Amrita School of Engineering, Bengaluru

Amrita Vishwa Vidyapeetham

Course: 19MAT204 – Mathematics for Intelligent Systems - 3

Course Instructor: Dr. Deepa Gupta

Course Project: Total Variation Minimization using ADMM

<u>Team – Dynamic Dudes:</u>

1.) BL.EN.U4AIE190**13** - **CHARAN TEJ K.**

2.) BL.EN.U4AIE190**14** - **CHAVALI SURYA TEJA**

3.) BL.EN.U4AIE190**62** - **SUGASH T.M.**

INTRODUCTION

Nowadays, the denoising or noise reduction technique is very prominent in the field of digital image processing and speech signal processing. For many years, it has been a great help in producing better quality images or better output signals. One such noise removal method is **Total Variation Minimization**.

Popularly known as **Total Variation Denoising (or) Total Variation Regularization**, is a technique to understand the behavior of the noise pattern and minimizes it to the maximum extent. The main goal behind this method's development is to preserve sharp edges in the underlying signal and mostly deals with those images and signals corrupted with **Gaussian noise**.

One way we should understand the problem is that denoising is a method of fitting a dense set of existing data while minimizing some added metric (regularization). Unlike any conventional "low pass filter", used in many for minimizing items in hardware, TV denoising is defined in terms of an optimization problem, and the result/output of the TV denoising function is obtained by minimizing a particular cost function.

Total variation is used not just for denoising, but for more general signal restoration problems, including deconvolution, interpolation, in-painting, compressed sensing, etc. Besides that, the concept of total variation has been generalized and extended in various ways.

Now that we’ve looked at the background of Total Variation Minimization, lets now proceed towards the mathematics part involved in it.

TOTAL VARIATION MINIMIZATION

The total variation function (TV) is defined as the sum of the absolute values of adjacent differences of the signal or data in it.

This is given by:

$$\mathbf{TV(x)} = \sum_{\mathbf{i=0}}^{\mathbf{N}} |\mathbf{f(x_{i+1}) - f(x_i)}|$$

where, $f(x_i)$ = previous value with respect to the current value

But, our whole idea of minimizing this function should be able to remove the Gaussian noise from the data. So, our total denoising problem looks like this:

$$\textbf{minimize} \quad \frac{1}{2}||\mathbf{x - b}||_2^2 + \lambda \sum_{\mathbf{i=0}}^{\mathbf{N}} |\mathbf{f(x_{i+1}) - f(x_i)}|$$

Here, we find a vector x that minimizes the sum-of-squares of the differences (which are the least-squares) between itself and the input b but also keep the TV of the new vector low. Also, a new variable ‘λ' has been used here, which is a regularization parameter, introduced to tune the algorithm performance for better accuracy and also to control the degree of smoothing.

Increasing ‘λ’ gives more weight to the second term which measures the fluctuation of the signal. An approach to set the parameter λ is described as the automatic method of tuning parameter choice for total variation denoising. The method is computationally much simpler than cross-validation.

Total variation denoising assumes that the noisy data $y(n)$ is of the form:

$$\mathbf{y(n) = x(n) + w(n), \qquad n = 0, \ldots, N - 1}$$

where, where $'\mathrm{x(n)}'$ is an (approximately) piecewise constant signal and '$\mathrm{w(n)}$' is white Gaussian noise.

This equation was not shown much importance in the further discussion of our topic implementation.

Notations used:

1) The N – point data used is represented as:

$$\boldsymbol{x = [x(0), \ldots, x(N - 1)]T}$$

2) Vector L1 Norm:

This length is sometimes called the taxicab norm or the Manhattan norm.

$$\boldsymbol{L^1(v) = \sum_{i=0}^{N} ||v||_1}$$

Ex: - $v = (a, b, c)$, then $\boldsymbol{L^1(v)} = |\boldsymbol{a}| + |\boldsymbol{b}| + |\boldsymbol{c}|$

3) Vector L2 Norm:

The L2 norm is calculated as the square root of the sum of the squared vector values. It is also known as the Euclidean norm as it is calculated as the Euclidean distance from the origin.

$$\boldsymbol{L^2(v) = \sum_{i=0}^{N} ||v||_2}$$

Ex: - Considering the previous example, $\boldsymbol{L^2(v)} = \sqrt{a^2 + b^2 + c^2}$

4) The first-order difference D of the above N – point data:

The matrix D is a sparse matrix and is defined as:

$$\mathbf{D} = \begin{bmatrix} -1 & 1 & & & & \\ & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & \\ & & & & -1 & 1 \end{bmatrix}.$$

where D is of size (N −1)×N and, for later that D*D^T is a tridiagonal matrix of the form. We took D*D^T so that we could extract more diagonals from the above sparse matrix, to get more data.

ALTERNATING DIRECTION METHOD OF MULTIPLIERS

Alternating Direction method of multipliers (ADMM) is known as one of the most efficient methods for solving convex problems. The ADMM is a substitute for the method of multiplies concept which uses the dual decomposition method.

Before moving on, let's look at some precursors and background theory for ADMM:

1.) Lagrangian Function:

We often come across some scenarios, where variables are required to satisfy certain constraints. At this time, we have to face a restriction to our function. In this case, the **Lagrange Multiplier Method** is very useful to deal with those constraints (restrictions).

We want to maximize the function f (x, y) where x and y are restricted to satisfy the equality constraint g (x, y) = c:

$$\max - f\,(x, y) \text{ subject to } g\,(x, y) = c$$

Then, we define the Lagrangian function, a modified version of the objective function that incorporates the constraint:

$$\mathbf{L}\,(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) = \mathbf{f}\,(\mathbf{x}, \mathbf{y}) + \boldsymbol{\lambda}\,[\mathbf{c} - \mathbf{g}\,(\mathbf{x}, \mathbf{y})]$$

where the term λ is an (n unknown) constant called a Lagrangian multiplier, associated with the constraint. (We will be denoting the Lagrange multiplier as '*y*' in the later sections of the report.

2.) Penalty Function:

The goal of penalty functions is to convert constrained problems into unconstrained problems by introducing an artificial penalty for violating the constraint.

Another way of visualizing penalty functions is that they act as a tuning parameter to make sure that the function moves in the right way while reaching a stationary point, without deviating away. If you are trying to maximize our objective function, the penalty will subtract the value.

3.) <u>Dual Problem:</u>

As defined previously, our convex problem is:

$$\text{minimize } \mathrm{f(x)}$$

$$\text{subject to } \mathrm{Ax} = \mathrm{b}$$

Therefore, our Lagrangian is defined as:

$$\mathbf{L}\,(\mathbf{x}, \mathbf{y}) = \mathbf{f}\,(\mathbf{x}) + \boldsymbol{y}^{\boldsymbol{T}}\,(\mathbf{Ax} - \mathbf{b})$$

where y is the dual variable (Lagrange multiplier);

and the dual problem is then defined as:

$$\text{maximize } \{\mathbf{g}(\mathbf{y}) = \boldsymbol{inf}_{(x)}\boldsymbol{L}(\boldsymbol{x}, \boldsymbol{y})\}$$

4.) <u>Dual Ascents Method:</u>

The dual ascent method is the exact opposite of the gradient descent method. We solve the dual problem using this.

- Perform an x – minimization step:
$$\boldsymbol{x}^{\boldsymbol{k}+\mathbf{1}} = \boldsymbol{argmin}\ \boldsymbol{L}(\boldsymbol{x}, \boldsymbol{y}^{\boldsymbol{k}})$$
- Then, calculate the $\nabla g(y) = Ax^{k+1} - b$ and then we update the dual variable using gradient descent:
$$\boldsymbol{y}^{\boldsymbol{k}+\mathbf{1}} = \boldsymbol{y}^{\boldsymbol{k}} + \boldsymbol{\alpha}^{\boldsymbol{k}}(\boldsymbol{A}\boldsymbol{x}^{\boldsymbol{k}+\mathbf{1}} - \boldsymbol{b})$$

5.) <u>Dual Decomposition:</u>

Suppose $\mathrm{f(x)} = f_1(x_1) + \cdots + f_N(x_N),\ x = (x_1, \ldots\ldots, x_N)$.

$$\text{maximize} \quad \{\mathbf{f(x)} = (\boldsymbol{x} + \boldsymbol{a})^{\boldsymbol{n}} = \sum_{i=1}^{N} \mathbf{f}_i(\boldsymbol{x}_i)$$

$$\text{subject to } \sum_{i=1}^{N} \mathbf{A}_i \boldsymbol{x}_i = \boldsymbol{b}$$

And also, the lagrange is separable as,

$$\mathbf{L}\,(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} \boldsymbol{L}_i(\boldsymbol{x}_{i,}\, \boldsymbol{y}) = \sum_{i=1}^{N} \mathbf{f}_i(\boldsymbol{x}_i) + \sum_{i=1}^{N} \boldsymbol{y}^{\boldsymbol{T}} * \mathbf{A}_i(\boldsymbol{x}_i) - \boldsymbol{y}^{\boldsymbol{T}} * \mathbf{b}$$

This is done in order to split the function for our ease, so that we'll be able to deduce each and every function more accurately and also to reduce the computational work.

After we then combine them all in to the original function to get back the original function. We see that in the next pre – requisite.

6.) <u>Method of Multipliers:</u>

This one uses the so called augmented Lagrangian (with a penalty parameter '**ρ'** **> 0**).

$$\mathbf{L}\,(\mathbf{x}, \mathbf{y}) = \mathbf{f(x)} + \boldsymbol{y}^{\boldsymbol{T}}\,(\mathbf{Ax} - \mathbf{b}) + \frac{\boldsymbol{\rho}}{\mathbf{2}} ||(\mathbf{Ax} - \mathbf{b})||_2^2$$

It can also be viewed as:

$$\mathbf{f(x)} = \frac{\boldsymbol{\rho}}{\mathbf{2}} ||(\mathbf{Ax} - \mathbf{b})||_2^2$$

subject to (Ax = b)

which, is equivalent to our original problem in dual problem.

Now, after forming this augmented Lagrangian function, we see that the decomposability property of the objective function is lost and we also can't bring the original function back to form with the penalty function included in there.

The solution is **ADMM**!

Alternating Direction Method of Multipliers:

The ADMM is the alternative for the method of multipliers and is a very powerful algorithm with good robustness of method of multipliers and can even support which can support decomposition even though the penalty function is included.

The main goal of ADMM is to bring back the decomposability property of our augmented lagrangian objective function by adding an extra function to it. The key to use ADMM is the separable terms in the given function. This allows the whole problem to be solved by iterating over the two sub-problems. A general notation of ADMM can be visualized as follows:

$$\textbf{Minimize} \quad \mathbf{f(x) + g(z)}$$

$$\textbf{Subject to} \quad \mathbf{Ax + Bz - c = 0}$$

We see that an extra function g(z) has been added to our function to improve the factorization.

As in the method of multipliers, we can construct our augmented lagrangian function of:

$$L_\rho \,(\mathbf{x, z}, y) = \mathbf{f\,(x) + g\,(z)} + y^T \mathbf{(Ax + Bz - c) + (\rho/2)||Ax + Bz - c||_2^2}$$

Now, ADMM iterates through every variable in the following ways, similar to that in dual ascent:

$$\boldsymbol{x^{k+1} = argmin\ L_\rho\left(x, z^k, y^k\right)}$$

$$\boldsymbol{z^{k+1} = argmin\ L_\rho\left(x^k,\ z, y^k\right)}$$

$$\boldsymbol{y^{k+1} =\ \ y^k + \rho(\mathbf{A}x^{k+1} + \mathbf{B}z^{k+1} - \mathbf{c})}$$

Here, we consider the Lagrange multiplier as y^T, where transpose has been considered, to satisfy the dimensions of the resultant matrix.

We now try to connect our Total Variation function and the above algorithm to minimize it using ADMM.

TOTAL VARIAITION WITH ADMM

To perform ADMM on the Total variation function, we need to write our problem closer to it. We also create a variable '*z*' since we only have one variable (*x*) in our problem of total variation denoising function.

This would easily lead us to a constraint $Fx - z = 0$, where F is the matrix form of our TV operator. And now, we write our function in the ADMM form as follows:

$$\textbf{minimize} \quad \frac{1}{2}||\mathbf{x}-\mathbf{b}||_2^2 + \lambda \mathbf{z}$$
$$\textbf{subject to} \quad \boldsymbol{Fx - z = 0}$$

We can compare the functions of conventional ADMM, with our newly above-formulated equations.

From the constrained equations discussed before, we can explicitly show how these problems line up:

f(x) $= \frac{1}{2}||\mathbf{x}-\mathbf{b}||_2^2$

g(z) $= \lambda \mathrm{z}$, **(where we can visualize '*z*' as our original TV function).**

A = F and B = - I

Therefore, we now define our own Lagrangian function for the above form:

$$L_\rho(x, z, y) = \frac{1}{2}||x - b||_2^2 + \lambda z + y^T(Fx - z - c) + \frac{\rho}{2}||Fx - z - c||_2^2$$

The above formulation can help us transform the **constrained** optimization problem in to the **unconstrained** optimization problem.

Since the objectives are separable, we can simplify the update steps a bit further:

$$x^{k+1} = argmin\left(f(x) + \frac{\rho}{2}||x - v||_2^2 \right)$$

$$\text{where, } v = -Bz + c + u,$$

where, 'u' is a scaled dual variable, $\left(u = \frac{1}{\rho} y^k \right)$

(Similarly, we do for all of the variables in the Lagrangian function)

Optimization problems can be formalized and subcategorized. Generally, if an optimization problem is linear it can be solved, if it is "convex" it can very often be solved. Least-squares fitting is a linear problem. TV denoising is a convex problem.

We have now split our function into two subproblems so that it much easier for us to solve our original problem. We also see that with each update we're only minimizing concerning one variable, as since our objective function is separable, this greatly simplifies the problem.

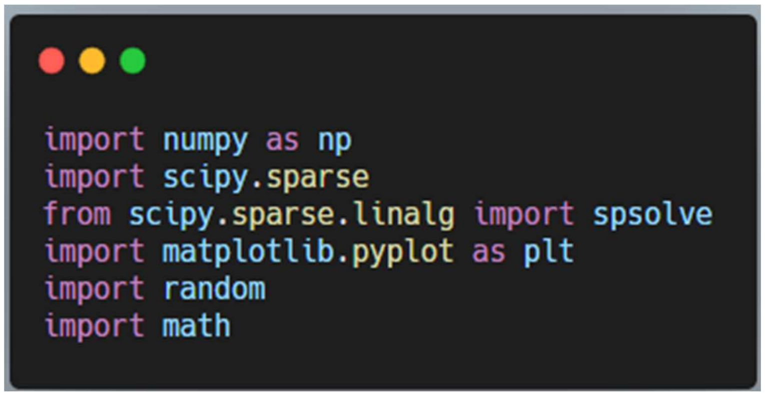
Notice that ADMM has helped us to decompose the function, even though we add a penalty function for violating the constraints in the beginning!

IMPLEMENTATION IN PYTHON

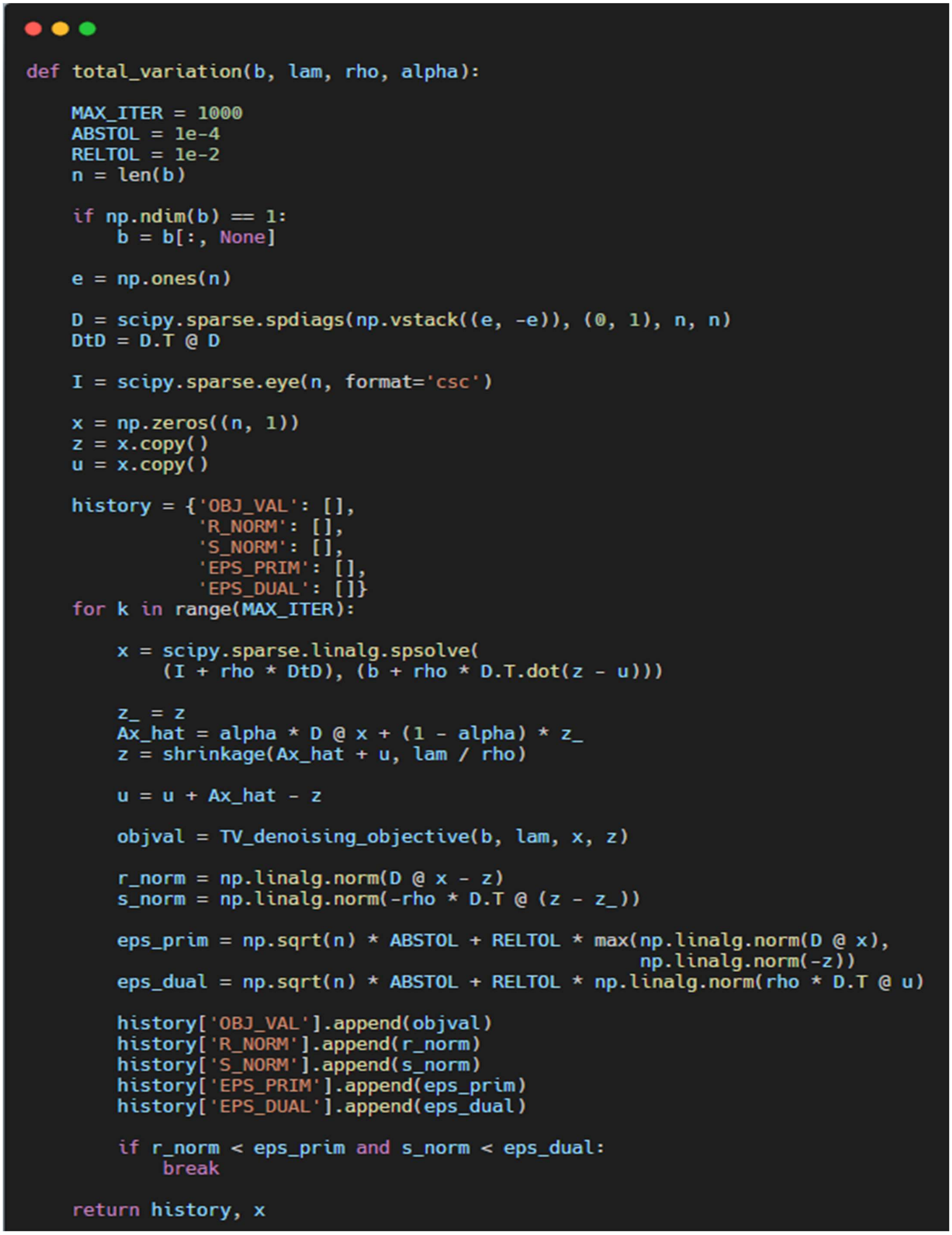
- At first, we import some important dependencies such as **SciPy, NumPy, Matplotlib.**

- Next, we defined a function that solves the total variation problem with the help of our ADMM function which returns the minimized values.
- For solving the raw total variation, we have defined another function called Total variation objective so that it returns the solution of the total variation function.
- We have another function called 'Shrinkage', this function is used for soft-thresholding (moves all values of its input toward 0).
- In the total variation, the l_2 norm should be minimized. So, we are using this shrinkage function to compress the value.
- Next, we will generate some random data on which we are going to apply our total variation function which we have defined using ADMM.

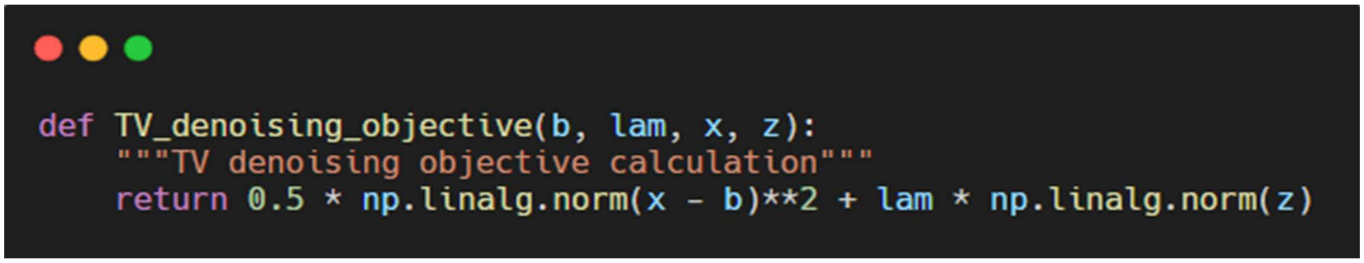
To get a better understanding of the above, go through the below code of our Total Variation Minimization problem implemented in python:



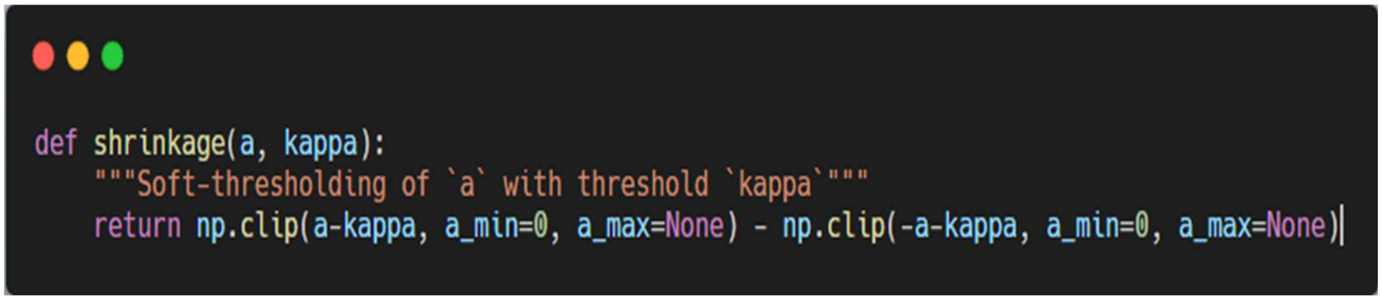
```
import numpy as np
import scipy.sparse
from scipy.sparse.linalg import spsolve
import matplotlib.pyplot as plt
import random
import math
```



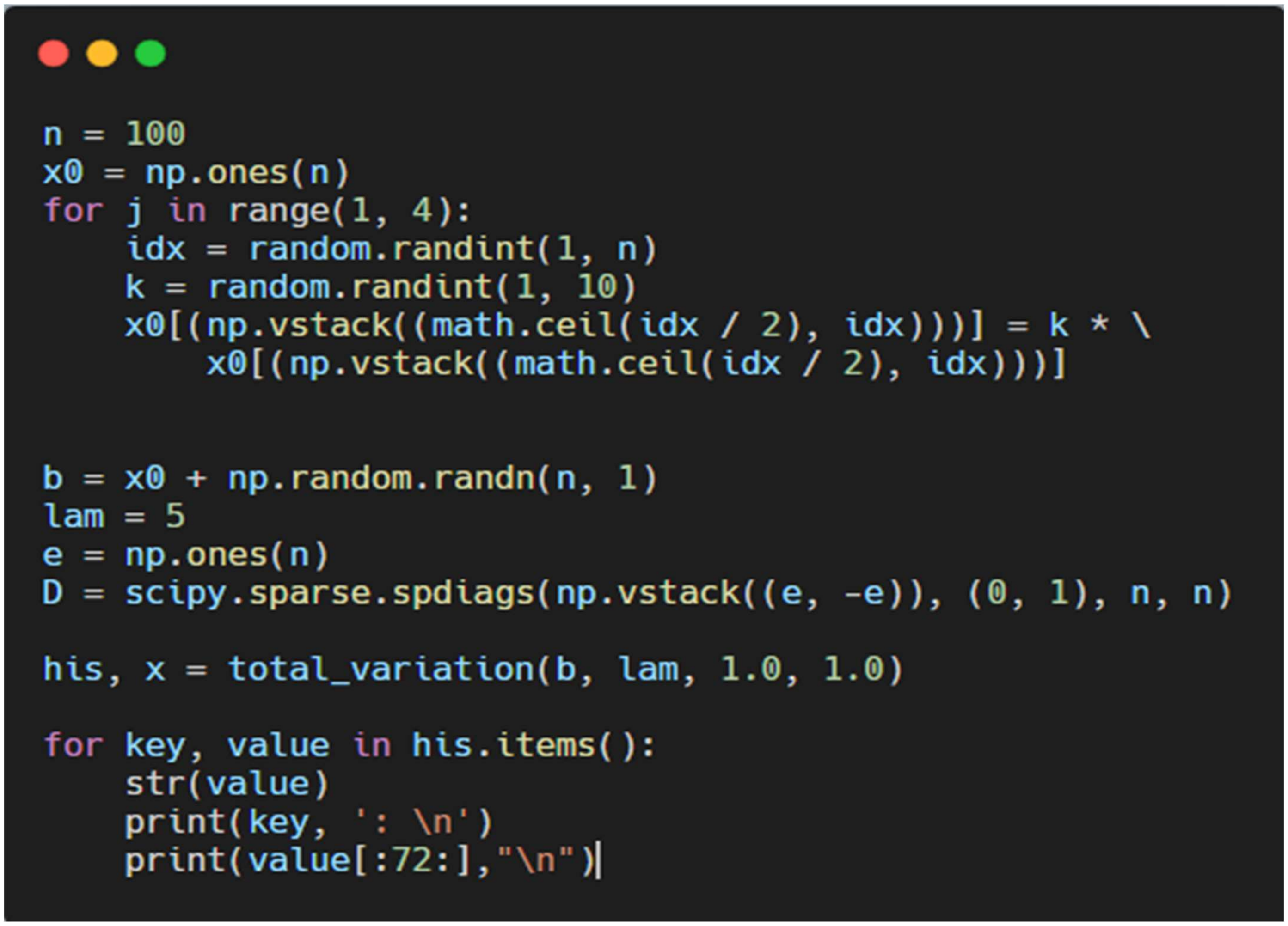
```
def total_variation(b, lam, rho, alpha):

    MAX_ITER = 1000
    ABSTOL = 1e-4
    RELTOL = 1e-2
    n = len(b)

    if np.ndim(b) == 1:
        b = b[:, None]

    e = np.ones(n)

    D = scipy.sparse.spdiags(np.vstack((e, -e)), (0, 1), n, n)
    DtD = D.T @ D

    I = scipy.sparse.eye(n, format='csc')

    x = np.zeros((n, 1))
    z = x.copy()
    u = x.copy()

    history = {'OBJ_VAL': [],
               'R_NORM': [],
               'S_NORM': [],
               'EPS_PRIM': [],
               'EPS_DUAL': []}
    for k in range(MAX_ITER):

        x = scipy.sparse.linalg.spsolve(
            (I + rho * DtD), (b + rho * D.T.dot(z - u)))

        z_ = z
        Ax_hat = alpha * D @ x + (1 - alpha) * z_
        z = shrinkage(Ax_hat + u, lam / rho)

        u = u + Ax_hat - z

        objval = TV_denoising_objective(b, lam, x, z)

        r_norm = np.linalg.norm(D @ x - z)
        s_norm = np.linalg.norm(-rho * D.T @ (z - z_))

        eps_prim = np.sqrt(n) * ABSTOL + RELTOL * max(np.linalg.norm(D @ x),
                                                      np.linalg.norm(-z))
        eps_dual = np.sqrt(n) * ABSTOL + RELTOL * np.linalg.norm(rho * D.T @ u)

        history['OBJ_VAL'].append(objval)
        history['R_NORM'].append(r_norm)
        history['S_NORM'].append(s_norm)
        history['EPS_PRIM'].append(eps_prim)
        history['EPS_DUAL'].append(eps_dual)

        if r_norm < eps_prim and s_norm < eps_dual:
            break

    return history, x
```



```python
def TV_denoising_objective(b, lam, x, z):
    """TV denoising objective calculation"""
    return 0.5 * np.linalg.norm(x - b)**2 + lam * np.linalg.norm(z)
```

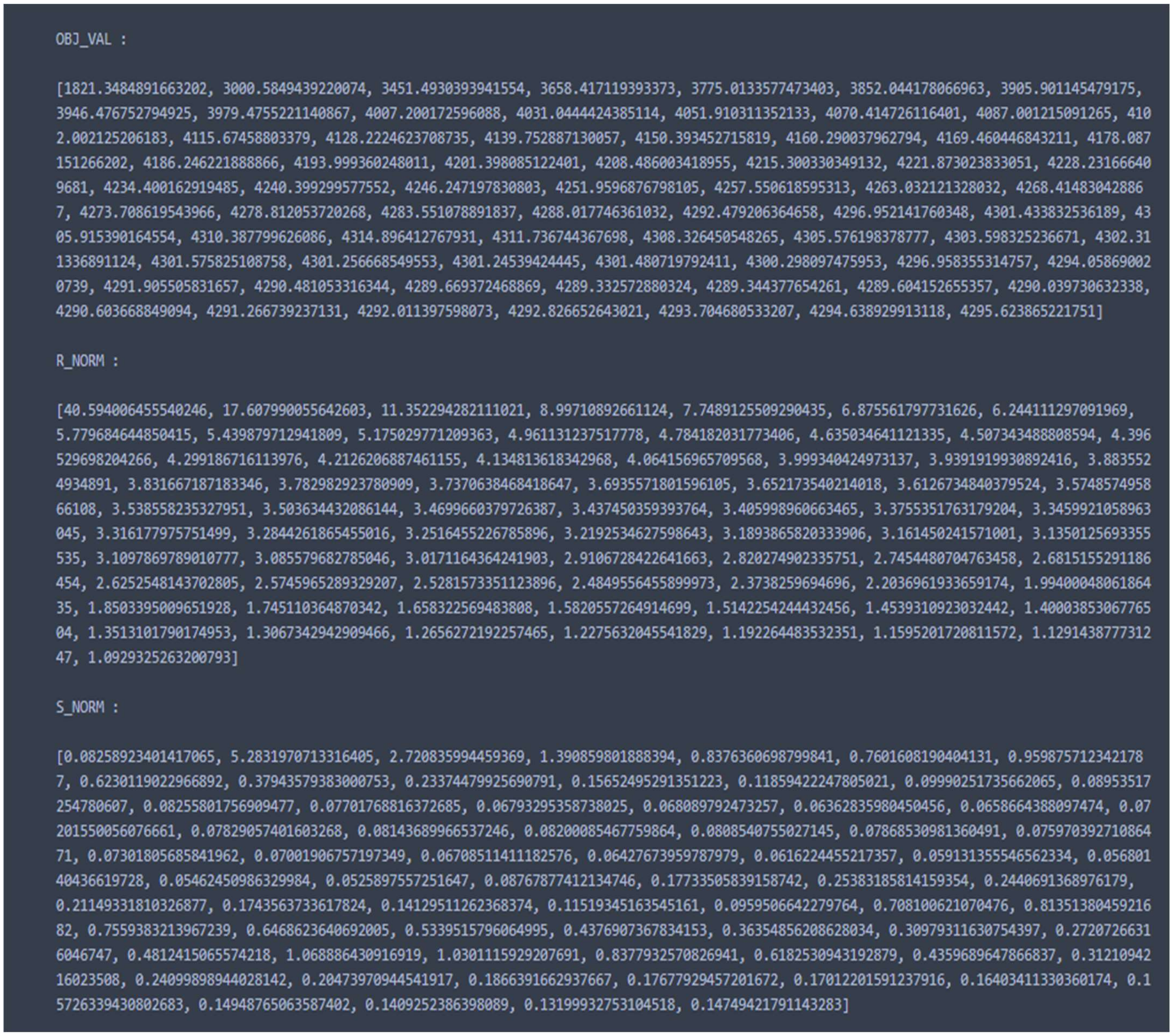


```python
def shrinkage(a, kappa):
    """Soft-thresholding of `a` with threshold `kappa`"""
    return np.clip(a-kappa, a_min=0, a_max=None) - np.clip(-a-kappa, a_min=0, a_max=None)
```

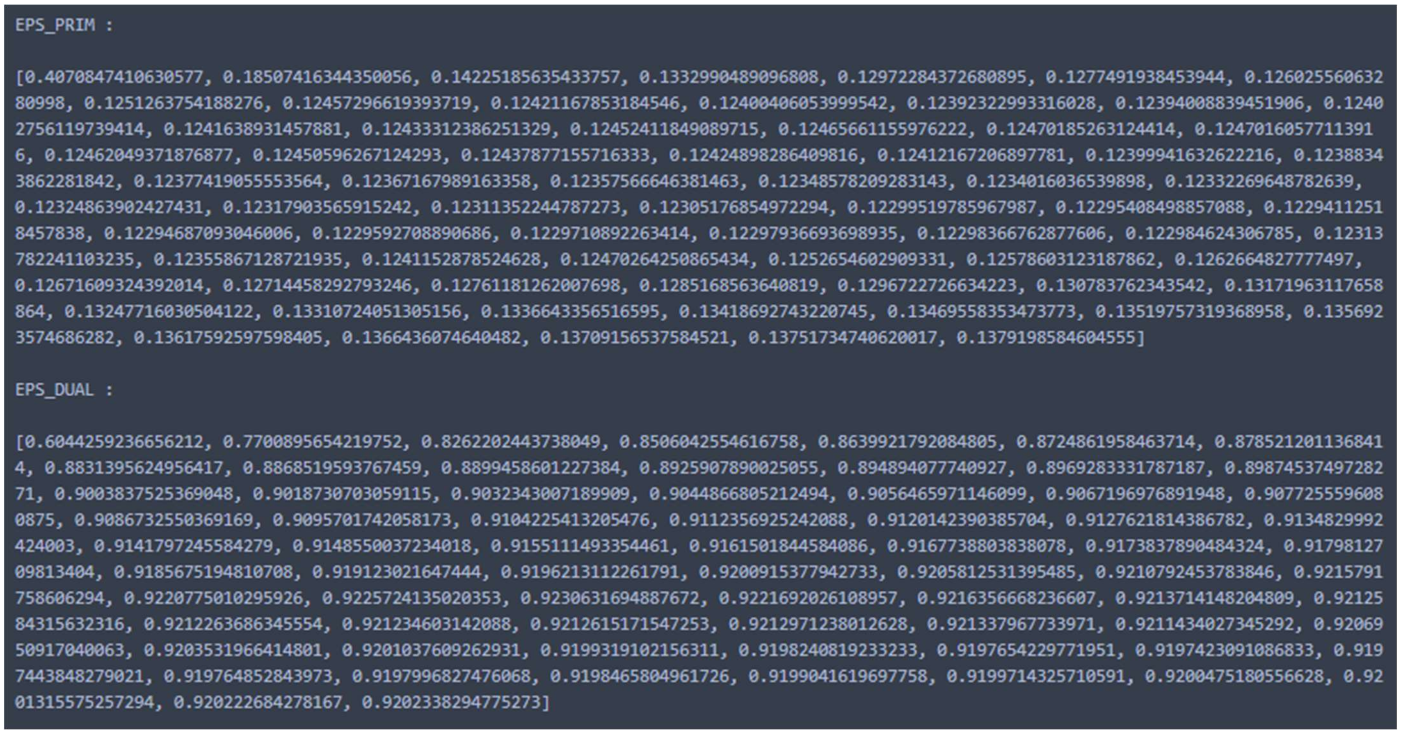


```python
n = 100
x0 = np.ones(n)
for j in range(1, 4):
    idx = random.randint(1, n)
    k = random.randint(1, 10)
    x0[(np.vstack((math.ceil(idx / 2), idx)))] = k * \
        x0[(np.vstack((math.ceil(idx / 2), idx)))]


b = x0 + np.random.randn(n, 1)
lam = 5
e = np.ones(n)
D = scipy.sparse.spdiags(np.vstack((e, -e)), (0, 1), n, n)

his, x = total_variation(b, lam, 1.0, 1.0)

for key, value in his.items():
    str(value)
    print(key, ': \n')
    print(value[:72:],"\n")
```

Output:



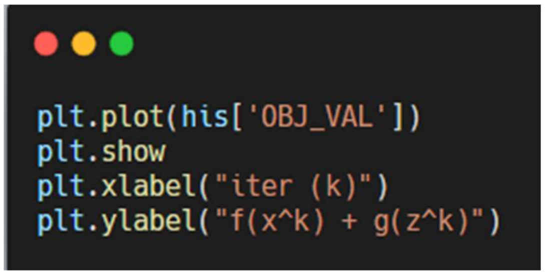
```
OBJ_VAL :

[1821.3484891663202, 3000.5849439220074, 3451.4930393941554, 3658.417119393373, 3775.0133577473403, 3852.044178066963, 3905.901145479175,
3946.476752794925, 3979.4755221140867, 4007.200172596088, 4031.0444424385114, 4051.910311352133, 4070.414726116401, 4087.001215091265, 410
2.002125206183, 4115.67458803379, 4128.2224623708735, 4139.752887130057, 4150.393452715819, 4160.290037962794, 4169.460446843211, 4178.087
151266202, 4186.246221888866, 4193.999360248011, 4201.398085122401, 4208.486003418955, 4215.300330349132, 4221.873023833051, 4228.23166640
9681, 4234.400162919485, 4240.399299577552, 4246.247197830803, 4251.9596876798105, 4257.550618595313, 4263.032121328032, 4268.41483042886
7, 4273.708619543966, 4278.812053720268, 4283.551078891837, 4288.017746361032, 4292.479206364658, 4296.952141760348, 4301.433832536189, 43
05.915390164554, 4310.387799626086, 4314.896412767931, 4311.736744367698, 4308.326450548265, 4305.576198378777, 4303.598325236671, 4302.31
1336891124, 4301.575825108758, 4301.256668549553, 4301.24539424445, 4301.480719792411, 4300.298097475953, 4296.958355314757, 4294.05869002
0739, 4291.905505831657, 4290.481053316344, 4289.669372468869, 4289.332572880324, 4289.344377654261, 4289.604152655357, 4290.039730632338,
4290.603668849094, 4291.266739237131, 4292.011397598073, 4292.826652643021, 4293.704680533207, 4294.638929913118, 4295.623865221751]

R_NORM :

[40.594006455540246, 17.607990055642603, 11.352294282111021, 8.99710892661124, 7.7489125509290435, 6.875561797731626, 6.244111297091969,
5.779684644850415, 5.439879712941809, 5.175029771209363, 4.961131237517778, 4.784182031773406, 4.635034641121335, 4.507343488808594, 4.396
529698204266, 4.299186716113976, 4.2126206887461155, 4.134813618342968, 4.064156965709568, 3.999340424973137, 3.9391919930892416, 3.883552
4934891, 3.831667187183346, 3.782982923780909, 3.7370638468418647, 3.6935571801596105, 3.652173540214018, 3.6126734840379524, 3.5748574958
66108, 3.538558235327951, 3.503634432086144, 3.4699660379726387, 3.437450359393764, 3.405998960663465, 3.3755351763179204, 3.3459921058963
045, 3.316177975751499, 3.2844261865455016, 3.2516455226785896, 3.2192534627598643, 3.1893865820333906, 3.161450241571001, 3.1350125693355
535, 3.1097869789010777, 3.085579682785046, 3.0171164364241903, 2.9106728422641663, 2.820274902335751, 2.7454480704763458, 2.6815155291186
454, 2.6252548143702805, 2.5745965289329207, 2.5281573351123896, 2.4849556455899973, 2.3738259694696, 2.2036961933659174, 1.99400048061864
35, 1.8503395009651928, 1.745110364870342, 1.658322569483808, 1.5820557264914699, 1.5142254244432456, 1.4539310923032442, 1.40003853067765
04, 1.3513101790174953, 1.3067342942909466, 1.2656272192257465, 1.2275632045541829, 1.192264483532351, 1.1595201720811572, 1.1291438777312
47, 1.0929325263200793]

S_NORM :

[0.08258923401417065, 5.2831970713316405, 2.720835994459369, 1.390859801888394, 0.8376360698799841, 0.7601608190404131, 0.959875712342178
7, 0.6230119022966892, 0.37943579383000753, 0.23374479925690791, 0.15652495291351223, 0.11859422247805021, 0.09990251735662065, 0.08953517
254780607, 0.08255801756909477, 0.07701768816372685, 0.06793295358738025, 0.068089792473257, 0.06362835980450456, 0.0658664388097474, 0.07
201550056076661, 0.07829057401603268, 0.08143689966537246, 0.08200085467759864, 0.0808540755027145, 0.07868530981360491, 0.075970392710864
71, 0.07301805685841962, 0.07001906757197349, 0.06708511411182576, 0.06427673959787979, 0.0616224455217357, 0.059131355546562334, 0.056801
40436619728, 0.05462450986329984, 0.0525897557251647, 0.08767877412134746, 0.17733505839158742, 0.25383185814159354, 0.2440691368976179,
0.21149331810326877, 0.1743563733617824, 0.14129511262368374, 0.11519345163545161, 0.0959506642279764, 0.708100621070476, 0.81351380459216
82, 0.7559383213967239, 0.6468623640692005, 0.5339515796064995, 0.4376907367834153, 0.36354856208628034, 0.30979311630754397, 0.2720726631
6046747, 0.4812415065574218, 1.068886430916919, 1.0301115929207691, 0.8377932570826941, 0.6182530943192879, 0.4359689647866837, 0.31210942
16023508, 0.24099898944028142, 0.20473970944541917, 0.1866391662937667, 0.17677929457201672, 0.17012201591237916, 0.16403411330360174, 0.1
5726339430802683, 0.14948765063587402, 0.1409252386398089, 0.13199932753104518, 0.14749421791143283]
```



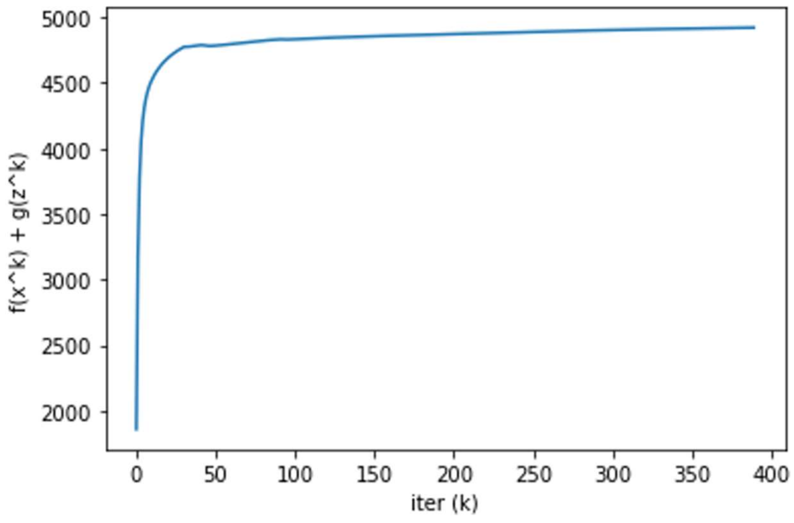
```
EPS_PRIM :

[0.4070847410630577, 0.18507416344350056, 0.14225185635433757, 0.1332990489096808, 0.12972284372680895, 0.1277491938453944, 0.12602556063280998, 0.1251263754188276, 0.12457296619393719, 0.12421167853184546, 0.12400406053999542, 0.12392322993316028, 0.12394008839451906, 0.12402756119739414, 0.1241638931457881, 0.12433312386251329, 0.12452411849089715, 0.12465661155976222, 0.12470185263124414, 0.1247016057711391 6, 0.12462049371876877, 0.12450596267124293, 0.12437877155716333, 0.12424898286409816, 0.12412167206897781, 0.12399941632622216, 0.12388343862281842, 0.12377419055553564, 0.12367167989163358, 0.12357566646381463, 0.12348578209283143, 0.1234016036539898, 0.12332269648782639,
0.12324863902427431, 0.12317903565915242, 0.12311352244787273, 0.12305176854972294, 0.12299519785967987, 0.12295408498857088, 0.12294112518457838, 0.12294687093046006, 0.1229592708890686, 0.1229710892263414, 0.12297936693698935, 0.12298366762877606, 0.122984624306785, 0.12313782241103235, 0.12355867128721935, 0.1241152878524628, 0.12470264250865434, 0.1252654602909331, 0.12578603123187862, 0.1262664827777497,
0.12671609324392014, 0.12714458292793246, 0.12761181262007698, 0.1285168563640819, 0.1296722726634223, 0.130783762343542, 0.13171963117658864, 0.13247716030504122, 0.13310724051305156, 0.1336643356516595, 0.13418692743220745, 0.13469558353473773, 0.13519757319368958, 0.1356923574686282, 0.13617592597598405, 0.1366436074640482, 0.13709156537584521, 0.13751734740620017, 0.1379198584604555]

EPS_DUAL :

[0.6044259236656212, 0.7700895654219752, 0.8262202443738049, 0.8506042554616758, 0.8639921792084805, 0.8724861958463714, 0.8785212011368414, 0.8831395624956417, 0.8868519593767459, 0.8899458601227384, 0.8925907890025055, 0.894894077740927, 0.8969283331787187, 0.8987453749728271, 0.9003837525369048, 0.9018730703059115, 0.9032343007189909, 0.9044866805212494, 0.9056465971146099, 0.9067196976891948, 0.9077255596080875, 0.9086732550369169, 0.9095701742058173, 0.9104225413205476, 0.9112356925242088, 0.9120142390385704, 0.9127621814386782, 0.9134829992424003, 0.9141797245584279, 0.9148550037234018, 0.9155111493354461, 0.9161501844584086, 0.9167738803838078, 0.9173837890484324, 0.9179812709813404, 0.9185675194810708, 0.919123021647444, 0.9196213112261791, 0.9200915377942733, 0.9205812531395485, 0.9210792453783846, 0.9215791758606294, 0.9220775010295926, 0.9225724135020353, 0.9230631694887672, 0.9221692026108957, 0.9216356668236607, 0.9213714148204809, 0.9212584315632316, 0.9212263686345554, 0.921234603142088, 0.9212615171547253, 0.9212971238012628, 0.921337967733971, 0.9211434027345292, 0.9206950917040063, 0.9203531966414801, 0.9201037609262931, 0.9199319102156311, 0.9198240819233233, 0.9197654229771951, 0.9197423091086833, 0.9197443848279021, 0.919764852843973, 0.9197996827476068, 0.9198465804961726, 0.9199041619697758, 0.9199714325710591, 0.9200475180556628, 0.9201315575257294, 0.920222684278167, 0.9202338294775273]
```

Let us now visualize the optimality conditions and various residual norms using graphs:

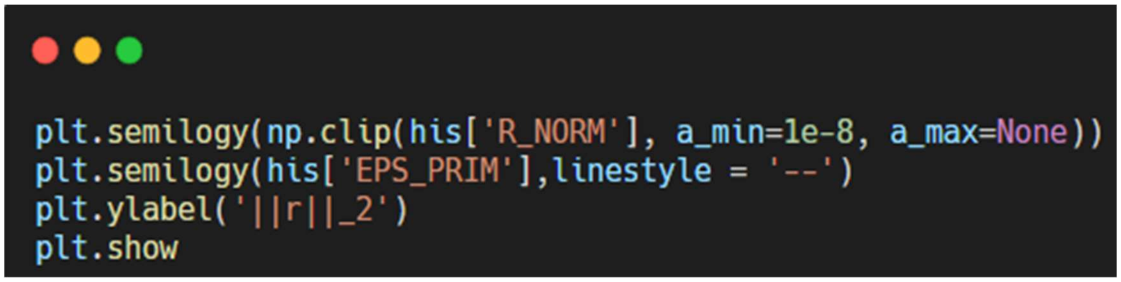
a.)



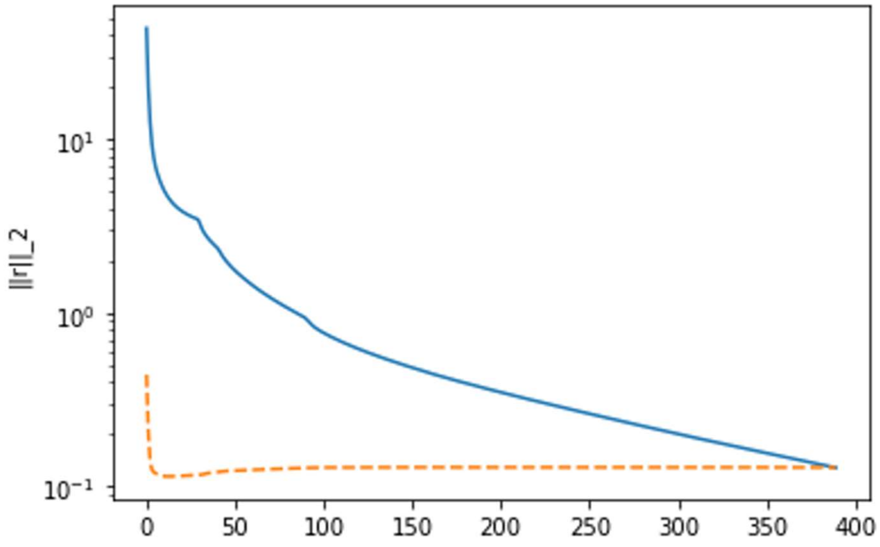
```
plt.plot(his['OBJ_VAL'])
plt.show
plt.xlabel("iter (k)")
plt.ylabel("f(x^k) + g(z^k)")
```



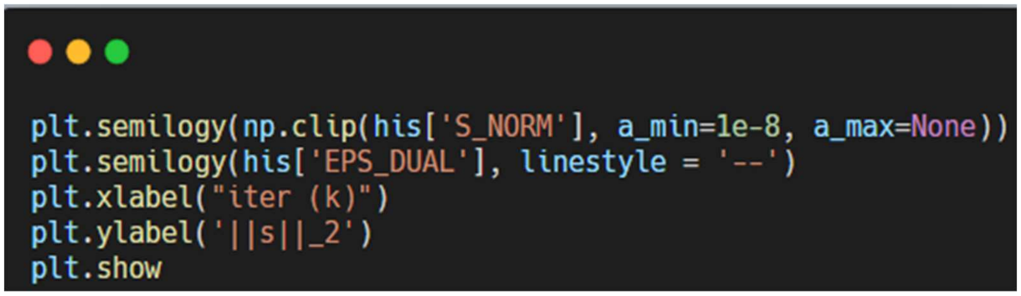
b.)



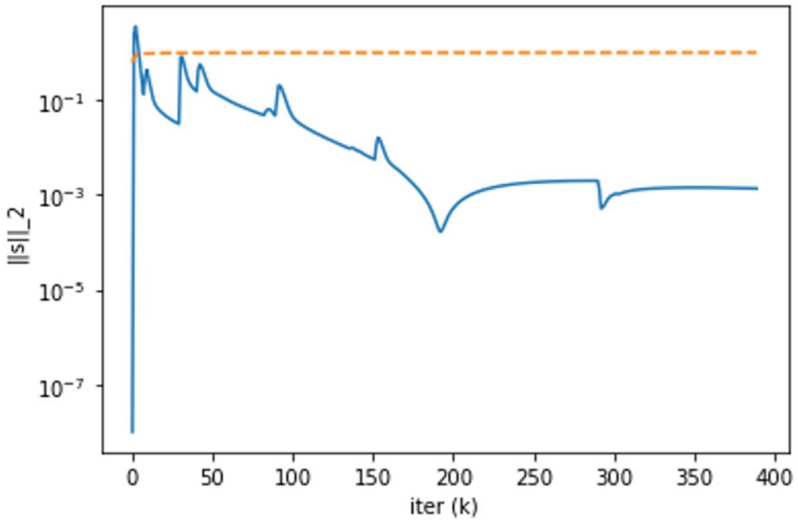
```
plt.semilogy(np.clip(his['R_NORM'], a_min=1e-8, a_max=None))
plt.semilogy(his['EPS_PRIM'],linestyle = '--')
plt.ylabel('||r||_2')
plt.show
```



c.)



```
plt.semilogy(np.clip(his['S_NORM'], a_min=1e-8, a_max=None))
plt.semilogy(his['EPS_DUAL'], linestyle = '--')
plt.xlabel("iter (k)")
plt.ylabel('||s||_2')
plt.show
```



APPLICATIONS OF TOTAL VARIATION MINIMIZATION

- As discussed previously, total variation minimization is used in Digital image processing, it helps in removing noise from the image:

    - The famous image of the first Blackhole was denoised by using this Total Variation minimization process, developed by Rudin-Osher-Fatemi.

- Total variation minimization also has its branches in fields of communication technology, such as removing the noise from the speech and video signals.

- The restoration of images is obtained with regularization methods involving TV operators and other algorithms like RLS smoothes out the edges in the restored image. This can be alleviated by methods relying on L1-norm regularization like TV regularization keeps the edges in the estimated image. The purpose is to recuperate a real image from an image distorted by several simultaneous phenomena such as blur and noise using the TV norm.

<u>CONCLUSION</u>

To the best of our knowledge, we have provided a complete summary of what **Total Variation Minimization** is, and we have also dealt with one of the most popular and powerful minimization techniques, which is the ADMM algorithm itself.

We have seen some important prerequisites to get a better apprehension about ADMM. We have also seen the implementation part of Total Variation Denoising using ADMM in python and successfully produced an output that minimizes the values of the random problem data generated and later visualized it, by plotting the variation of the function over all the iterations.

Although for some noise models, particular algorithms have better reconstruction performance, such as the MM algorithm (Majorization – Minimization), we expect that in the coming years' TV will improve its current reconstruction performance and hopefully become competitive with state-of-the-art alternative denoising methods.

Finally, I would conclude by saying that we as a team understood the importance of dealing with noisy data in images and speech signals, and these all put together to form the foundations for AI models.

--***--**