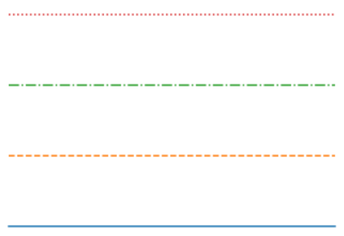
Prof. Dr. Natalie Packham
LV-Nr. 425093

Summer term 2019

Computational Finance and FinTech – Problem Set 3 with solutions

Exercise 1. Write a program that plots lines of different line styles. The output should look something like this:



Solution 1.

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

points = np.ones(5) # Draw 5 points for each line

linestyles = ['-', '--', '-.', ':']
for y in range(len(linestyles)):
    plt.plot(y * points, linestyle=linestyles[y])
    plt.axis('off')
    plt.show()
```

Exercise 2. Write a function that calculates the Black-Scholes price of a European call option on a stock with price process $(S_t)_{t\geq 0}$.
"European" in this context means that the option can be exercised only at maturity. A call option with strike K gives the owner the right, but not the obligation to purchase a share of stock for a price of K. Hence, the payoff of the call option at maturity T is

$$C_{K,T} = \max(S_T - K, 0),$$

where S_T is the stock price at maturity.
The Black-Scholes model assumes that a stock's log-returns are normally distributed. In this model the price of a call option at time $t = 0$ with current stock price S_0 is given by the *Black-Scholes formula*:

$$C(0, S_0) = S_0\,\mathrm{N}(d_+) - e^{-rT} K\,\mathrm{N}(d_-),$$

with

$$d_\pm = \frac{\ln(S_0/K) + (r \pm \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}},$$

where $\mathrm{N}(x)$ is the cumulative distribution function of the standard normal distribution.
The parameters are K: strike price, T: maturity, r: risk-free interest rate and σ: volatility.
(Hint: Use `from scipy.stats import norm` to access the normal distribution function.)

Solution 2.

```
import math from scipy.stats import norm

def bs(S,K,T,sigma,r):
    d1=(math.log(S/K) + (r + 0.5 * sigma**2)*T) / (sigma * math.sqrt(T))
    d2=(math.log(S/K) + (r - 0.5 * sigma**2)*T) / (sigma * math.sqrt(T))
    return S*norm.cdf(d1) - math.exp(-r *T) * K * norm.cdf(d2)
```

Exercise 3. Load the data from `tr_eikon_eod_data.csv` (see Chapter 5 on Financial Time Series). Create a data frame with log-returns and calculate the relevant summary statistics. Volatility corresponds to the annualised standard deviation of log-returns. To calculate volatilities, scale the standard deviation of daily log-returns by $\sqrt{252}$ and multiply by 100 to express it in percentages. Compare the volatilities of single stocks with the volatility of the S&P 500 index (`.SPX`). Can you find a reasonable explanation for the difference?

Solution 3.

```
import numpy as np
import pandas as pd
from pylab import mpl, plt
plt.style.use('seaborn')

filename = './data/tr_eikon_eod_data.csv'
data = pd.read_csv(filename, index_col=0, parse_dates=True)
rets = np.log(data / data.shift(1))

rets.aggregate([min, np.mean, np.std, np.median, max]).round(4)
rets.std()*100*np.sqrt(252) # volatilities
```

The S&P 500 volatility is lower than single-stock volatilities. This seems reasonable, as the S&P corresponds to a well-diversified portfolio, in which only systematic risk remains.

Exercise 4. Load the data from `tr_eikon_eod_data.csv` (see Chapter 5 on Financial Time Series). Create a data frame with log-returns and a data frame with discrete returns. Calculate the correlations of the log-returns with the discrete returns. Plot the respective returns in scatter plots. What can you conclude about daily returns?

Solution 4.

```
import numpy as np
import pandas as pd
from pylab import mpl, plt
plt.style.use('seaborn')

filename = './data/tr_eikon_eod_data.csv'
data = pd.read_csv(filename, index_col=0, parse_dates=True)
rets_c = np.log(data / data.shift(1))
rets_d = data.pct_change()
rets_c.corrwith(rets_d)

i=1;
plt.figure()
for s in rets_c.columns:
    plt.subplot(3,4,i)
    plt.scatter(rets_c[s], rets_d[s])
    i = i + 1
```

For daily returns, discrete and log-returns are approximately equal. Correlations are well above 99% and the scatter plots show that discrete returns are a nearly linear function of log-returns.

Exercise 5. Load the data from the file `hprice.xls` into a data frame. The file contains data on $N = 546$ houses sold in Windsor, Canada. The dependent variable Y is the sales price of the house in Canadian dollars. The explanatory variables included in this data set are:

- the lot size of the property (in square feet)
- the number of bedrooms
- the number of bathrooms
- the number of storeys (excluding the basement)
- A dummy variable = 1 if house has a driveway (= 0 otherwise)
- A dummy variable = 1 if house has a recreation room
- A dummy variable = 1 if house has a basement
- A dummy variable = 1 if house has gas central heating
- A dummy variable = 1 if house has air conditioning
- The size of garage (number of cars it will hold)
- A dummy variable = 1 if house is in a desirable neighbourhood

Conduct a simple regression of `saleprice` on `#bedroom` and conduct a multiple regression of `saleprice` on `lot size`, `# bedroom`, `# bath` and `# stories`.

(a) What are the coefficients associated with the number of bedrooms in each regression? Can you explain why they are different?

(b) What is the R^2 in the multiple regression? What is the interpretation?

(c) Are all coefficients statistically significant? Explain!

Solution 5.

```python
import numpy as np
import pandas as pd
from pylab import mpl, plt
import statsmodels.api as sm
plt.style.use('seaborn')

hprice = pd.read_excel('./data/hprice.xls')
Y = hprice['sale price']
X = hprice['#bedroom']
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.summary()

X=hprice[['lot size', '#bedroom', '#bath', '#stories']]
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.summary()
```

(a) The coefficient associated with the number of bedrooms in the simple regression is 13270 CAD, in the multiple regression the coefficient is 2825 CAD. The coefficient in a regression can be interpreted as the marginal effect of a change in the x-variable on the y-variable, *keeping all other coefficients fixed.* In the simple regression, the coefficient measures the marginal effect between houses with fewer bedrooms compared to houses with more bedrooms: On average, houses with one more bedroom are 13270 CAD more expensive. In the multiple regression we find that on average, houses with one more bedroom, but the same number of bathrooms, the same lot size and the same number of floors cost 2825 CAD more.

(b) The R^2 in the multiple regression is 53.2%, which means that 53.2% of the variance in the sale price data is explained by the regression model.

(c) All slope coefficients are statistically significant at the 5% level as their p-values are smaller than 0.05. The number of bedrooms is not statistically significant at the 1% level. p-values

Exercise 6 (optional)**.** Write an abstract base class `FinancialInstrument` with a `getPrice()` method. Then write concrete classes `Bond`, `Stock` and `EuropeanCallOption`.

- The bond should take a coupon, a maturity and an interest rate as arguments when instantiated.
- The stock should take a dividend, growth rate and discount rate. Implement `getPrice()` via the Gordon Growth model, which assumes that the current stock price is equal to the present value of all future cash flows.
- The call option price at time $t = 0$ and current stock price S_0 is given by the *Black-Scholes formula.*

Determine the prices of the following financial instruments:

- Bond with coupon 5, maturity of five years and a discount rate of 10%.
- Share or stock with dividend of 5, growth rate of 5.2% and a discount rate of 10%.
- European call option with initial stock price 100, strike price 100, maturity of 0.5 years, volatility of 20% and risk-free rate of 5%.

Solution 6.

```python
from abc import ABC, abstractmethod # package for abstract base classes

class FinancialInstrument(ABC):
    @abstractmethod
    def getPrice(self):
        pass

class Bond(FinancialInstrument):
    def __init__(self, coupon, n, r):
        self.coupon = coupon
        self.r = r
        self.n = n

    def getPrice(self):
        annuity_factor = 1/self.r * (1-1/(1+self.r)**self.n);
        return annuity_factor * self.coupon + 100/(1+self.r)**self.n

class Stock(FinancialInstrument):
    def __init__(self, dividend, g, r):
        self.dividend = dividend
        self.g = g
        self.r = r

    def getPrice(self):
        return self.dividend / (self.r-self.g) # perpetuity

class EuropeanCallOption(FinancialInstrument):
    def __init__(self, S, K, T, sigma, r):
        self.__S = S # attributes are private
        self.__K = K
        self.__T = T
        self.__sigma = sigma
        self.__r = r

    def getPrice(self):
        return bs(self.__S, self.__K, self.__T, self.__sigma, self.__r)
```

```
Bond(5,5,0.1).getPrice()

81.04606615295775
```

```
Stock(5,0.052,0.1).getPrice()

104.16666666666664
```

```
EuropeanCallOption(100,100,0.5,0.2,0.05).getPrice()

6.888728577680624
```