# Submission Details

The HW 2 is due at 11:59PM on Friday November 13, 2020.

Please refer to the HW2 template HW2.zip.
1. You will find the data needed for solving the homework problems in the directory “data” of each problem directory.
2. Write a pdf file with solution if it is applicable and save it in the directory “/pdf solution/”

3. Save the program code in the directory “/code/”
4. Save the obtained results in the directory “/results/”

## Problem 1

### Linear regression:

In this problem, we will implement a linear regression using matlab. Please use the DQE/Problem1/data.mat file containing the different measurements of height (variable y) of an individual according to its age (variable x).

You can visualize all the data using the following code

```
load data.mat
figure;
plot(x, y, `x');
xlabel(`Age');
ylabel(`hauteur');
```

The linear regression assumes that the variable y (height) is a linear combination of the parameter vector x (age):

$$f(x) = W^T x = \sum_{i=0}^{D} w_i x_i$$

where $x_0 = 1$, D is the dimension of the space.
Linear regression consists in finding the parameters $W$ which minimizes the least square error criteria

$$E(W) = \sum_{t=1}^{N} (f(x_t) - y_t)^2$$

We will try to find the linear regression parameters W in three different ways

## I) Exact solution

Remember the linear regression exact solution that minimizes the least square error is given by

$$(w_0, w_1) = A^{-1}B$$

$$A = \begin{pmatrix} N & \sum_t x_t \\ \sum_t x_t & \sum_t (x_t)^2 \end{pmatrix}$$

$$B = \left(\sum_t y_t \, , \sum_t y_t \, x_t\right)^T$$

1. Calculate the values of $(w_0, w_1)$ that minimize the least square error?
2. Plot the curve corresponding to these parameters with the data as well?
3. Predict the values of the height for both following persons aged of 3.5 and 7 years.

## II) Gradient decent solution

In this part, the gradient descent algorithm (see optimization course) will be used to find the best parameters. We will use the batch version (not the online one) and the recurrence formula is giving as follow:

$$W_{j+1} = W_j - \eta \frac{1}{N} \sum_{t=1}^{N} (f(x_t) - y_t)\, x_t$$

1. Implement gradient descent method with a learning step $\eta = 0.07$ and starting from the origin of the search space W = 0. Wait until the training converge?
   a. Use the $norm(\frac{1}{N}\sum_{t=1}^{N}(f(x_t) - y_t)\, x_t) < 0.001$ as stopping criteria.
2. Did you obtain the same results as the exact solution?
   Display in 3D the curve of the error based on the following code. (save the .fig file?)

```
error= zeros(100, 100);
w0 = linspace(-0.1, 1, 100);
w1 = linspace(-0.2, 0.45, 100);
 for i = 1:length(w0)
          for j = 1:length(w1)
             W = [w0(i); w1(j)];
             error(i,j) = ....
   end
end

error = error';
figure;
surf(w0, w1, error)
xlabel('w_0'); ylabel('w_1')
```

3. What is the connection between this figure and the values found by the gradient descent algorithm?

**III) Newton optimization solution**

In this part, we use Newton method to find the best parameters to the linear regression.

1. Implement the Newton method starting from the origin of the search space W = 0. Wait until the training converge?
   a. Use the $norm(\frac{1}{N}\sum_{t=1}^{N}(f(x_t) - y_t)\, x_t) < 0.001$ as stopping criteria.
2. Display in the same figure of question II-3 the curve of the error. (save the .fig file?)
3. Comment the speed between both gradient decent and Newton methods.

## Problem 2: 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*** = ***D Y***, 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 estimated through the following minimization:

$$\hat{Y} = \underset{Y}{argmin} \; \|X - DY\|_2^2 + \lambda |Y|_1$$

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.

**<u>A little story: Cassini-Huygens</u>**

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 (https://saturn.jpl.nasa.gov/galleries/images/) 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 (http://esamultimedia.esa.int/docs/titanraw/index.htm) 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 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 $\boldsymbol{I}$ be any image (expressed as a vector). The wavelet transform $\boldsymbol{F}$ of the image can be computed as

$$\boldsymbol{F} = \boldsymbol{W}\boldsymbol{I}$$

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

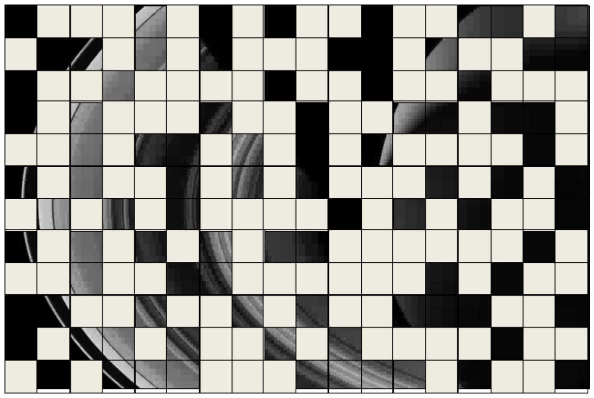
The image itself can be recovered from the transform as

$$\boldsymbol{I} = \boldsymbol{W}_{inv}\boldsymbol{F}$$

where $\boldsymbol{W}_{inv}$ is the inverse transform matrix. For images, $\boldsymbol{F}$ is generally sparse, *i.e.* most of the components of $\boldsymbol{F}$ are zero or close to zero. **We will use this to our benefit**.

### 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



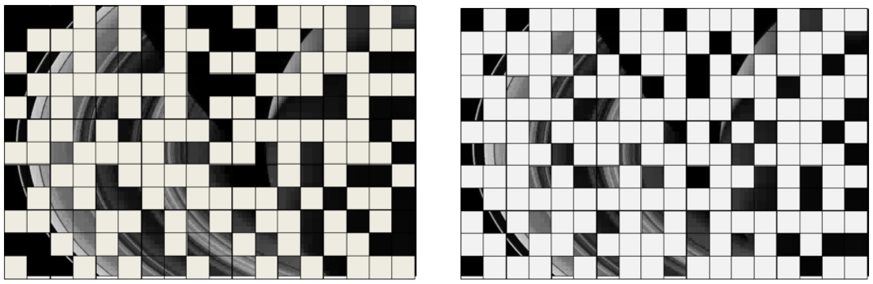
and computes a ***single*** integral value

$$\sum_{i,j} m_{i,j}\, I_{i,j}$$

where $\boldsymbol{m_{i,j}}$ is the value of the mask at pixel ***(i,j)***, and $\boldsymbol{I_{i,j}}$ is the actual image value. Representing the mask as the vector $\boldsymbol{m_1}$, where the subscript *1* is to indicate that this is the first such mask applied, Huygens' first measurement would be

$$P_1 = m_1^T I$$

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



Huygens can now get a *series of measurements* $\boldsymbol{P_1,P_2,P_3,\ldots,P_k}$ of (scalar) measurements, where $P_i$ has been obtained using mask $m_i$, and $\boldsymbol{K}$ is the total number of measurements obtained in this manner. Representing all of the masks as a single matrix $\boldsymbol{M = [m_1,m_2,m_3,\ldots,m_k]^T}$, and the measurements $\boldsymbol{P_i}$ collectively as a vector $\boldsymbol{P = [P_1,P_2,P_3,\ldots,P_k]^T}$, we can write

$$\boldsymbol{P=MI}$$

Note that $\boldsymbol{K}$ may be far fewer than the number of pixels in the image. Huygens can now simply transmit $\boldsymbol{P_1,P_2,P_3,\ldots,P_k}$ and save on power. If $\boldsymbol{K < N}$, where $\boldsymbol{N}$ 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.

## 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 $I = W_{inv}F$, where $F$ is sparse. Hence we can write

$$P = MW_{inv}F$$

Let $R = MWinv$. Remember that the random masks applied to the camera are *known*, i.e. $M$ is known. The inverse wavelet transform matrix $W_{inv}$ of course known. Hence $R$ is known. We can therefore write

$$P = RF$$

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

To recover the image $I$ from the ***Kx1*** compressed measurement vector $P$, it is sufficient to recover the *Nx1* sparse vector $F$, since as $I = W_{inv}F$. We can do so using the sparse recovery technique we discussed in the context of dictionary-based representations, which we recalled above:

$$\hat{F} = \underset{F}{argmin} \, \|P - RF\|_2^2 + \lambda |F|_1$$

The complete image can now be recovered as $I = W_{inv}\hat{F}$. 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.

---

## Homework Problem 2a

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

The file cassini128.png is the original image. This is only for reference, to compute error; we will not use it in recovery computations. It is a 128x128 image with a total of 16384 pixels. Note that the sparse wavelet transform F of the image will also have 16384 components, although most of them will be very small or 0.

This file compresseddata.tar contains three sets of compressed measurements “P1639.mat”, “P2048.mat” and “P4096.mat”, and their corresponding measurement matrices “R1639.mat”, “R2048.mat”, and “R4096.mat”. In each case the numbers represent the number of measurements. Thus “P1639.mat” contains a 1639x1 vector, representing 1639 one-pixel measurements, and “R1639.mat” is a 1639x16384 matrix that gives the linear transform from the 16384-component sparse transform ***F*** to the measurement vector in “P1639”. In the directory you will also find files name “SXXXX” (where XXXX is 1639, 2048 or 4096). This file is required by matlab for image reconstruction, and is not part of the actual recovery algorithm.

You are required to use the sparse recovery procedure to recover ***F*** for each of these three measurements, and to reconstruct the image.

1) For each of the three measurements, solve

$$\hat{F} = \underset{F}{argmin} \|P - RF\|_2^2 + \lambda |F|_1$$

We recommend setting $\lambda = 0.001$ max $(abs(R^T P)$, although you may also try other values. You may use any solver for the above problem. As a default, we included a matlab implementation (DQE/Problem2/code/l1_ls.m) of a least-squares $l_1$solver from Stephen Boyd in the folder. Note however that this is not the best algorithm; there are other better algorithms for $l_1$minimization.

2) For each of the three measurements, reconstruct the original image using the recovered *F*. You can do so using matlab through the following commands:

```
S=load ('SXXXX.mat');
Irec = waverec2(reshape(F,128,128),S,'db1');
```

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. Report this error.

## Problem 3: Gaussian classifier

In this problem we shall be building a digit recognition system using the MNIST dataset. This data set can be downloaded from http://yann.lecun.com/exdb/mnist/. The MNIST dataset is a set of handwrit- ten digits, and your job is to build a classifier that takes as input an image of a digit, and outputs the digit class.

You should download these four files and use the files loadMNISTImages.m and loadMNISTLabels.m available in problem3/Code/ to load these into MATLAB arrays.

The digit class chosen is simply the one that yields to the highest posterior probability of the class given as follow:

Here x represents the image, or more precisely, the pixel values of the image formatted as a vector, and c represents the digit, which can be 0,1,..,9. Images are grayscale and of size 28x28, which, if flattened, yields a vector of length 28x28=784

P(c|x) is the probability that the class is c given the data x. It is called the posterior. P(x|c) is the probability that the data x belongs to the class c. It is called the likelihood. P(c) is the prior of each class.

You will notice that P(x) is constant for all values of c in P(c|x). Using this information, we can simplify our problem so that, in order to choose which digit given an image, all we need to do is calculate this argmax (notice P(x) is removed):

$$c^* = argmax_c P(c|x) = argmax_c \frac{P(x|c).P(c)}{P(x)} \quad c^* = argmax_c P(x|c).P(c)$$

## II. Methodology

Build a MATLAB code for all these questions given below and please use the two MATLAB functions provided to read both images and their corresponding labels.

- Build a Gaussian model to estimate the likelihood probability P(x|c) with diagonal covariance matrice. Estimate the prior of each class. Apply Bayes rule and classify all the images in the test file. To have stability number estimate the log P(c|x). Report both classification accuracy by class and the total.

  – Visualize the means of each digit class (Reproduce the image of the means)

- Applying PCA followed by both full and diagonal covariance matrix Gaussian classifier. Select the number of eigen-vectors that capture 95% of the information present on the eigen-values. Report both classification accuracy by class and the total.

## Submission Instructions

Please follow these instructions closely. We will deduct points for not sticking to the template. Solutions should be uploaded to Blackboard.

Create a ”results” folder within Problem3 as follows”Problem3/results/ with the accuracies reported for all four parts in a single .txt file named ”results.txt”. Additionally your MATLAB code should generate a ”results.txt” file every time it runs (in case we’ll go ahead and run your code to verify it).

Note: Points will be deducted if your code doesn’t match the required output format mentioned above. Your code should not output anything else.