Department of Mathematics and Computer Science
University of Southern Denmark, Odense

February 15, 2018
Marco Chiarandini

DM559 – Linear and integer programming

Sheet 0, Spring 2018 [pdf format]

For this session on Wednesday 8th February you must have read the Python tutorial. In class we will discuss the following exercises.

Exercise 1* Python: One liner quizzes

The goals of this exercise are:

- to refresh programming skills in Python
- to give examples of interactive and one line programming
- to examplify as Python can be used in a functional programming way
- to highlight similarities and differences between the Python programming language and the mathematical language and defintions.

The basic data structures in Pyton are lists, tuples, sets and dictionaries.

- Construct in Python the set $S = \{x \in \mathbb{R} \mid x \geq 0 \wedge x \bmod 3 \equiv 1\}$

  Solution:

  ```
  S = {x for x in range(50) if x % 3 == 1}
  ```

- Using comprehension lists make a list for $\{(i, j) \mid i \in \{1, 2, 3, 4\}, j \in \{5, 7, 9\}\}$

  Solution:

  ```
  [(i+1,j) for i in range(4) for j in [5,7,9]]
  ```

- Calculate the inverse of a function or the index function for an invertible function (ie, bijective = injective + surjective) given in form of a dictionary.

  Solution:

  ```
  {d[k]:k for k in d}
  {v:k for k in d.keys() for v in d.values}
  {v:k for k,v in d.items()}
  ```

- What is the result of the following lines?

  ```
  map(lambda x: x%3, range(5))
  filter(lambda x: x%2==0, range(5))
  ```

  (In Python 3.x, you have to enclose those lines in the list constructor `list()`.)

In this course we will use numpy and scipy to deal with arrays. However, without these two modules, vectors and matrices can be implemented in Python as lists. How?

- Generate a couple of numerical examples for vectors and matrices. Experiment with the operators + and *. Do they yield the same result as expected from linear algebra?
- Write a one line function for the sum of two vectors using list comprehension.
- Write a one line function for the multiplication of a vector by a scalar.
- Write a one line function for the sum of two matrices using list comprehension.
- Write a one line function for the multiplication of a matrix by a scalar.
- Write a function for the multiplication of two matrices not necessarily square. Raise a ValueError exception if the size of the matrices is not compliant.

Solution:

It is important to note that the operators + and * are overloaded for list. They concatenate or replicate the two lists, respectively. This is definetly not what we learned to be the definition of those operations.

```
def sumVec(a,b): return [a[i]+b[i] for i in range(len(a))]
def multScalVec(alpha,a): return [alpha * a[i] for i in range(len(a))]


def sumMat(M,N)
  result = [[0 for x in range(len(N[0]))] for y in range(len(M))]
  for i in range(len(M)):
      for j in range(len(N[0])):
          result[i][j]=M[i][j]+N[i][j]
  print(result)


def mult(M,N):
    if (len(M[0])!=len(N))
        raise ValueError("The inner size of the martices does not match")

    result = [[0 for x in range(len(N[0]))] for y in range(len(M))]

    for i in range(len(M)):
        for j in range(len(N[0])):
            for k in range(len(N)):
                result[i][j] = result[i][j] + M[i][k] * N[k][j]

    return result

M=[[1,2,3],[1,2,3]]
N=[[1,2],[1,2],[1,2]]

try mult(M,N) except ValueError: print("Oops, a ValueError occurred")

def printMatrix(M):
    for row in M:
        print(["%3.0f" % a for a in row])
    print("\n")
```

The modules `numpy` and `scipy` make available another data structure in Python, the 'array' type. The next exercise guides you to the discovery of how operators are overloaded for the 'array' type module.

Exercise 2*

Generate in python two matrices A and B of size 3×2 and 2×4, respectively, made of integers numbers randomly drawn from the interval $[1, \ldots, 10]$. (Hint: consider using the function `numpy.random.randint` .) Calculate the following results, first by hand and then checking the correctness of your answer in Python:

- $A+B, A-B$
- $A \cdot B$
- A/B

[In IPython and Jupyter it is possible from command line to ask for completion via tab. This can be used to explore which functions are available for a given module. Try for example to type

```
import numpy as np
np.
```

followed by a tab. You should see a list of available functions. Among them there are two submodules that will be useful for us: `random` and `linalg`. The first implements a function to generate random numbers and matrices. The second implements functions from linear algebra. It is possible to get a manual for each function by following the function with a question mark. For example: `np.random.randint?`.]

Solution:

```
A=np.random.randint(1,10,(3,2))
B=np.random.randint(1,10,(2,4))
A+B
A-B
A*B
```

All these produce an error because the `+,-` operators, as in linear algebra perform an element-wise addition and subtraction, which requires the two matrices to have exactly the same size.
The `*` operator performs also an element-wise multiplication, which require the two matrices to have exactly the same size. However, contrary to addition and multiplication, this operation is not defined element-wise in linear algebra.
The correct way to obtain the matrix product is:

```
np.dot(A,B)
```

The division by a matrix is not defined in linear algebra. In Python it does an element-wise operation if the matrices have the same size.

Exercise 3*

Generate in Python two vectors $\mathbf{v}$ and $\mathbf{u}$ of size 4. Are they represented as column or row vectors? Calculate $\langle \mathbf{u}, \mathbf{v} \rangle$.

Solution:

As for matrices, the `*` makes the element-wise calculation. The inner product is calculated by the function `np.dot`. Vectors are represented as row vectors in Python. However, the `dot` function takes care to put the vectors in the right form without need to transpose.

```
a=np.array([1,2,3])
b=np.array([1,2,3])
np.dot(a,b)
```

Exercise 4*

Find the inverse of the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

by hand and compare your result with the result found in python.

Solution:

Expenses	A	B	C
Raw materials	0.10	0.30	0.15
Labor	0.30	0.40	0.25
Overhead and miscellaneous	0.10	0.20	0.15
	0.50	0.90	0.55

Table 1: Production Costs per Item (dollars)

```
np.linalg.inv(A)
```

Exercise 5

Numpy and scipy both implement their procedures in a compiled language such as fortran and C, rather than the interpreted language python. Hence, whenever possible the functions of those libraries must be used to gain efficiency. Let A be a large enough matrix of random integer numbers. Undertake the following experimental analysis:

- Compare the running time to calculate the column-wise sum of the matrix A using the base `sum` function, the `numpy.sum` function and with a for loop.
- Compare the runnning time to multiply A by itself using the numpy and scipy libraries and the function you wrote in Exercise 1.
- Compare the running time of the numpy and scipy libraries to calculate the inverse of a matrix.
- Compare the use of `vectorzie` against the use of a for loop for a non element-wise function of your choice.

The easiest way to measure the execution time of a single statement is to use IPython's magic function %timeit. As the execution time of a single statement can be extremely short, the statement is placed in a loop and executed several times (option `-n`). Moreover since other tasks running in the computer may influence the result, the loop is repeated a number of times and the best result is taken (option `-r`). The best results is the minimum time.
Alternatively, one can use the module `timeit` from Python. Here the parameter `number` and `repeat` refer to the number of times the command to track is executed and to the number of times the experiment is repeated, respectively.

Exercise 6

Let u be a one dimensional array. Construct another array y with values $y_i = (u_i + u_{i+1} + u_{i+2})/3$. In statistics this array is called the moving average of u. Try to avoid the use of for loops in your script.

Solution:

```
[sum(u[i:i+1]) for i in range(len(u)-2)]
```

Exercise 7* Production Costs

The following is Application 1 of Section 1.3 from [Le].
A company manufacture three products. Its production expenses are divided into three categories. In each category, an estimate is given for the cost of producing a single item of each product. An estimate is also made of the amount of each product to be produced per quarter. These estimates are given in Tables 1 and 2. At its stockholders' meeting, the company would like to present a single table showing the total cost for each quarter in each of the three categories: raw materials, labor and overhead.

Moreover the company would like to decide a unitary price for each of the three products with the possibility to account for seasonal variation, that is, the price can vary between the quarters. Choose the prices in such a way that it is convenient for the company to produce the decided amount.

Product	Summer	Fall	Winter	Spring
A	4000	4500	4500	4000
B	2000	2600	2400	2200
C	5800	6200	6000	6000

Table 2: Amount produced per Quarter

- Put the production quantity matrix described in the text in form of a vector and consequently the cost matrix in a form that is convenient for the multiplication with vector. Calculate in python the total cost of "Row materials", "Labor" and "Overhead and misc." for the decided production.
- Next, calculate the total revenue obtained by selling all the amount produced of the three products using the prices you decided.
- Finally, calculate the profit, that is, the surplus after removing the costs from the revenue.

Solution:

The price of a unit of each single prodcut must be at least equal to the cost for producing it, otherwise we would not gain anything by selling. Hence the unitary price of product A is in dollars at least:

Expenses	A	B	C
	0.10	0.30	0.15
	0.30	0.40	0.25
	0.10	0.20	0.15
	0.50	0.90	0.55

So let's decide for $0.60, 1.00, 0.65$ as the prices of the three products. If we want to calcuate the total revenue:

$$RP = \begin{bmatrix}0.6 & 1. & 0.65\end{bmatrix} \begin{bmatrix}4000 & 4500 & 4500 & 4000\\ 2000 & 2600 & 2400 & 2200\\ 5800 & 6200 & 6000 & 6000\end{bmatrix} = [8170, 9330, 9000, 8500]$$

And the total sum is 35000. In Python:

```
R = np.array([0.6,1.0,0.65])
P = np.array([[4000,2000,5800],[4500,2600,6200],[4500,2400,6000],[4000,2200,6000]]).
    transpose()
S = np.dot(R,P)
sum(S) # 35000.
```

The exercise asks to consider the possibility to set different prices per quarter. This cannot be done maintaining the matrix form proposed. We need to pass to a vector notation. Another advantage of the vector approach is that we do not need to use a non-mathematical operation such as the sum of all elements of a matrix as done above to calculate 35000.0 from the matrix RP but we can use the dot product.
In vector form we reshape the matrices R and P into:

$$\mathbf{r} = \begin{bmatrix} 0.6 & 1. & 0.65 & 0.6 & 1. & 0.65 & 0.6 & 1. & 0.65 & 0.6 & 1. & 0.65 \end{bmatrix}^T$$

$$\mathbf{p} = \begin{bmatrix} 4000 & 2000 & 5800 & 4500 & 2600 & 6200 & 4500 & 2400 & 6000 & 4000 & 2200 & 6000 \end{bmatrix}^T$$

Thus we can calculate the total revenue by dot product between vectors:

$$\langle \mathbf{r}, \mathbf{p} \rangle = \mathbf{r} \cdot \mathbf{p}^T = 35000.0$$

In Python:

```
np.dot(np.hstack([R,R,R,R]), P.T.reshape(-1,1))=array([ 35000.])
```

To change the price throughout the quarters it is enough to change the appropriate entry in the vector. In the calculation above we used the notation that the vector $\mathbf{r}$ is made by elements r_{it} where $i \in \{A, B, C\}$ and $t \in \{1Q, 2Q, 3Q, 4Q\}$. Similarly the vector of production $\mathbf{p}$ is made by entries p_{it}.
In order to calculate the total profit we need to calculate the total expenses:

$$\text{profit} = \text{revenue} - \text{expenses}$$

If the costs are the same throughout the quarters then the calculation is explained in [Le]. If we want to have the possibility to model different costs per each quarter then we need to create an appropriate cost matrix $C = (c_{ijt})$ where i is the raw material, j is the product and t is the quarter:

$$\begin{bmatrix} c_{1A1} & c_{1B1} & c_{1C1} & & & & & & \\ c_{2A1} & c_{2B1} & c_{2C1} & & \mathbf{0} & & & \mathbf{0} & \\ c_{3A1} & c_{3B1} & c_{3C1} & & & & & & \\ & & & c_{1A2} & c_{1B2} & c_{1C2} & & & \\ & \mathbf{0} & & c_{2A2} & c_{2B2} & c_{2C2} & & \mathbf{0} & \\ & & & c_{3A2} & c_{3B2} & c_{3C2} & & & \\ & & & & & & c_{1A3} & c_{1B3} & c_{1C3} \\ & \mathbf{0} & & & \mathbf{0} & & c_{2A3} & c_{2B3} & c_{2C3} \\ & & & & & & c_{3A3} & c_{3B3} & c_{3C3} \end{bmatrix}$$

Exercise 8

In python, create two arrays of size 1×3 of type double containing zeros. Write an "if... then" expression in which you print a message if the two arrays are the same. What is the outcome? Is it what you expected? Did you use == or <? Try complementing them with the attributes `any` and `all`.

Solution:

```
import numpy
a=array([1,2,3])
b=array([4,2,5])

a==b # array([False,True,False])
a<b # array([True,False,False])
(a==b).any() # True
(a==b).all() # False
```

Generate then an array of the same size as above of uniform random numbers from the interval $[0, 1] \times 10^{-3}$. Construct then another vector by summing to the previous one a small error decided at random from the interval $[0, 1] \times 10^{-16}$. Compare the two arrays as you did above. We wish to have that the two vectors are considered to be the same but what is the result? In order to check the equality of the two arrays up to machine precision there is the function `numpy.allclose`. Read the documentation of this function and set its parameters such that the equality of the two arrays is again evaluated to false.

Solution:

```
data = random.rand(3)*1e-3
small_error = random.rand(3)*1e-16
(data == data + small_error).any() # False
allclose(data,data+small_error) # True
allclose(data,data+small_error,atol=1e-08,rtol=1e-05) # absolute(‘a‘ - ‘b‘) <= (‘atol‘ + ‘
    rtol‘ * absolute(‘b‘))
```

Exercise 9* Gaussian Elimination

Let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 2 & 8 \end{pmatrix}$$

Use the augmented matrix $[A|I]$ to find the inverse matrix, using elementary row operations, in Python. Use the module `fractions` to cary out the operations in fraction mode (See the Python tutorial).

Solution:

```
AI[1]-=3*AI[0]
AI[2]-=6*AI[0]
print(AI)
AI[1]=-1/2.*AI[1]
AI[2]+=10*AI[1]
print(AI)
AI[2]=1/10.*AI[2]
AI[1]-=2*AI[2]
AI[0]-=2*AI[1]+3*AI[2]
AI
```

Exercise 10

Redo the previous exercise using elementary matrices.

Exercise 11* System of linear equations

Solve the system $A\mathbf{x} = \mathbf{b}$ for

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 2 & 8 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},$$

using:

1. the augmented matrix form $[A|b]$ and Gauss elimination,
2. the inverse matrix
3. `numpy.linalg.solve`

Do you achieve the same result? Why, why not?

Exercise 12

How many solutions have these linear systems? Use one of the methods from the previous exercise. Moreover, use the graphical approach.

$$\begin{aligned} 6x &- 10y = 2 \\ 3x &- 4y = 5 \end{aligned}$$

$$\begin{aligned} 3x &- 4y = 5 \\ 6x &- 8y = 10 \end{aligned}$$

$$\begin{aligned} 3x &- 4y = 5 \\ 6x &- 8y = 3 \end{aligned}$$

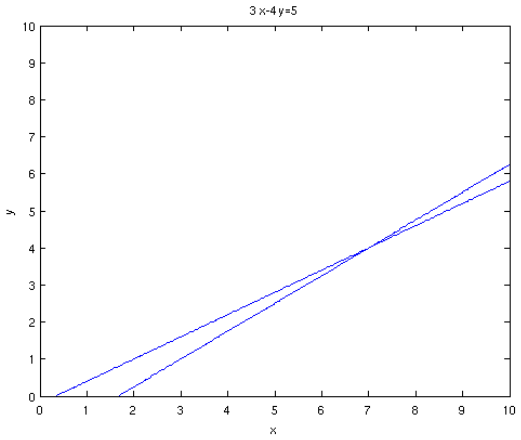
Solution:

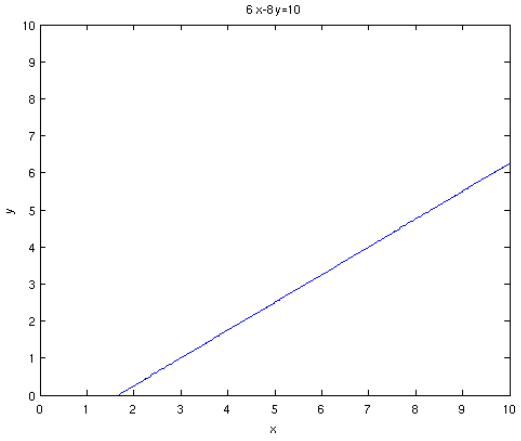
$$\begin{aligned} 6x &- 10y = 2 \\ 3x &- 4y = 5 \end{aligned}$$

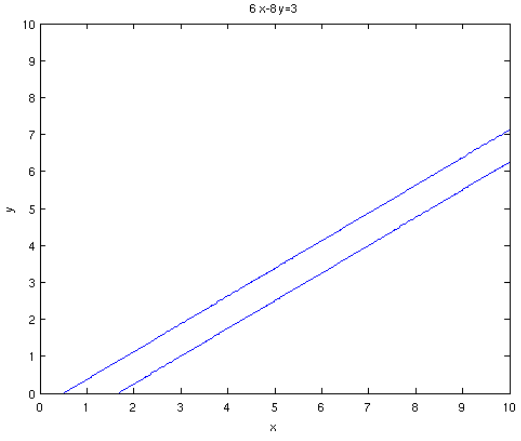
has one single solution

$$\begin{aligned} 3x &- 4y = 5 \\ 6x &- 8y = 10 \end{aligned}$$

has infinite solutions


3 x-4 y=5
y
x


6 x-8 y=10
y
x


6 x-8 y=3
y
x

$$\begin{array}{rcrcl} 3x & - & 4y & = & 5 \\ 6x & - & 8y & = & 3 \end{array}$$

has no solution

Exercise 13

Solve the following system of equations $A\mathbf{x} = \mathbf{b}$ by reducing the augmented matrix to reduced row echelon form (tip: check the function `rref` of the module SymPy in Python).

$$\begin{array}{rrrrrcr} x_1 & +5x_2 & +3x_3 & +7x_4 & +x_5 & = & 2 \\ 2x_1 & +10x_2 & +3x_3 & +8x_4 & +5x_5 & = & -5 \\ x_1 & +5x_2 & +x_3 & +3x_4 & +3x_5 & = & -4 \end{array} \tag{1}$$

Solution:

```
A= np.array([[1.,5,3,7,1],[2,10,3,8,5],[1,5,1,3,3]])
AB= np.array([[1.,5,3,7,1,2],[2,10,3,8,5,-5],[1,5,1,3,3,-4]])
import sympy
sympy.Matrix(AB).rref()
```

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -7-5s-t-4u \\ s \\ 3-2t+u \\ t \\ u \end{pmatrix} = \begin{pmatrix} -7 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} + s \begin{pmatrix} -5 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + v \begin{pmatrix} -1 \\ 0 \\ -2 \\ 1 \\ 0 \end{pmatrix} + u \begin{pmatrix} -4 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

```
p= np.array([[-7,0,3,0,0]]).T
v1 = np.array([[-5,1,0,0,0]]).T
v2 = np.array([[-1,0,-2,1,0]]).T
v3 = np.array([[-4,0,1,0,1]]).T
np.dot(A,p)
print("{}\n\n{}\n\n{}".format(np.dot(A,v1),np.dot(A,v2),np.dot(A,v3)))
```