Homework 3

NMF - Clustering - Sparse Optimization - SVMs

CMU 11-755/18-797: Machine Learning for Signal Processing (Fall 2022)

OUT: October 25th, 2022

DUE: **November 10th, 2022, 11:59 PM Eastern Time**

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution independently: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.
- **Submitting your work:** Assignments should be submitted to Canvas unless explicitly stated otherwise. Please submit all derivation/explanation results as **report_{YourAndrewID}.pdf**. **Each derivation/proof should be completed on a separate page**. Submissions can be handwritten, but should be labeled and clearly legible. Alternatively, submissions are strongly encouraged to be typeset using LaTeX. Please refer to Piazza for detailed instructions for joining Canvas and submitting your homework.
- **Programming**: All programming portions of the assignments should be submitted to Canvas as well. Please submit all codes and output files as **programming_{YourAndrewID}.zip**. We will not be using this for autograding, but rather for plagiarism detection, meaning you may use any language you would like to program.
- **Late submissions**: You have **in total 7 slack days** that you can **freely apply** to any homework. Any homework submitted after running out of slack days will receive zero credit. Please make sure you submit on time.

1 Understanding Clustering, Regression and SVMs

1.1 Clustering - Centroids election

In this problem, you will see why we define the centroids as the mean of the points in a cluster.

1. Let $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$ be a set of n points in a d-dimensional space. Show that the sum of the squared distances of the $\mathbf{a}_i$ to any point $\mathbf{x}$ equals the sum of the squared distances to the centroid of the $\mathbf{a}_i$ plus n times the squared distance from $\mathbf{x}$ to the centroid. That is,

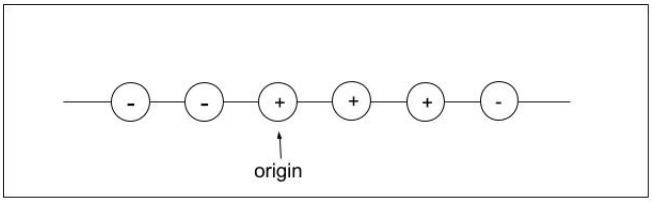
$$\sum_{i=1}^{n} \|\mathbf{a}_i - \mathbf{x}\|_2^2 \quad = \quad \sum_{i=1}^{n} \|\mathbf{a}_i - \mathbf{c}\|_2^2 + n\|\mathbf{c} - \mathbf{x}\|_2^2 \tag{1}$$

   where $\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{a}_i$ is the centroid of the set of points.

2. Conclude that the sum of squared distances of the $\mathbf{a}_i$ to a point $\mathbf{x}$ is minimized when $\mathbf{x}$ is the centroid, namely $\mathbf{x} = \frac{1}{n}\sum_i \mathbf{a}_i$

1.2 Support Vector Machines

Use the following dataset in 1-D space which consists of 3 positive data points $\{0, 1, 2\}$ and 3 negative data points $\{-2, -1, 3\}$.



1. Find a feature map $\varphi : \mathbb{R}^1 \to \mathbb{R}^2$ which will map the data in the original 1-D input space (x) to a 2-D feature space $\varphi(x) = (y_1, y_2)$ so that the data becomes linearly-separable. Plot the dataset after mapping in 2-D space.

2. Write down the equation for the separator hyperplane $w_0 + w_1 y_1 + w_2 y_2 = 0$ given by hard-margin linear SVM in the 2-D feature space. Draw this hyperplane on your plot and mark the corresponding support vector(s).

1.3 Regularized least squares

Given X, a $m \times n$ matrix, show that the solution to the regularized least-squares problem

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - X\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2 \tag{2}$$

is

$$\widehat{\boldsymbol{\beta}} \quad = \quad (X^\top X + \lambda I_n)^{-1} X^\top \mathbf{y} \tag{3}$$

where I_n is the identity matrix of size n and $\lambda > 0$.

2 Signal Separation using Non Negative Matrix Factorization

In this problem you will use NMF to separate a mixed signal into its component signals. Specifically, your goal is to separate music and speech sounds from a signal consisting of both.

The basic idea is that you will use NMF to first learn bases for speech and music. Then you will use the learned bases to separate out music and speech from a recording consisting of both.

Do the following processing steps.

1. In the directory `hw3materials_f22/problem2` you will find `speech.wav` file. Read the speech signal and compute its spectrogram. You already did this in previous homework 1. Use the `stft` function followed by magnitude computation, as before;

```
% matlab
[s, fs] = audioread('filename');
spectrogram = stft(s',2048,256,0,hann(2048))
M = abs(spectrogram)
```

```
# python
audio, sr = librosa.load("filename", sr=None)
spectrogram = librosa.stft(audio, n_fft=2048, hop_length=256, center=False, win_length=2048)
M = abs(spectrogram)
phase = spectrogram/(M + 2.2204e-16)
```

2. Do the same for `music.wav` file.

If we call M_m and M_s the magnitude spectrum of the music and speech file respectively, then the dimensionality of both matrices should be 1025×976.

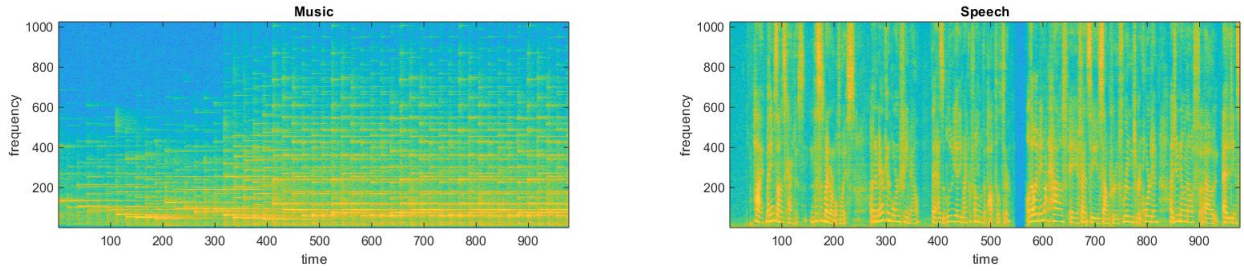


Figure 1: Magnitude spectrum (in log scale) of files `music.wav` and `speech.wav`

Now using the spectrograms for music M_m and speech M_s you are first going to learn bases, B_m and B_s for music and speech respectively. Therefore, we given the non-negative matrices M_m and M_s, we need to find the non-negative matrices B_m, W_m, B_s and W_s such that

$$M_m \approx B_m W_m \tag{4}$$
$$M_s \approx B_s W_s \tag{5}$$

2.1 NMF Estimation: Learning Bases

In this homework, given a non-negative matrix $M \in \mathbb{R}^{p \times q}$, we will consider the algorithm to estimate the non-negative matrices $B \in \mathbb{R}^{p \times k}$ and $W \in \mathbb{R}^{k \times q}$ that attempts to minimize the *KL divergence* between M and BW. The KL divergence between matrix A and matrix B is given by

$$D(A \,||\, B) = \sum_{ij} \left(A_{ij} \log \left(\frac{A_{ij}}{B_{ij}} \right) - A_{ij} + B_{ij} \right) \tag{6}$$

Then, the algorithm that minimizes $D(M||BW)$ is the following:

- Initialize B and W randomly.
- Iteratively update B and W using the following rule:

$$B = B \otimes \frac{\left(\frac{M}{BW}\right) W^\top}{\mathbf{1}_{p\times q} W^\top} \tag{7}$$

$$W = W \otimes \frac{B^\top \left(\frac{M}{BW}\right)}{B^\top \mathbf{1}_{p\times q}} \tag{8}$$

where $\otimes$ denotes component-wise matrix multiplication, $\frac{\cdot}{\cdot}$ denotes the component-wise matrix division, and $\mathbf{1}_{p\times q}$ is the $p \times q$ matrix which each of its component is 1.

1. Write the function `NMF_train` which receives as inputs:
   - `M`: a $p \times q$ non-negative matrix.
   - `B_init`: a $p \times k$ non-negative matrix. This matrix is the initial value of matrix B.
   - `W_init`: a $k \times q$ non-negative matrix. This matrix is the initial value of matrix W.
   - `n_iter`: a positive integer value. It determines the number of iterations of the algorithm.

   Based on the algorithm previously described, the outputs of this function must be:
   - `B`: a $p \times k$ non-negative matrix.
   - `W`: a $k \times q$ non-negative matrix.

   Submit your code.
2. In the directory `hw3materials_f22/problem2` you can find the files `Bm_init.csv` and `Wm_init.csv`. These files are matrices which must be used as initial condition to run your function `NMF_train` over the magnitude spectrum of the music signal M_m. Consider values for `n_iter` in $\{0, 50, 100, 150, 200, 250\}$. Run your function using these parameters and plot $D(M_m||B_m W_m)$ as function of `n_iter`. Attach this plot to your report and write the value of $D(M_m||B_m W_m)$ for `n_iter` $= 250$.
3. Similarly, you can also find in the directory `hw3materials/problem2` the files `Bs_init.csv` and `Ws_init.csv`. As before, these files are matrices which must be used as initial condition to run your function `NMF_train` over the magnitude spectrum of the speech signal M_s. Repeat the previous task your this signal considering the same setting. Attach your plot to the report and write the value of $D(M_s||B_s W_s)$ for `n_iter` $= 250$.

2.2 Signal Separation

Now, using the bases you learned for speech and music, we will separate a signal which is a mix between a different speech signal and a music signal.

In the directory `hw3materials_f22/problem2` you can find a file named `mixed.wav`. Compute its magnitude spectrum, that we will call M_{mixed}. The following figure visualize this matrix.

Our model considers that the bases we learned in the previous part are good enough to represent this audio file. More precisely, in this case, given M_{mixed}, B_s and B_m, we need to learn the weights W such that

$$M_{\text{mixed}} \quad \approx \quad [B_s \; B_m] W \tag{9}$$

In this equation, $[B_s \; B_m]$ represents the concatenation of the bases you already learned; so if M_{mixed} is a $p \times q$ matrix, $[B_s \; B_m]$ is a $p \times 2k$ matrix. Moreover, the matrix W gives us the contribution of each element in this *new* set of bases. To learn W we can use the previous algorithm, but without updating the bases B (we already know B).

1. Write the function `separate_signals` which receives as inputs:

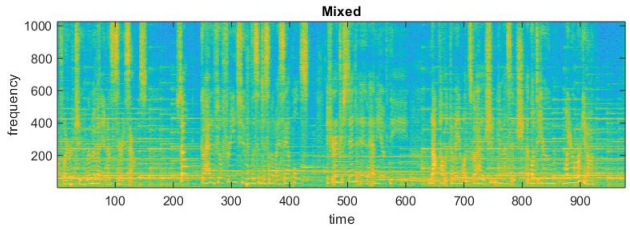


Figure 2: Magnitude spectrum (in log scale) of file `mixed.wav`

- `M_mixed`: a $p \times q$ non-negative matrix.
- `B_speech`: a $p \times k$ non-negative matrix. This matrix is the bases you have to represent speech.
- `B_music`: a $p \times k$ non-negative matrix. This matrix is the bases you have to represent music.
- `n_iter`: a positive integer value. It determines the number of iterations of the algorithm.

The output of this function must be:

- `M_speech_rec`: a $p \times q$ non-negative matrix. It is the recovered speech magnitude spectrum.
- `M_music_rec`: a $p \times q$ non-negative matrix. It is the recovered music magnitude spectrum.

Submit your code

2. Using the bases you learned from the previous problem when you considered `n_iter` = 250, apply your function `separate_signals` using the magnitude spectrum of the mixed signal and taking `n_iter` = 500. Submit your output as `M_speech_rec.csv` and `M_music_rec.csv`.
3. Now using the phase for the mixed signal along with reconstructed spectrograms, reconstruct time domain music and speech signals. You did this in homework 1 as well. You can use the `stft` function again to do this. Save the generated signals as `speech_rec.wav` and `music_rec.wav`. Submit these two files.

3 Sparse recovery

We have previously noted in class that given a dictionary D and a data X that can be composed as a *sparse* combination of dictionary entries, i.e. we can write $X = DY$, where $\|Y\|_0 \leq k$ for some small value of k, then Y can be recovered from X and D using sparse recovery techniques. Specifically, we noted that this can be estimtated through the following minimization:

$$\hat{Y} \quad = \quad \arg\min_{Y} \|X - DY\|_2^2 + \lambda \|Y\|_1 \tag{10}$$

We will now see how this simple principle can be used to *subsample* data and still recover it. This principle is known as "compressive sensing".

We will use an example to explain the concept. The example will also be your homework problem.

A little story: Cassini-Huygens

The Cassini spacecraft was launched on 15 Oct 1997 and entered into orbit around Saturn on 1 July 2004. Ever since it has been orbiting the ringed planet, taking thousands of pictures of Saturn, its rings, and its moons, and beaming them back to Earth. Here are some of the gorgeous pictures sent by Cassini to earth. Cassini is powered by about 33 kg of Plutonium-238, which will continue to provide power to the spacecraft until the end of its mission.

On Christmas day 2004 Cassini launched the Huygens probe towards Saturn's moon, Titan. Huygens landed on Titan on 14 Jan 2005. Unlike Cassini, Huygens was powered by a battery. By design, the module had no more than three hours of battery life, most of which was planned to be used during the descent onto Titan. Engineers expected to get at most only 30 minutes of data from the surface, much of which was in the form of pictures of Titan's surface and topography. Here are the incredible pictures sent by Huygens.

Our "story" ends with this note: Huygens could only send 350 pictures before its batteries ran out of power. (An additional 350 pictures were lost because of a software bug due to which the channel they were sent on was ignored).

Compressive sensing to the rescue

One of the main consumptions of on-board battery is for the *transmission* of the pictures. Each image must be adequately "wrapped" with error-correcting code and transmitted to the recipient (Huygens transmitted to both Cassini and the Earth). The amount of energy consumed to transmit a picture directly relates to the number of pixels captured – more pixels require more energy both to capture and to transmit.

Let us now consider an alternate scheme.

(a) ***A note on image sparsity in the wavelet domain***

It is well known that when considered in the wavelet transform domain, images are sparse. A wavelet transform can be viewed as a matrix operation (just as a discrete Fourier transform can). Let I be any image (expressed as a vector). The wavelet transform $\mathcal{F}$ of the image can be computed as

$$\mathcal{F} \quad = \quad WI \tag{11}$$

where W is the transform matrix. (In reality, the transform is viewed as a *filterbank*, but we will assume the simpler model above for our discussion).

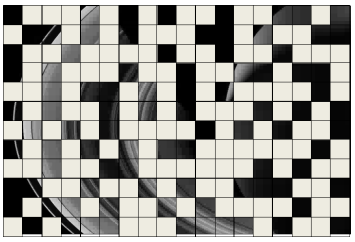
The image itself can be recovered from the transform as

$$I \quad = \quad W^{-1}\mathcal{F} \tag{12}$$

For images, $\mathcal{F}$ is generally sparse, i.e. most of the components of $\mathcal{F}$ are zero or close to zero. **We will use this to our benefit**.

(b) ***Capturing one-pixel masked integral images***

Instead of simply snapping a picture directly, consider a scheme where Huygens would instead apply a *random mask* to its camera and computes an *integral* instead. So, for instance, instead of capturing the entire image, Huygens applies a random mask to get the following picture



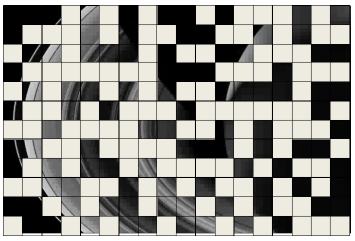
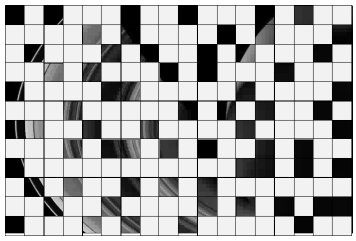
and computes a **single** integral value

$$P_1 \quad = \quad \sum_{i,j} m_{i,j} I_{i,j} \tag{13}$$

where $m_{i,j}$ is the value of the mask at pixel (i, j), and $I_{i,j}$ is the actual image value. Representing the mask as the vector $\mathbf{m}_1$, where the subscript 1 is to indicate that this is the first such mask applied, Huygens' first measurement would be

$$P_1 \quad = \quad \mathbf{m}_1^\top I \tag{14}$$

Similarly, applying a series of other such masks, e.g.

Huygens can now get a *series of measurements* $P_1,\ P_2,\ P_3, \cdots, P_K$ of (scalar) measurements, where P_i has been obtained using mask $\mathbf{m}_i$, and K is the total number of measurements obtained in this manner. Representing all of the masks as a single matrix $\mathbf{M} = [\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3 \cdots \mathbf{m}_K]^\top$, and the measurements P_i collectively as a vector $\mathbf{P} = [P_1\ P_2\ P_3 \cdots P_K]^\top$, we can write

$$\mathbf{P} \quad = \quad \mathbf{M}I \tag{15}$$

Note that K may be far fewer than the number of pixels in the image. Huygens can now simply transmit $P_1 \cdots P_K$, and save on power. If $K < N$, where N is the number of pixels in the image, Huygens could use the saved energy to take more pictures and transmit them. Since we are sending far fewer numbers than we would if we were to actually send the entire image, we will call these *compressed* measurements.

(c) ***Recovering the full image from the compressed measurements***

But how then can the N-pixel pictures themselves be recovered from this sequence of K compressed measurements? For this we exploit the sparsity of images in the wavelet domain.

Recall from our earlier discussion that we can represent $I = W^{-1}\mathcal{F}$, where $\mathcal{F}$ is sparse. Hence we can write

$$\mathbf{P} \quad = \quad \mathbf{M}W^{-1}\mathcal{F} \tag{16}$$

Let $\mathbf{R} = \mathbf{M}W^{-1}$. Remember that the random masks applied to the camera are *known*, i.e. $\mathbf{M}$ is known. The inverse wavelet transform matrix W^{-1} of course known. Hence $\mathbf{R}$ is known. We can therefore write

$$\mathbf{P} \quad = \quad \mathbf{R}\mathcal{F} \tag{17}$$

In other words, we are expressing the compressed measurements $\mathbf{P}$ as a sparse combination of the columns in a dictionary matrix $\mathbf{R}$.

To recover the image I from the $K \times 1$ compressed measurement vector $\mathbf{P}$, it is sufficient to recover the $N \times 1$ sparse vector $\mathcal{F}$, since $I = W^{-1}\mathcal{F}$. We can do so using the sparse recovery technique we discussed in the context of dictionary-based representations, which we recalled above:

$$\hat{\mathcal{F}} \quad = \quad \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 + \lambda\|\mathcal{F}\|_1 \tag{18}$$

The complete image can now be recovered as $I = W^{-1}\hat{\mathcal{F}}$. This is the basic concept behind compressive sensing.

We have managed to come up with a solution that enables Huygens to send far fewer numbers (only K) than the size of the image itself (N pixels) and still recover the image. Huygens can now use the conserved battery to take more pictures. We have thus rescued NASA's space exploration program!!

Having achieved these lofty goals, let us now return to a more mundane issue: homework problems.

3.1 ℓ_1 minimization

In this problem we give you compressed measurements from an image taken by Cassini (not Huygens). We also give you the "measurement matrix" $\mathbf{R}$. You must recover the full image. (Note: we are referring to $\mathbf{R}$ in the above equations as the measurement matrix; more conventionally $\mathbf{M}$ would be called the measurement matrix).

You have been provided of the original image (`hw3material_f22/problem3`). This is only for reference, to compute error; **we will not use it in recovery computations**. It is a 128×128 image with a total of 16384 pixels. Note that the sparse wavelet transform $\mathcal{F}$ of the image will also have 16384 components, although most of them will be very small or 0.

In the directory `hw3materials_f22/problem3/data` you also can find three sets of compressed measurements `P1024.csv`, `P2048.csv` and `P4096.csv`, and their corresponding measurement matrices `R1024.csv`, `R2048.csv` and `R4096.csv`. In each case the numbers represent the number of measurements. Thus `P1024.csv` contains a 1024×1 vector, representing 1024 one-pixel measurements, and `R1024.csv` is a 1024×16384 matrix that gives the linear transform from the 16384-component sparse transform $\mathcal{F}$ to the measurement vector in `P1024`. In the same directory you will also find the file named `S.csv`. This file is required by matlab for image reconstruction, and is not part of the actual recovery algorithm. For Python, image reconstruction should be done using the `pywavelets` package.

You are required to use the sparse recovery procedure to recover $\mathcal{F}$ for each of these three measurements, and to reconstruct the image.

1. For each of the three measurements, solve

$$\hat{\mathcal{F}} \quad = \quad \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 + \lambda\|\mathcal{F}\|_1 \tag{19}$$

   We recommend setting $\lambda = 100$, although you may also try other values. For those using MATLAB, We included an implementation of a least-squares ℓ_1 (`l1_ls.m`) solver from Stephen Boyd (`hw3materials_f22/problem3`). Note however that this is not the best algorithm; there are other better algorithms for ℓ_1 minimization. The Stanford sparselab page has several nice solvers. You may try these; some of them should give you better results.

If you are using Python, we would recommend to use Lasso from Sklearn. However, feel free to explore other solvers written in Python.

For each of the three measurements, reconstruct the original image using the recovered $\mathcal{F}$. You can do so using matlab through the following commands:

```
S = csvread('S.csv');
Irec = waverec2(F',S,'db1');
```

or if you are using Python, use the `F2Coeffs.py` in the handout code.

Compute the error $E = \sum_{i,j}(I(i,j) - I_{rec}(i,j))^2$, where $I(i,j)$ and $I_{rec}(i,j)$ are the (i,j)th pixel value in the original and recovered image respectively.

You will be required to include your errors of three measurements and show the reconstructed image in your report (specify which solver and what parameters do you use), and submit your code. Submissions that report lower error will obtain better scores for this problem, so it is in your benefit to explore the parameter space.

3.2 Iterative Hard Thresholding

In the previous part we solved the problem of sparse recovery as one of ℓ_1 minimization. We do so since it is well known that minimizing the ℓ_1 norm also often minimizes the ℓ_0 norm of the solution; however our actual objective in sparse recovery is to find a solution that has minimal ℓ_0 norm, i.e. the smallest number of non-zero elements! Often, the optimal ℓ_1 minimized solution will differ significantly (and can be significantly worse) than the optimal ℓ_0 minimized solution.

In this part of the problem we will solve the alternate optimization problem:

$$\hat{\mathcal{F}} \quad = \quad \arg\min_{\mathcal{F}} \|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 \quad \text{such that} \quad \|\mathcal{F}\|_0 \leq K \tag{20}$$

where K is the level of sparsity desired in $\mathcal{F}$. This is clearly an ℓ_0 *constrained* optimization problem. $\|\mathcal{F}\|_0 \leq K$ specifies a *feasible set*, which is the set of all vectors that have no more than K non-zero elements (note that this is a non-convex set). Recall from our lecture on optimization that we can use the method of projected gradients for this problem.

The gradient of $\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2$ w.r.t $\mathcal{F}$ is given by

$$\nabla_{\mathcal{F}}\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2 \quad = \quad -\mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F}) \tag{21}$$

Representing the n^{th} iterate in an iterative estimate of $\mathcal{F}$ by $\mathcal{F}^n$, the projected gradient descent algorithm to estimate $\mathcal{F}$ would be given by

$$\mathcal{F}^{n+1}_{\text{unconstrained}} \quad = \quad \mathcal{F}^n + \eta\mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F}^n) \tag{22}$$

$$\mathcal{F}^{n+1} \quad = \quad P_K(\mathcal{F}^{n+1}_{\text{unconstrained}}) \tag{23}$$

where η is a step size. The operator P_K in the second step of the algorithm simply sets the $N - K$ smallest elements (in magnitude) of $\mathcal{F}^{n+1}_{\text{unconstrained}}$ to zero, forcing it to be a $K - sparse$ vector.

The second step in the above iteration is the "projection" step of the projected gradient algorithm. It provably *projects* $\mathcal{F}^{n+1}_{\text{unconstrained}}$ onto the feasible set for $\mathcal{F}$, namely the set of all K-sparse vectors.

The above iteration is guaranteed to converge under specific conditions on $\mathbf{R}$, K and η. This is captured nicely in the following algorithm known as "**Iterative Hard Thresholding**" (IHT), proposed by Tom Blumensath.

Iterative Hard Thresholding

As a first step, the IHT algorithm assumes that the individual columns of the measurement matrix $\mathbf{R}$ are vectors of norm ≤ 1. So, if this condition is not satisfied, multiply $\mathbf{R}$ by a scalar α such that the norm of the columns of $\alpha\mathbf{R}$ is less than 1. Note that it is NOT convenient that α is too small. Determine the largest value of α to reach the condition.

The IHT algorithm then performs iterations of the following computation:

1. Choose a maximum sparsity level K for your solution.
2. Initialize $\mathcal{F}^0_{\text{unconstrained}} = 0$. (Alternately, you could set it to any random value).
3. Iterate:
$$\mathcal{F}^{n+1} = P_K(\mathcal{F}^n + \mathbf{R}^\top(\mathbf{P} - \mathbf{R}\mathcal{F}^n))$$
until the error $\|\mathbf{P} - \mathbf{R}\mathcal{F}\|_2^2$ converges, or the maximum number of iterations is achieved.
4. The final output is $\hat{\mathcal{F}} = \mathcal{F}_{\text{converged}}$, where $\mathcal{F}_{\text{converged}}$ is the final estimate derived from the previous step.

What you need to do are as follows.

i. Implement the IHT algorithm. Include your IHT implementation to your code as a single file (iht.m or iht.py)

ii. For each of the three measurements in the previous problem, find the estimate $\hat{\mathcal{F}}$ and reconstruct the original image using the recovered $\hat{\mathcal{F}}$, as you already did it. Compute the error $E = \sum_{i,j}(I(i,j) - I_{rec}(i,j))^2$, where $I(i,j)$ and $I_{rec}(i,j)$ are the (i,j)th pixel value in the original and recovered image respectively. Include these errors, show the reconstructed images in your report, and submit your code.

Note: In this problem, use $K = round(max(\frac{M}{4}, \frac{N}{10}))$, where M is the number of measurements, and N is the size of the image. Set the maximum number of iterations to 500.

Note 2: If you need to redefine the dictionary as $\alpha\mathbf{R}$, then, you will find $\mathcal{F}$ such that $\mathbf{P} \approx \alpha\mathbf{R}\mathcal{F}$. However, you need to find $\mathcal{F}$ such that $\mathbf{P} \approx \mathbf{R}\mathcal{F}$. Make sure you account for this when you form the final image.

iii. You should find that the IHT solution is considerably worse than the ℓ_1 minimization result. Can you give any intuition as to why this is so? Include your analysis to your report.

4 Music Decomposition using NMF

Modifying the spectrogram computed for `FoxTitle.wav` in preceding homeworks by a tiny bit as such to avoid diving 0 during NMF iteration:

```
% matlab
[s, fs] = audioread('filename');
spectrogram = stft(s',1024,160,0,hann(1024))
M = abs(spectrogram) + eps
```

```
# python
audio, sr = librosa.load("filename", sr=None)
spectrogram = librosa.stft(audio, n_fft=1024, hop_length=160, center=False, win_length=1024)
M = abs(spectrogram) + 2.2204e-16 # a.k.a. np.finfo(float).eps
phase = spectrogram / M
```

This time we are going to perform NMF to attempt music transcription, still, without knowing the waveform of the individual notes.

To explain this, given $\mathbf{M}$, the (magnitude) spectrogram of the music, which is a matrix of size $D \times T$, where D is the size of the Fourier Transform and T is the number of spectral vectors in the signal, we need to find the non-negative matrices $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N]$ and $\mathbf{W} = [\mathbf{w}_1^\top, \mathbf{w}_2^\top, \ldots, \mathbf{w}_N^\top]^\top$ such that $\mathbf{M} \approx \mathbf{BW}$. The idea is that the bases $\mathbf{b}_i$ represents the component signal for the original signal $\mathbf{M}$ and $\mathbf{w}_i^\top$ gives the contribution of $\mathbf{B}_i$ in $\mathbf{M}$.

1. Use STFT to analyze the music signal. Please use a window size of 1024 samples and a hop size of 160 samples. You can analyze the signal either in linear scale or in logarithmic scale, and with any other STFT parameters you find that perform the best. Then, decompose the signal with NMF by assuming $N = 11$ bases in $\mathbf{B}$. Use the files `B_init.csv` and `W_init.csv` in the folder `hw3materials_f22/problem4` to initialize the bases and weights matrices and set the `n_iter` as 150.

   Submit the matrix $\mathbf{B}$ consisting of 11 components as `musicbases.csv`.

2. In the folder `hw3materials_f22/problem4` we also have the ground truths of the individual notes for you to validate your results. Using a windows size of 1024 samples and a hop size of 160 samples, every individual note recording will be segmented to 62 frames. Here we view the part starting from the 21st frame to the 40th frame as the steady part with a consistent spectrum. The average spectrum of these frames is the spectrum of this building block note. Compare the 11 building blocks you've got using NMF with each of the ground truth notes by calculating the absolute value of the cosine similarity between your bases and each of the ground truth bases to find the match with the highest similarity score. How many unique notes can you actually restore and how close are your answers? Please report the note names of the best match of your 11 bases, and their respective absolute cosine similarity scores. Write your thoughts about why NMF is / isn't proper for this task.