# 1 Question 1

**Statement 1 ▶ Timescale Invariance**

$$d_1 = \frac{\log(S/E) + (r + \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}}$$

$$d_1 = \frac{\log(S/E) + (r - \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}}$$

Prove the following identity:

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

```
from sympy import symbols, log, sqrt, pretty_print


S, E, r, σ, T, t = symbols('S,□E,□r,□σ,□T,□t')

numerator_l = log(S/E) + r*(T-t)
numerator_r = σ**2*(T-t)/2
denominator = σ*sqrt(T-t)

d1 = (numerator_l + numerator_r) / denominator
d2 = (numerator_l - numerator_r) / denominator

# Process:
__Δ = d1 - d2
#    = 2*numerator_r / denominator
#    = σ**2*(T-t) / σ*sqrt(T-t)
#    = σ * sqrt(T-t)
#    =        _______
#       σ·□□ T - t

pretty_print(__Δ.simplify())
```

## 2 Question 2

**Statement 2 ▶ Put-Call Parity**

With $t = 0$, $S_0 = 5$, $E = 4$, $T = 1$, $\sigma = 0.3$ and $r = 0.05$, find the option values and verify the put-call parity.

```python
from math import log, sqrt, erf, exp


S, E, r, σ, T, t = 5, 4, 0.05, 0.3, 1, 0

numerator_l = log(S/E) + r*(T-t)
numerator_r = σ**2*(T-t)/2
denominator = σ*sqrt(T-t)

d1 = (numerator_l + numerator_r) / denominator
d2 = (numerator_l - numerator_r) / denominator

N = lambda d: (1+erf(d/sqrt(2)))/2

print('    d_1 =', d1)
print('    d_2 =', d2)
print(' N(d_1) =', N(d1))
print(' N(d_2) =', N(d2))
print('N(-d_1) =', N(-d1))
print('N(-d_2) =', N(-d2))


C = S*N(d1) - E*exp(-r*(T-t))*N(d2)
P = E*exp(-r*(T-t))*N(-d2) - S*N(-d1)

print('              P+S =', P+S)
print('C+E*exp(-r*(T-t)) =', C+E*exp(-r*(T-t)))
```

# 3 Question 3

**Statement 3 ▶ Study Note**

1. Write a study note of Black-Scholes' 73 paper.
2. Find a sequence of works following this paper and sort out hot topics nowadays.

## 3.1 The Black-Scholes World

The Black–Scholes model assumes that the market consists of at least one risky asset, usually called the stock, and one riskless asset, usually called the money market, cash, or bond. Now we make assumptions on the assets (which explain their names):

- (riskless rate) The rate of return on the riskless asset is constant and thus called the risk-free interest rate.
- (random walk) The instantaneous log return of stock price is an infinitesimal random walk with drift; more precisely, the stock price follows a geometric Brownian motion, and we will assume its drift and volatility are constant (if they are time-varying, we can deduce a suitably modified Black–Scholes formula quite simply, as long as the volatility is not random).
- The stock does not pay a dividend.

Assumptions on the market:

- There is no arbitrage opportunity (i.e., there is no way to make a riskless profit).
- It is possible to borrow and lend any amount, even fractional, of cash at the riskless rate.
- It is possible to buy and sell any amount, even fractional, of the stock (this includes short selling).
- The above transactions do not incur any fees or costs (i.e., frictionless market).

## 3.2 Black–Scholes Equation

As above, the **Black–Scholes equation** is a partial differential equation, which describes the price of the option over time. The equation is:

$$\frac{\partial V}{\partial t}+\frac{1}{2}\sigma^2S^2\frac{\partial^2 V}{\partial S^2}+rS\frac{\partial V}{\partial S}-rV=0. \tag{1}$$

### 3.3 Black–Scholes Formula

The value of a call option for a non-dividend-paying underlying stock in terms of the Black–Scholes parameters is:

$$\begin{aligned} C(S_t, t) &= N(d_1)S_t - N(d_2)PV(K) \\ d_1 &= \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right] \\ d_2 &= d_1 - \sigma\sqrt{T-t} \\ PV(K) &= K\exp\left(-r(T-t)\right) \end{aligned} \tag{2}$$

The price of a corresponding put option based on put–call parity is:

$$\begin{aligned} P(S_t, t) &= K\exp\left(-r(T-t)\right) - S_t + C(S_t, t) \\ &= N(-d_2)K\exp\left(-r(T-t)\right) - N(-d_1)S_t. \end{aligned} \tag{3}$$

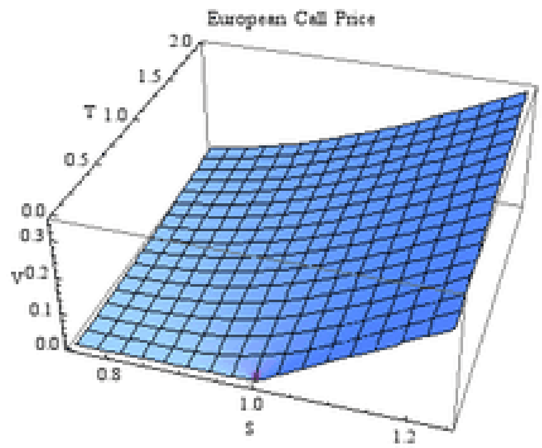



Figure 1: A European call valued using the Black-Scholes pricing equation for varying asset price $S$ and time-to-expiry $T$

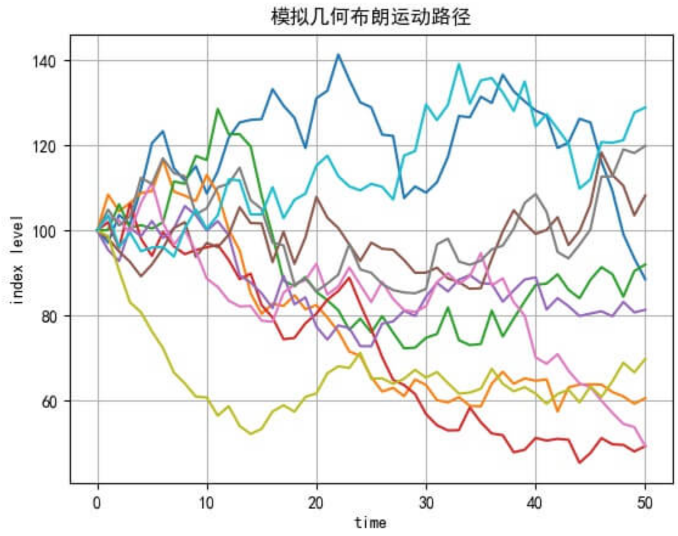
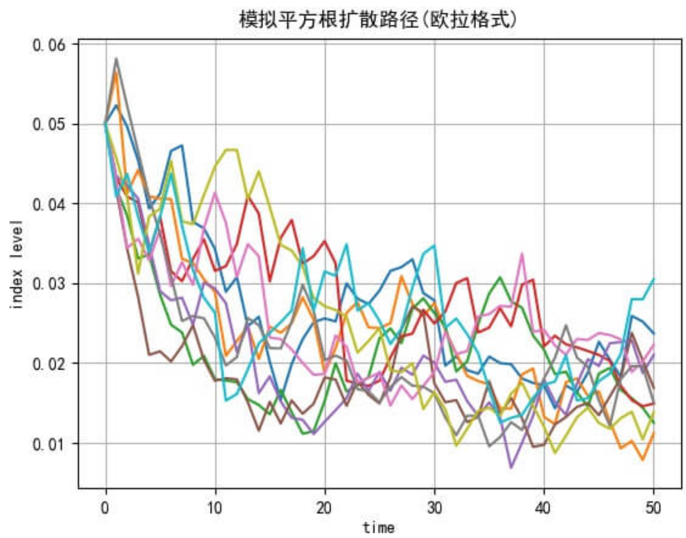
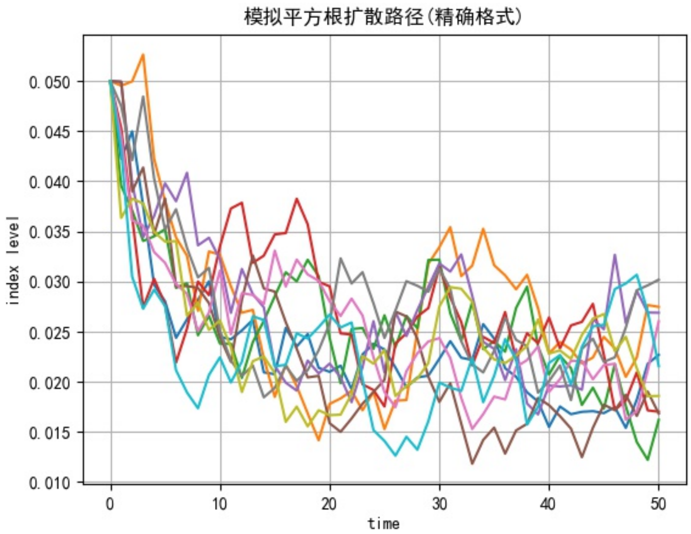



Figure 2: Simulated geometric Brownian motions with parameters from market data





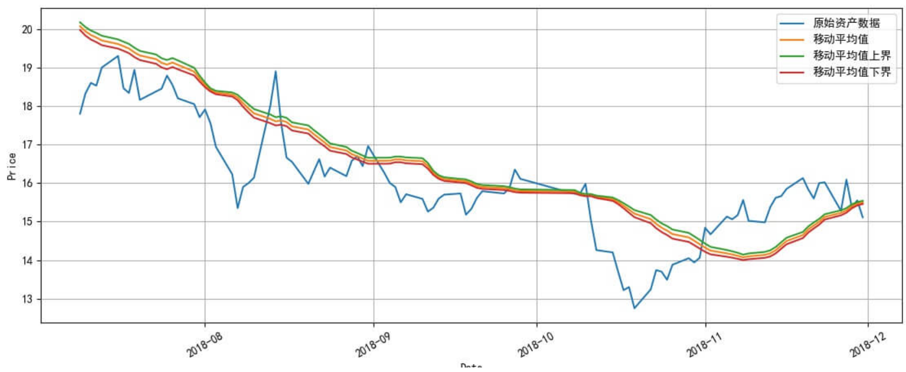




Figure 3: Bolling graph


Figure 4: Candle graph

# A Simulation Program

simulation.py

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
# 比较模拟结果的分布特性.
import scipy.stats as scs
# 字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']


def print_statistics(a1, a2):
    """
    Prints selected statistics.

    Parameters
    ==========
    a1, a2 : ndarray objects
    results object from simulation
    """
    # 主体
```

```
    sta1 = scs.describe(a1)
    sta2 = scs.describe(a2)
    print("%14s %14s %14s"%("statistic", "data set 1", "data set 2"))
    print("-" * 45)
    print("%14s %14.3f %14.3f"%("size", sta1[0], sta2[0]))
    print("%14s %14.3f %14.3f"%("min", sta1[1][0], sta2[1][0]))
    print("%14s %14.3f %14.3f"%("max", sta1[1][1], sta2[1][1]))
    print("%14s %14.3f %14.3f"%("mean", sta1[2], sta2[2]))
    print("%14s %14.3f %14.3f"%("std", np.sqrt(sta1[3]), np.sqrt(sta2[3])))
    print("%14s %14.3f %14.3f"%("skew", sta1[4], sta2[4]))
    print("%14s %14.3f %14.3f"%("kurtosis", sta1[5], sta2[5]))


S0    = 100    # initial value
r     = 0.05   # constant short rate
sigma = 0.25   # constant volatility
T     = 2.0    # in years
I     = 10000  # number of random draws

ST1   = S0 * np.exp(
            (r-sigma**2/2)*T + sigma*np.sqrt(T)*npr.standard_normal(I)
        )
ST2   = S0 * npr.lognormal(mean  = (r-sigma**2/2)*T,
                           sigma = sigma*np.sqrt(T),
                           size  = I)
print_statistics(ST1, ST2)

"""
plt.hist(ST1, bins=50)
plt.xlabel("index level")
```

```
plt.ylabel("frequency")
plt.grid(True)
plt.show()
"""


# 几何布朗运动
S0      = 100     # initial value
r       = 0.05    # constant short rate
sigma   = 0.25    # constant volatility

M  = 50
T  = 2.0
I  = 10000
dt = T / M
S  = np.zeros((M+1, I))
S[0] = S0
for t in range(1, M+1):
    S[t] = S[t-1] * npr.lognormal(mean  = (r-sigma
        **2/2)*dt,
                                  sigma = sigma*np.
                                      sqrt(dt),
                                  size  = I)

plt.plot(S[:, :10], lw=1.5)
plt.title("模拟几何布朗运动路径")
plt.xlabel("time")
plt.ylabel("index␣level")
plt.grid(True)
plt.show()


# 平方根扩散(欧拉格式)
x0    = 0.05
kappa = 3.0
theta = 0.02
sigma = 0.1

M  = 50
```

```
T   = 2.0
I   = 10000
dt  = T / M
x   = np.zeros((M+1, I))
x[0] = x0
clamp = lambda x: np.maximum(x, 0)
for t in range(1, M+1):
    x[t] = x[t-1] + kappa*(theta-clamp(x[t-1]))*dt \
        + sigma*np.sqrt(clamp(x[t-1]))*np.sqrt(dt)*
          npr.standard_normal(I)
x = clamp(x)

plt.plot(x[:, :10], lw=1.5)
plt.title("模拟平方根扩散路径(欧拉格式)")
plt.xlabel("time")
plt.ylabel("index␣level")
plt.grid(True)
plt.show()


# 平方根扩散(精确格式)
x0     = 0.05
kappa  = 3.0
theta  = 0.02
sigma  = 0.1

M   = 50
T   = 2.0
I   = 10000
dt  = T / M
y   = np.zeros((M+1, I))
y[0] = x0
for t in range(1, M+1):
    df = 4*theta*kappa / sigma**2
    c  = (sigma**2 * (1-np.exp(-kappa*dt))) / (4*
        kappa)
    nc = np.exp(-kappa*dt) * y[t-1] / c
    y[t] = c * npr.noncentral_chisquare(df, nc, size
        =I)
```

```
plt.plot(y[:, :10], lw=1.5)
plt.title("模拟平方根扩散路径(精确格式)")
plt.xlabel("time")
plt.ylabel("index␣level")
plt.grid(True)
plt.show()


# 比较欧拉格式与精确格式
print_statistics(x[-1], y[-1])


# 随机波动率
r     = 0.05
v0    = 0.1
kappa = 3.0
theta = 0.25
sigma = 0.1
rho   = 0.6

M  = 50
T  = 2.0
I  = 10000
dt = T / M
v  = np.zeros((M+1, I))
v[0] = v0

ran_num  = npr.standard_normal((2, M+1, I))
corr_mat = np.array([[1.0,rho], [rho,1.0]])
cho_mat  = np.linalg.cholesky(corr_mat)
clamp    = lambda x: np.maximum(x, 0)
for t in range(1, M+1):
    ran = np.dot(cho_mat, ran_num[:,t,:])
    v[t] = v[t-1] + kappa*(theta-clamp(v[t-1]))*dt \
           + sigma*np.sqrt(clamp(v[t-1]))*np.sqrt(dt)*
             ran[1]
v = clamp(v)

```

```
w0 = 100
w  = np.zeros((M+1, I))
w[0] = w0
for t in range(1, M+1):
    ran = np.dot(cho_mat, ran_num[:,t,:])
    w[t] = w[t-1] * np.exp((r-v[t])*dt + np.sqrt(v[t])*np.sqrt(dt)*ran[0])

fig,(ax1,ax2) = plt.subplots(2, 1, sharex=True, figsize=(7,6))
ax1.plot(w[:,:10], lw=1.5)
ax1.set_title("模拟随机波动率模型路径")
ax1.set_ylabel("index level")
ax1.grid(True)
ax2.plot(v[:,:10], lw=1.5)
ax2.set_xlabel("time")
ax2.set_ylabel("volatility")
ax2.grid(True)

fig.show()
```

# B Plot Function

bolling.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time     : 2018/01/23 14:00
# @Author   : Iydon
# @File     : bolling.py

# 绘图
import matplotlib.pyplot as plt
# 字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
```

```python


def date_to_num(dates:list):
    """
    ["2017-01-01", "2017-01-02", ...]
    =>
    [736330.0, 736331.0, ...]
    """
    # 数据
    import datetime
    from matplotlib.pylab import date2num
    # 主体
    strptime = datetime.datetime.strptime
    return [date2num(strptime(date, "%Y-%m-%d")) for
        date in dates]

def get_data():
    """
    Get demo data.
    """
    # 数据
    import tushare as ts
    # 主体
    data = ts.get_k_data("002738", "2018-06-01", "
        2018-12-01")
    ret  = data.values
    ret[:,0] = date_to_num(ret[:,0])
    return ret

def bolling(asset:list, samples:int=20, alpha:float
    =0, width:float=2):
    """
    According to MATLAB:

    BOLLING(ASSET,SAMPLES,ALPHA,WIDTH) plots
        Bollinger bands for given ASSET
    data vector. SAMPLES specifies the number of
        samples to use in computing
    the moving average. ALPHA is an optional input
```

```
        that specifies the exponent
    used to compute the element weights of the moving average. The default
    ALPHA is 0 (simple moving average). WIDTH is an optional input that
    specifies the number of standard deviations to include in the envelope. It
    is a multiplicative factor specifying how tight the bounds should be made
    around the simple moving average. The default WIDTH is 2. This calling
    syntax plots the data only and does not return the data.

    Note: The standard deviations are normalized by (N-1) where N is the
    sequence length.
    """
    # build weight vector
    import numpy as np
    # 主体
    r = len(asset)
    i = np.arange(1,samples+1) ** alpha
    w = i / sum(i)
    # build moving average vectors with for loops
    a = np.zeros((r-samples, 1))
    b = a.copy()
    for i in range(samples, r):
        a[i-samples] = np.sum( asset[i-samples:i] * w )
        b[i-samples] = width * np.sum(np.std( asset[i-samples:i] * w ))
    return a,a+b,a-b


data = get_data()
date = data[:,0]

samples = 20
```

```python
mav,uband,lband = bolling(data[:,1], samples, 0, 2)
ind = range(samples, len(data[:,1]))

fig,ax = plt.subplots(figsize=(15,5))
plt.plot(date[ind], data[ind,1])
plt.plot(date[ind], mav)
plt.plot(date[ind], uband)
plt.plot(date[ind], lband)
plt.legend(['原始资产数据','移动平均值','移动平均值上界','移动平均值下界'])

plt.xticks(rotation=30)
ax.xaxis_date()
plt.xlabel('Date')
plt.ylabel('Price')

plt.grid(True)
plt.show()
```

candle.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Time    : 2018/01/23 14:00
# @Author  : Iydon
# @File    : candle.py

# 绘图
import matplotlib as mpl
import matplotlib.pyplot as plt
import mpl_finance as mpf


def date_to_num(dates:list):
    """
    ["2017-01-01", "2017-01-02", ...]
    =>
    [736330.0, 736331.0, ...]
```

```
    """
    # 数据
    import datetime
    from matplotlib.pylab import date2num
    # 主体
    strptime = datetime.datetime.strptime
    return [date2num(strptime(date, "%Y-%m-%d")) for
        date in dates]

def get_data():
    """
    Get demo data.
    """
    # 数据
    import tushare as ts
    # 主体
    data = ts.get_k_data("002738", "2018-06-01", "
        2018-12-01")
    ret  = data.values
    ret[:,0] = date_to_num(ret[:,0])
    return ret


data = get_data()

fig,ax = plt.subplots(figsize=(15,5))
mpf.candlestick_ochl(ax, data[:,:-2], width=0.6,
    colorup='g', colordown='r', alpha=1.0)

plt.xticks(rotation=30)
ax.xaxis_date()
plt.xlabel('Date')
plt.ylabel('Price')

plt.grid(True)
plt.show()
```