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

October 16, 2024
Marco Chiarandini

DM587/AI511 – Linear Algebra and Applications

Sheet 0, Autumn 2024 [pdf format]

Solution:

Included.

Exercise 1

Write a list of the first 100 numbers in which any number divisible by three is replaced by the word "fizz" and any divisible by five by the word "buzz". Numbers divisible by both become "fizz buzz". Can you create a numpy array containig this list?

Solution:

```
for num in xrange(1,100):
  if num % 5 == 0 or num % 3 == 0:
    print("fizzbuzz")
  elif num % 5 == 0:
    print("buzz")
  elif num % 3 == 0:
    print("fizz")
  else:
    print(num)
```

Exercise 2 Data Types

Revise the difference between the main data types in Python: list, tuples, dictionaries and sets. Write an example for each of them in which you define and initialize a variable for each type and then print the content looping through the elements of the variable.
Numpy arrays can only contain data of the same type hence we cannot create an array from a list contining both numbers and strings.

Solution:

```
# lists my_list = [10,20,30,40,50,60] for i in my_list: print(i)

# tuples
my_tuple = (1,2,3,4,5,6,7,8,9)
for i in my_tuple:
  print(i)

# dictionaries
my_dict = {'name': 'Esau', 'age': 2, 'occupation': 'My dog'}
for key,val in my_dict.iteritems():
  print("My {} is {}".format(key,value))

# set
my_set = {20,30,40,50,60,20,30,40,50}
for i in my_set:
  print(i)
```

Exercise 3 Python: One liner quizzes

Write a one line Python code for the following tasks:

a) Construct 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}
```

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

Solution:

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

c) 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()}
```

d) 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()`.)

Exercise 4* Matrix Calculus in basic Python

The basic data structures in Pyton are lists, tuples, sets and dictionaries. Vectors and matrices can be implemented in Python as lists. How?

a) 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?

b) Write a function for the sum of two vectors using list comprehension.

c) Write a function for the multiplication of a vector by a scalar.

d) Write a function for the sum of two matrices using list comprehension.

e) Write a function for the multiplication of a matrix by a scalar.

f) 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")
```

Exercise 5* Matrix Calculus in numpy and scipy

The modules numpy and scipy make available another data structure in Python, the 'array' type. This exercise guides you to the discovery of how operators are overloaded for the 'array' type module. Generate in Python two matrices A and B of size 3×2 and 2×4, respectively, made of integer numbers randomly drawn from the interval $[1, \dots, 10]$. Calculate the following results, first by hand and then checking the correctness of your answer in Python numpy:

a) $A + B, A - B$

b) $A \cdot B$

c) 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 operations if the matrices have the same size.

Exercise 6* Matrix Operations

For some aribtrary dimension and some random numbers in the matrices:

a) Construct an array of zeros

b) Concatenate an identity matrix to a matrix

c) Insert a row in between other two

d) Print the dimensions of an array

e) Multiply matrices

f) Print the matrix transpose

g) Print the rank of a matrix

Exercise 7* Matrix Inverse

Calculate the inverse of these two matrices:

$$A = \begin{pmatrix} -1 & \frac{3}{2} \\ 1 & -1 \end{pmatrix} \qquad B = \begin{pmatrix} -1 & \frac{3}{2} \\ \frac{2}{3} & -1 \end{pmatrix}.$$

Exercise 8* Indexing and slices

Construct an array of dimension 3×3 containing the first 9 natural numbers. Remember that indices start at 0.

a) print the element at (0,0)

b) print all rows starting from the second

c) print the second row of the matrix

d) print a vector filled by the 3rd column of the matrix

e) print the type of the data contained in the matrix

f) change the type of the data in the matrix to be a floating point number with 64 bits.

g) print the size of each dimention of the matrix

h) flatten the matrix using reshape and ravel

i) copy the flattened version and change a value in the elements of the copy. Show the difference in these operations between shallow and deep copy.

Exercise 9* Construction of matrices and vectors

a) Construct a matrix of size 3×4 containing only zeros. Which type of data contains the matrix created?

b) Repeat the previous task with a matrix of all ones.

c) Create an identity matrix of size 4×4

d) Create a diagonal matrix of size 4×4 containing the first four natural numbers in the diagonal

e) Create a matrix of size 3×4 containing random numbers between 0 and 1, extremes included.

f) Create a matrix of size 3×4 containing random integer numbers between 1 and 10, extremes included.

g) Create an array containing the first 5 integer numbers.

h) Create an array of incremental numbers from 0 to 5 with intervals of 0.5.

i) Concatenate two random matrices of size 3×4 first vertically and then horizontally.

Exercise 10* Solving Systems of Linear Equations

Solve by Gaussian elimination the following system of linear equations $A\mathbf{x} = \mathbf{b}$ where

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

First carry out the calculations by hand and then try using Python numpy. Finally, compare your result with the one of `numpy.linalg.solve`.

Solution:

```
A=np.array([[3,1,1,1],
  [2, 4, 1, 1],
  [2,1,2,2]])
b=np.array([ 2, 2, 1])

# np.linalg.solve(A,b)
print np.linalg.matrix_rank(A)
AA=np.column_stack([A,b])
AA=column_stack([A,b])
AA[0,:] = f(1,3) * AA[0,:] # remember indices start from 0
AA[1,:] = -2 * AA[0,:] + AA[1,:]
AA[2,:] = -2 * AA[0,:] + AA[2,:]
printm(AA)
AA[1,:] = f(3,10)* AA[1,:]
AA[2,:] += -f(1,3) * AA[1,:]
printm(AA)
AA[2,:] = f(10,13) * AA[2,:]
printm(AA)
AA[1,:] += -f(1,10) * AA[2,:]
AA[0,:] += -f(1,3) * AA[2,:]
printm(AA)
AA[0,:] += -f(1,3) * AA[1,:]
printm(AA)
```

After elementary row operations the augmented matrix looks like:

```
[1, 0, 0, 0, 9/13]
[0, 1, 0, 0, 3/13]
[0, 0, 1, 1, -4/13]
```

From here we can write directly the solution:

$$\begin{aligned} x_1 &= 9/13 \\ x_2 &= 3/13 \\ x_3 &= -4/13 - t \\ x_4 &= t \end{aligned}$$

which in vector notation is equivalent to

$$\mathbf{x} = \begin{bmatrix} 9/13 \\ 3/13 \\ -4/13 \\ 0 \end{bmatrix} + t \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix}$$

Hence the solution subspace is expressed by an affine combination.

Exercise 11 3D Arrays in NumPy

a) Create a 3D array of shape $(3, 3, 3)$ filled with random integers between 0 and 9.

Solution:

```
import numpy as np

array_3d = np.random.randint(0, 10, (3, 3, 3))
print("3D Array:\n", array_3d)
```

b) Access the element at position $(1, 2, 1)$ in the 3D array.

Solution:

```
element = array_3d[1, 2, 1]
print("Element at position (1, 2, 1):", element)
```

c) Extract a 2D slice from the 3D array. Slice the first matrix (i.e., the 0th index along the first axis).

Solution:

```
slice_2d = array_3d[0, :, :]
print("2D Slice from the 3D array (0th matrix):\n", slice_2d)
```

d) Set all elements in the second matrix (i.e., the 1st index along the first axis) to 5.

Solution:

```
array_3d[1, :, :] = 5
print("3D Array after modification:\n", array_3d)
```

e) Calculate the sum of the elements along axis 0 (collapsing the first dimension).

Solution:

```
sum_axis_0 = np.sum(array_3d, axis=0)
print("Sum along axis 0:\n", sum_axis_0)
```

f) Reshape the 3D array into a 2D array of shape (9, 3).

Solution:

```
reshaped_array = array_3d.reshape(9, 3)
print("Reshaped 2D Array (9, 3):\n", reshaped_array)
```

g) Create a 3D identity matrix of shape (3, 3, 3) where each matrix along the first axis is an identity matrix.

Solution:

```
identity_3d = np.array([np.eye(3) for _ in range(3)])
print("3D Identity Matrix:\n", identity_3d)
```

Exercise 12 Runtime

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 square matrix of random integer numbers large enough. Undertake the following experimental analysis:

a) Compare the runnning time to multiply A by itself using the numpy and scipy libraries and the function you wrote in Exercise 1.

b) 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.

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.