Zainab Al-Suwaykit
Daniel González Esparza

AMCS CS 212
Numerical Optimization
Assignment 6

1. Image Reconstruction.

(1)

$$(U_{ij}-U_{i-1,j})^2 + (U_{ij}-U_{i,j-1})^2$$
$$= U_{ij}^2 - 2U_{ij}U_{i-1,j} + (U_{i-1,j})^2 + U_{ij}^2 - 2U_{ij}U_{i,j-1} + (U_{i,j-1})^2$$
$$= 2U_{ij}^2 + (U_{i-1,j})^2 + (U_{i,j-1})^2 - 2(U_{ij})(U_{i-1,j}) - 2(U_{ij})(U_{i,j-1})$$

$$C = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

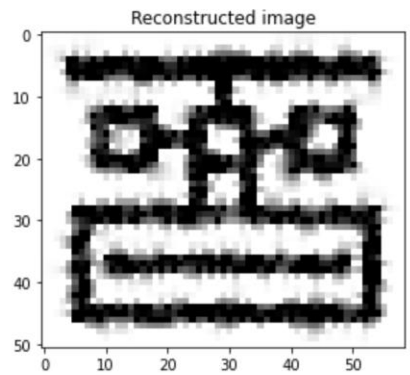
$$\text{minimize} \quad \sum_{i=2}^{m}\sum_{j=2}^{n} \begin{bmatrix} U_{ij} & U_{i-1,j} & U_{i,j-1} \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} U_{ij} \\ U_{i-1,j} \\ U_{i,j-1} \end{bmatrix}$$

subject to: $AX = b$

where A, b is known pixeles values.

```python
import cvxpy as cp
from cvxpy.atoms.atom import Atom
UO = np.array(U0)
UC = np.array(U1)
A = np.zeros((m, n))

for i in range(1,m):
    for j in range(1,n):
        if UO[i, j] == UC[i, j]:
           A[i, j] = 1
# Define and solve the CVXPY problem.
U = cp.Variable(shape=(m,n))
exp = cp.sum((U[1:,:]-U[:-1,:])**2) + cp.sum((U[:,1:]-U[:,:-1])**2)
constraints = [cp.multiply(U,A) ==  cp.multiply(UC,A)]
prob = cp.Problem(cp.Minimize(exp),constraints)
prob.solve(verbose=True)
print("optimal objective value: {}".format(U.value))
```

Zainab Al-Suwaykit
Daniel González Esparza



2. Contact problem in 1D.

```
import cvxpy as cp
import numpy as np


print('-------------------FIRST POINT----------------------')
print('----------------------------------------------------')
x = cp.Variable(2)
k1=1
k2=10
k3=2
l=1
w=0.2

A = np.array(([-1,0],[1,-1],[0,1]))
KK= np.array(([(k1+k2),-k2],[-k2,(k3+k2)]))
KL=np.array(([0,-l*k3]))
b = np.array([-w/2,-w,l-w/2])

# Construct the problem.

objective = cp.Minimize((1/2)*cp.quad_form(x, KK)+np.transpose(KL) @ x+0.5*k3*(l)**2)
constraints = [A@x <= b]
prob = cp.Problem(objective, constraints)
# The optimal objective val, returned by `prob.solve()`
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print('The optimal value is ',x.value)
# The optimal Lagrange multiplier for first constraint
print('The lagrange multipliers are: ')
print(constraints[0].dual_value)
```

What is the significance of the multipliers in this problem?
The optimal value x is [0.53333333 0.73333333] and the Lagrange multipliers are:[0 1.46666667 0.], We can realize that the first and third constrains are not active, therefore do not modify the solutions in anyway and only depends on the second restriction.

Zainab Al-Suwaykit
Daniel González Esparza

```python
print('-------------------LAST POINT-----------------------')
print('---------------------------------------------------')
x = cp.Variable(4)
k1=1
k2=10
k3=2
c1=2
c2=4
l=1
w=0.2

A = np.array(([-1,0,1/2,0],[1,-1,1/2,1/2],[0,1,0,1/2]))
b = np.array([-w/2,-w,l-w/2])
KK= np.array(([(k1+k2),-k2,(k2-k1)/2,k2/2],[-k2,(k3+k2),-k2/2,(k3-k2)/2],\
              [(k2-k1)/2,-k2/2,c1+(k1+k2)/2,k2/4],[k2/2,(k3-k2)/2,k2/4,c2+(k3+k2)/2]))
KL=np.array(([0,-l*k3,0,-k3*l/2]))

# Construct the problem.
objective = cp.Minimize((1/2)*cp.quad_form(x, KK)+np.transpose(KL) @ x+0.5*k3*(l)**2)
constraints = [A@x <= b]
prob = cp.Problem(objective, constraints)
# The optimal objective val, returned by `prob.solve()`
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print('The optimal value is ',x.value)
# The optimal Lagrange multiplier for first constraint
print('The lagrange multipliers are: ')
print(constraints[0].dual_value)
```

Reformulating the problem, we realized that, as in the first question, the constrain number 1 and 3 are still inactive, what means that the blocks are touching. This make sense because the stiffness of the spring between both blocks are so much larger those in both sides.

The optimal value is [0.47707911 0.75334686 0.09087221 0.06166329], The lagrange multipliers are: [0. 1.568357 0]

3. Transformation of L1-norm minimization.

(3)

$\underset{x}{\text{minimize}} \quad \|Ax - b\|_1$ ← non smooth

$\text{minimize} \quad t$ ← smooth

Subject to $|Ax - b| \leq t$

Zainab Al-Suwaykit
Daniel González Esparza

canonical form.

minimiz t

s.t $|Ax-b| - t + s = 0$

$t \geq 0$

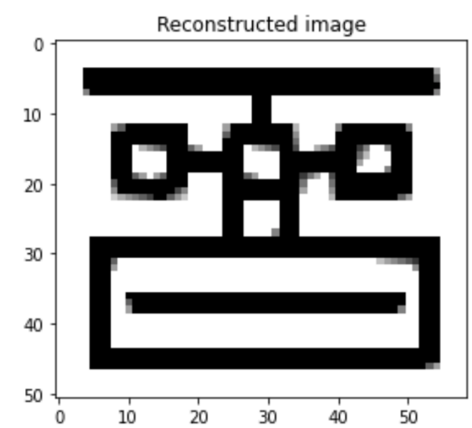
$s \geq 0$

where $t \in R^m$, $t = (t_1, t_2 \dots t_m)^t$

4. Image reconstruction revisited.

```python
#Q4
U1 = cp.Variable(shape=(m,n))
exp = cp.sum(cp.norm1(U1[1:,:]-U1[:-1,:])) \
        + cp.sum(cp.norm1(U1[:,1:]-U1[:,:-1]))
constraints = [cp.multiply(U1,A) ==  cp.multiply(UC,A)]
prob = cp.Problem(cp.Minimize(exp),constraints)
prob.solve(verbose=True)


print("optimal objective value: {}".format(U1.value))
```

```python
import matplotlib.pyplot as plt

plt.imshow(U1.value, cmap='gray')
plt.title('Reconstructed image')
plt.show()
```



From the slide

Zainab Al-Suwaykit
Daniel González Esparza

L_1 objective:

$$\text{minimize} \quad \begin{bmatrix} \mathbf{0}^T & \mathbf{1}^T & \mathbf{1}^T \end{bmatrix} \begin{bmatrix} x \\ u \\ v \end{bmatrix}$$

$$\text{subject to} \quad \begin{bmatrix} B & -I & \\ -B & -I & \\ C & & -I \\ -C & & -I \end{bmatrix} \begin{bmatrix} x \\ u \\ v \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} A & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ v \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

$$\begin{bmatrix} x \\ u \\ v \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

5. Compressed sensing.

Due to the simplicity of the linear system of equations, applying the KKT conditions we could obtain the solution solving the system of equations, for that, first we solve for x values in the linear combinations of the gradients, and finally substitute its values in the constrains for having the Lagrange multipliers.

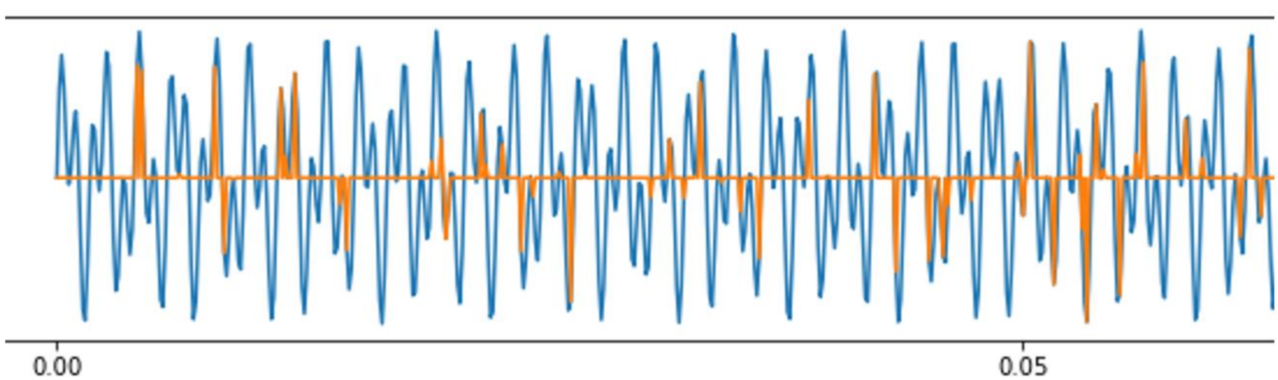
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
import random
import cvxpy as cp


n = 2500
m = 250
fs = 8192                               # sampling frequency
t = np.arange(n) / fs
y = ( np.sin(2*np.pi*521*t) + np.sin(2*np.pi*1233*t) ) /2.0

D = scipy.fftpack.dct(np.eye(n), norm='ortho').transpose()
k = sorted(random.sample(range(n), m))  # pick m random points for the reconstruction
A = D[k, :]
b = y[k]

plt.figure(figsize=(10.24, 2.56))       # change aspect ratio of plot
plt.plot(t[k], b, 'ko', markersize=1)   # plot data to use to reconstruct
plt.show()
x=t[k]



print('-------------Applying KKT conditions-----------------')
print('-----------------------------------------------------')

#Applying the KKT conditions
#Ax=b, therefore x=A\b...
NU=np.linalg.solve(A@np.transpose(A),-2*b)
xx=-np.transpose(A)@NU/2
y1=D@xx
```

Zainab Al-Suwaykit
Daniel González Esparza

Using the CVXPY routine, we found that the maximum error between both solutions was around 10^{-15}. What means there is no significant difference between both solutions.

```python
print('------------------FIRST POINT----------------------')
print('---------------------------------------------------')
x = cp.Variable(2500)

I= np.identity(2500)

# Construct the problem.

objective = cp.Minimize(cp.quad_form(x, I))
constraints = [A@x == b]
prob = cp.Problem(objective, constraints)
# The optimal objective val, returned by `prob.solve()`
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print('The optimal value is ',x.value)
# The optimal Lagrange multiplier for first constraint
print('The lagrange multipliers are: ')
print(constraints[0].dual_value)

Y2=D@x.value

Error=max(np.absolute(Y2-y1))

plt.figure(figsize=(50, 2.56))
plt.plot(t,y)
plt.plot(t,Y2)
```

The comparison between both signals is shown in the next figure. We can observe that, though both signals are similar in some zones, the behavior could not be consider as similar. Therefore, the quadratic form is not a good approximation.

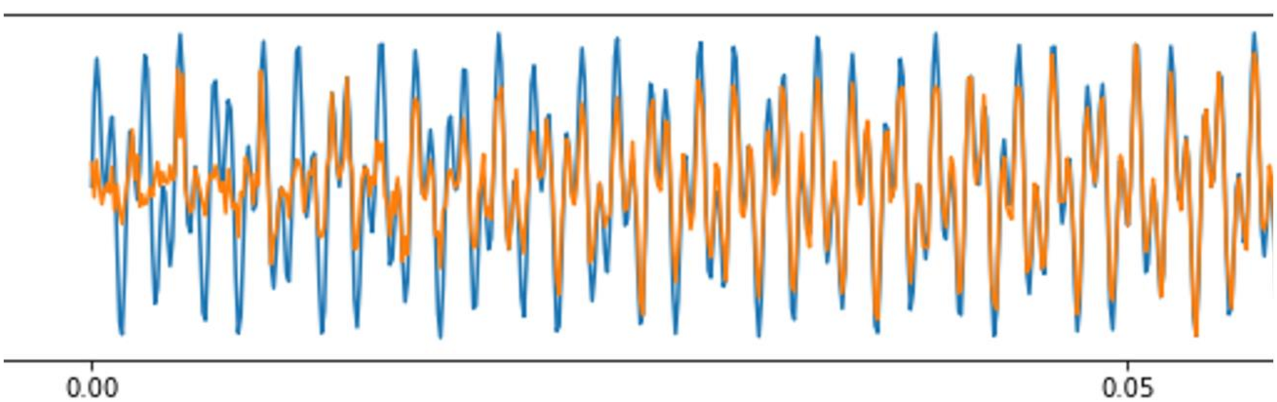


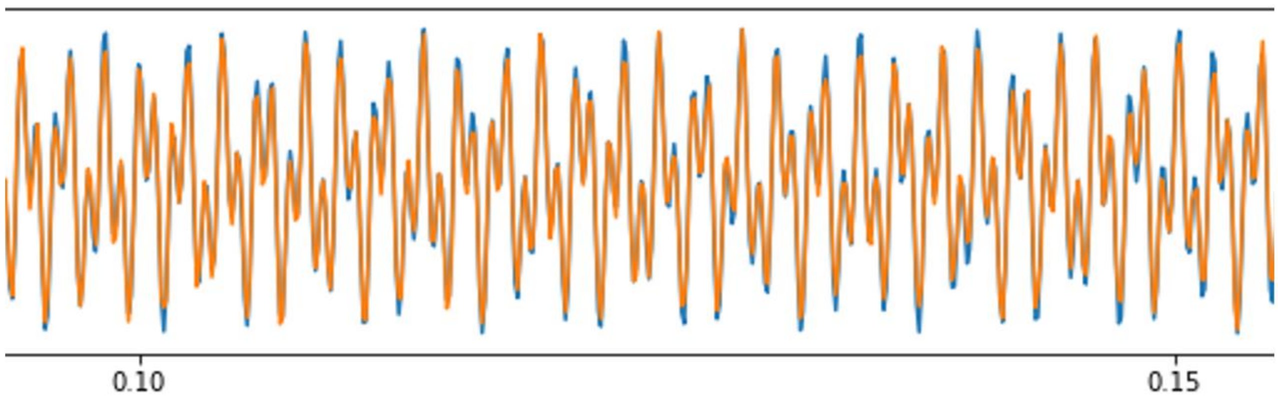
On the other hand, we reformulate the objective function and the results were quite differents.

Zainab Al-Suwaykit
Daniel González Esparza

```
print('-------------------SECOND POINT----------------------')
print('-----------------------------------------------------')
x = cp.Variable(2500)

I= np.identity(2500)

# Construct the problem.

objective = cp.Minimize(cp.norm(x, 1))
constraints = [A@x == b]
prob = cp.Problem(objective, constraints)
# The optimal objective val, returned by `prob.solve()`
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print('The optimal value is ',x.value)
# The optimal Lagrange multiplier for first constraint
print('The lagrange multipliers are: ')
print(constraints[0].dual_value)

Y2=D@x.value

plt.figure(figsize=(50, 2.56))
plt.plot(t,y)
plt.plot(t,Y2)
```

We can observe in the next images that minimizing the error with the new objective function the results were better. In the beginning the behavior is quite similar but the amplitudes not, but in the center is also the same behaviour.




0.10
0.15