Exercise 00

Introduction to Linux and Python

For next week you should do some pratice with the Linux commands and solve three exercises with Python (+NumPy). At the end of each line you can find [between square brackets] the reference to the "Linux and Python" manual with the functions necessary.
At the end, you have to send you solution to `luca.donati@fu-berlin.de`. This first session of exercises will be corrected, but not evaluated.

1 Linux I

1. Create a folder called "TCexercises" in your home directory. [1.3]
2. Enter in the folder and create a subdirectory "exercise00". [1.3]
3. Create a Python file for each exercise. [1.4.1]

2 Python

1. (a) Create the following matrix 5×5 (2 dimensional NumPy array):

$$M = \begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix}$$

   What is the sum of the rows? What is the sum of the columns? [2.4.1, 2.4.5]

   (b) Exchange the first row with the first column. [2.4.3]

   (c) Create a vector (1d NumPy array) $\vec{u} = (1, 2, 3, 4, 5)$ and sum each element of the first column with each element of the vector. [2.4.3, 2.4.4]

   (d) Multiply by 2 the last two elements of the first and of the second column. [2.4.3, 2.4.4]

   (e) Replace by 3 all the elements of the matrix greater than 15. [2.4.3]

   (f) Compute the trace (sum of the elements of the diagonal) of the matrix M. There are many ways to compute the trace of a matrix, try the following:

      i. Implement a for loop to extract all the elements of the diagonal. [2.6.1]
      ii. Use the function `diagonal()` to extract all the elements of the diagonal. [2.4.5]
      iii. Use the function `np.trace()` to check your previous results. [2.4.5]

   (g) Find the eigenvalues and the eigenvectors of the matrix M and verify the relation $M\vec{v} = \lambda\vec{v}$. [2.4.5, 2.4.4]

   (h) Create a second 5×5 matrix A of random integer numbers (from a uniform distribution) in the interval [0,9]. [2.4.2, 2.4.5]

   (i) Compute the determinant of the matrices and verify the relation $\det(M \cdot A \cdot M^{-1}) = \det(M) \cdot \det(A) \cdot \det(M^{-1})$.[2.4.5, 2.4.4]

2. (a) In this second exercise, you will practise with the plot functions, so first of all, import the matplotlib.
   Generate a vector x of $N = 10000$ equally spaced points in the interval $[-50, 50]$ and compute $y_1 = \frac{\sin(x)}{x}$ and $y_2 = |\frac{\sin(x)}{x}|$. [2.4.2, 2.4.6]

   (b) plot y_1 and y_2 using two different colors and two different line styles [2.5].

   (c) Reduce the x-axis to $[-4\pi, 4\pi]$ and set ticks only for $-4\pi, -2\pi, 0, 2\pi, 4\pi$. [2.5]

   (d) Add title, x-label, y-label and legend to the graph and save it in the EPS format. [2.5]

   (e) Compute $y_3 = \exp(x)$ with $x \in \mathbb{R}$ and $y_4 = \log(x)$ with $x \in \mathbb{R}^+$. [2.4.3, 2.4.6]

(f) Plot in a new figure y_1 and y_2 using two different colors and two different line styles. [2.5]

(g) Reduce the y-axis to $[-3, 2]$. [2.5]

(h) Add title, x-label, y-label and legend to the graph and save it in EPS format. [2.5]

3. (a) Generate an array P of size $N \times 3$, with $N = 1000$ random numbers (from a uniform distribution). Each row of the array represents the position in the 3d space of a particle. [2.4.1]

(b) Create a Python file where to define a new function (you can also use the file userfunctions.py created during the tutorial) that generates a symmetric matrix d of size $N \times N$, where each element d_{ij} is the euclidean distance between each pair of points (i, j):

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

There are many methods to implement this function, you should implement the following three.

i. The first method should use two for loops. [2.6.1]

```
def distance1(P):    #P is the array $Nx3$
        x = P[:,0]       #we store each column in a vector
        y = ...
        ...
        for i in range(N):
                for j in range(N):
                        d[i,j] = np.sqrt((x[i] - x[j])**2 + ... )

        ...
```

ii. Inside the double for loop, we have computed the difference between each pair of elements of the vectors x, y and z. This method is quite slow, but we can improve it vectorizing the algorithm.
We can create the same matrices x_i - x_j doing a difference between the "row vector" x and the "column vector" x. To do this, we need the function np.newaxis that adds "a new dimension to the array". Basically it just transforms a row vector in a column vector.
If X is a vector, X[:,np.newaxis]-X is a matrix M where each element M_{ij} is X_i - X_j. You should generate these matrices for each dimension and use them to compute the distance with the "elementwise opeartions". [2.4.4]

```
def distance2(P):    #P is the array $Nx3$
        x = P[:,0]
        y = ...
        ...
        dx = x[:,np.newaxis]-x #dx is a matrix NxN
        ...
        d = np.sqrt(dx**2 + ... )
        ...
```

iii. Import from SciPy the function distance and check your previous results:

```
import scipy.spatial.distance as dist

d = dist.cdist(x,x)
```

To measure the execution time, you have to use the function time as follows:

```
import time

t_begin = time.time()
#instructions....
t_end = time.time()

print "Elapsed time: ", t_end - t_begin
```

Plot the results

3 Linux II

You can try the following linux commands with the three python files.

1. List the created files and check their dimension. [1.5.1]
2. Create a folder "backup" and copy all the files in it. [1.5.2]
3. Enter in the folder "backup" and delete the files 00-1.m and 00-2.m. [1.3, 1.5.2]
4. Go back in the up directory and delete the full directory "backup". [1.3, 1.5.2]
5. Create a TAR archive of the exercise folder. [1.6]