Compressive Sensing - 2022 - Computer Lab Correction

claude.petit

December 2022

1 Part 1 - Denoising 1D signals

1.1-1.2.
Code in Python:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate a vector x
n = 128
k = 5
x = np.zeros([1, n])
for i in range(0, k):
    x[0, i] = (i+1)/k
x = np.random.permutation(x.T).T
# Plot x
fig, axs = plt.subplots(1, figsize=(13, 5))
axs.stem(x.T, use_line_collection=True)
axs.set_title('x', fontsize=31)
plt.show()
# Add noise and generate y
y = np.empty([1, n])
y = x + np.random.normal(loc=0.0, scale=0.05,
size=(1, n))
# Plot y
fig, axs = plt.subplots(1, figsize=(13, 5))
axs.stem(y.T, use_line_collection=True)
axs.set_title('y + noise', fontsize=31)
plt.show()
```

1.3. Tykhonov regularization.

Let

$$\phi(z) = \frac{1}{2}||z-y||_2^2 + \frac{\lambda}{2}||z||_2^2 \tag{1}$$

ϕ is a function from $\mathbb{R}^n$ into $\mathbb{R}$. To find the minimum, we have to evaluate the gradient of the function (using the nabla notation). We recall that

$$\nabla||x||_2^2 = 2x \tag{2}$$

Then it comes immediately that

$$\nabla\phi(z) = 0 \tag{3}$$
$$\iff z - y + \lambda z = 0 \tag{4}$$
$$\iff z = \frac{y}{1+\lambda} \tag{5}$$
$$\tag{6}$$

This is indeed a minimum because the Hessian matrix is definite negative (verify this !).
You can also work coordinate by coordinate:

$$\phi(z) = \sum_{i=1}^{n}(z_i - y_i)^2/2 + \lambda\sum_{i=1}^{n} z_i^2/2 \tag{7}$$

The i-th coordinate of the gradient is

$$\frac{\partial\phi}{\partial z_i}(z) = z_i - y_i + \lambda z_i \tag{8}$$

Which is zero and changes of sign if, and only if,

$$z_i = \frac{1}{\lambda+1}y_i \tag{9}$$

and we recover the same solution as previously
So the solution of the minimization problem is:

$$\widehat{x} = \frac{y}{1+\lambda} \tag{10}$$
$$\tag{11}$$

1.4. The solution is not sparse because it is proportional to y which is not sparse also. The solution of the problem (1) isn't sparse because of the presence of the euclidian norm.

Code in Python :

```
# Compute estimate as a function of lambda
def getEst(y, lambda):
    xHat = (1/(1+lambda) * y.T).T
    return(xHat)
lambda = 0.01
xHat = getEst(y, lambda)
L = [0.01, 0.05, 0.1, 0.2]
xHatLst = []
# Plot curves
for i in range(len(L)):
    xHat = getEst(y, L[i])
    xHatLst.append(xHat)
    fig, axs = plt.subplots(1, figsize=(13, 5))
axs.stem(xHatLst[2].T, use_line_collection=True)
axs.set_title('$\hat{x}$ with lambda = 0.1',
fontsize=31)
plt.show()
```

1.5. The following minimization problem is called Basis Pursuit Denoising (BPDN) and is equivalent to the LASSO regression method (in Lagrangian formulation):

$$\widehat{x} = \arg\min_{z\in\mathbb{R}^n} \left(\frac{1}{2}||z-y||_2^2 + \lambda||z||_1\right) \tag{12}$$

First, remark that

$$||z-y||_2^2 = (z-y)'(z-y) = z'z - 2y'z + y'y \tag{13}$$

$y'y$ is a constant, so the minimization problem is the same as finding

$$\arg\min_{z\in\mathbb{R}^n} \left(-y'z + ||z||_2^2/2 + \lambda||z||_1\right) \tag{14}$$

$$\iff \arg\min_{z\in\mathbb{R}^n} \sum_{j=1}^{n} \left(-y_j z_j + z_j^2/2 + \lambda|z_j|\right) \tag{15}$$

which is the sum of n independent optimization problems. We can thus solve them independently in $\mathbb{R}$. Let

$$\phi(z) = -yz + z^2/2 + \lambda|z| \tag{16}$$

ϕ is function from $\mathbb{R}$ to $\mathbb{R}$ of class C^∞ except in 0 where it is non continuous. We split the problem in two cases :

- if $y > 0$. Then $z > 0$ (why ?). In that case,

$$\phi'(z) = 0 \iff z = y - \lambda \tag{17}$$

Due to $z > 0$, this is possible only if $y > \lambda$. If not, the gradient doesn't cancel and the minimum is reached only if $z = 0$; indeed, $\phi(z) = z^2/2 + (\lambda - y)z$ for $z \geq 0$ if minimum for $z = 0$.

- if $y < 0$. Then $z < 0$. As previously, $\phi'(z) = 0$ if, and only if $z = y + \lambda$. Due to $z < 0$ this happens only if $y < -\lambda$ otherwise the minimum is reached in $z = 0$.

We can now define the solution by defining the function "soft thresholding" as:

$$\sigma_\lambda(y) = \begin{cases} y+\lambda & \text{if } y \leq -\lambda \\ 0 & \text{if } -\lambda \leq y \leq \lambda \\ y-\lambda & \text{if } y \geq \lambda \end{cases} \tag{18}$$

for $y \in \mathbb{R}$. We extend this notation to $\mathbb{R}^n$ coordinate by coordinate (without changing the name of the function):

$$z = \sigma_\lambda(y) \iff z_i = \sigma_\lambda(y_i)\ \forall i = 1, ..., n \tag{19}$$

and conclude that the solution of the LASSO is

$$\widehat{x} = \sigma_\lambda(y) \tag{20}$$

Basis Pursuit (BP) and the LASSO are not the same algorithms. In fact, there exists four methods of optimization whose definitions are close : LASSO, BP, BPDN and QCBP (quadratically constrained basis pursuit).

$$\text{BP} \iff \begin{cases} \min_{z\in\mathbb{R}^n} ||z||_1 \\ \text{s.t. } Az = b \end{cases} \tag{21}$$

$$\text{LASSO} \iff \begin{cases} \min_{z\in\mathbb{R}^n} ||Az-b||_2 \\ \text{s.t. } ||x||_1 < \tau \end{cases} \tag{22}$$

$$\text{BPDN} \iff \arg\min_{z\in\mathbb{R}^n} \left(\frac{1}{2}||z-y||_2^2 + \lambda||z||_1\right) \tag{23}$$

$$\text{QCBP} \iff \begin{cases} \min_{z\in\mathbb{R}^n} ||z||_1 \\ \text{s.t. } ||Az-b||_2 < \epsilon \end{cases} \tag{24}$$

It can be shown (see Foucart p. 60-65 and Proposition 3.2. p. 64 for complete proofs) that LASSO,

BPDN and QCBP have same solutions (up to adapting constants).

Optimization methods are of the form:

$$\min \bullet \text{ s.t. } \star \tag{25}$$

In the BP algorithm we do not take into account the full vector z, but only a linear transformation of some of its components. This is achieved through a matrix-vector multiplication. In the next part we will do this by applying the Fourier transform operator

1.6.

```
def SoftThresh(y,lambd):
  xhat = np.maximum(np.abs(y)-lambd,0)*np.sign(y)
  return(xhat)
ST = SoftThresh(np.linspace(-10,10,100),2)
plt.plot(np.linspace(-10,10,100), ST)
```

other solution

```
def SoftTresh(y, lam):
    n = y.shape[1]
    xHat = np.empty([1, n])
    for i in range(n):
        if (y[0, i] <= -lam):
            xHat[0, i] = y[0, i] + lam
        elif (abs(y[0, i]) < lam):
            xHat[0, i] = .0
        elif (y[0, i] >= lam):
            xHat[0, i] = y[0, i] - lam
    return(xHat)
```

```
lambd = 0.1
xhat = SoftThresh(y,lambd)
plt.stem(x)
plt.stem(range(len(x)),xhat,linefmt = 'red')
plt.figure()
plt.stem(y)
plt.stem(range(len(x)),xhat,linefmt = 'red')
ran = np.arange(-10, 10.1, 0.1)
u = np.empty([1, ran.shape[0]])
u[0, :] = ran
lam = 2.
xHat = SoftTresh(u, lam)
plt.plot(u.T, xHat.T)
plt.show()
```

If y is small, it gets set equal to zero. When y is larger than λ instead, its value is decreased by a quantity equal to λ.

1.7.
Python code :

```
lamLst = [0.01, 0.05, 0.1, 0.2]
xHatLst = []
for i in range(len(lamLst)):
    xHat = SoftTresh(y, lamLst[i])
    xHatLst.append(xHat)
fig, axs = plt.subplots(3, 1, figsize=(13, 9))
axs[0].stem(xHatLst[2].T, use_line_collection=True)
axs[0].set_title('$\hat{x}$ obtained
with lambda = 0.1', fontsize=21)
axs[0].set_ylim(-0.1, 1.1)
axs[1].stem(x.T, use_line_collection=True)
axs[1].set_title('Original vector x', fontsize=21)
axs[1].set_ylim(-0.1, 1.1)
axs[2].stem(x.T-xHatLst[2].T, use_line_collection=True)
axs[2].set_title('Difference x - $\hat{x}$', fontsize=21)
axs[2].set_ylim(-0.1, 1.1)
plt.subplots_adjust(hspace=0.95)
plt.show()
```

The solution $\hat{x}$ is sparse (even if not perfectly). The position of the non zero values is perfectly reconstructed. However, as we can see from the last panel, the amplitude of the non zero entries of the original vector is not perfectly obtained. In particular, the reconstructed values are smaller than the original ones.

2 Part 2 - Random frequency domain sampling and aliasing

2.1.
Code in Python :

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

```
# generate a row vector
n = 128
k = 5
x = np.zeros([1, n])
for i in range(0, k):
    x[0, i] = (i+1)/k
    x[0, i] = (i+1)/k * 100.
# permute the positions
x = np.random.permutation(x.T).T
plt.stem(x.T, use_line_collection=True)
plt.show()
```

2.2-2.3.
Code in Python

```
X = np.fft.fft(x)
# X = np.fft.fftshift(x)
X.shape
plt.plot(X.T.real)
plt.show()
Xu = np.zeros([1, n], dtype=np.complex_)
for i in range(0, n, 4):
    Xu[0, i] = X[0, i]
plt.plot(Xu.T)
plt.show()
xu = np.fft.ifft(Xu)*4
# get real an imaginary parts
xuReal = xu.real
xuComp = xu.imag
fig, axs = plt.subplots(3, 1, figsize=(20, 10))
#fig.suptitle('Vertically stacked subplots')
axs[0].stem(x.T, use_line_collection=True)
axs[1].stem(xuReal.T, use_line_collection=True)
axs[2].stem(xuComp.T, use_line_collection=True)
axs[0].title.set_text('x')
axs[1].title.set_text('xuReal')
axs[2].title.set_text('xuImag')
plt.show()
```

xu is periodic because it has been obtained from a periodic sampling of the Fourier transform of x. From this result we will not be able to reconstruct the original result as we miss information about the position of the non-zero entries.

2.4.
Code in Python:

```
indexes = np.arange(0, n, 1)
indexes = np.random.permutation(indexes.T).T
indexes = indexes[0:32]
Xr = np.zeros([1, n], dtype=np.complex_)
for i in (range(len(indexes))):
    Xr[0, indexes[i]] = X[0, indexes[i]]
xr = np.fft.ifft(Xr)*4
# get real an imaginary parts
xrReal = xr.real
xrComp = xr.imag
fig, axs = plt.subplots(3, 1, figsize=(20, 10))
#fig.suptitle('Vertically stacked subplots')
axs[0].stem(x.T, use_line_collection=True)
axs[1].stem(xrReal.T, use_line_collection=True)
axs[2].stem(xrComp.T, use_line_collection=True)
axs[0].title.set_text('x')
axs[1].title.set_text('xrReal')
axs[2].title.set_text('xrImag')
plt.show()
```

In opposition with the previous case, we are now able to make an infer on the position of the non-zero elements of vector x. This is hampered by the fact that the recovered $\hat{x}$ is not sparse and there are some sizable non-zero entries that do not correspond to non-zero entries of vector x. We could therefore say that the vector $\hat{x}$ is affected by noise.

This example shows that a random sampling is desirable.

The frequential undersampling adds noise to the signal. This noise (called incoherent aliasing) is not a real one and comes from the signal. We should be able to suppress it). By random unsampling, we've changed the ill-conditioned problem into a sparse signal denoising problem.

3 Part 3 - Reconstruction from randomly sampled frequency domain data

a.

Code in Python:

```
import numpy as np
import matplotlib.pyplot as plt
# generate a row vector
n = 128
k = 5
x = np.zeros([1, n])
for i in range(0, k):
    x[0, i] = (i+1)/k
# permute the positions
x = np.random.permutation(x.T).T
# Compute the Fourier transform X
# and undersample
X = np.fft.fft(x)/np.sqrt(n)
indexes = np.arange(0, n, 1)
indexes = np.random.permutation(indexes.T).T
indexes = indexes[0:32]
Y = np.zeros([1, n], dtype=np.complex_)
for i in (range(len(indexes))):
    Y[0, indexes[i]] = X[0, indexes[i]]
y = np.fft.ifft(Y*np.sqrt(n)*4)
# get real an imaginary parts
yReal = y.real
yComp = y.imag
fig, axs = plt.subplots(3, 1, figsize=(20, 10))
#fig.suptitle('Vertically stacked subplots')
axs[0].stem(x.T, use_line_collection=True)
axs[1].stem(yReal.T, use_line_collection=True)
axs[2].stem(yComp.T, use_line_collection=True)
axs[0].set_title('$x$', fontsize=21)
axs[1].set_title('yReal', fontsize=21)
axs[2].set_title('yImag', fontsize=21)
plt.subplots_adjust(hspace=0.5)
plt.show()
#
# implement the Projection Over Convex Sets
# (POCS) # algorithm
def SoftTresh(y, lam):
    n = y.shape[1]
    xHat = np.empty([1, n], dtype='cfloat')
    for i in range(n):
        if (np.absolute(y[0, i]) < lam):
            xHat[0, i] = .0
        elif (np.absolute(y[0, i]) >= lam):
            # xHat[0, i] = y[0, i] /
            # (np.absolute(y[0, i]) *
            # (np.absolute(y[0,
            # i])-lam)).astype(complex)
            real = y[0, i].real / np.absolute(
            y[0, i]) * (np.absolute(
            y[0, i])-lam)
            imag = y[0, i].imag / np.absolute(
            y[0, i]) * (np.absolute(
            y[0, i])-lam)
            xHat[0, i] = real + imag*1j
    return(xHat)
#
def POCS(Y, lam, nIte, x):
    XHat = Y  # initialise
    #print(np.linalg.norm(Y, ord=2))
    xHatLst = []
    errorLst = []
    for i in range(0, nIte):
        xHat = np.fft.ifft(XHat*np.sqrt(n))
        #print(np.linalg.norm(xHat, ord=2))
        xHat = SoftTresh(xHat, lam)
        XHat = np.fft.fft(xHat)/np.sqrt(n)
        #print(np.linalg.norm(XHat, ord=2))
        #print("")
        #indexes = np.where(Y == 0)[1]
        indexes2 = np.where(Y != 0)[1]
        #XHat[0, indexes] = XHat[0, indexes]
        XHat[0, indexes2] = Y[0, indexes2]
        xHatLst.append(xHat)
        errorLst.append(sum((xHat.T.real-x.T)**2))
    return(xHatLst, errorLst)
#
nIte = 100
lam = 0.01
xHatLst001, errorLst001 = POCS(Y, lam, nIte, x)
lam = 0.05
xHatLst005, errorLst005 = POCS(Y, lam, nIte, x)
lam = 0.1
xHatLst01, errorLst01 = POCS(Y, lam, nIte, x)
# Plot the estimate at each iteration
for i in range(len(xHatLst001)):
    plt.stem(xHatLst001[i].T.real, use_line_collection=True)
    plt.title('Evaluation {}, lambda = 0.01'.format(i))
    plt.show()
#
plt.stem(x.T, use_line_collection=True)
plt.title('x')
plt.show()
#
plt.stem(x.T, use_line_collection=True)
plt.title('x')
plt.show()
#
for i in range(len(xHatLst005)):
```

```
    plt.stem(xHatLst005[i].T.real, use_line_colle
    plt.title('Evaluation {}, lambda = 0.05'.form
    plt.show()
#
for i in range(len(xHatLst01)):
    plt.stem(xHatLst01[i].T.real, use_line_collec
    plt.title('Evaluation {}, lambda = 0.1'.forma
    plt.show()
#
plt.stem(x.T, use_line_collection=True)
plt.title('x')
plt.show
# f. Plot The Error At Each Iteration
fig, axs = plt.subplots(3, 1, figsize=(15, 15))
axs[0].plot(errorLst001, 'bs')
axs[0].set_title('Error, lambda = 0.01',
fontsize=21)
# #axs[0].set_ylim(-0.1, 1.1)
axs[1].plot(errorLst005, 'bs')
axs[1].set_title('Error, lambda = 0.05',
fontsize=21)
# #axs[1].set_ylim(-0.1, 1.1)
axs[2].plot(errorLst01, 'bs')
axs[2].set_title('Error, lambda = 0.1',
fontsize=21)
# #axs[2].set_ylim(-0.1, 1.1)
# #plt.subplots_adjust(hspace=0.95)
plt.show()
# g. Repeat the iterative reconstruction for the
# equispaced undersampled signal
Y = np.zeros([1, n], dtype=np.complex_)
for i in range(0, n, 4):
    Y[0, i] = X[0, i]
y = np.fft.ifft(Y*np.sqrt(n))*4
nIte = 100
lam = 0.01
xHatLst001, errorLst001 = POCS(Y, lam, nIte, x)
lam = 0.05
xHatLst005, errorLst005 = POCS(Y, lam, nIte, x)
lam = 0.1
xHatLst01, errorLst01 = POCS(Y, lam, nIte, x)
# Plot estimate at each iteration
for i in range(len(xHatLst001)):
    plt.stem(xHatLst001[i].T.real, use_line_collection=True)
    plt.title('Evaluation {}, lambda = 0.01'.format(i+1))
    plt.show()
for i in range(len(xHatLst005)):
    plt.stem(xHatLst005[i].T.real, use_line_collection=True)
    plt.title('Evaluation {}, lambda = 0.05'.format(i+1))
    plt.show()
#     if (i == len(xHatLst005)-1):
#         plt.stem(x.T, use_line_collection=True)
#         plt.title('x'.format(i))
#         plt.show()
plt.stem(x.T, use_line_collection=True)
plt.title('x')
plt.show()
#
for i in range(len(xHatLst01)):
    plt.stem(xHatLst01[i].T.real, use_line_collection=True)
    plt.title('Evaluation {}, lambda = 0.1'.format(i+1))
    plt.show()
#     if (i == len(xHatLst01)-1):
#         plt.stem(x.T, use_line_collection=True)
#         plt.title('x'.format(i))
#         plt.show()
```