Thomas Kuntz: MF 703 Assignment 5

November 5, 2021

Problem 1:

(1)

```
SELECT MAX(close)
FROM instrument_prices
WHERE ticker = givenTicker AND
quote_date BETWEEN givenDate1 AND givenDate2
```

(2)

Columns that are available are ”instrument_id” and ”ticker”. The preferable column is ”instrument_id” since that should be a unique identifier and therefore should result in no collisions when joining the tables.

(3)

```
SELECT instrument_info.shares_outstanding, ISNULL(NULLIF(instrument_prices.close, ”), 0.0)
FROM instrument_info
LEFT JOIN instrument_prices
ON instrument_info.instrument_id = instrument_prices.instrument_id;
```

(4)

```
SELECT sectors.sector_name, instrument_info.shares_outstanding * instrument_prices.close AS mktVal
FROM sectors AS s
INNER JOIN instrument_info AS insInfo
ON s.sector_id = insInfo.sector_id
INNER JOIN instrument_prices AS insPr
ON insInfo.instrument_id = insPr.instrument_id
WHERE mktVal ¿ givenVal
GROUP BY sectors.sector_name
ORDER BY mktVal
```

Problem 2:

(1)

There are multiple ways this can be done, but bootstrapping and regression are the best for long periods of missing data. Regression seems like the best candidate here, where we can use an estimate of the data that we have to mimic the realized values that we are not filling in. this will do the best job at maintaining the integrity of the data in terms variances and other important information contained in the data. The higher the R-squared value, the better our chances are of seeing an accurate estimate of the missing data, and less bias.

(2)

```
start_date = '2000-01-01'
end_date = '2020-12-31'
df1 = data.get_data_yahoo(['GS','IBM'], start = start_date, end = end_date)['Adj Close']
df1 = df1.pct_change().dropna()
dfMissing = df1['2019-01-01':'2020-01-01']
dfFill = df1[df1.index.min():end_date]
ols = sm.OLS(dfFill['GS'], dfFill['IBM']).fit()
beta = float(ols.params)
gsFill = dfMissing['IBM'] * beta
dfMerge = pd.concat([dfMissing, df1[start_date:'2001-01-01']])
ols2 = sm.OLS(dfFill['GS'], dfFill['IBM']).fit()
beta2 = float(ols.params)
gsFill = dfMissing['IBM'] * beta
```

The advantage if insertion sort by binary search is that it is done in place so it has O(1) space complexity. This is good for mostly linear data, but where it is not good is that it can have a long time complexity of O(n Log(n)) because it is a log(n) operation and this needs to be performed on each element in the array with length n.

```
def binary_search(arr, low, high, x):
    if high >= low:
        mid = (high + low) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)
        else:
            return binary_search(arr, mid + 1, high, x)
    else:
        return -1
# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10
# Function call
result = binary_search(arr, 0, len(arr)-1, x)
if result != -1:
    print("Element is present at index", str(result))
else:
    print("Element is not present in array")
```

(3)

The template method is a design that is utilized in object oriented programming. It lets the user define and construct base and super classes so that we can let other sub-classes and their methods override methods defined in the super class so that we don't change the skeleton of the super class. This may look like polymorphism and inheritance which are highly dependant on the template method. This is because the super class can't change the subclass functionality.

(4)

Local repository resides on the computers of team members. A remote repository is hosted on a server that is accessible for all team members. A Local repository has a working copy: a directory where some version of your project's files are checked. A Remote repository doesn't have this: it only consists of the bare '.git' repository folder.

First, use 'git status' to check all of changes, then use 'git add' to add a new local files to a local repository, then use 'git commit' to take changes to all local files and commit them to the local repository.

Second, use 'git push' to push all committed changes from the local repository to the a remote repository, then use 'git merge' to merge changes with from the remote repository.

The actual work of the project only happens in a local repository. All modifications must be done locally and committed. These changes are uploaded to a remote repository to share with other people. A remote repository isn't intended for working with files, but only as a way to share and exchange code between developers. Thus why you might check in your code to the Local Repository but not the Remote Repository.

Problem 3:

(1)

The BS model will give a higher price then the the Bachelier model because it is log-normally distributed and the Bachelier model is normally distributed. This leads to a slightly higher price when we solve the SDEs and this is because the BS model follows a geometric Brownian motion and the Bachelier model follows an arithmetic Brownian motion. Combining these two factors leads to a higher option price.

(2)

The best P&L for us is when we see the stock price is just barely in the money at expiration. This means that both options will be exercised, and we will get exercised against on the short position on the European put, but we will only see minimal losses, maybe even a small gain depending on the premium we received for selling the put. The second part of this strategy is that we will wan the stock path to take a massive dip before rising back up to the strike price. This will make the Lookback very profitable and we will make a sizable amount of money off of the Lookback.

The worst P&L is when the stock price ends just in the money ate expiry, but never dips below the strike price until expiry. The Lookback is essentially a European put now and does not benefit us anymore then going long the European put. We will get exercised on the short position again, and this time we do not have the Lookback gains to offset these losses. Thus we lose the difference between the premium we paid for the Lookback and the premium we collected selling the European put.

(3)

If implied volatility is consistently greater then the realized volatility, options will be rich. This favors options sellers and so we would want to be short options prices with the idea of collecting the premiums. We do this because it will let us take advantage of the convergence between implied and historical realized volatilities. As implied volatility moves towards the realized volatility values, the options prices will trade down and we profit from the loss of value.

(4)

If the S&P has a negative one day coefficient, then is is a mean reverting process so we would expect to see it go up tomorrow if it went down today. We would take advantage of this by shorting the S&P on days following an up day, and then reverse the trade and go long the next day. The VIX is slightly different because it is calculated as levels of forward implied volatility. This means we have to difference it to turn it into a return series. The same coefficient is nice to see, but it doesn't help us much as investors because we can't directly trade the VIX. We could trade futures but but this is not an ideal strategy, and we still are not directly trading the asset. Thus the mean reverting strategy really only applies to the S&P.

Problem 4:

(1)

θ is the mean of the asset.

κ is the speed at which the persistence of reversion decays.

σ is absolute dollar variation

The rest of the parameters are all the same as in our other models. We need to turn the SDE from a relative change measure to an absolute change model. WE can do this by essentially dropping the the S_t from the model provided, so it goes from,

$$dS_t = \kappa(\theta - S_t)dt + \sigma dW_t$$

to

$$dS_t = \kappa(\theta)dt + \sigma dW_t \tag{1}$$

(2)
This model would not work well for in index since indexes often have a positive or negative drift, and because this model is mean reverting, it does not fit the underlying characteristics of the index. Thus i would not use it to model an index. I would use it to model rates and volatility because they are highly mean reverting, There is a specific adaptation of this model called the Vasicek model made specifically for rate modeling.
(3)
The attached jupyter notebook has the code for part 3.
(4)
The attached jupyter notebook has the code for part 4.
(5)
AmericanUpAndOutOption(S0, K, B, kappa, theta, t, r, sigma, n, num_trials)
AmericanUpAndOutOption(100, 100, 108, 0.2, 100, 1, 0.0, 10.0, 252, 1000)

We can see that: S0=100, K=100, B=108, Kappa=0.2, theta=100, t=1, r=0.0, sigma=10.0, number of periods=252, num_trials=1000
We can really only mess with sigma and Kappa, so we will want to find the point or multiple points where the stock path gets close to the barrier value, but never actually touches it, the earlier we get close to the barrier the better because we can exercise the option for a quick and profitable trade.
(6)
We could asset that the call price is never less then zero. We could assert that the option is cheaper then then a vanilla American call. This may not seem intuitive at fist, but because upside is limited, it will be cheaper then the vanilla counterpart which has unlimited upside. A third unit test could be to be to check that the option is pricing at zero when the stock price rises above the barrier price.

Problem 5:
(1)

```
def mergeDF(df1, df2):
    dataframe = pd.merge(df1, df2, how="outer", left_index=True, right_index=True)
    return dataframe
```

(2)

```
def rolling_overlapping(df1, df2, length):
    dataframe = mergeDF(df1, df2)
    def multi_period_return(period_returns):
        return np.prod(period_returns + 1) - 1
    for column in list(dataframe.columns):
        close_price = dataframe[column].dropna()
        rolling_return = close_price.pct_change().rolling(length).apply(multi_period_return)
```

```
            rolling_std = close_price.pct_change().rolling(length).std()*(252**0.5)
            dataframe2 = pd.concat([rolling_return, rolling_std], axis=1)
            dataframe2.columns = ['rolling_return', 'rolling_std']
            dataframe = pd.merge(dataframe, dataframe2, how="left", left_index=True, right_index=True)
        return dataframe
(3)
def conditionalVol(df1, d22, df_market_regime):
        dataframe = mergeDF(df1, df2)
        dataframe["return_equity1"] = dataframe.columns[0].pct_change()
        dataframe["return_equity2"] = dataframe.columns[1].pct_change()
        dataframe = pd.merge(dataframe, df_market_regime, how="right", left_index=True, right_index=True)
        dataframe = pd.get_dummies(dataframe)
        rtn1 = []
        vol1 = []
        rtn2 = []
        vol2 = []
        col_name = []
        for i in range(4, len(dataframe.columns)):
            col_name.append(dataframe.columns[i])
            df = dataframe[dataframe.columns[i] == 1]
            mean_return_equity1 = df["return_equity1"].mean()
            rtn1.append(mean_return_equity1)
            std_return_equity1 = df["return_equity1"].std()
            vol1.append(std_return_equity1)
            print(f'With market regime condition dataframe.columns[i], Average return of equity1 is
mean_return_equity1, volatility is std_return_equity1.')
            mean_return_equity2 = df["return_equity2"].mean()
            rtn2.append(mean_return_equity2)
            std_return_equity2 = df["return_equity2"].std()
            vol2.append(std_return_equity2)
            print(f'With market regime condition dataframe.columns[i], Average return of equity2 is
mean_return_equity2, volatility is std_return_equity2.')
        dfCondition = pd.DataFrame(list(zip(rtn1, vol1, rtn2, vol2)), index =[rtn1, vol1, rtn2, vol2], columns
= col_name)
        return dfCondition
(4)
def fibonacci(n):
        if n ¡= 1:
            return n
        else:
            return(fibonacci(n-1) + fibonacci(n-2))
    nterms = 10
    if nterms ¡= 0:
        print("Plese enter a positive integer")
    else:
```

```
print(”Fibonacci sequence:”)
for i in range(nterms):
   print(fibonacci(i))
```

Problem 6:

(1)
in attached jupyter notebook
(2)
in attached jupyter notebook
(3)
in attached jupyter notebook
(4)
in attached jupyter notebook

Problem 7:

(1)
The argv parameter stores the arguments entered in the command line, and argc is the number of strings pointed to argv. argv is an argument vector such that each element is passed in from the command line. It is also a pointer to an array of characters i.e. an array of C-strings. To change argv to a double:

```
char* my_char_string = argv[2];
double y = atof(my_char_string.c_str());
```

(2)
The output of the function is:

```
calling default constructor
5
calling copy constructor
5
5
10
calling constructor with bar
5
5
5
5
calling constructor with bar
5
```

Explain each line of output, what is causing it and which object it is coming from.
”calling default constructor”: gets printed out when Foo f; on line 33 gets called since it is the default constructor and it is coming from object f

”5”: gets printed out when std :: cout ¡¡ f.bar ¡¡ std :: endl ; on line 35 gets called since object f has property bar which is initialized to 5 and is coming from object f

”calling copy constructor” : gets printed out when Foo f1(f); on line 37 gets called and comes from object f1

”5”: gets printed out when std :: cout ¡¡ f.bar ¡¡ std :: endl ; on line 38 gets called since object f1 has property bar initialized to 5 and comes from object f1

”5”: gets printed out when std :: cout ¡¡ f.bar ¡¡ std :: endl ; on line 41 gets called since f.bar was initialized to 5 on line 34 and is coming from object f

”10”: gets printed out when std :: cout ¡¡ f1.bar ¡¡ std :: endl ; on line 42 gets called since f1.bar and was initialized to 10 on line 40 and comes from object f1

”calling constructor with bar”: gets printed out when Foo f2(f.bar); on line 44 gets called since we want to initialize object f2 with an instance of another object f. Now, f2.bar is equal to f.bar.

”5”: prints out when std :: cout ¡¡ f2.bar ¡¡ std :: endl ; on line 47 gets called since f2 still has value of f.bar which is 5 since nothing was overridden. This comes from object f2

”5”: prints out when std :: cout ¡¡ bar ¡¡ std :: endl ; on line 48 gets called since all that getBarValue method does is assign current bar value to a variable and return bar. f2.bar is still 5, even though temp_bar = 25, f2.getBarValue returns 5 since f2.bar = 5. This comes from object f2.

”5”: prints out when std :: cout ¡¡ f2.bar ¡¡ std :: endl ; on line 51 gets called since no changes to f2 were made. This comes from object f2.

’5”: prints out when std :: cout ¡¡ bar2 ¡¡ std :: endl ; on line 52 gets called since when getBarRef gets called, it initializes the passed in parameter to bar but then returns bar. Since f2.bar has not been changed and is still 5, when f2. getBarRef (temp_bar); on line 50 gets called, it’s output is f2.bar. This comes from object f2.

”calling constructor with bar”: prints out when Foo f3 = Foo::makeFoo(f2.bar); on line 54 gets called. This calls the static constructor on line 27 and gets initialized to f3, and it is created with a bar value, so it gets created with the constructor and not default constructor, and prints the message from line 9. This comes from object f3.

”5”: prints out when std :: cout ¡¡ f3.bar ¡¡ std :: endl ; on line 55 gets called. As f3 was already created with constructor with bar value of f2.bar and since f2.bar is 5, f3.bar is also 5. This comes from object f3.

(3)
Set variable bar to be a private member variable. This change happens on line 29.
private: int bar;

(4)
```
Foo operator=(Foo & rhs) {
        std::cout¡¡”overloading assignment operator”¡¡std::endl;
```

```
        if(this == &rhs){
            return *this; // return reference to this
        }
        bar = rhs.bar; // copy
        return *this; // return reference to this
    }
```

(5)
Code output is:

calling default constructor
calling copy constructor
calling constructor with bar
calling copy constructor
overloading assignment operator

”calling default constructor”: prints out when Foo f; on line 1 gets called. Object f is now created but it’s variables are not initialized to anything.

”calling copy constructor”: prints out when Foo f1 = f; on line 4 gets called. This is allowed since we overloaded the assignment operator. Object f1 now has f1.bar identical to f.bar.

”calling constructor with bar”: prints out when Foo f2 (50); on line 11 gets called. Here, a new f2 object is created with it’s bar set to 50.

”calling copy constructor”: prints out when Foo f3 = f2; on line 13 gets called. A new f3 object is created and set to copy all the properties from f2, then copy constructor for f2 gets called when f3 is created.

”overloading assignment operator” : these two print out when f2 = f; on line 16 gets called. First, the assignment operator overload method gets called since we set one instance of a Foo object equal to another. Now, f2.bar = f.bar and ”overloading assignment operator” prints out.

The if statement on line 8 will throw an error since fCond() is not a method defined in the Foo class.

Line of code we add to initialize fPtr:

Foo fPtr;

This produces an error in the output since now fPtr has two definitions, shared_ptr ¡Foo¿ and Foo.

Problem 8:

(1)
Driver Code:

```
Foo foo;
foo.func1();
foo.func2();
```

Output:

```
Foo constructor is being called
Foo func1 is being called
Foo func2 is being called
```

Foo destructor is being called

(2)

Driver Code:

```
FooKid kid;
kid.func1();
kid.func2();
```

Output:

```
FooKid constructor is being called
FooKid func1 is being called!
FooKid func2 is being called!
FooKid destructor is being called
Foo destructor is being called
Foo destructor is being called
```

(3)

We can overload the comparison ¿ operator outside of the two classes since at the end of the day, a sort only compares against one value.

```
bool operator¡(const Foo& a, const FooKid& b){

        return a.GetValue() ¡ b.GetValue();
}
```

Then we have a vector (vec) of Foo and FooKid objects, to sort the list we can call:

```
std::sort(vec.begin(), vec.end());
```

we can also create a method outside the two classes that compares the two values from Foo and FooKid classes:

```
bool compare(const Foo& a, const FooKid& b){

        return a.GetValue() ¡ b.GetValue();
}
```

Then, since we want a sortedList, we can make sure the list is of type vector and we call:

```
std::sort(vec.begin(), vec.end(), compare);
```

(4)

Foo constructor is being called
Foo constructor is being called
FooKid constructor is being called
Foo func1 is being called
Foo func2 is being called
FooKid func1 is being called!
FooKid func2 is being called!
Foo constructor is being called
FooKid constructor is being called
FooKid func1 is being called!
Foo func2 is being called
Foo constructor is being called
FooKid constructor is being called
FooKid func1 is being called!
FooKid func2 is being called!
Foo func1 is being called
Foo func2 is being called
Foo destructor is being called
FooKid destructor is being called
Foo destructor is being called
FooKid destructor is being called
Foo destructor is being called
FooKid destructor is being called
Foo destructor is being called
Foo destructor is being called

”Foo constructor is being called”: prints from std::shared_ptr ¡Foo¿ foo1(new Foo()); on line 2 with object foo1 from base constructor method.

”Foo constructor is being called”: prints from std :: shared_ptr ¡FooKid ¿ foo2 (new FooKid ()); on line 3 To create child class, base class needs to be created first, which is an object that has no name since it is implicitly called, so we call base class constructor.

”FooKid constructor is being called”: prints from std :: shared_ptr ¡FooKid ¿ foo2 (new FooKid ()); on line 3 since the base object for FooKid is made, now call child constructor for object foo2

”Foo func1 is being called”: prints from foo1 - ¿ func1 (); on line 5 since now point from object foo1 and call it’s func1 method.

”Foo func2 is being called”: prints from foo1 - ¿ func2 (); on line 6 since now point to object foo1 and call it’s func2 method.

”FooKid func1 is being called!”: prints from foo2 - ¿ func1 (); on line 9 since foo2 is a child class (object foo2) and we call it’s func1 method.

”FooKid func2 is being called!”: prints from foo2 - ¿ func2 (); on line 10 since foo2 is a child class (object foo2) and we call it’s func2 method.

”Foo constructor is being called” : prints from foo3 = std :: shared_ptr ¡Foo ¿(new FooKid ()); on line 12 since to create child class, base class needs to be created first, which is an object that has no name since it is implicitly called. Hence, we call base class constructor.

”FooKid constructor is being called”: prints from foo3 = std :: shared_ptr ¡Foo ¿(new FooKid ()); on line 12 since the base parent class is created, now we call child constructor for object foo3.

”FooKid func1 is being called!”: prints from foo3 - ¿ func1 (); on line 13 using child object foo3 method func1 can be overridden because it is virtual, calls and outputs func1 method from child class.

”Foo func2 is being called”: prints from foo3 - ¿ func2 (); on line 14 using foo3’s Foo object. Since parent class has func2 set to static, child cannot override it, and we see why output is from Foo object and not foo3 from FooKid.

”Foo constructor is being called”: prints from FooKid foo4; on line 16 since when calling constructor for child class, parent object must first be made, so we call Foo constructor.

”FooKid constructor is being called”: prints from FooKid foo4; on line 16 since parent Foo has been made, we can call child constructor and create object foo4.

”FooKid func1 is being called!”: prints from foo4 . func1 (); on line 19 since foo4 is a child class and we call it’s func1 method and because it is dynamic, it can be overridden.

”FooKid func2 is being called!”: prints from foo4 . func2 (); on line 20 since foo4 is a child class and we call it’s func2 method

”Foo func1 is being called”: prints from foo5 . func1 (); on line 22 since foo5 was created on line 17 and is of type Foo (parent) and we call it’s func1 method.

”Foo func2 is being called”: prints from foo5 . func2 (); on line 23 since now that foo5 is created, we call it’s func2 method.

Now, all destructors get implicitly called:

Foo destructor is called -¿ foo5 (base) FooKid destructor is called -¿ foo4 (child) Foo destructor is called -¿ foo4’s parent FooKid destructor is called -¿ foo3 (child) Foo destructor is called -¿ foo3’s parent FooKid destructor is called -¿ foo2 (child) Foo destructor is called -¿ foo2’s parent Foo destructor is called -¿ foo1 (base)