programming?

A process is an execution instance of a program: it is an execution of a program.

A program is a blueprint: a process is its invocation.

A thread is the minimum unit of a process

## Q1: Why do we need multi-threading programming?

It is a parallel programming skill to enhance execution efficiency.

## Q1: Why do we need multi-threading programming?

It is a parallel programming skill to enhance execution efficiency.

Executing a program consists of a set of threads say, load variables, generate random numbers, calculating payoffs…

## Q1: Why do we need multi-threading programming?

It is a parallel programming skill to enhance execution efficiency.

Executing a program consists of a set of threads say, load variables, generate random numbers, calculating payoffs…

Multi-threading programming can let these processes share CPUs and other resources in a parallel way to improve efficiency

## Q1: Why do we need multi-threading programming?

It is a parallel programming skill to enhance execution efficiency.

Executing a program consists of a set of threads say, load variables, generate random numbers, calculating payoffs…

Multi-threading programming can let these processes share CPUs and other resources in a parallel way to improve efficiency

Type top in your terminal (for Mac/Linux/cygwin users), you will see all running processes in your machine

**process=program code + its resulting states**

```
Processes: 265 total, 11 running, 8 stuck, 246 sleeping, 1129 threads                 21:17:26
Load Avg: 9.32, 9.33, 9.47  CPU usage: 99.26% user, 0.73% sys, 0.0% idle
SharedLibs: 128M resident, 12M data, 14M linkedit.
MemRegions: 45113 total, 2038M resident, 90M private, 799M shared.
PhysMem: 6469M used (1121M wired), 1716M unused.
VM: 710G vsize, 535M framework vsize, 37337(0) swapins, 49473(0) swapouts.
Networks: packets: 1380847/1224M in, 1191563/237M out.
Disks: 714039/33G read, 619187/19G written.

PID   COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP PPID STATE
7229  screencaptur 0.1  00:00.15 6     4    58    2764K  20K    0B     335  335  sleeping
7225- mdworker32   0.0  00:00.17 3     0    54    6400K  0B     0B     7225 1    sleeping
7224  QuickLookSat 0.0  00:00.18 9     0    97    11M    12K    0B     7224 1    sleeping
7223  quicklookd   0.0  00:00.11 4     0    91    6792K  0B     0B     7223 1    sleeping
7222  top          2.3  00:20.50 1/1   0    25    3976K  0B     0B     7222 7214 running
7220  mdworker     0.0  00:00.32 4     0    48    9324K  0B     0B     7220 1    sleeping
7214  bash         0.0  00:00.02 1     0    17    888K   0B     0B     7214 7213 sleeping
7213  login        0.0  00:00.29 2     0    28    1588K  0B     0B     7213 471  sleeping
7212  Google Chrom 0.0  00:06.56 15    0    141   96M    0B     0B     406  406  sleeping
7176  netbiosd     0.0  00:00.02 2     1    36    2748K  0B     0B     7176 1    sleeping
7174  ocspd        0.0  00:00.40 4     0    65    2980K  0B     0B     7174 1    sleeping
7117  helpd        0.0  00:00.01 2     0    42    1292K  0B     0B     7117 1    sleeping
7096  applessdstat 0.0  00:00.00 3     1    35    868K   0B     0B     7096 1    stuck
7070  mdworker     0.0  00:00.88 4     0    48    9392K  0B     0B     7070 1    sleeping
7040  mdworker     0.0  00:00.12 4     0    45    9092K  0B     0B     7040 1    sleeping
6874  mdworker     0.0  00:00.81 4     0    48    11M    0B     0B     6874 1    sleeping
6861  SCIM         0.0  00:01.44 3     0    187   17M    0B     0B     6861 1    sleeping
6860  imklaunchage 0.0  00:00.04 2     0    69    2316K  0B     0B     6860 1    sleeping
6805- Microsoft Ex 0.0  01:35.43 14    1    186   97M    0B     0B     6805 1    sleeping
6797  mdworker     0.0  00:01.38 4     0    48    11M    0B     0B     6797 1    sleeping
6692  periodic-wra 0.0  00:00.00 2     1    28    576K   0B     0B     6692 1    sleeping
6595  aslmanager   0.0  00:00.01 2     1    27    1180K  0B     0B     6595 1    sleeping
5784  com.apple.ap 0.0  00:00.01 2     1    27    556K   0B     496K   5784 1    sleeping
5778  com.apple.hi 0.0  00:00.02 2     0    32    736K   0B     404K   5778 1    sleeping
```

```
① import threading
② import time
③ def addHarmonicSeries(n):
④     sum=0.0
⑤     for i in range(1,n):
⑥         sum=sum + 1.0/i
⑦         print('{:5d} {:12.6f}'.format(i, sum))
⑧ start_time = time.clock()

⑨ no_thread = 100
⑩ for i in range(no_thread):
11     t = threading.Thread(target=addHarmonicSeries, args=(i,))
12     t.start()
13     print("\n-->"+t.getName() + "\n")
14     time.sleep(1)
```

Will have a process to calculate harmonic series

Function parameters

## Q2: What's wrong with the following student's codes in quiz 1?

```
①  sum = 0

   for n in range(1,10000):
       numerator = (-1) ** (n + 1)
       multiplier = (2/3) ** n
       sum += (numerator / n) * multiplier

   print sum
```

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\left(\frac{2}{3}\right)^n$$

## Q2 : WHAT's wrong with the following student's codes in quiz 1?

① 

```
sum = 0

for n in range(1,10000):
    numerator = (-1) ** (n + 1)
    multiplier = (2/3) ** n
    sum += (numerator / n) * multiplier

print sum
```

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\left(\frac{2}{3}\right)^n$$

## A little bit professional version: make sure your codes work for both python2 and python3 to remove possible confusion

```
sum = 0.0

for n in range(1,1000000):

   l  = math.pow(-1.0, n+1)/n
   r  = math.pow((2.0/3.0), n)
   sum = sum + l * r

print("\n sum: {:12.9f} ". format(sum) +"\n")
```

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\left(\frac{2}{3}\right)^n$$

## Q3: what are basic ideas of MC simulations?

## Q3: what are basic ideas of MC simulations?

Idea: try all possible cases for $S_T$ through a large number of simulations to calculate an average option price!

## Q3: what are basic ideas of MC simulations?

① Generate a large number N numbers of random numbers from N(0,1)

② Compute all possible stock prices at T for each random number by using the result from BSM model: $S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}z\right)$

③ Calculate all possible payoffs

④ Simulations: calculate the average of all payoffs and divided it by e^rT (r: interest rate)

## PANDAS→Python's data analytics toolkit: we always need to use in financial data analytics

http://pandas.pydata.org/pandas-docs/stable/

1600+ pages in 2015

2000+ pages in 2016

Visit OFTEN this webpage to get familiar with PANDAS

Quick start: http://pandas.pydata.org/pandas-docs/stable/10min.html

Python for financial data analytics just starts→ it is a great chance to follow the trend to become an expert!

## Python Data Analysis Library: Pandas

① It is combined with the IPython toolkit built on top of NumPy, SciPy, matplotlib.

② The number 1 tool for data analytics!

③ It handles financial time series data very well.

A general misconception that students hold: If I can call some commands in Pandas, then I can do data analytics

To become a professional data anayticist, you MUST do serious coding as well.

## Pandas has two basic data structures: series and dataFrame

① Series: is like an ordered dictionary that permits duplicates

② What's dictionary data structure in python (I assume students know python)?

## Pandas has two basic data structures: series and dataFrame

① Series: is like an ordered dictionary that permits duplicates (a labeled one-dimensional array)

② What's dictionary data structure in python?

③ Dictionary: a key:value pair structure (like a phonebook)

```
contacts= {'bill': '353-1234',
           'rich': '269-1234',
            'jane': '352-1234'}

print("my mini-phonebook is\n"+str(contacts)+"\n")  # output contacts as a string
```

General signature: pandas.Series(data, index)

## A series is a dictionary: A series is a labeled array

```
① import pandas as pd

② contacts2 = pd.Series([1235678, 2568833, 3338733, 5903333],\
③                       index=['alex', 'bill', 'tom', 'david'])

④ print("\n a series-->"+str(contacts2) + "\n")

⑤ x=range(10)
⑥ a=pd.Series([math.sin(x[2]), 5, math.pow(2, x[9]), 11, 29])
⑦ print(a)
⑧ a.mean()
⑨ a.std()
⑩ a.sum()
11 a.quantile(0.95)
12 a.cumsum()
13 a.sort()
14 print(a)
```

# Dataframe: a labeled matrix (a spreadsheet)!

```
In [6]: goog.head()
Out[6]:
              Open    High     Low   Close  Volume
Date
2010-06-01  239.97  245.28  239.82  240.94     NaN
2010-06-02  243.10  246.69  240.49  246.44     NaN
2010-06-03  247.31  253.75  247.10  252.55     NaN
2010-06-04  249.61  254.37  248.10  249.11     NaN
2010-06-07  249.28  250.20  241.33  242.52     NaN
```

Top 5 lines of Google stock data starting 06/01/2010 retrieved from web by using Pandas.io.data

# Dataframe: a labeled matrix (a spreadsheet)!

```
In [6]: goog.head()
Out[6]:
              Open    High     Low   Close  Volume
Date
2010-06-01  239.97  245.28  239.82  240.94     NaN
2010-06-02  243.10  246.69  240.49  246.44     NaN
2010-06-03  247.31  253.75  247.10  252.55     NaN
2010-06-04  249.61  254.37  248.10  249.11     NaN
2010-06-07  249.28  250.20  241.33  242.52     NaN
```

Top 5 lines of Google stock data starting 06/01/2010 retrieved from web by using Pandas.io.data
Each row in a dataframe is a series

It is still called data frame in R, but in Matlab, it is called data.matrix

# Let's start financial data analytics by using Pandas

We need to retrieve the real data from Internet at first!

I want to get all Google Stock information in recent 5 years: 6/1/2010—6/1/2015

How can I do it?

We need to use pandas.io.data and pandas.io.ga to extra data from Internet sources such as Google Finance, Yahoo Finance

1. https://www.google.com/finance
2. https://finance.yahoo.com/

```
import numpy as np
import pandas as pd
import pandas.io.data as web

# Retrive all google stock from the s_date to the end_date

s_date='6/1/2010'  # start date
e_date='6/1/2015'  # end date

# Retrieve google stock ( 'GOOG') from google finance

goog = web.DataReader('GOOG', data_source='google',  start=s_date, end=e_date)
# print all google stock
print(goog)

# show time index for stock: here time is the key
g_idx=goog.index
print(g_idx)

# First 5 business days of stock
g_head=goog.head()
# Last 5 business days of stock
g_tail=goog.tail()
```

```
print(g_tail)
Open    High    Low   Close   Volume
Date
2015-05-26  538.12  539.00  529.88  532.32  2406512
2015-05-27  532.80  540.55  531.71  539.79  1525019
2015-05-28  538.01  540.61  536.25  539.78  1029849
2015-05-29  537.37  538.63  531.45  532.11   2597407
2015-06-01  536.79  536.79  529.76  533.99  1904332
```

**Want to access the 'High' and 'Open' column data?**

```
print(g_tail[['Open', 'High']])
print(g_tail[[0, 1]])
```

# How to access each row?

Similar to dictionary, we use a date as a key to access data

goog.loc['20100601']

```
In [225]: goog.loc['20100601']
Out[225]:
Open       239.97
High       245.28
Low        239.82
Close      240.94
Volume        NaN
Name: 2010-06-01 00:00:00, dtype: float64
```

Data missing→ they are not available!

.loc[] a way to do index in pandas: loc can be understood as "location"

# How to access each row Cont'd?

goog.loc['20100601']

```
In [225]: goog.loc['20100601']
Out[225]:
Open      239.97
High      245.28
Low       239.82
Close     240.94
Volume       NaN
Name: 2010-06-01 00:00:00, dtype: float64
```

Same meaning: goog[0:1] (1st raw of data)

```
In [225]: goog.loc['20100601']
Out[225]:
Open      239.97
High      245.28
Low       239.82
Close     240.94
Volume       NaN
Name: 2010-06-01 00:00:00, dtype: float64
```

Data from the 1001th to 1010th row

```
In [234]: goog[1000:1010]
Out[234]:
              Open    High     Low   Close   Volume
Date
2014-05-21  532.90  539.18  531.91  538.94  1193389
2014-05-22  541.13  547.60  540.78  545.06  1611837
2014-05-23  547.26  553.64  543.70  552.70  1929632
2014-05-27  556.00  566.00  554.35  565.95  2100298
2014-05-28  564.57  567.84  561.00  561.68  1647717
2014-05-29  563.35  564.00  558.71  560.08  1350657
2014-05-30  560.80  561.35  555.91  559.89  1766794
2014-06-02  560.70  560.90  545.73  553.93  1434989
2014-06-03  550.99  552.34  542.55  544.94  1861921
2014-06-04  541.50  548.61  538.75  544.66  1812084
```

## Want to save your data in Excel?

Use to_excel(…) function

```
# save it as an excel file
filename='google_stock_data.xlsx'
goog.to_excel(filename, sheet_name='sheet1', index=False)

## check if the file exists
import os.path
if (os.path.isfile(filename)==True):
    print("\n "+filename + " is saved!\n" )
```

**You can type 'ls' in Ipython to see the file.**

## Q4: How to know the risk of stock?

An important measure is volatility→ it can viewed as standard deviation (variance) of the stock

The higher the volatility, the riskier the security

## How to compute the volatility of a security (Stock data)?

## How to compute the volatility of a security (stock data)?

We can use history data!

In other words, we can look back to calculate its volatility.

Note: it does not mean its future volatility will follow the value you get!!

## Details of volatility computing

① Given n+1 number of observations (e.g., daily observation): 0, 1,2,...,n;

## Details of volatility computing, Cont'd

① Given n+1 number of observations (e.g., daily observation): 0, 1,2,...,n;

② We use $S_i$ to represent the stock price (close price) at the end of ith interval: [i-1, i]

## Details of volatility computing, Cont'd

① Given n+1 number of observations (e.g., daily observation): 0, 1,2,...,n;

② We use $S_i$ to represent the stock price (close price) at the end of ith interval: [i-1, i]

③ Let $\tau$ represent the length of time interval in years (e.g.,1/252: 252 total trading days), then the volatility can be estimated as

## Details of volatility computing, Cont'd

① Given n+1 number of observations (e.g., daily observation): 0, 1,2,...,n;

② We use $S_i$ to represent the stock price (close price) at the end of ith interval: [i-1, i]

③ Let $\tau$ represent the length of time interval in years (e.g.,1/252: 252 total trading days), then the volatility can be estimated as

$$\hat{\sigma} = \frac{s}{\sqrt{\tau}}$$

A normalized standard deviation w.r.t time

$s$ is the estimation of standard deviation of $u_i = \ln(\frac{S_i}{S_{i-1}})$ which can be estimated as

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} u_i^2 - \frac{1}{n(n-1)}(\sum_{i=1}^{n} u_i)^2}$$

The standard error of such estimate is about $\hat{\sigma}/\sqrt{2n}$

## How can we use Pandas to do it?

1) We need to calculate $u_i = \ln(\frac{S_i}{S_{i-1}})$ first, where $S_i$ is the close price of each transaction day
2) We need to calculate the standard deviation s of the sequence $u_i = \ln(\frac{S_i}{S_{i-1}})$
3) We need to estimate the volatility as $\hat{\sigma} = \frac{s}{\sqrt{\tau}}$

   we choose $\tau$ =1/252

# What do we need to do?

We have a dataFrame: goog that has stock from 06/01/2010 to 06/01/2015

We need to

1) calculate the sequence u: log of the ratio of two neighbor close prices

```
S_i          = goog['Close']          # close price
S_i_minus_1  = goog['Close'].shift(1) # move/shift the original one one slot
```

## 0. Bookkeeping

① S_i = goog['Close'] # close price
② S_i_minus_1 = goog['Close'].shift(1) # move/shift the original one one slot
③ S_i[0:10]
④ S_i_minus_1[0:10]

```
In [50]: [S_i[0:10]]
Out[50]:
[Date
 2010-06-01    240.94
 2010-06-02    246.44
 2010-06-03    252.55
 2010-06-04    249.11
 2010-06-07    242.52
 2010-06-08    242.15
 2010-06-09    236.77
 2010-06-10    243.26
 2010-06-11    244.01
 2010-06-14    241.35
 Name: Close, dtype: float64]
```

```
In [56]: S_i_minus_1[0:10]
Out[56]:
Date
2010-06-01       NaN
2010-06-02    240.94
2010-06-03    246.44
2010-06-04    252.55
2010-06-07    249.11
2010-06-08    242.52
2010-06-09    242.15
2010-06-10    236.77
2010-06-11    243.26
2010-06-14    244.01
Name: Close, dtype: float64
```

## 1. Create U_sequence

① goog['U_sequence'] = np.log(S_i/S_i_minus_1)

② U_sequence = goog['U_sequence']

```
In [297]: U_sequence[0:10]
Out[297]:
Date
2010-06-01         NaN
2010-06-02    0.022571
2010-06-03    0.024491
2010-06-04   -0.013715
2010-06-07   -0.026810
2010-06-08   -0.001527
2010-06-09   -0.022468
2010-06-10    0.027042
2010-06-11    0.003078
2010-06-14   -0.010961
Name: U_sequence, dtype: float64
```

2. compute s: we need to compute the standard deviation of the U sequence (n is the number of observations we have)→

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} u_i^2 - \frac{1}{n(n-1)}(\sum_{i=1}^{n} u_i)^2}$$

**We can code it directly**

**We can use a method in Pandas to do this better: roll_std()**

① s=pd.rolling_std(goog['U_sequence'], window=252)

- **The U-sequence data we get is a time series data.**
- **For time series data, Pandas has a series of functions to calculate its statistics: they call start from rolling: e.g.: "rolling_std"**
- http://pandas.pydata.org/pandas-docs/stable/computation.html

# 3. Compute the volatility

① goog['Volatility'] = pd.rolling_std(U_sequence, window=252) * np.sqrt(252)

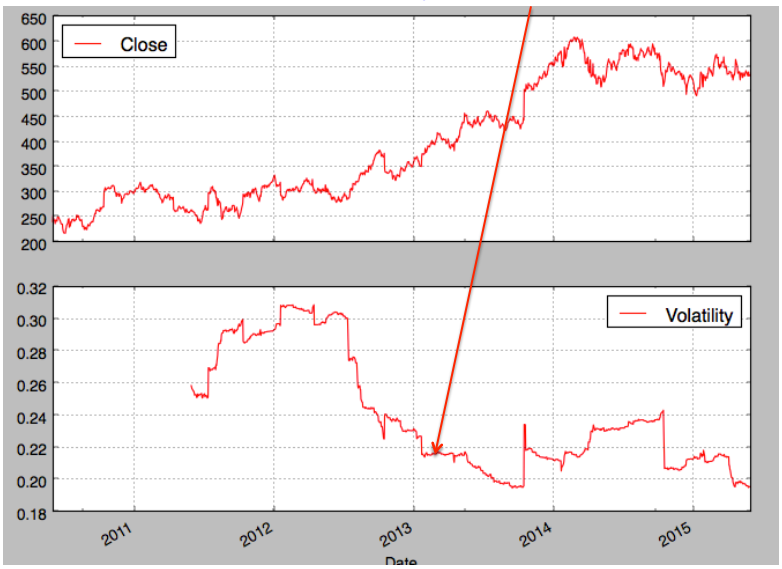
# Why *np.sqrt(252) here?

① goog['Volatility'] = pd.rolling_std(U_sequence, window=252) * np.sqrt(252)

# 4. plot the volatility by using matplotlib

① # invoke the matplotlib
② %matplotlib
③ goog[['Close', 'Volatility']].plot(subplots=True, color='red',figsize=(8, 6));

GOOG becomes a less risk security since 2013



## Lab: Compute the volatility of AAPL from 06/01/2008—09/01/2016