SHI BO

MF703

MIDTERM EXAM

Problem 1 (7 points) SQL Programming

1.1)

SELECT TOP 1 close

FROM instrument_prices

WHERE close IN (SELECT close FROM instrument_prices WHERE quote_date >= 'date1' AND quote_date <= 'date2' AND ticker='tickername' ORDER BY close DESC)

1.2)

instrument_id and ticker are both available to join the instrument_info and instrument_prices tables.

But instrument_id is better than ticker, because It is best practice to JOIN on int data types rather than strings.

1.3)

SELECT close, share_outstanding

FROM instrument_info

LEFT OUTER JOIN instrument_prices

ON instrument_info.instrument_id = instrument_prices.instrument_id

SET close = 0.0

WHERE close IS NULL

1.4)

SELECT SUM(share_outstanding*close) AS market_value

FROM instrument_prices INNER JOIN instrument_info

ON instrument_info.instrument_id = instrument_prices.instrument_id

INNER JOIN sectors ON sectors.sector_id =instrument_info.sector_id

GROUP BY sectors.sector_id

ORDER BY market_value DESC

HAVING market_value > 'given value'

Problem 2 (10 points) Data Cleaning & Algorithms

2.1)

```
import pandas as pd

SPY = pd.read_csv('SPY.csv',index_col = 'Date')
SPY = SPY['Adj Close']
print('Summation of Missing data(year 2012) is: ',SPY.isna().sum())

def interpolate_missing_data(data):
    data = pd.DataFrame(data)
    data = data.interpolate(method = 'linear').ffill().bfill()
    return data

SPY = interpolate_missing_data(SPY)
print('Summation of Missing data after using interpolate method is: ',SPY.isna().sum())
```

```
Date
2011-08-24     98.086784
2011-08-25     96.591537
2011-08-26     97.995422
2011-08-29    100.811424
2011-08-30    101.077255
                  ...
2012-01-10           NaN
2012-01-11           NaN
2012-01-12           NaN
2012-01-13           NaN
2012-01-17           NaN
Name: Adj Close, Length: 100, dtype: float64

In [56]: runfile('D:/用户文件/桌面/未命名0.py', wdir='D:/用户文件/桌面')
Summation of Missing data(year 2012) is:  250
Summation of Missing data after using interpolate method is:  0
```

```
Date
2011-08-24     98.086784
2011-08-25     96.591537
2011-08-26     97.995422
2011-08-29    100.811424
2011-08-30    101.077255
                  ...
2012-01-10    105.932405
2012-01-11    106.012076
2012-01-12    106.091747
2012-01-13    106.171419
2012-01-17    106.251090
Name: Adj Close, Length: 100, dtype: float64
```

The method I am going to use is called linear interpolation which is an imputation technique that **assumes a linear relationship** between data points and utilizes non-missing values from adjacent data points to compute a value for a missing data point. The advantage of this method is that it has the advantages that it is fast, make data smoother and least sophisticated However, it is not very accurate and it will skew the original distribution since the data may be non-linear.

2.2)

```
def binary_insert_sort(l) :
    length = len(l)
    for i in range(1, length) :
        be, en = 0, i - 1
        mid = (en + be) // 2

        while True :
            if  mid < 0 or l[mid] == l[i] or (be >= en):
                if  mid >= 0 and l[mid] > l[i] :
                    l.insert(mid, l.pop(i))
                else :
                    l.insert(mid + 1, l.pop(i))
                break
            elif l[mid] > l[i] :
```

```
        en = mid - 1
    else :
        be = mid + 1
    mid = (en + be) // 2
return l
```

The advantages of binary search algorithm are-

- It eliminates half of the list from further searching by using the result of each comparison.
- It indicates whether the element being searched is before or after the current position in the list.
- This information is used to narrow the search.
- For large lists of data, it works significantly better than linear search.
- binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient.
- The time complexity of this algorithm is generally O(nlogn), it is more efficient than original insertion sort algorithm.
-

The disadvantages of binary search algorithm are-

- It employs recursive approach which requires more stack space.
- Programming binary search algorithm is error prone and difficult.
- In the worst case the time complexity is also $O(n^2)$ and this algorithm requires the list to be sorted
- The interaction of binary search with memory hierarchy i.e. caching is poor. (because of its random access nature)

2.3)

The template method can be used to create a universal method to support a variety of different formal parameters, and avoid the repeated design of the method body of the overloaded methods.

Its biggest feature is treat the data type as a parameter.

It is used in object-oriented programming for polymorphism concept.

```
class BSM:
    def __init__(self, args):
        pass
    def price_paths(self):
        pass
    
class Option:
    def __init__(self, args):
        pass
    def pricing(self,args):
        pass
    
class lookback(Option):
    def __init__(self, args):
        pass
    def payoff(self, price_paths, other args):
        # Some codes for lookback option
        
class European(Option):
    def __init__(self, args):
        pass
    def payoff(self, price_paths, other args):
        # Some codes for European option
```

2.4)

Local Working Copy: Working copy is your current working directory. It can get updates from local repo and local changes on the working copy can be committed to local repo.

Local Repository: Local Repository can get updates from remote repo. When publication is desired, the modifications are pushed to remote repo from local repo.

Remote Repository: Remote Repository is a public repository, others can get code from your remote repo.

Use 'commmit' to check in the changes to my local repo, use 'push' to check in the changes to my remote repo.

When I do not want the changes go public, I can check in the changes to my local repo.

Problem 3 (10 points) Finance Application

1) Based on the initial value provided, the price of European call by using BSM is approximately 0.4401345232422509 and by using Bachelier model is

approximately 0.0. It is obvious that the Bachelier will lead to lower price since the volatility part of stock price is constant and is not amplified
by St.

2) We assume that the price if lookback put is P1 and European put is P2 and P1 > P2, then two conditions appear. For the first condition, the minimum price along the path is higher than the strike so the investor will incur a loss of P1-P2 regardless of what condition at expiry is. For the second condition, the minimum price along path is lower than strike. The investor will have P&L min{St,K}-min{St} for 0<=t<=T + P2 -P1, which implies that the lower the minimum price the better, as for the final price, it would be better if terminal price is higher of equal to the strike.
3) A rolling realized volatility is the sum of squared log daily returns of asset for rolling days. Generally, the premium (which also means the implied vol is greater than realized vol) is positive except for a few of exceptional periods like 2008 crisis. The premium is high when market is doing well and is low when shocks hit. Recall what we concluded in HW3, we found that there is a negative relationship of profit and loss against the premium of straddle between implied and realized volatility since the payoff of this straddle is positively related to the realized volatility but the option prices that you need to pay is positively related to the implied volatility. As a investor, we could find that options premiums are assumed to be overvalued so that we can make money by shorting options in this period.
4) A statistically significant negative coefficient in S&P index means that there is a negative autocorrelation between current and previous day's return. This also turns out that no momentum factor is associated with index so that as a investor we might not reasonably expect the movements over the upcoming several days (the leading time series) to match those of the lagging time series and to move upward. Same explanation also applies in VIX index. So, we can assume that the implied volatility will increase of decrease persistently and we could take this as an advantage to long options.

Problem 4 (15 points) Simulation Algorithms

4.1）

C = Call premium
S0=Current stock price at time 0
T=time until option exercise
K= strike price
N = Cumulative standard normal distribution
M=Maximum value of the Asset over a period
v=volatility (σ)
C0 = payoff

dSt =Change in Stock price st time t

θ = long term mean level

κ = speed of reversion

B = barrier level

The only difference between OU mdoel and Bachelier model is that the OU model utilizes the mean reversion method instead of a constant factor which is r(interest rate). If we want to simplify this model to Bachelier what we only need to modify is to replace the mean reversion part $\kappa = 0$.

4.2）

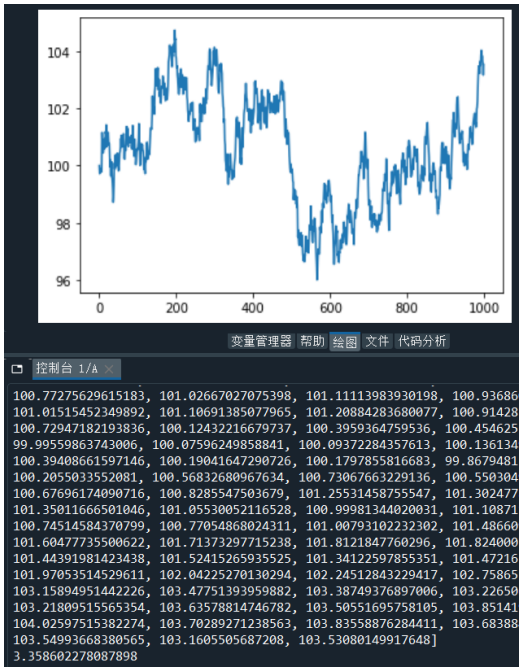
This is not a good model for equity index since we assume that OU process is continuous-time model and stationary (identically distributed) so that it fits very well for volatility and interest rates because interest rates vary over time both in terms of level and volatility, but it seems reasonable that the mean interest rate around which they vary is constant over time and not unreasonable to neglect the volatility changes as a first approximation. However, it is not compatible for equity index since the stock price may have 'jumps' over time so it will not be continuous. As a result, it is optimal to use OS model for volatility and interest rate modeling.

4.3）

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt




def OH_Process(s0, kappa, theta, sigma, t_start,t_end, N):

    t = np.linspace(t_start,t_end,N) # define time axis
    dt = np.mean(np.diff(t))

    drift = lambda y,t: kappa*(theta-y) # define drift term
    diffusion = lambda y,t: sigma # define diffusion term
    noise = np.random.normal(loc=0.0,scale=1.0,size=N)*np.sqrt(dt) #define noise process
    # solve SDE
    short_s = [0]*N # Create array to store rates
    short_s[0] = s0 # Initialise rates at $r_0$
    for i in range(1,N):
        short_s[i] = short_s[i-1] + drift(short_s[i-1],i*dt)*dt + diffusion(short_s[i-1],i*dt)*noise[i]
    return short_s
```

4.4）

```
def up_and_out_call( N, s0, kappa, theta, sigma, t_start,t_end, barrier, K, r):
    
    sim = OH_Process(s0, kappa, theta, sigma, t_start,t_end, N)
    print(sim)
    plt.plot(sim)
    
    if max(sim) > barrier:
        return 0
    else:
        T = t_end - t_start
        price = np.exp(-r*T) * max((sim[-1] -K),0)
    return price
    
    
price = up_and_out_call( 1000, 100, 0.4, 90, 10, 0,1, 105, 100, 0.05)
print(price)
```




```
100.77275629615183, 101.02667027075398, 101.11113983930198, 100.93686
101.01515452349892, 101.10691385077965, 101.20884283680077, 100.91428
100.72947182193836, 100.12432216679737, 100.3959364759536, 100.454625
99.99559863743006, 100.07596249858841, 100.09372284357613, 100.136134
100.39408661597146, 100.19041647290726, 100.1797855816683, 99.8679481
100.2055033552081, 100.56832680967634, 100.73067663229136, 100.550304
100.67696174090716, 100.8285547503679, 101.25531458755547, 101.302477
101.35011666501046, 101.05530052116528, 100.99981344020031, 101.10871
100.74514584370799, 100.77054868024311, 101.00793102232302, 101.48660
101.60477735500622, 101.71373297715238, 101.8121847760296, 101.824000
101.44391981423438, 101.52415265935525, 101.34122597855351, 101.47216
101.97053514529611, 102.04225270130294, 102.24512843229417, 102.75865
103.15894951442226, 103.47751393959882, 103.38749376897006, 103.22650
103.21809515565354, 103.63578814746782, 103.50551695758105, 103.85141
104.02597515382274, 103.70289271238563, 103.83558876284411, 103.68388
103.54993668380565, 103.1605505687208, 103.53080149917648]
3.358602278087898
```

4.5)

The simulation parameters that we must choose is the Kappa, Theta and sigma since we need the sigma*dw(wiener process) to simulate the path of price and Kappa determines how fast this price can revert to its mean. Theta turns out that we must set a long-term mean up to approach.

From the charts above I simulated the price for 1000 steps, initial price = 100, speed of reversion = 0.4, theta = 90, sigma = 10, time = 1, and barrier = 105, strike price = 100 and r = 0.05. Among those parameters, once the price of the stock goes above 105 it will knocked out so that the premium for the option will be 0. But in our medel this time, it did not touch the line and the final price is 3.3586. The speed of mean reversion can be fast or slow depending on what number we

choose for example we can choose kappa equals to 1.5 which means that the speed is fast since it is greater than 1 vice versa. From the prospective of my model, the optima will be 0.4 to 0.6 which will generate literally proper result for this model and we can clearly see how the stochastic process do.

4.6）

i. The first test for the SDE is test to test for the final stock price accuracy at specific time by subjecting values whose stock price evaluation is known.

ii. The second is to test for change in stock price using the SDE formula, by subjecting known values as well.

iii. Third unit test is to test the American up-and-out payoff evaluation algorithm using known values and outcome.

Problem 5 (10 points) Python Concepts

5.1)

```
def merge(df1,df2):
    df1.merge(df2, how='outer', left_on='date', right_on='date')
    return df1
```

5.2)

```
def rolling(df1,df2,period_length):
    df = merge(df1,df2)
    df = df.dropna()
    df_daily_return = (df - df.shfit(1))/df.shfit(1)
    df_rolling_return = df_daily_return.rolling(period_length).mean() * 252
    df_rolling_std = df_daily_return.rolling(period_length).std() *math.sqrt(252)
    new_df = pd.concat([df,df_rolling_return,df_rolling_std],axis = 1)
    return new_df
```

5.3)

```
def regime(df,mkt):
    df = df.join(mkt)
    df1 = df.groupby(['market regime']).mean()[['return x','return y']]
    df2 = df.groupby(['market regime']).std()[['return x','return y']]
    return df1.join(df2)
```

5.4)

```
def Fibonacci(n):
    if n <= 1:
        return n
    else:
        return Fibonacci(n - 1) + Fibonacci(n - 2)


def Fibonacci_Recursion(n):
    result_list = []
    for i in range(1, n + 1):
        result_list.append(Fibonacci(i))
    return result_list
```

Problem 6 (20 points) Object Oriented Programming in Python

6.1)

```
6.1
import abc
class OptionsPricer(object):
    __metaclass__ = abc.ABCMeta
    @abc.abstractmethod
    def price(self):
        return

    def __init__(self,S0,sigma,frequency,t,r,days):
        self.S0 =S0
        self.sigma = sigma
        self.frequency = freqyency
        self.T = T
        self.r = r
        self.days = days
        return
```

6.2)

```
class EuropeanOptionPricer(OptionsPricer):
    def __init__(self):
        return
    def price(self):
        return

class LookbackOptionPricer(OptionsPricer):
    def __init__(self):
        return
    def price(self):
        return
```

6.3)

```
import abc
class Base(object):
    __metaclass__ = abc.ABCMeta
    @abc.abstractmethod
    def simulation(self):
        return

    def __init__(self):
        return

class BlackScholes(Base):
    def __init__(self):
        return
    def simulation(self):
        return
    def __eq__(self):
        return
    def __lt__(self):
        return
    def __gt__(self):
        return

class Bachelier(Base):
    def __init__(self):
        return
    def simulation(self):
        return
    def __eq__(self):
        return
    def __lt__(self):
        return
    def __gt__(self):
        return
```

6.4)

```
6.4
def option_price(S0, K, T, r, sigma, days, op_type, frequency, sto_type, option_type):
    try:
        if sto_type == 'Bachelier':
            Bachelier_sim = Bachelier(S0, sigma, frequency, T, r,days)
            Bachelier_sim_path = Bachelier_sim.simulation()

            if option_type == 'European':
                option = EuropeanOptionPricer(K, T, r, op_type, Bachelier_sim_path)
            elif option_type == 'Lookback':
                option = LookbackOptionPricer(K, T, r, op_type, Bachelier_sim_path)
            else:
                print('option type error')

        elif sto_type == 'BSM':
            BSM_sim = BlackScholes(S0, sigma, frequency, T, r, days)
            BSM_sim_path = BS_sim.simulation()

            if option_type == 'European':
                option = EuropeanOptionPricer(K, T, r, op_type, Bachelier_sim_path)
            elif option_type == 'Lookback':
                option = LookbackOptionPricer(K, T, r, op_type, Bachelier_sim_path)
            else:
                print('option type error')

        else:
            print('stochastic type error')

        return option.price()

    except:
        print('Input parameters not correct')
```

BSM_sim_path

BSM_sim_path

Problem 7 (8 points) C++ Concepts

7.1)

argv is a pointer array, the parameters entered on the command line are stored in argv.

string s = argv[1];

cout << stod(s) << endl;

7.2)

calling default constructor // this comes from line 33, we constructed a instance of Foo and have no parameters, so it called the default constructor.

5 //this comes from line 35, after f.bar is assigned a value 5, then we print the value of f.bar.

calling copy constructor // this comes from line 37, when we initialize an instance from another instance we will call copy constructor

5 // comes from line 38, because we call copy constructor to initialize f1, so f1.bar is assigned the same value of f.bar

5 // comes from line 41, f.bar is not changed, so it is still 5

10 // comes from line 42, because we assigned 10 to f1.bar, so f1.bar is 10 now.

from line 45 the code will not run successfully, because bar is a local variable that not declared, as well as bar2.

7.3)

private: int bar;

7.4）

```
Foo& operator=(const Foo& t) {
            bar = t.bar;
        std::cout << "calling assignment operator" << std::endl;
            return *this;
}
```

7.5）

calling default constructor //comes from line 1, because we initialize an instance and have no parameters, so it called default constructor.

calling copy constructor // comes from line 4, because we initialize an instance f1 from an old instance f.

calling constructor with bar // comes from line 11, because we initialize an instance by passing a parameter 'bar', so it called an overloaded constructor.

calling copy constructor // comes from line 13, because we initialize an instance f3 from an old instance f2.

calling assignment operator // comes from line 16, because we already overloaded assignment operator, so when we assign from an instance to another instance, it called the overloaded assignment operator.

fPtr = std::make_shared<Foo>();

output will be:

calling default constructor

calling copy constructor

calling default constructor

calling constructor with bar

calling copy constructor

calling assignment operator

Problem 8 (20 points) Object Oriented Programming in C++

8.1）

```
class Foo {
public:
        Foo() {std::cout << " calling Foo constructor " << std::endl;}

       ~Foo() {std::cout << " calling Foo destructor " << std::endl;}

       virtual void func1() { std::cout << " calling Foo func1 " <<
std::endl; }
       void func2() { std::cout << " calling Foo func2 " << std::endl; }
};
```

8.2）

```
class FooKid :public Foo {
public:
        FooKid() { std::cout << " calling FooKid constructor " << std::endl; }

        ~FooKid() { std::cout << " calling FooKid destructor " << std::endl; }

        void func1() { std::cout << " calling FooKid func1 " << std::endl; }
```

```
        void func2() { std::cout << " calling FooKid func2 " << std::endl; }
};
```

8.3）

If we want to sort instances we might have a key in the member variables of class, that means we need to sort the instances based on the key value, for example, define 'int bar' as a member variable.

When we know the sort is based on the key value. We can overload the '<' or '>' operators to implement instance sort based on the key value.

```
bool operator <(const Foo& f)
{
   if(bar < f.bar)
   {
     return true;
   }
   return false;
}

bool operator >(const Foo& f)
{
   if(bar > f.bar)
   {
     return true;
   }
   return false;
}
```

8.4）

The output is:

calling Foo constructor // comes from line 2, because we initialize an instance of Foo class by calling 'new Foo()'

calling Foo constructor // comes from line 3, because we initialize an instance of derived class FooKid and it will call base class constructor at first.

calling FooKid constructor // comes from line 3, because we initialize an instance of derived class FooKid, so it will call itself constructor after the base class constructor.

calling Foo func1 // comes from line 5, because foo1 is a pointer of Foo class, so it will call the func1 method in Foo class.

calling Foo func2 // comes from line 6, because foo1 is a pointer of Foo class, so it will call the func2 method in Foo class.

calling FooKid func1 // comes from line 9, because foo2 is a pointer of FooKid class, so it will call the func1 method in FooKid class.

calling FooKid func2 // comes from line 10, because foo2 is a pointer of FooKid class, so it will call the func2 method in FooKid class.

calling Foo constructor // comes from line 12, because when we call FooKid class constructor it will automatically call the base class constructor(Foo class).

calling FooKid constructor // comes from line 12, after calling the base class constructor, FooKid constructor will be called.

calling FooKid func1 // comes from line 13, because foo3 is a pointer of Foo class, but foo3 is bind to an instance of FooKid, and func1 in Foo class is dynamic bind, so it will call func1 in FooKid.

calling Foo func2 // comes from line 14, because foo3 is a pointer of Foo class and bind to an instance of FooKid, also the func2 in Foo is static bind, polymorphism does not work here, so it will call func2 in Foo.

calling Foo constructor // comes from line 16, because foo4 is an instance of FooKid, when we initialize an instance of derived class it will automatically call the base class constructor.

calling FooKid constructor // comes from line 16, after calling the base class constructor, it will call itself constructor(Fookid constructor).

calling FooKid func1 // comes from line 19, because foo4 is an instance of FooKid, so it will call func1 in FooKid.

calling FooKid func2 // comes from line 20, because foo4 is an instance of FooKid, so it will call func2 in FooKid.

calling Foo func1 // comes from line 22, because foo5 is an instance of Foo, although it is intialized by copy constructor, but there is no polymorphism happens, so it calls func1 in Foo.

calling Foo func2 // comes from line 23, because foo5 is an instance of Foo, although it is intialized by copy constructor, but there is no polymorphism happens, so it calls func2 in Foo.