Statistical Programming and Data Bases (SPDB)
Facultat de Matemàtiques i Estadística
Final Exam, June 7, 2022
`Corrected Version`

Answer the questions concisely and precisely
Duration: 2:15 hour

The Final exam should consists on a `.zip` *of a folder.*

- *The name of this folder should contain your name (to avoid errors).*
- *The folder should contain two files.*
  - *A Jupyter file with the solutions of Exercise I and II*
  - *A Zeppelin file with the solution of Exercises III, IV and V.*

Remind: answer the Exercises I, II in a *unique Jupyter file also containing* your name *at the beginning of the solutions.*

Exercise I (1.5 points, Python) Consider the following code:

```python
def gm(x):
    return -(np.exp(-(x[0]+1.5)**2/(3.1**2))+
                np.exp(-(x[1]-0.5)**2/(0.3**2)))
from  scipy import optimize
x0=np.array([1]*2)
result = optimize.minimize(gm,x0=x0,method='SLSQP')
```

Listing 1: Exercise I

1. (0.5 Points) Explain the code line by line, and its purpose.
2. (1 Point) Count the number of `gm()` function evaluations with help of a decorator, without modifying the function.

——————————— SOLUTIONS ———————————

1. (0.5 Points) Explain the code line by line, and its purpose.
   - Function `gm` defines a two dimensional function, evaluating to two exponentials on each axis centered at `(-1.5, 0.5)`.
   - The module `optimize` is imported from `scipy`.
   - Point `x0` represents an initial guess placed at `(1,1)`.
   - The method `minizime` from `scipy.optimize` is called with an initial guess at `(1,1)` by using the Sequential Least SQuares Programming optimizer.

2. (1 Point) Count the number of `gm()` function evaluations with help of a decorator, without modifying the function.

```
In [75]: c = 0
         def dgm(inf):
             def outf(x):
                 global c
                 c += 1
                 return(inf(x))
            return(outf)

         @dgm
         def gm(x):
             return -(np.exp(-(x[0]+1.5)**2/(3.1**2))+
                       np.exp(-(x[1]-0.5)**2/(0.3**2)))
         from  scipy import optimize
         x0=np.array([1]*2)
         result = optimize.minimize(gm,x0=x0,method='SLSQP')
         result


Out[75]:        fun: -1.999999965379848
                jac: array([ 7.51316547e-05, -9.67323780e-04])
            message: 'Optimization terminated successfully'
               nfev: 29
                nit: 9
               njev: 9
             status: 0
            success: True
                  x: array([-1.49963902,  0.49995646])

In [76]: print(c)
          29
```

Exercise II (3.5 points, Python) Given the *burning ship* fractal series:

$$z_{n+1} = (|\text{Re}\,(z_n)\,| + i|\text{Im}\,(z_n)\,|)^2 + c$$

and given an initial element of the series $z_0 = 0 + i0$

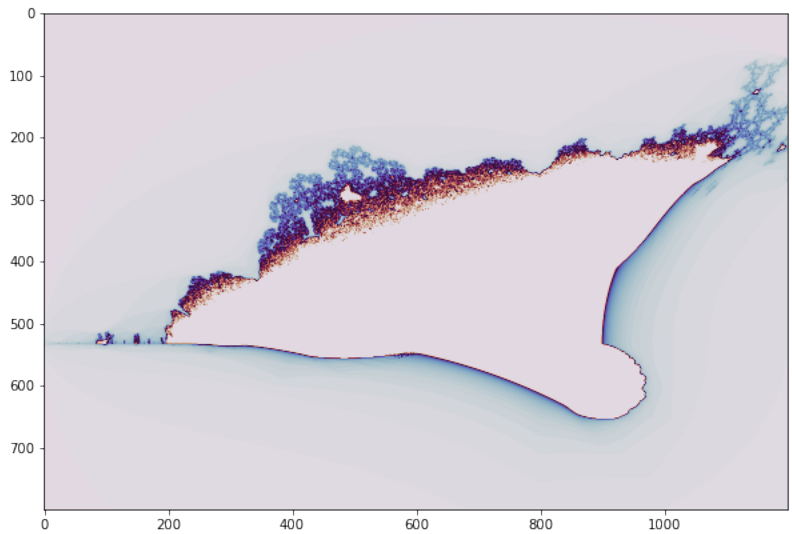
1. (2 Points) Write a function $f(c, A, N_{max})$, which returns the number of iterations N of the previous series necessary for the module $|z| > A$, where $A \in \mathbb{R}$, being N_{max} a maximum number of iterations to test.

2. (1.5 Points) Calculate and plot a 2D image representing N over the domain of c comprised by $Re(c) \subset [-2, 1]$, $Im(c) \subset [-2, 1]$ with the parameters ($A = 2$, $n_{max} = 100$, $z_0 = 0i$, Image resolution = (800,1200) pixels).

Note: `numpy`'s `meshgrid` function may be useful.

SOLUTIONS

```
In [2]: def fa(z,c): return((abs(z.real) + 1j*(abs(z.imag)))**2 + c)

        def fc(c=1,f=fa,z=0j,A=2,n_max=100):
            n=0
           while (abs(z)<A):
               z = f(z,c)
               n +=1
               if (n > n_max):
                   break
           return(n)

In [3]: import numpy as np
        dimx = 1200
        dimy = 800
        n_max=100

        rec = np.linspace(-2,1,dimx)
        imc = np.linspace(-2,1,dimy)

        # Usem meshgrid de numpy
        recs, imcs = np.meshgrid(rec,imc)

        #  Empaquetem els valors en tuples
        cs = zip( recs.ravel(),imcs.ravel())

In [5]: Ns = map(lambda x: fc(c=complex(x[0],x[1])),cs)
        N = np.array(list(Ns))
        N = N.reshape((dimy,dimx))

In [8]: import pylab as pl
        pl.figure(figsize=(10,7))
        pl.imshow(N,  cmap= plt.get_cmap("twilight"));
```



Remind: answer the Exercises III, IV and V in a unique Zeppelin file containing your name at the beginning and the solutions.

Exercise III (1.5 Points) Prefix Sum.
The prefixSum of a list of integers `List(a0, a1, a2, a3)` it the list

`L(a0, a0+a1, a0+a1+a2, a0+a1+a2+a3)`

Remind that in Scala you have the `leftScan`. For instance

`List(1,3,8).scanLeft(100)(((s, x)=>s+x))`

returns `List(100, 101, 104, 112)`. That is

`List(100, 100+1, 100+1+3, 110+1+3+8)`

Formally, the `leftScan` is defined as

`List(a1, a2, a3).scanLeft(a0)(f) = List(b0, b1, b2, b3)`

such that `b0 = a0, b1 = f(b0, a1), b2 = f(b1, a2), b3 = f(b2, a3)`.

(1.5 Points) Using `leftScan` design a function

`prefixSum(L : List[Int]) : List[Int]`

such that `prefixSum(List(2, 5, 8))` returns `List[Int] = List(2, 7, 15)`.

——————————— SOLUTIONS ———————————

Solution 1. Split `L` into `Head = L.head` and `Tail = L.tail` and aplly the `scanLeft`.

```
def prefixSum_2(L: List[Int]):List[Int] = {
     var Head = L.head
     var Tail = L.tail
     Tail.scanLeft(Head)((s, x)=>s+x)
}
```

Solution 2. Similar to *Solution 1* but encoded in a more compact way.

```
%spark
def prefixSum(L: List[Int]):List[Int] = L.tail.scanLeft(L.head)((s, x)=>s+x)
```

Solution 3. As we are dealing with prefix sum, we can work with the list `L.scanLeft(0)` = `List(0, 2, 5, 8)`, apply prefix sum

`List(0, 0+2, 0+2+5, 0+2+5+8) = List(0, 2, 7, 15)`

and take the `tail`.

```
%spark
def prefixSum_3(L: List[Int]):List[Int] = {
     L.scanLeft(0)((s, x)=>s+x).tail
}
```

Exercise IV: (2 Points) Looking at the customers behaviour.
In order to solve this exercise, remid that you have two possibilities when working with relations.

- Once we have created a `DataFrame`, you can create a `temporary view` so that SQL instruction can be executed "directly".
- You can work directly with Spark `DataFrame` with no temporary views. in such a case, you have to adapt the SQL syntax.

Following, let us work with the relations `customer` and `orderinfo` of the `bpsimple` DB.

(1 Point) Load `customer` and find the customers in a given city for instance `'Binham'`.
remind the `select ... where ...`

The result is:

```
+-------+------+
| fname | lname|
+-------+------+
|Richard|Stones|
|  Ann  |Stones|
|  Dave |Jones |
+-------+------+
```

(1 Point) Load `orderinfo` and for all customers, find the `date_of_shipped` of each orderinfo.

- In the case you use temporary views, remind that the SQL92 syntax uses variations of `JOIN` to specify how tables relate. For instance

  ```
  SELECT column_list FROM table INNER JOIN other_table ON join_condition
  ```

  To avoid ambiguities we can give a name to the different views. For instance, you can give name `c` to the temporary view of the customer relation so called `customer2`.

  ```
  select c.customer_id, ...  from customer2 c inner join ....  on c.customer_id = ...
  ```

- If you use Spark directly (no temporary views), remind that the syntax for an inner join is a little bit different.

  ```
  customer.join(orderinfo, customer("customer_id")=== ...,"inner")
  ```

The result is:

```
+-----+-------+------------+--------------------+
|fname| lname |orderinfo_id|    date_shipped    |
+-----+-------+------------+--------------------+
| Alex|Matthew|           1|2004-03-17 00:00:...|
| Ann | Stones|           5|2004-07-24 00:00:...|
| Ann | Stones|           2|2004-06-24 00:00:...|
|Laura|  Hardy|           4|2004-09-10 00:00:...|
|David| Hudson|           3|2004-09-12 00:00:...|
+-----+-------+------------+--------------------+
```

———————————— SOLUTIONS ————————————

Solution of the first question. We consider two possibilities, using temporary views or not.

Solution with a Temporary View.

```
%spark
val customer = spark.read.format("csv")
        .option("header", "true")
        .option("inferSchema", "true")
        .csv("/notebook/data/bpsimple/customer.csv")

customer.createOrReplaceTempView("customer2")

spark.sql("select fname, lname from customer2 where town = 'Bingham'").show()
```

Solution with no Temporary View.

```
%spark
import spark.implicits._

val customer = spark.read.format("csv")
        .option("header", "true")
        .option("inferSchema", "true")
        .csv("/notebook/data/bpsimple/customer.csv")

var living_in_city = customer
                .where(customer("town") === "Bingham")
                .select(customer("fname"), customer("lname"))

living_in_city.show()
```

Solution of the second question. We also consider two possibilities, using temporary views or not.

Solution with a Temporary Views. We assume from before the temporary view `customer2`.

```
%spark
val orderinfo = spark.read.format("csv")
        .option("header", "true")
        .option("inferSchema", "true")
        .csv("/notebook/data/bpsimple/orderinfo.csv")

orderinfo.createOrReplaceTempView("orderinfo2")

spark.sql("select c.fname, c.lname, e.orderinfo_id, e.date_shipped
           from customer2 c  inner join orderinfo2 e
           on c.customer_id = e.customer_id").show()
```

Solution with no Temporary View.

```
%spark
val orderinfo = spark.read.format("csv")
        .option("header", "true")
        .option("inferSchema", "true")
        .csv("/notebook/data/bpsimple/orderinfo.csv")

var customer_inner_orderinfo = customer.
    join(orderinfo,
        customer("customer_id")=== orderinfo("customer_id"),
        "inner")

customer_inner_orderinfo.select("fname", "lname", "orderinfo_id", "date_shipped").show()
```

Exercise V (1.5 Point) Relating Topics

Explain (shortly) the link between programming, data bases and learning algorithms.

——————————— SOLUTIONS ———————————

Usually there is a long way between the raw information and the strongly structured information neded to apply learning algorithms. In a little bit artificial distinction we structure thewhole process in three main steps.

- In many cases, raw information need to be processed (with the help of progamming languages like `Phyton` or `Scala`) in order to isolate and structurate useful information. Examples of such a process was counting the words in `mobydick.txt` or finding the longest words in `ManifestComunistParty.txt`.
- A classic way to stucture information is through relational data bases (DB for short). A BD contains many relations, or tables, dealing with the different aspects of the information. We describe the `bpsimple` DB. This DB give us a way to organize clients, customers, orderlines and items in a small business. To deal with the queries to a DB, the SQL language is fundamental and quite clear. Moreover, `Scala` give us the opportunity to deal with `SQL` queries in a friendly way.
- Once, information is structured into a DB, quering this DB we can isolate the inportant aspects and to start with the selection of features needed for machine learning.
