Project

Information

- The codes will be in ONE (1) python or jupyter-notebook file called YOURNAME.py (or .ipynb)
- Your project file need to be **uploaded on Moodle** assignment called TP_project before the **1st of May at noon**
- Your answers will be written as comments,

```
# answers
#
"""
answers

"""
```

Exercise 1. Remove noise from a signal.
The goal here is to smooth a noisy signal with a gradient method. Let $y = (y_i)_{i=1,\dots,N}$ be a noisy signal measured at times $(t_i)_{i=1,\dots,N}$. To remove the noise we will minimize in $\mathbb{R}^N$ the following cost function

$$f(x) = \frac{1}{2}\sum_{i=1}^{N}(x_i - y_i)^2 + \frac{\lambda}{2}\sum_{i=1}^{N-1}(x_{i+1} - x_i)^2$$

where the first sum is related to the data, it is minimal when x is exactly y. The second sum is a regularization that penalizes the large variations of the signal. This second term is minimal if x is constant. The parameter $\lambda \geqslant 0$ denotes the weight of the regularization, if $\lambda = 0$ the solution of the minimization problem is $x = y$ and for large λ the solution will be far from the original signal.

1. Re-write the minimization problem as follows : $f(x) = \frac{1}{2}\|x - y\|^2 + \frac{\lambda}{2}\|Dx\|^2$, where $D \in \mathcal{M}_N(\mathbb{R})$.
   Write a function `D_mat` that, given the dimension N, returns the matrix D.
2. Verify that f is a differentiable function then write two functions that compute `f` and `grad_f` the gradient of f given a vector x.
3. Use the following function `generate_signal` to generate a noisy signal and display the obtained signal, we will choose $N = 100$.

```
def generate_signal(N,sigma=0.05):
    t = np.linspace(0,1,N)
    t1 = 0.1+0.25*np.random.random()
    t2 = 0.35+0.25*np.random.random()
    yi = np.array([-0.1,0.8,0.2])
    y = np.zeros(N)
    for i in range(y.size):
        if t[i]<=t1:
            y[i]=yi[0]
        elif t[i]>t1 and t[i]<=t2:
            y[i] = yi[1]
        else:
            y[i] = yi[2]
    y += sigma*(2*np.random.random(y.size)-1)
    return t,y
```

4. Fill in the following `gradient_met` function that implements the fixed step gradient method:

```
def gradient_met(grad_f, x0, alpha, eps, Nmax):
  """
  grad_f a function that compute the gradient of f given x
  x0 the initialization
  alpha the step (constant)
  eps the precision (stop criteria)
  Nmax the maximum number of iterations
  """

    ### TO FILL IN


    return x,n, cvg # returns
                # x the minimum,
                # n the number of iterations
                # cvg a boolean that indicates if the algorithm has
                                                    converged
```

5. Use a fixed step gradient method to reduce the noise from the signal, and plot the noisy signal and the signal without noise on the same figure with the number of iterations of gradient method in legend.

6. If you take $\lambda = 1$ and $\alpha = 0.5$ what happens? Is it expected ?

7. Vary λ from 1 to 4 by step of 1 and plot the obtained signals on the same figure (with λ values in the legend). What do you observe?

8. You should have noticed that the previous method does not well capture the initial signal. We thus change the regularization term and consider the cost function

$$f_\mu(x) = \frac{1}{2}\sum_{i=1}^{N}(x_i - y_i)^2 + \lambda \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + \mu^2},$$

   where μ is a small parameter (typically of order 10^{-2}).

   Write a function that computes the gradient of f_μ.

9. Use the fixed step gradient method (with $\mu = 0.01$, you might need to adapt the algorithm parameters *i.e* step, max number of iterations...) to minimize the cost function and display the obtained signal. Compare to the previous regularization and comment your results.

10. We now propose to use a gradient method with a step (α_k) that varies, using the Barzilai–Borwein method, for all $k \in \mathbb{N}$, when $\Delta g_k \neq 0$:

$$\alpha_k = \frac{\langle \Delta g_k, \Delta x_k \rangle}{\|\Delta g_k\|^2}$$

with $\Delta x_k = x_k - x_{k-1}$ and $\Delta g_k = \nabla f(x_k) - \nabla f(x_{k-1})$.
Hint: we see that to compute the step α_k we need 2 previous points x_k and x_{k-1}, so the algorithm needs 2 initialization points, we can choose $x_{-1} = -x_0$ with $x_0 \neq 0$.

- Fill in the following function that computes the Barzilai-Borwein step:

```
def BBstep(grad_f, x, xm1) :
  """
  grad_f a function that compute the gradient of f given x
  x0 the initialization point x_0
  xm1 the initialization point x_-1
  """

  ### TO FILL IN

  return bbs # the step in the method of Barzilai-Borwein
```

- Fill in the following `barzilai_borwein` function that implements Barzilai-Borwein method:

```
def barzilai_borwein(grad_f, x0, eps, Nmax):
  """
  grad_f a function that compute the gradient of f given x
  x0 the initialization
  eps the precision (stop criteria)
  Nmax the maximum number of iterations
  """

    ### TO FILL IN


    return x, n, cvg # returns
                # x the minimum,
                # n the number of iterations
                # cvg a boolean that indicates if the algorithm
                                         has converged
```

- Use Barzilai-Borwein method to remove noise from your signal and compare the result with the previous method. Comment your results.

Exercise 2. Remove noise from an image.

The two previous denoising methods (exercise 1) can also be applied to images. We will consider only black and white images here.

The discrete gradient ∇^d is useful when you do image processing. It is defined using the finite differences in each direction :

$$\nabla^d : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \quad \nabla^d u = (\delta_x u, \delta_y u)$$

with (x-direction) :

$$(\delta_x u)_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i \neq m \\ 0 & \text{otherwise} \end{cases}$$

and (y-direction) :

$$(\delta_y u)_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j \neq n \\ 0 & \text{otherwise} \end{cases}$$

The following Python code loads the image of Mona Lisa (available on Moodle), converts it to an array and displays it.

```
from matplotlib.image import imread
image = imread('Grey_Mona_lisa.jpg')

imageArray = np.asarray(image,dtype=np.float64)[:,:,0].copy()/255. #
                                        convert to array
# plot with grey levels
plt.imshow(imageArray, cmap='gray')

g = imageArray.copy()
```

1. Write two functions that given an (m, n) array compute the discrete derivatives in each direction (δ_x and δ_y). Display your results for the Mona Lisa image on 2 different images.

2. Write a function that computes the norm of the discrete gradient : the matrix $\mathcal{M}_{m,n}(\mathbb{R})$ defined by

$$M_{i,j} = \sqrt{(\delta_x)_{i,j}^2 + (\delta_y)_{i,j}^2}$$

   Display your result for the Mona Lisa. What do you remark ?

3. Solve with a gradient method the following minimization problem to reduce noise from Mona Lisa image.

$$\min_{v \in \mathbb{R}^{m \times n}} J(v) \text{ with } J(v) = \frac{1}{2}\|v - g\|^2 + \frac{\lambda}{2}\|\nabla^d v\|^2,$$

   where g is the array representing the noisy image.
   You will test different values for the parameter λ and comment your results.

4. Can you change the cost function as in the first exercise, introducing a function J_μ ? Compare with the previous method, and comment the results.

5. Load an image of your choice and convert it to an array.

   (a) Create a gaussian white noise and add it your image. You will obtain a noisy image, display both images (the level of noise will be display in the image title).

   (b) Use the previous methods to reduce the noise of the created noisy image.

   (c) Compare the different methods: quantify the differences between the initial image (without noise) and the results of the methods.

   (d) Comment the results.