CS598 PS 2017 – PROBLEM SET 1 SOLUTIONS

Problem 1. A probability problem

Define the cancer event as c, the not-cancer event as $\bar{c}$, and the positive test event as p. From the problem definition we know that:

$$P(c) = 0.008, \quad P(p \mid c) = .9, \quad P(p \mid \bar{c}) = 0.07$$

We want to estimate the probability of the patient having cancer, that is $P(c \mid p)$. We do so using the Bayes' rule:

$$P(c \mid p) = \frac{P(p \mid c)P(c)}{P(p)}$$

We know all of the right hand side variables except for $P(p)$. We can estimate it using:

$$P(p) = P(p \mid c)P(c) + P(p \mid \bar{c})P(\bar{c}) = 0.9 \times 0.008 + 0.07 \times 0.992 = 0.076$$

Which gives us:

$$P(c \mid p) = \frac{P(p \mid c)P(c)}{P(p)} = \frac{0.9 \times 0.008}{0.07} \approx 0.094$$

Therefore the actual risk of cancer given a positive test is about 9%, a long way off from what most doctors guessed (think of that next time you go to the hospital).

For additional reading on this experiment and other interesting things about probability and how we perceive it, you can the following New York Times article:

http://opinionator.blogs.nytimes.com/2010/04/25/chances-are/

Problem 2. Manipulating data using linear algebra

1.a

Let us assume that the data is contained in matrix $\mathbf{X}$. We first compute the the average column vector of $\mathbf{X}$. We can do that with a matrix multiplication:

$$\bar{\mathbf{x}} = \frac{1}{K}\mathbf{X} \cdot \mathbf{1}$$

wherein $\mathbf{1}$ is a column vector that contains only ones. This product effectively sums all the columns of $\mathbf{X}$, and the division by K results in obtaining the average. In order to make this average vector a matrix we use the vec-transpose operator $\bar{\mathbf{x}}^{(M)}$, which will result in an $M \times N$ matrix containing the average image.

1.b

If we had an image in an $M \times N$ matrix $\mathbf{Y}$, in order to keep only the values in its upper half we would have to multiply with an identity matrix whose bottom half would be set to zero. We can construct this matrix using:

$$\mathbf{U} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \mathbf{I}_{M/2}$$

where $\mathbf{I}_{M/2}$ stands for an identity matrix with size $M/2$. However, we have the vectorized form of the image, $\mathbf{y} = \mathrm{vec}(\mathbf{Y})$, and not the image matrix itself. Therefore to perform this calculation in the vectorized domain we need to use the following identity:

$$\mathrm{vec}(\mathbf{A} \cdot \mathbf{B}) = (\mathbf{I} \otimes \mathbf{A}) \cdot \mathrm{vec}(\mathbf{B})$$

Substituting the upper-half selection matrix $\mathbf{U}$ and the vectorized image $\mathbf{y}$ we get the vectorized upper-half values of the image as $(\mathbf{I} \otimes \mathbf{U}) \cdot \mathbf{y}$. Since we need to obtain the average of that we use the averaging technique in the previous section to obtain:

$$\mathbf{u} = \frac{1}{\frac{M}{2}N} \mathbf{1} \cdot (\mathbf{I}_{M/2} \otimes \mathbf{U}) \cdot \mathbf{X}$$

Note that now we multiply with $\mathbf{X}$, the matrix containing all of our vectorized images in order to process all the images in one step. This multiplication will result in the vector $\mathbf{u}$ which will contain the averages of the top half pixels of all the vectorized images in $\mathbf{X}$.

Likewise we can construct a lower selection matrix with:

$$\mathbf{L} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \mathbf{I}_{M/2}$$

and equivalently obtain the average of the lower-half of the images as:

$$\mathbf{l} = \frac{1}{\frac{M}{2}N} \mathbf{1} \cdot (\mathbf{I}_{M/2} \otimes \mathbf{L}) \cdot \mathbf{X}$$

We can now construct the matrix $\mathbf{Z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{l} \end{bmatrix}$ and compute its covariance using $\mathbf{C} = \frac{1}{K}\left(\mathbf{Z} - \frac{1}{K}\mathbf{Z} \cdot 1_{K,1} \cdot 1_{1,K}\right) \cdot \left(\mathbf{Z} - \frac{1}{K}\mathbf{Z} \cdot 1_{K,1} \cdot 1_{1,K}\right)^{\top}$, which will be the covariance of the

upper and lower average[1]. The vectors $1_{K,1}$ and $1_{1,K}$ have the sizes denoted by their subscript and contain all ones. Multiplying **Z** with the first vector computes the mean of **Z**, and multiplication with the second one replicates that mean so that we can remove it from **Z** (remember that we need to remove the mean before we compute the covariance). Note that it is possible to obtain **Z** directly with one expression, but you probably have a headache by now so I'll skip that (if you feel like it, what's the expression for it?)

2.a

We can obtain the solution by using the following expression:

$$\bar{\mathbf{x}} = \left(\left(\left(\left(\left(\frac{\mathbf{1}_K}{K} \otimes \frac{\mathbf{1}_3}{3}\right) \otimes \mathbf{I}_N\right) \otimes \mathbf{I}_M\right) \cdot \mathrm{vec}\left(\mathbf{X}\right)\right)\right)^{(M)}$$

Where $\mathbf{1}_L$ is a row vector of length L whose elements are all ones, and equivalently $\mathbf{I}_L$ is an identity matrix of size L. Each of the matrices in the Kronecker products tells us how to combine each dimension. You can think of it as multiplying each of these matrices with its equivalent dimension to determine how to combine the variables along that dimension. The matrix $\mathbf{I}_M$, being the identity, maintains the structure of the first dimension, as does $\mathbf{I}_N$ for the second one, whereas the matrix $\frac{\mathbf{1}_3}{3}$ takes the average of the third dimension as does $\frac{\mathbf{1}_K}{K}$ for the fourth one. The vec-transpose transforms the result to an $M \times N$ matrix.

2.b

Using the intuition from above we can now repeat this process only we need to change the matrix corresponding to the third dimension so that it only selects the red channel (the first of the three variables along that dimension). We do so using:

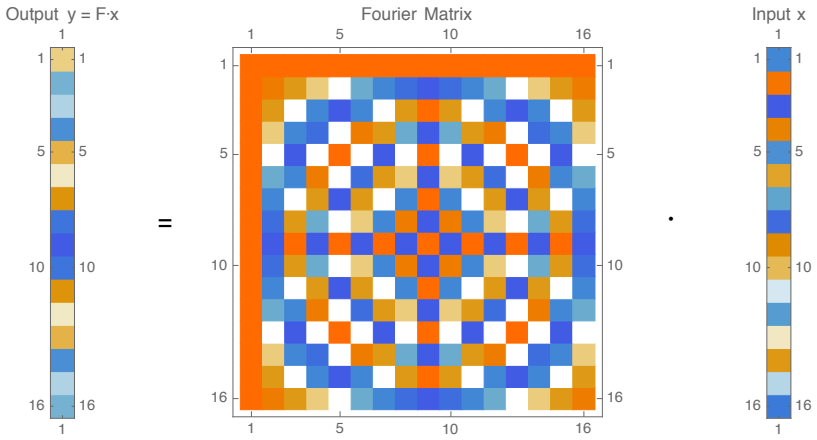
$$\bar{\mathbf{x}}_R = \left(\left(\left(\left(\left(\frac{\mathbf{1}_K}{K} \otimes \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\right) \otimes \mathbf{I}_N\right) \otimes \mathbf{I}_M\right) \cdot \mathrm{vec}\left(\mathbf{X}\right)\right)\right)^{(M)}$$

Had we wanted to select the blue channel we would use [0 0 1], etc.

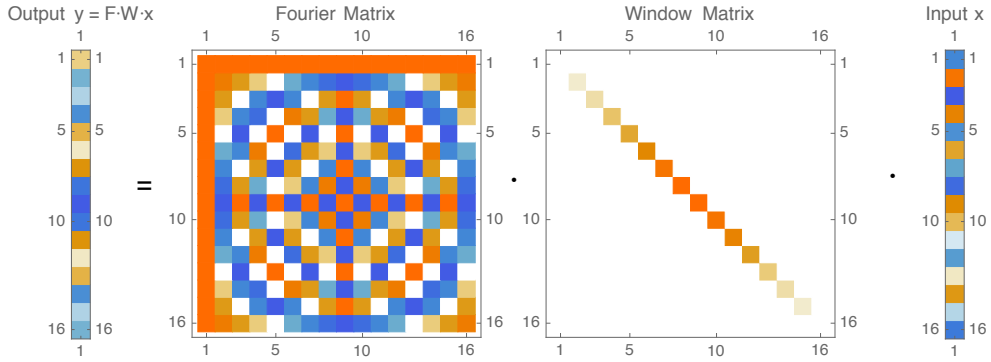
[1] To get an unbiased estimate of the covariance you should divide by K-1. Dividing by K will introduce a slight bias, but makes for less daunting formulas and doesn't result in a significantly different estimate in real-life.

Problem 3. Signal Processing is Linear Algebra

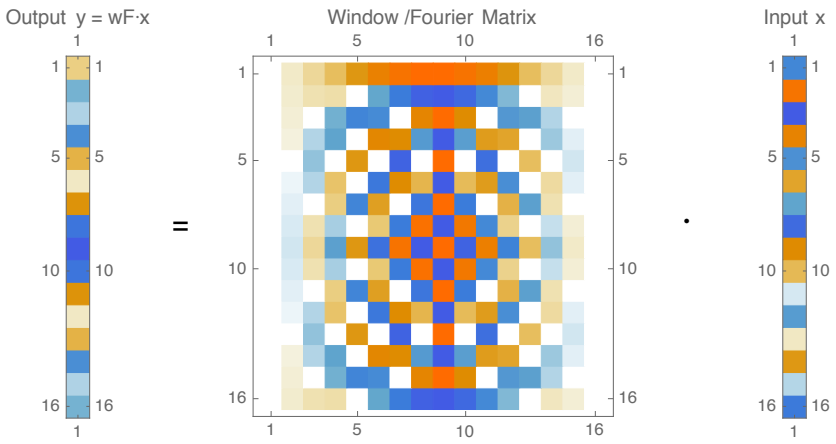
Let us consider the general case of a spectrogram using a Fourier transform size of length N and a hop size of N/2 (in the problem set N = 1,024, here we use N = 16 to make the plots more legible). Let us first get the window business out of the way. Suppose that the input vector **x** was 1,024 samples long, in which case we can only apply one Fourier transform. Ignoring the window, we would simply multiply **x** with the $N \times N$ Fourier matrix **F**:



If we were to add the window we would have to element-wise multiply the input vector **x** with a vector that contains a Hann window (a smooth taper from zero to one and back to zero). In order to do that we can left-multiply x with a diagonal matrix **W** that contains the window values along the diagonal, i.e.:

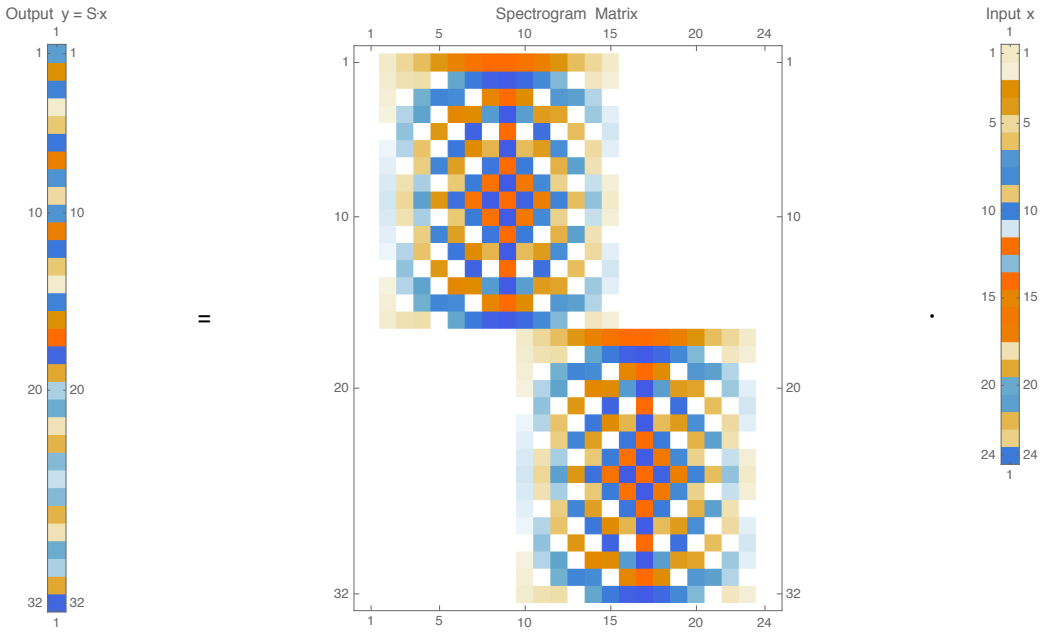


And we can combine the two matrices as one to simplify the operation:

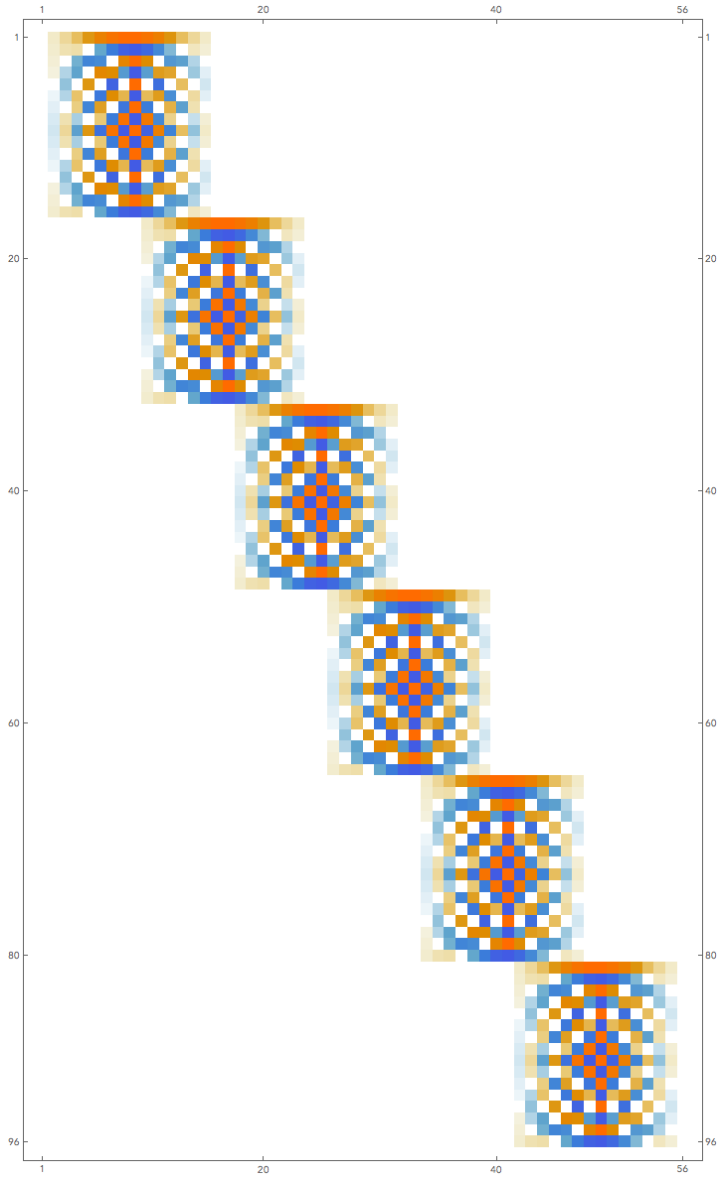


What happens now is that we taper the rows of the Fourier matrix using the window (hence the fading towards the sides). This in part will transfer to the **x** vector once we take the dot products, thereby applying the window on the subset of **x** that we analyze.

Now let's see how we can construct this a spectrogram matrix. Suppose that the length of the input vector x was 1.5 × *N* samples. that means that we would only be able to take two windowed DFTs, one starting at the first sample, and one starting at the *N*+1 sample. We would need to do so with a 2*N* × 1.5 *N* matrix. Performing the first windowed DFT is easy, it means that the first to N^{th} row and first to N^{th} column will contain the matrix above. These first *N* columns will multiply the first *N* samples of **x** and will produce the first *N* Fourier coefficients at the output. For the second transform we would need to place a Fourier matrix under the first one (so that the next *N* points will be the next set of output coefficients), but we would need it to operate on the $N+1^{st}$ to the last sample of **x**. That means that the matrix will span from the $N+1^{st}$ to 2 N^{th} row and from the $N/2+1^{st}$ to the last column. The overall operation will look like this, with each block in the spectrogram matrix being a separate windowed Fourier matrix:



In order to generalize to additional transforms for a longer vector **x** we would need to add more Fourier matrix blocks, which will be placed one under the other and each will be offset by $N/2$ columns to the right of the previous one. That multiplication will apply a windowed DFT on each frame of **x** thus resulting in the desired transformation. Shown below is a matrix for a longer vector **x**.



Bonus: In order to take the result of this transform and make it a spectrogram matrix we would have to reshape the resulting vector to a matrix. We can easily do that using the vec-transpose operator or order N. However, if we do that it would also plot the symmetric part of the spectrum (over the frequencies) which we don't care about. Therefore we need to also left multiply with a half identity matrix that will select only one half of the frequencies.

For those of you that were brave enough to try this on a real sound, you undoubtedly discovered that the spectrogram matrix was humongous and that the overall multiplication was extremely slow. This is because most of the storage and computations involved zeros and were unnecessary. You should instead form the spectrogram matrix as a sparse matrix (`help sparse`) which would eliminate all that redundancy.