ComputerSession1Subject

January 20, 2025

1 Computer session 1

1.1 Basics in Python

[1]:
```
import numpy
import matplotlib.pyplot as plt
```

1.1.1 Exercice 2.16

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 defined by $1 \leq i, j \leq 3, c_{ij} = a_{ij}$.
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 $y_i = \sum_{j=1}^{4} d_{ij}$.

1.2 Plotting curves

The basic command for plotting curves is `plot(x,y)`. In this expression, `x` et `y` are (`numpy.array`) with the same size which can be either declared or generated. You should use

`numpy.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:

```
def f(x) :
    return .... # The function

xx = linspace (a ,b , N) #
plot( xx , f(xx) , 'color') # 'color' is optional but allows to choose the color of the curve
axis('equal') # the two axis are at the same scale
title("Graph")
legend ("f")
xlabel("\$ x\$-axis")
ylabel("\$y\$-axis")
```

1.2.1 Exercice 2.17

1. Plot the graph of $f : x \mapsto x^2 \cos(10x)$ 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 `numpy.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:

```
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.

1.2.2 Exercise 2.18

Plot 20 level sets of

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

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

1.3 Basics of gradient descent

1.3.1 Exercise 2.19

1. We first consider the minimisation problem

$$\inf_{x \in \mathbb{R}^d} \frac{1}{2} \langle Ax, x \rangle - \langle b, x \rangle .$$

We work in dimension 2. Write a python function f that takes as arguments A, b and x and returns $\frac{1}{2} \langle Ax, x \rangle - \langle b, x \rangle$.

2. Choose any symmetric, positive definite matrix A and generate a random vector b. Solve the equation

$$Ax = b$$

using the gradient descent method (you are of course allowed to use, here, the explicit expression for the gradient). Can you illustrate the order of convergence of your method?

3. Plot the level sets of the function f, and represent the successive iterates on the graph.

1.4 Exercice 2.20

We now consider a function f of d variables.

1. We aim at approximation, numerically, the gradient of a function f. In order to do so, we resort to a centred difference methods. In other words, we write:

$$\frac{\partial f}{\partial e_i} \simeq \frac{f(x+\delta e_i) - f(x-\delta e_i)}{2\delta}$$

for $\delta > 0$ small enought where $\{e_i\}_{i=1,\dots,d}$ is the canonical basis of $\mathbb{R}^d$. First define the vectors e_i in Python. Second, write a function `gradient(F,x,d,delta=1e-05)` that returns the discrete gradient of the Python function `F`. You can test your code on the following toy function:

[2]:
```
# Test function
def F(x):
    return x[0]**2 + x[1]**2
```

2. Write a code that takes, as arguments, x_0 and a step size τ and returns the sequence generated by the gradient descent initialised at x_0 with fixed step size τ. Test your code on `F`.

We now apply this to the least square problem to test our algorithm: fix $\alpha_0 = 3$, $\alpha_1 = 2$, an integer N and generate a random vector of size N by calling `Xi=numpy.random.rand(N)*2-1` (this generates numbers between -1 and 1). Generate a "noise" vector `Wi=numpy.random.rand(N)*2-1`. Finally, set `Yi=alpha0+alpha1*Xi+Wi` (the set of noisy observations we have access to). We want to see whether we can recover the values of α_0 and α_1 from the knowledge of `(Xi, Yi)`. To do so, we define, for an array $A = (a_0, a_1)$,

$$E(A) = \frac{1}{N}\sum_{k=1}^{N} |y_i - (a_0 + a_1 x_i)|^2$$

.

3. Write the function $A \mapsto E(A)$ in python, A being coded as an `numpy` array. The vectors `Xi` and `Yi` are `numpy` array defined globally.

4. Plot 20 level lines of E. Is what you observe coherent with we might expect?

5. Minimise E with a constant step-size gradient descent using the discretised gradient you computed earlier. Run your code with several values of τ and find the best value for τ.

6. Check that we find a good approximation for (α_0, α_1). Show datas `(Xi,Yi)` and the line $y = a_0 + a_1 x$ that you obtain, for $x \in [-1, 1]$.

[ ]:

[ ]: