

Financial Markets - Exercises

Matteo Sani
Staff Quants - MPS Capital Services
matteo.sani@mpscapitalservices.it

October 4, 2020

Chapter 1

Introduction to python

Exercise 1.1

What is the built-in function that python uses to iterate over a number sequence ? Write an example that uses it.

Solution 1.1

The built-in function used to iterate over a sequence of numbers is `range`. It returns a sequence of numbers taking three parameters that represents respectively the lower boundary of the sequence, the upper boundary of the sequence and the step. If just one parameter is passed the default lower boundary is 0 and the step is 1. Note that the upper boundary is excluded from the sequence.

```
for i in range(10, 20, 2):  
    print (i)
```

```
10  
12  
14  
16  
18
```

```
for i in range(5):  
    print (i)
```

```
0  
1  
2  
3  
4
```

Exercise 1.2

What is a string in python ? Declare one string variable and try to manipulate it (concatenate, make uppercase, capitalize, replace characters, split...).

Solution 1.2

A string is simply a sequence of characters.

```
aString = "this is a string"

aString = aString + ", just an example"
print (aString)
this is a string, just an example

print (aString.upper())
'THIS IS A STRING, JUST AN EXAMPLE'

print (aString.capitalize())
'This is a string, just an example'

print (aString.replace("just an", "for"))
'this is a string, for example'

print (aString.split(","))
['this is a string', ' just an example']

if (aString.endswith("example")):
    print ("This string is really an example.")
else:
    print ("This string is not an example")
This string is really an example.
```

Exercise 1.3

What does the continue do in python ? Show an example of its usage.

Solution 1.3

continue is used to skip cycles in for loops. Note that % is the module operator, it returns the remainder of division.

```
for i in range(10):
    if i%2 == 0:
        continue
    else:
        print (i)
```

1
3
5
7
9

Exercise 1.4

When should you use the break in python ? Show an example of its usage.

Solution 1.4

break is the command used to interrupt a while loop even if the while condition is still satisfied.

```
i = 0
while i < 11:
    if (i/2 > 3):
        break
    print (i)
    i = i + 1
```

```
0
1
2
3
4
5
6
```

Exercise 1.5

Which python function will you use to convert a number to a string ? Show an example.

Solution 1.5

`str()` is the correct function to use in order to cast a number to a string.

```
x = 2.34
print ("This {} is of type {}".format(x, type(x)))
print ("This {} is of type {}".format(str(x), type(str(x))))
```

```
This 2.34 is of type <class 'float'>
This 2.34 is of type <class 'str'>
```

Exercise 1.6

Import the math module and compute the logarithm of 2.09, the exponential of 1.57 and the area of a circle of radius 6 cm (circle area = $\pi \cdot r^2$).

Solution 1.6

```
import math

print ("log(2.09) = {}".format(math.log(2.09)))
print ("exp(1.57) = {}".format(math.exp(1.57)))
print ("area of circle (R=6cm) is about {:.2f} cm2".format(math.pi*6**2))
```

```
log(2.09) = 0.7371640659767196
exp(1.57) = 4.806648193775178
area of circle (R=6cm) is about 113.10 cm2
```

Exercise 1.7

Given the following variables

```

S_t = 800.0 # spot price of the underlying
K = 600.0 # strike price
vol = 0.25 # volatility
r = 0.01 # interest rate
ttm = 0.5 # time to maturity, in years

```

write out the Black Scholes formula and save the value of a call in a variable named 'call_price' and the value of a put in a variable named 'put_price'.

Hint: remember that there are many modules available in python that let you save a lot of time. In this case we need the cumulative distribution function of the standard normal distribution which can be found in `scipy.stats` module, the name of the function is `norm`.

Solution 1.7

The BS equation for the price of a call is:

$$C(S, t) = S_t N(d_1) - Ke^{-r(T-t)} N(d_2)$$

where

- S_t is the spot price of the underlying
- K is the strike price
- r is the risk-free interest rate (expressed in terms of continuous compounding)
- $N(\cdot)$ is the cumulative distribution function of the standard normal distribution
- $T - t$ is the time to maturity
- σ is the volatility of the underlying

$$d_1 = \frac{\ln(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma^2)(T - t)}{\sigma\sqrt{T - t}}$$

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

```
from math import log, exp, sqrt
# You'll need the Gaussian cumulative distribution function
from scipy.stats import norm

S_t = 800.0
ttm = 0.5
K = 600.0
vol = 0.25
r = 0.01

d1_num = (log(S_t/K)+(r+0.5*pow(vol, 2))*ttm)
d1_den = vol*sqrt(ttm)
d1 = d1_num /d1_den
d2 = d1 - d1_den

call_price = S_t * norm.cdf(d1) - K * exp(-r*ttm)*norm.cdf(d2)
put_price = - S_t * norm.cdf(-d1) + K * exp(-r*ttm)*norm.cdf(-d2)

print("{:.3f} {:.3f}".format(call_price, put_price))

205.472 2.480
```


Chapter 2

Data Containers

Exercise 2.1

What is a dictionary in python programming ? Create a dictionary, modify it and then print all its items.

Solution 2.1

A dictionary is a container that maps a key (any object) to a value (any object), contrary to lists which map an integer (the index) to a value (any object).

```
dictionary = {"calculus":28, "physics":30, "chemistry":25}

dictionary["laboratory"] = 27
dictionary["chemistry"] = 24

print ("Exam\t\tVote")
for k, v in dictionary.items():
    print ("{}:\t{}".format(k, v))
```

Exam	Vote
calculus:	28
physics:	30
chemistry:	24
laboratory:	27

Exercise 2.2

Write code which, given the following list

```
input_list = [3, 5, 2, 1, 13, 5, 5, 1, 3, 4]
```

prints out the indices of every occurrence of

```
y = 5
```

Solution 2.2

```
l = [3, 5, 2, 1, 13, 5, 5, 1, 3, 4]

for i in range(len(l)):
    if l[i] == 5:
        print (i)

1
5
6
```

Exercise 2.3

Write a Python program to convert a list of tuples into a dictionary. Input:

```
l = [("x", 1), ("x", 2), ("x", 3), ("y", 1), ("y", 2), ("z", 1)]
```

Solution 2.3

```
l = [("x", 1), ("x", 2), ("x", 3), ("y", 1), ("y", 2), ("z", 1)]

d = {}
for item in l:
    d[item[0]] = item[1]

print (d)

{'x': 3, 'y': 2, 'z': 1}
```

Note that there is just one occurrence of the key x and y because keys has to be unique and setting the same key to a different value simply overwrite the existing entry.

Exercise 2.4

Write a Python program to replace last value of tuples in a list. Input:

```
l = [(10, 20, 40), (40, 50, 60), (70, 80, 90)]
```

Solution 2.4

```
l = [(10, 20, 40), (40, 50, 60), (70, 80, 90)]

for i in range(len(l)):
    new_tuple = l[i][0:2] + (l[i][2] + 10,)
    l[i] = new_tuple

print (l)

[(10, 20, 50), (40, 50, 70), (70, 80, 100)]
```

Exercise 2.5

Write a Python program to count the elements in a list until an element is a tuple. Input:

```
{[1, 5, 'a', (1,2), {'test':1}]}
```

Solution 2.5

```
l = [1, 5, "a", (1,2), {"test":1}]

number_of_items = 0
for item in l:
    if type(item) != tuple:
        number_of_items = number_of_items + 1
    else:
        break

print ("There are {} items before a tuple.".format(number_of_items))

There are 3 items before a tuple.
```

Exercise 2.6

Write a Python script to concatenate following dictionaries to create a new one. Input:

```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50, 6:60}
```

Solution 2.6

```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}

dic_tot = dict()
dic_tot.update(dic1)
dic_tot.update(dic2)
dic_tot.update(dic3)

print (dic_tot)

{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

Exercise 2.7

Write a Python script to check whether a given key already exists in a dictionary.

Solution 2.7

```
dic = {"a":1, "b":2, "c":3}

print ("z" in dic)
print ("a" in dic)

False
True
```

Exercise 2.8

Write a Python program to combine two dictionary adding values for common keys. Input:

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
```

Solution 2.8

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}

d = {}
d.update(d1)

for k in d2.keys():
    if k in d:
        d[k] = d[k] + d2[k]
    else:
        d[k] = d2[k]

print (d)

{'a': 400, 'b': 400, 'c': 300, 'd': 400}
```

Exercise 2.9

Given the following dictionary mapping currencies to 2-year zero coupon bond prices, build another dictionary mapping the same currencies to the corresponding annualized interest rates.

```
d = {
    'EUR': 0.98,
    'CHF': 1.005,
    'USD': 0.985,
    'GBP': 0.97
}
```

Solution 2.9

The price of a n-years zero coupon bond is:

$$P = \frac{M}{(1+r)^n} = M \cdot D$$

where * M is the value of the bond at the maturity * r is the risk-free rate * n is the number of years until maturity

Hence:

$$D = \frac{1}{(1+r)^n} \Rightarrow r = \left(\frac{1}{D}\right)^{\frac{1}{n}} - 1$$

```
from math import exp

# initialize an empty dictionary in which to store result
rates = {}

maturity = 2
discount_factors = {
    'EUR': 0.98,
    'CHF': 1.005,
    'USD': 0.985,
    'GBP': 0.97
}

# loop over the input dictionary to get the currencies
for currency, df in discount_factors.items():
    # calculate the rate and store it in the output dictionary
    rates[currency] = pow(1/df, 1/maturity) - 1

for r in rates.items():
    print (r)

('EUR', 0.010152544552210818)
('CHF', -0.002490663892367073)
('USD', 0.007585443719756668)
('GBP', 0.015346165133619083)
```


Chapter 3

Date and Time

Exercise 3.1

Write code that:

- print the day of the week of your birthday
- print the weekday of your birthdays for the next 120 years

Solution 3.1

```
import datetime

birthday = datetime.date(1974, 10, 20)
print (birthday.weekday()) # remeber it starts form 0
```

6

```
from dateutil.relativedelta import relativedelta

for i in range(120):
    print ((birthday + relativedelta(years=i)).weekday())
```

6

0

2

3

4

5

0

1

2

3

4

...

Exercise 3.2

Write code to determine whether a given year is a leap year and test it with 1800, 1987 and 2020.

Solution 3.2

```
years = [1800, 1987, 2020]

for y in years:
    if y % 400 == 0:
        print("{} is a leap year ".format(y))
    elif y % 100 == 0:
        print("{} is NOT a leap year ".format(y))
    elif y % 4 == 0:
        print("{} is a leap year ".format(y))
    else:
        print("{} is NOT a leap year ".format(y))

1800 is NOT a leap year
1987 is NOT a leap year
2020 is a leap year
```

Exercise 3.3

Write code to print next five days starting from today.

Solution 3.3

```
d = datetime.date.today()
for i in range(1, 6):
    print(d + relativedelta(days=i))

2020-08-04
2020-08-05
2020-08-06
2020-08-07
2020-08-08
```

Exercise 3.4

Build again dates as in Exercise 3.1 (i.e. the weekday of your birthdays for the next 120 years) and count how many of your birthdays is a Monday, Tuesday, ... , Sunday until 120 years of age. Print out the result using a dictionary. (expected output something like: {6: 10, 0: 10, 2: 9, 3: 10, 4: 10, 5: 10, 1: 9})

Solution 3.4


```

import datetime
from dateutil.relativedelta import relativedelta
birthday = datetime.date(1974, 10, 20)

d = {}
for i in range(120):
    wd = ((birthday + relativedelta(years=i)).weekday())
    if wd in d.keys():
        d[wd] = d[wd] + 1
    else:
        d[wd] = 1

print (d)

{6: 17, 0: 18, 2: 17, 3: 17, 4: 17, 5: 17, 1: 17}

```

Exercise 3.5

In the next lessons we will create many contracts (e.g. swaps) which take in input lists of dates like for example the payment dates. Since it would be very boring to write long list of dates for each of these contracts. The goal of this exercise is to write code which given a start date and a number of months, returns a list of dates of **annual** frequency from the start date to the ending of the period after the specified number of months.

For example

- 2019-11-10 start date 12 months → 2019-11-10, 2020-11-10
- 2019-11-10 start date 24 months → 2019-11-10, 2020-11-10, 2021-11-10

Note that if the number of months is not a multiple of 12, the last period should simply be shorter than 12 months. For example:

- 2019-11-10 start date 9 months → 2019-11-10, 2020-08-10
- 2019-11-10 start date 15 months → 2019-11-10, 2020-11-10, 2021-02-10

Solution 3.5

```

from finmarkets import generate_swap_dates
from datetime import date
from dateutil.relativedelta import relativedelta

def generate_swap_dates(start_date, n_months):
    dates = []
    for i in range(0, n_months, 12):
        dates.append(start_date + relativedelta(months=i))
    dates.append(start_date + relativedelta(months=n_months))

```

[illegible]

Chapter 4

Function and Classes

4.1 Exercises

Exercise 4.1

Take the code for the Black-Scholes formula from Exercise 1.7 and wrap it in a function. Then, use this function to calculate the prices of calls with various strikes, using the following data.

```
s = 800
# strikes expressed as % of spot price
moneyness = [ 0.5, 0.75, 0.825, 1.0, 1.125, 1.25, 1.5 ]
vol = 0.3
ttm = 0.75
r = 0.005
```

The output should be a dictionary mapping strikes to call prices.

Solution 4.1

```

from math import log, exp, sqrt
from scipy.stats import norm

def d1(S_t, K, r, vol, ttm):
    num = log(S_t/K) + (r + 0.5*pow(vol, 2)) * ttm
    den = vol * sqrt(ttm)
    return num/den

def d2(S_t, K, r, vol, ttm):
    return d1(S_t, K, r, vol, ttm) - vol * sqrt(ttm)

def call(S_t, K, r, vol, ttm):
    return S_t * norm.cdf(d1(S_t, K, r, vol, ttm)) \
        - K * exp(-r * ttm) * norm.cdf(d2(S_t, K, r, vol, ttm))

s = 800
# strikes expressed as % of spot price
moneyness = [ 0.5, 0.75, 0.825, \
              1.0, 1.125, 1.25, 1.5 ]
vol = 0.3
ttm = 0.75
r = 0.005

result = {}
for m in moneyness:
    result[s*m] = call(s, m*s, r, vol, ttm)
print(result)

{400.0: 401.66074527896365,
 600.0: 213.9883852521275,
 660.0: 166.85957363897393,
 800.0: 84.03697017660357,
 900.0: 47.61880394696229,
1000.0: 25.632722952585738,
1200.0: 6.655275227771156}

```

Exercise 4.2

Write two classes, Circle and Rectangle that given the radius and height, width respectively allow to compute area and perimeter of the two shapes. Test them with the following:

```

a_circle = Circle(5)
print ("My circle has an area of {} m**2".format(a_circle.area()))

a_rectangle = Rectangle(3, 6)
print ("My rectangle has a perimeter of {} m and an area of {} m**2" \
      .format(a_rectangle.perimeter(), a_rectangle.area()))

```

Solution 4.2

```

from math import pi
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return pi*self.radius**2

class Rectangle:
    def __init__(self, width, height):
        self.height = height
        self.width = width

    def area(self):
        return self.width*self.height

    def perimeter(self):
        return self.width*2 + self.height*2

circle = Circle(5)
print ("My circle area is {:.1f} m**2".format(circle.area()))

rect = Rectangle(3, 6)
print ("My rect area is {:.1f} m**2 and the "\
      "perimeter is {} m".format(rect.area(), rect.perimeter()))

My circle area is 78.5 m**2
My rect area is 18.0 m**2 and the perimeter is 18 m

```

Exercise 4.3

Define a class `Songs`, its `__init__` should take as input a dictionary (lyrics that contains lyrics line by line). Define a method, `sing_me_a_song` that prints each element of the lyrics in his own line. Also test it with the following input.

```

lyrics = {"Wonderwall":["Today is gonna be the day",
                      "That they're gonna throw it back to you",
                      "By now you should've somehow", "..."],
         "Wish you were here": ["So, so you think you can tell",
                                "Heaven from hell",
                                "Blue skies from pain", "..."]}

```

Solution 4.3

```

class Songs:
    def __init__(self, lyrics):
        self.lyrics = lyrics

    def sing_me_a_song(self, title):
        song = self.lyrics[title]
        print ("Title: {}".format(title))
        print ("*****")
        for line in song:
            print (line)

lyrics = {"Wonderwall":["Today is gonna be the day",
                        "That they're gonna throw it back to you",
                        "By now you should've somehow", "..."],
          "Vado al massimo": ["Voglio veder come va a finire",
                              "Andando al massimo senza frenare",
                              "Voglio vedere se davvero poi",
                              "Si va a finir male", "..."]}

songs = Songs(lyrics)
songs.sing_me_a_song("Wonderwall")

Title: Wonderwall
*****
Today is gonna be the day
That they're gonna throw it back to you
By now you should've somehow
...

```

Exercise 4.4

Define a `Point2D` class that represent a point in a plane. Its `__init__` method should accept the point coordinates `x` and `y`. Write a method `distanceTo` that compute the distance of the point to another passed as input. Test the class by printing the distance of the point $P = (4, 5)$ to the origin $P = (0, 0)$ and to $P = (3, 4)$.

Solution 4.4

```
from math import sqrt
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distanceTo(self, x, y):
        dist = sqrt((self.x-x)**2 + (self.y - y)**2)
        return dist

    def distanceTo_v2(self, p):
        dist = sqrt((self.x-p[0])**2 + (self.y - p[1])**2)
        return dist

    def distanceTo_v3(self, p):
        dist = sqrt((self.x-p.x)**2 + (self.y - p.y)**2)
        return dist

point = Point2D(4, 5)
p0 = (0, 0)
point0 = Point2D(0, 0)
print ("distance to p0: {:.2f}".format(point.distanceTo(p0[0], p0[1])))
print ("distance_v2 to p0: {:.2f}".format(point.distanceTo_v2(p0)))
print ("distance_v3 to p0: {:.2f}".format(point.distanceTo_v3(point0)))

p1 = (3, 4)
point1 = Point2D(3, 4)
print ("distance to p1: {:.2f}".format(point.distanceTo(p1[0], p1[1])))
print ("distance_v2 to p1: {:.2f}".format(point.distanceTo_v2(p1)))
print ("distance_v3 to p1: {:.2f}".format(point.distanceTo_v3(point1)))

distance to p0: 6.40
distance_v2 to p0: 6.40
distance_v3 to p0: 6.40
distance to p1: 1.41
distance_v2 to p1: 1.41
distance_v3 to p1: 1.41
```


Chapter 5

Data Manipulation and Its Representation

Exercise 5.1

Using pandas import data stored in `stock_market.xlsx` (you can find it in the same web page besides this notes). With the resulting dataframe determine:

1. remove duplicates and missing data (how many rows are left ?)
2. stocks with positive variation;
3. the first five stocks with the lowest price.

Solution 5.1

1. First load the excel file into a dataframe and look at data structure.

```
import pandas as pd

df = pd.read_excel("stock_market.xlsx")

print (len(df))
df.head()
```

51

	Symbol	Name	Price	Change	Change%	Volume (M)	\
0	GE	General Electric Company	6.07	-0.19	-0.0304	142.732	
1	NOK	Nokia Corporation	4.78	0.33	0.0742	117.960	
2	F	Ford Motor Company	6.61	-0.13	-0.0193	115.394	
3	PINS	Pinterest, Inc.	34.29	9.10	0.3613	111.864	
4	AAPL	Apple Inc.	425.04	40.28	0.1047	93.574	

	Avg Volume (M)	Market Cap (B)
0	102.268	53.132
1	31.296	27.083
2	87.719	26.288
3	15.550	20.110
4	35.035	1821.000

As usual if we are not sure that our data is *clean* we should check for duplicates and NaN and take care of them. The `df.duplicated()` method returns the status of each row (duplicate or not, True or False). If we would like just to see the duplicated entries we could combine the `df.duplicated()` method with the selection syntax like this:

```
df[df.duplicated() == True]
```

	Symbol	Name	Price	Change	Change%	Volume (M)	Avg Volume (M)	\
40	RUN	Sunrun Inc.	36.69	0.02	0.0005	20.113	3.604	

	Market Cap (B)
40	4.489

So it looks like we have just one duplicate and we can remove it:

```
print ("Before duplicates removal: {}".format(len(df)))
df = df.drop_duplicates()
print ("After duplicates removal: {}".format(len(df)))
```

Before duplicates removal: 50
After duplicates removal: 50

Then we need to take care of the NaN, again if we want to check the rows with NaN we can select (here the syntax is a little bit more complicated since we need to use `any` to look for NaN in every column):

```
df[df.isna().any(axis=1)]
```

	Symbol	Name	Price	Change	Change%	\
23	NCLH	Norwegian Cruise Line Holdings Ltd.	13.64	-0.53	-0.0374	
47	NBL	Noble Energy, Inc.	NaN	-0.23	-0.0225	

	Volume (M)	Avg Volume (M)	Market Cap (B)
23	28.402	64.895	NaN
47	18.462	13.535	4.795

Since we don't want to artificially modify our sample we just drop rows with NaN:

```
print ("Before NaN removal: {}".format(len(df)))
df = df.dropna()
print ("After NaN removal: {}".format(len(df)))
```

Before NaN removal: 51

After NaN removal: 49

- The second point asks to determine the companies with a daily positive variation. Clearly we have to apply to the dataframe a selection on the "Change" (or "Change%") column requiring positive values.

```
pos_var = df[df.loc[:, "Change"] > 0]
```

```
print (len(pos_var))
pos_var.head() # just printing the first 5 rows
```

16

	Symbol	Name	Price	Change	Change%	Volume (M)	\
1	NOK	Nokia Corporation	4.78	0.33	0.0742	117.960	
3	PINS	Pinterest, Inc.	34.29	9.10	0.3613	111.864	
4	AAPL	Apple Inc.	425.04	40.28	0.1047	93.574	
6	BAC	Bank of America Corporation	24.88	0.04	0.0016	62.039	
8	FB	Facebook, Inc.	253.67	19.17	0.0817	53.030	

	Avg Volume (M)	Market Cap (B)
1	31.296	27.083
3	15.550	20.110
4	35.035	1821.000
6	72.793	215.562
8	24.521	723.726

So in origin we had 48 stocks and just 16 have a positive variation of its price.

- The last question requires to print the first 5 stocks with the lowest prices. In this case it is enough to sort by price the dataframe (ascending) and then just select the first 5 entries.

```
highest_price = df.sort_values(by=['Price'], ascending=True)[:5]
```

highest_price

	Symbol	Name	Price	Change	Change%	Volume (M)	\
25	ABEV	Ambev S. A.	2.68	-0.16	-0.0563	26.136	
32	BBD	Banco Bradesco S. A.	4.22	-0.32	-0.0705	22.129	
1	NOK	Nokia Corporation	4.78	0.33	0.0742	117.960	
15	OPK	OPKO Health, Inc.	5.15	-0.76	-0.1286	35.762	
33	MRO	Marathon Oil Corporation	5.49	-0.02	-0.0036	21.249	

	Avg Volume (M)	Market Cap (B)
25	36.654	41.999
32	22.046	36.739
1	31.296	27.083
15	17.792	3.450
33	34.098	4.339

Exercise 5.2

Given the following discount factors plot the resulting discount curve, possibly adding axis labels and legend.

```
dfs = [1.0, 1.0014907894567657, 1.0031038833235129, 1.0047764800189012,
        1.0065986105304596, 1.014496095021891, 1.022687560553011,
        1.0303585751965112, 1.0369440287181253, 1.0422287558021188,
        1.0461834022163963, 1.0489228953047331, 1.0505725627906783,
        1.0513323539753632, 1.0513777790851995, 1.0508768750534248,
        1.049935905228433, 1.0486741093761602, 1.047175413484517,
        1.0455115431993336, 1.0437147446170034, 1.0418294960952215,
        1.0398823957504923, 1.0378979499878478, 1.0358789099539805,
        1.0338409767365169, 1.031791178324756, 1.0297378455884902,
        1.0276772747965244, 1.0256154380560942, 1.0235543974485939,
        1.0214974135391857, 1.0194401540150835, 1.0173862951028778]
```

```
pillars = [datetime.date(2020, 8, 3), datetime.date(2020, 11, 3),
            datetime.date(2021, 2, 3), datetime.date(2021, 5, 3),
            datetime.date(2021, 8, 3), datetime.date(2022, 8, 3),
            datetime.date(2023, 8, 3), datetime.date(2024, 8, 3),
            datetime.date(2025, 8, 3), datetime.date(2026, 8, 3),
            datetime.date(2027, 8, 3), datetime.date(2028, 8, 3),
            datetime.date(2029, 8, 3), datetime.date(2030, 8, 3),
            datetime.date(2031, 8, 3), datetime.date(2032, 8, 3),
            datetime.date(2033, 8, 3), datetime.date(2034, 8, 3),
            datetime.date(2035, 8, 3), datetime.date(2036, 8, 3),
            datetime.date(2037, 8, 3), datetime.date(2038, 8, 3),
            datetime.date(2039, 8, 3), datetime.date(2040, 8, 3),
            datetime.date(2041, 8, 3), datetime.date(2042, 8, 3),
```

```

datetime.date(2043, 8, 3), datetime.date(2044, 8, 3),
datetime.date(2045, 8, 3), datetime.date(2046, 8, 3),
datetime.date(2047, 8, 3), datetime.date(2048, 8, 3),
datetime.date(2049, 8, 3), datetime.date(2050, 8, 3)]

```

Solution 5.2

```

import datetime

dfs = [1.0, 1.0014907894567657, 1.0031038833235129, 1.0047764800189012, ...]

pillars = [datetime.date(2020, 8, 3), datetime.date(2020, 11, 3), ...]

from matplotlib import pyplot as plt
import matplotlib.dates as mdates

plt.plot(pillars, dfs, marker="o", label="EUR6M d.f.")

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
# this one instead rotate labels to avoid superimposition
plt.xticks(rotation=45)
plt.xlabel("Pillar dates")
plt.ylabel("Discount Factors")
plt.grid(True)
plt.legend()
plt.show()

```


Chapter 6

Discount Factors and Forward Rates

Exercise 6.1

Python has a useful command called `assert` which can be used for checking that a given condition is satisfied, and raising an error if the condition is not satisfied.

The following line does not cause an error, in fact it does nothing

```
assert 1 < 2
```

This causes an error

```
assert 1 > 2
```

`assert` can take a second argument with a message to display in case of failure.

```
assert 1 > 2, "Two is bigger than one"
```

Take the `df` function from lesson 3 and modify it by adding some assertions to check that:

- the pillar date list contains at least 2 elements;
- the pillar date list is the same length as the discount factor list;
- the first pillar date is equal to the today date;
- the value date argument 'd' is greater or equal to the first pillar date and also less than or equal to the last pillar date.

Then try using the function with some invalid data to make sure that your assertions are correctly checking the desired conditions

Solution 6.1

```

# import modules and objects that we need
from datetime import date
import numpy
import math

# define the input data
today_date = date(2017, 10, 1)
pillar_dates = [date(2017, 10, 1),
                 date(2018, 10, 1),
                 date(2019, 10, 1)]
discount_factors = [1.0, 0.95, 0.8]

# define the df function
def df(t):
    ##### CHECKS #####
    # Check that there are at least 2 pillar dates
    assert len(pillar_dates) >= 2, "need at least 2 pillar dates"

    # Check that the number of pillar dates
    # is equal to the number of pillar discount factors
    assert len(pillar_dates) == len(discount_factors), \
        "number of pillar dates should be equal to \
        the number of pillar discount factors"

    # Check that the first pillar date is the today date
    assert today_date == pillar_dates[0], \
        "first pillar date should be the today date"

    # Check that the value date argument is between
    # the first and last pillar dates
    assert pillar_dates[0] <= t <= pillar_dates[-1], \
        "Invalid value date %s" % (d)
    ##### END OF CHECKS #####

    log_discount_factors = []
    for discount_factor in discount_factors:
        log_discount_factors.append(math.log(discount_factor))

    pillar_days = []
    for pillar_date in pillar_dates:
        pillar_days.append((pillar_date - today_date).days)

    t_days = (t - today_date).days

    interpolated_log_discount_factor = \
        numpy.interp(t_days, pillar_days, log_discount_factors)

    return math.exp(interpolated_log_discount_factor)

df(date(2019, 1, 1))

0.9097285910181567

```


Exercise 6.2

Create a file `finmarket.py` and copy into it the function used to compute Black Scholes formula used in Exercise 4.1. This is the initial implementation of our financial library. Repeat Exercise 4.1 now using the version of the Black and Scholes formula in the `finmarket` module.

Solution 6.2

```
import finmarket

s = 800
# strikes expressed as % of spot price
moneyness = [ 0.5, 0.75, 0.825, \
              1.0, 1.125, 1.25, 1.5 ]
vol = 0.3
ttm = 0.75
r = 0.005

result = {}
for m in moneyness:
    result[s*m] = call(s, m*s, r, vol, ttm)
result

{400.0: 401.66074527896365,
 600.0: 213.9883852521275,
 660.0: 166.85957363897393,
 800.0: 84.03697017660357,
 900.0: 47.61880394696229,
1000.0: 25.632722952585738,
1200.0: 6.655275227771156}
```

Exercise 6.3

Add to `finmarket` module the `DiscountCurve` class developed during the lesson. Using the `libor` and `eonia` curves defined in the examples compute the forward 6m `libor` coupon.

Solution 6.3

```
npv = eonia_curve.df(date(2020, 4, 1)) * \
      libor_curve.forward_rate(date(2020,4, 1),
                              date(2020, 10, 1))

# Compute it in the pre-2008 way
npv_pre_2008 = libor_curve.df(date(2020, 4, 1)) * \
               libor_curve.forward_rate(date(2020, 4, 1),
                                       date(2020, 10, 1))

print (npv)
print (npv_pre_2008)

0.37932346377238657
0.38139410902305737
```

Chapter 7

Swap and Bootstrapping

Exercise 7.1 (Analytic bootstrapping)

Consider two 5% coupon paying bonds and a par value of €100 with the clean market prices (exclusive of accrued interest) of €99.50 and €98.30 and having time for the maturity of 6 months and 1 year respectively. Determine the spot rate for the 6-month and 1-year bond.

Solution 7.1

Consider two 5% coupon paying bonds and a par value of €100 with the clean market prices (exclusive of accrued interest) of €99.50 and €98.30 and having time for the maturity of 6 months and 1 year respectively. Determine the spot rate for the 6-month and 1-year bond.

At the end of 6 months the bond will pay a coupon of €2.5 (= €100 * 5% / 2) plus the principal amount (= €100) which sums up to 102.50. To determine the 6m spot rate we can write the following equation, :

$$\frac{102.5}{(1 + S_{6m}/2)} = 99.5 \rightarrow S_{6m} = 2 \cdot \left(\frac{102.5}{99.5} - 1 \right) = 6.03\%$$

At the end of another 6 months the second bond will pay a coupon of €2.5 (= €100 * 5% / 2) plus the principal amount (= €100) which sums up to €102.50. The bond is trading at €98.30, therefore, the 1-year spot rate S_{1y} can be calculated using S_{6m} as,

$$\frac{2.5}{(1 + S_{6m}/2)} + \frac{102.5}{(1 + S_{1y}/2)^2} = 98.30$$

$$\frac{102.5}{(1 + S_{1y}/2)^2} = 98.30 - \frac{2.5}{(1 + 0.03015)}$$

$$(2 + S_{1y})^2 = \frac{4 \cdot 102.5}{98.87317} = 4.276428$$

$$S_{1y}^2 + 4 \cdot S_{1y} - 0.276428 = 0$$

$$S_{1y} = -2 \pm \sqrt{4 + 0.276428} = \begin{cases} -4.06795 & \text{not valid} \\ 6.80\% \end{cases}$$

Exercise 7.2

Take the OvernightIndexSwap class, add it to finmarket module and try importing and using it. In particular read the OIS market data from the excel file used during the lesson and construct the corresponding swaps.

Solution 7.2

```
import pandas, datetime
from finmarket import OvernightIndexSwap, generate_swap_dates

observation_date = datetime.date.today()
df = pandas.read_excel('ois_data.xlsx')

market_quotes = {}
for i in range(len(df)):
    key = df.loc[i, 'months']
    value = df.loc[i, 'quote']
    market_quotes[key] = value

swaps = []
for months, rate in market_quotes.items():

    swap = OvernightIndexSwap(1e6,
                               generate_swap_dates(observation_date,
                                                    months),
                               0.01 * rate
                               )

    swaps.append(swap)
```

Exercise 7.3 (OvernightIndexSwap fair value)

Take the OvernightIndexSwap class from the lesson and add a new method called fair_value_strike which takes a discount curve object and returns the fixed rate which would make the OIS have zero NPV.

Hints: first take the formulas for the NPV of the fixed and floating legs, put one equal to the other and solve for K.

Solution 7.3

As the hint suggested the two NPV equations are compared:

$$\text{NPV}_{\text{fix}} = NK \sum_{i=1}^n D(d_i) \frac{d_i - d_{i-1}}{360}$$

$$\text{NPV}_{\text{float}} = N \cdot [D(d_0) - D(d_n)]$$

$$K \sum_{i=1}^n D(d_i) \frac{d_i - d_{i-1}}{360} = [D(d_0) - D(d_n)]$$

$$K = \frac{[D(d_0) - D(d_n)]}{\sum_{i=1}^n D(d_i) \frac{d_i - d_{i-1}}{360}}$$

Now in python:

```
class OverNightIndexSwap:
    ...
    def fair_value_strike(self, discount_curve):
        den = 0
        for i in range(1, len(self.payment_dates)):
            start_date = self.payment_dates[i-1]
            end_date = self.payment_dates[i]
            tau = (end_date - start_date).days / 360
            df = discount_curve.df(end_date)
            den += df * tau
        num = (discount_curve.df(self.payment_dates[0]) -
              discount_curve.df(self.payment_dates[-1]))

        return num/den
```

Finally add this method to the class implementation in finmarket.

Chapter 8

Interest Rate Swap and Swaptions

8.1 Exercises

Exercise 8.1 (Forward rate curve class)

Write a `ForwardRateCurve` class (for EURIBOR/LIBOR rate curve) which doesn't compute discount factors but only interpolates forward rates; then add it to the `finmarkets` module (this function is used to define the Libor curve needed throughout this lesson).

Solution 8.1

In this case it is enough to write a new class that has three attributes: a today date, a set of `pillar_dates` and the corresponding rates. There will be just a single method `forward_rate` which returns the corresponding interpolated rate.

```

import numpy

# an EURIBOR or LIBOR rate curve
# doesn't calculate discount factors, only interpolates forward rates
class ForwardRateCurve(object):

    # the special __init__ method defines how to
    # construct instances of the class
    def __init__(self, pillar_dates, rates):

        # we just store the arguments as attributes of the instance
        self.today = pillar_dates[0]
        self.rates = rates

        self.pillar_days = [
            (pillar_date - self.today).days
            for pillar_date in pillar_dates
        ]

    # interpolates the forward rates stored in the instance
    def forward_rate(self, d):
        d_days = (d - self.today).days
        return numpy.interp(d_days, self.pillar_days, self.rates)

```

Exercise 8.2 (Monte Carlo Simulation I)

Using the function `randint` of the module `random` make a Monte Carlo simulation of rolling three dices to check the probability of getting the same values on the three of them.

From the probability theory you should expect:

$$P_{d1=d2=d3} = \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} \cdot 6 = \frac{1}{36} = 0.0278$$

Solution 8.2


```

from random import seed, randint

seed(1)

trials = 10000000
success = 0
for _ in range(trials):
    d1, d2, d3 = randint(1, 6), randint(1, 6), randint(1, 6)
    if d1 == d2 and d2 == d3:
        success += 1

print("The probability to get three equal dice is {:.4f}".format(success/trials))

```

The probability to get three equal dice is 0.0278

Exercise 8.3 (Analytic bootstrapping)

Using the function `normal` of `numpy.random` simulate the price of a stock which evolves according to a log-normal stochastic process with a daily rate of return $\mu = 0.1$ and a volatility $\sigma = 0.15$ for 30 days.

Also plot the price. Try to play with μ and σ to see how the plot changes.

Solution 8.3

```

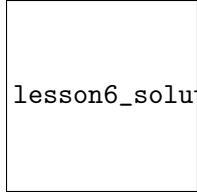
from numpy.random import normal, seed
from matplotlib import pyplot as plt
import math

S = 100
mu = 0.1
sigma = 0.15
T = 1

seed(1)
historical_series = [S]
for i in range(30):
    S = S * math.exp((mu - 0.5 * sigma * sigma) * T + sigma * math.sqrt(T) * normal())
    historical_series.append(S)

plt.plot(range(31), historical_series)
plt.xlabel("days")
plt.ylabel("Price of stock X")
plt.show()

```


 lesson6_solutions_files/lesson6_solutions_5_0.png

Exercise 8.4 (Analytic bootstrapping)

Suppose that the Libor Forward rates are those defined here in ([curve_data.py](#)). Determine the value of an option to pay a fixed rate of 4% and receives LIBOR on a 5 year swap starting in 1 year. Assume the notional is 100 EUR, the exercise date is on October, 30th 2020 and the swap rate volatility is 15%.

Solution 8.4

```
from finmarkets import InterestRateSwap
from datetime import date
from dateutil.relativedelta import relativedelta
from curve_data import discount_curve, libor_curve
from scipy.stats import norm
import math

pricing_date = date.today()
start_date = date.today() + relativedelta(years=1)
exercise_date = date(2020, 10, 30)

irs = InterestRateSwap(start_date, 100, 0.04, 12, 5)
sigma = 0.15

A = irs.annuity(discount_curve)
S = irs.swap_rate(discount_curve, libor_curve)
T = (exercise_date - pricing_date).days / 365
d1 = (math.log(S/irs.fixed_rate) + 0.5 * sigma**2 * T) / (sigma * T**0.5)
d2 = (math.log(S/irs.fixed_rate) - 0.5 * sigma**2 * T) / (sigma * T**0.5)
npv = irs.notional * A * (S * norm.cdf(d1) - irs.fixed_rate * norm.cdf(d2))

print("Swaption NPV: {:.3f} EUR".format(npv))

Swaption NPV: 13.587 EUR
```