COMPUTER SESSION # 1

The goal of this first computer session is to review some basic manipulations in Python and to get down to business regarding basic optimisation algorithms. The basic packages we will be using throughout are the `numpy` and `matplotlib` packages; do not forget to load them.

```
import numpy
import matplotlib.pyplot as plt
```

0.0.1. *Basics in Python.* The `numpy` library deals with arrays and matrices, and is particularly useful for linear algebra.

Exercise 0.1. *Consider the matrix A defined as:*

$$\begin{pmatrix} 4 & 6 & -2 & 3 \\ 2 & -1 & 0 & 1 \\ -7 & 0 & 1 & 12 \end{pmatrix}.$$

(1) *Define A as a* `numpy.array()`.
(2) *Print the first line and the second column of A.*
(3) *Create a new matrix* `Ac` *as a copy of A. Modify it by multiplying the first two lines by 2 and (then) divide its last column by 3.*
(4) *Define the new matrix B*

$$\begin{pmatrix} 4 & 5 & 6 \\ 5 & 10 & 15 \\ 1 & 1 & 1 \end{pmatrix},$$

(5) *Go back to the initial matrix A. Create a matrix $C \in M_{33}\mathbb{R}$ as a sub-matrix of A: for any $1 \leq i, j \leq 3$, $c_{ij} = a_{ij}$.*
(6) *Matrix product*
   - *Create $D = BA$ (using* `numpy.dot()`*).*
   - *Create $E = B \cdot C$, where $\cdot$ denotes the Hadamard product of matrices:*

$$\forall 1 \leq i, j \leq 3, \quad e_{ij} = c_{ij} b_{ij}.$$

(7) *Compute the sum of all elements of E, and create the vector $Y \in \mathbb{R}^3$ whose coordinates are given, for $1 \leq i \leq 3$, by $y_i = \sum_{j=1}^{4} d_{ij}$.*

0.0.2. *Plotting curves.* The basic command for plotting curves is `plot(x,y)`. In this expression, `x` et `y` are (`array`) with the same size which can be either declared or generated. You should use `linspace(a,b,N)` to represent as a list the interval (a, b) with N (uniform) discretisation points.

The functions `title, axis, legend, x/ylabel` are useful for the presentation of the graphs.The typical code will write as follows:

```
1
2  def f(x) : return .... # The function
3
4  xx=linspace(a,b,N) #
5  plot(xx, f(xx),'color') # 'color'  is optional but allows to
       choose the color of the curve
6
7  axis('equal') # the two axis are at the same scale
8  title("Graph")
9  legend("f")
10 xlabel("\$ x\$-axis")
11 ylabel("\$ y\$-axis")
```

Exercise 0.2. (1) *Plot the graph of* $f : x \mapsto x^2\cos(10*x)$ *on* $(-\pi;\pi)$ *in green.*
(2) *Plot, on the same graph, the functions* $f_n : x \mapsto x^2\cos(nx)$ *pour* $x \in (-\pi,\pi)$ *for* $n = 0, 1, \ldots, 10$. *Observe that if we create a graph using* `plt.plot(xx,f(xx),...)`*, we can call back the function* `plt.plot(xx,g(xx),....)` *to draw another graph on the same figure.*

Finally, the `contour` function is extremely useful to plot level-sets of functions of several variables. To do so, we consider a function f of two variables and we generate an array containing all the values of f as follows: once f and the two domains of definition are give, we can write:

```
1 z=[[f(x,y) for x in ... ] for y in ...]
```

The function fonction `plt.contour(`x `domain,`y `domain,Z,N)` plots N level sets. The `plt.colorbar()` command allows for a tuning of the colour scheme.

Exercise 0.3. *Plot 20 level sets of*

$$f : (x, y) \mapsto e^{-x^2}\sin(\pi x - y)$$

fo $(x, y) \in (-4, 4)^2$. *Toy around with the options* `cmap='inferno',cmap='plasma'`*...*

0.0.3. *Sequences and convergence rates.* We consider a sequence $\{x_k\}_{k\in\mathbb{N}} \in (\mathbb{R}^d)^{\mathbb{N}}$ and a point $x^* \in \mathbb{R}^d$. For $\alpha \in (0; 1)$, we say that $\{x_k\}_{k\in\mathbb{N}}$ converges to x^* linearly at rate α if there exists a constant C such that

$$\forall k \in \mathbb{N},\ \|x_k - x^*\| \leq C\alpha^k.$$

We say that the convergence is super-linear if

$$\forall \alpha \in (0; 1),\ \frac{\|x_k - x^*\|}{\alpha^k} \underset{k\to\infty}{\to} 0.$$

We say it is sub-linear if

$$\forall \alpha \in (0; 1),\ \frac{\|x_k - x^*\|}{\alpha^k} \underset{k\to\infty}{\to} +\infty(\text{along a subsequence}).$$

The sequences that will be considered in this class are all of recursive type, meaning they are defined by a relation of the type

$$x_{k+1} = G_k(x_0, \ldots, x_k).$$

In order to study the convergence rate of sequences, the usual way to proceed, without knowing what the limit is, is the following:

(1) Compute a large number N of terms of the sequence, and consider x_N as some approximation of the would-be limit point x^*.
(2) Use the `semilogy` function of python to plot $\|x_k - x*\|$ in logarithmic scale. What do you expect to see in this logarithmic scale if the convergence is linear? super-linear? sub-linear?

Exercise 0.4. *Study the (sub/super/∅)linear convergence rates for the following sequences:*

(1) $x_0 = 1$ *and, for any* $k \in \mathbb{N}$,
$$x_{k+1} = x_k \left(1 - \frac{x_k}{2}\right).$$
(2) $x_0 = 1$ *and, for any* $k \in \mathbb{N}$,
$$x_{k+1} := \frac{x_k}{2} + \frac{1}{x_k}.$$
Numerical warning (Machine Precision): For sequences that converge very quickly, the numerical evaluation of $\|x_k - x^*\|$ *will eventually drop below machine precision (roughly* 10^{-16} *in Python). As the* `semilogy` *function cannot process* $\log_{10}(0)$*, this will result in an error message. To properly visualise the sequence convergence, you can instead plot* $\|x_k - x^*\| + \varepsilon$ *with* $\varepsilon =$ `1e-16`.

0.0.4. *Optimality conditions and Hessians.* The `numpy.linalg` submodule contains standard routines for linear algebra, such as `solve(A,b)` to solve the linear system $Ax = b$, and `eigvals(A)` to compute the eigenvalues of a matrix A.

Exercise 0.5. (1) ***Quadratic functions.*** *Consider the function* $f : \mathbb{R}^3 \to \mathbb{R}$ *defined by* $f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$*, with*
$$A = \begin{pmatrix} 4 & -1 & 1 \\ -1 & 4 & -2 \\ 1 & -2 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}.$$
(a) *Compute (analytically)* $\nabla f(x)$ *and* $\nabla^2 f(x)$.
(b) *Using* `numpy.linalg.solve`*, find the unique critical point* x^* *of* f.
(c) *Using* `numpy.linalg.eigvals`*, compute the eigenvalues of the Hessian* $\nabla^2 f(x)$*. Is the function* f *coercive? Convex? What is the nature of the critical point* x^**?*
(d) *Replace* A *with* $A' = A - 4I_3$ *(where* I_3 *is the identity matrix, which can be generated using* `numpy.eye(3)`*). Re-evaluate the critical point and its nature. What has changed?*

(2) ***Non-linear functions.*** *Consider* $g : \mathbb{R}^2 \to \mathbb{R}$ *defined by* $g(x, y) = x^4 + y^4 - 4xy + 1$.
(a) *Compute (analytically) the critical points of* g.
(b) *Write a Python function* `hessian_g(x, y)` *that returns the Hessian matrix of* g *evaluated at* (x, y) *as a* 2×2 `numpy.array`.
(c) *Evaluate the eigenvalues of the Hessian at each critical point. Conclude on their local optimality (local minimum, local maximum, or saddle point).*

(d) *Bonus: Verify your findings by plotting the level sets of g using* `contour` *on the domain* $(-2, 2)^2$.