1 Simple Matrix Problems

1.1

This would only work in a column vector with a same size. Let's say it's a size-N vector.

$$vec(\mathbf{a} \times \mathbf{b^T}) = b \otimes a$$

$vec(\mathbf{a} \times \mathbf{b^T})$ would have copies of $\mathbf{a}$ weighted with each element of $\mathbf{b}$ appended at the bottom, resulting in a column vector of size N*N. Since $\mathbf{a}$ is a column vector, the Kronecker product would result in same weighted copy in same shape.

1.2

$$vec(\mathbf{A})^T * vec(\mathbf{B}) = \mathbf{tr}(\mathbf{A^T} * \mathbf{B})$$

Left side is going to equal to a sum of multiples of all elements at their respective position.

$$\begin{bmatrix} A_{11} & A_{21} & ... & A_{1n} & A_{21} & A_{22} & ... & A_{2n} & ... & Amn \end{bmatrix} * \begin{bmatrix} B_{11} \\ B_{21} \\ ... \\ B_{1n} \\ B_{21} \\ B_{22} \\ ... \\ B_{2n} \\ ... \\ Bmn \end{bmatrix}$$

Right side, the trace operation is going to sum up all the diagonal entries. $\mathbf{tr}(\mathbf{A^T} * \mathbf{B})$ ends up something like this:
$\mathbf{A_{11}B_{11}} + \mathbf{A_{21}B_{21}} + ... + \mathbf{A_{1n}B_{1n}} + \mathbf{A_{12}B_{12}} + ... + \mathbf{A_{mn}B_{mn}}$

They are equivalent.

1.3

$\mathbf{x^T A A}^T\mathbf{x}$ is equivalent to $(\mathbf{A^T x})^\mathbf{T}(\mathbf{A^T x})$ which equals to norm of $(\mathbf{A^T x})$. Which is always larger than or 0. So, this is positive semi-definite.

2 Probability

$P(B) = 0.7\%$
$P(M \mid B) = 90\%$
$P(M \mid \neg B) = 8\%$
$P(M) = \frac{P(MB)P(B)}{P(B)} + \frac{P(M\neg B)P(\neg B)}{P(\neg B)} = 0.7\% * 90\% + 8\% * 99.3\%$
$P(B \mid M) = \frac{P(M|B)P(B)}{P(M)} = \frac{90\% * 0.7\%}{8.475\%} = \mathbf{7.34}\%$

3 Problem 3

3.1 Matrix Version

3.1.1

We start with a (M x N) by K matrix. Let's define few things:
- $[\mathbf{1}]_{\mathbf{n}}$ is a column vector with a size of n.
- Our image matrix is called **img**

The following operation would get the mean image in M by N format:

$$\frac{(\mathbf{img} * [\mathbf{1}]_{\mathbf{K}})^{\{M\}}}{K}$$

Multiplying the matrix of column image vectors with column of 1s would result in a summation of each image. Reshaping this sum would result in M by N image and dividing by K would return a mean image.

3.1.2 The hard one

The Problem is in two steps. We have to come up with a matrix with a size of 2 by K. This matrix would have column vectors where the entries are average of top and average of bottom of each image.

Let's name this matrix **X**. Here's how to get **X**:

1. Transpose **img**. This would result in K by (M x N) matrix.

2. Multiply $\mathbf{img}^T$ with $([\mathbf{1}]^{\mathbf{T}}_{\mathbf{M/2}} \otimes \mathbf{I} \otimes [\mathbf{1}]^{\mathbf{T}}_{\mathbf{N}})^T$

This would result in a sum of all the K by 2 matrix where each row vector is a sum of tops and sum of bottoms of each image.

3. Transpose the product so that it would be in 2 by K format.

4. Divide the transpose of the product by $\frac{M}{2} \times N$ to get the average.

Let's define another matrix $\mathbf{Y}$.
This matrix equals to $\mathbf{X} - \mathbf{X} \times [\mathbf{1}]^{\mathbf{T}}_{\mathbf{K}} \otimes \left[\frac{1}{K} \ \frac{1}{K}\right]^T$ which is a mean-subtracted $\mathbf{X}$.

Our final covariance matrix is then $\frac{1}{K}\mathbf{Y}\mathbf{Y}^T$

3.2 4D Tensor Version

3.2.1

We start with a 4D tensor with dimension of M x N x 3 x K. Let's name this **img**. We'll begin our series of operations with applying vec() on our tensor.

$$\mathbf{A} = vec(\mathbf{img})$$

Now $vec(\mathbf{img})$ is going to return a A, a matrix of size (M X N X 3) by K. We are going to multiply this with K by 1 column vector of 1s. This would return sum of all images at their respective pixel and channel.

$$\mathbf{V_{RGB}} = \mathbf{A} * \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix}_K$$

So now we have $\mathbf{V_{RGB}}$ which is a sum of all images at their respectivel pixel and channel. We need to reshape this so that we have (M X N) by 3 matrix. This matrix would have M x N image but color-sorted in each column. We can then multiply by a column vector of 1s similar to what we did before to sum up each pixel over all color channels.

$$\mathbf{V_{Greyscale}} = \mathbf{V_{RGB}} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Now we have a (M X N) column vector with each pixel summed over all images and color. We need to divide this by 3 * K to get the mean image.

$$\mathbf{V}_\mu = \mathbf{V_{Greyscale}}/(3 * K)$$

Vec-transposing this matrix by M would result in a mean image.

3.2.2

We start once again with applying vec() on our tensor.

$$\mathbf{A} = vec(img)$$

Again, *vec*(**img**) is going to return a A, a matrix of size (M X N X 3) by K. We are going to multiply this with K by 1 column vector of 1s. This would return sum of all images at their respective pixel and channel.

$$\mathbf{V_{RGB}} = \mathbf{A} * \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix}_K$$

So now we have $\mathbf{V_{RGB}}$ which is a sum of all images at their respectivel pixel and channel. Vec-transpose this by (M x N), then have (M X N) by 3 matrix. This matrix would have M x N image but color-sorted in each column.

This time we only want red, so let's multiply by column vector with 1,0,0 so that we only sum up red.

$$\mathbf{V_{Greyscale}} = \mathbf{V_{RGB}}^{M*N} * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Divide this matrix by K would and vec-transposing by M would result in a mean image.

4 This problem was good

We are given a vector X and we are supposed to come up with a matrix that when multiplied that returns a vec-operated coefficient matrix.
This is a two step process:

1. We need to apply a Hann Window on each frame.
2. Calculate DFT coefficients.

The number of frames we are going to get with an hop size of 512 and frame size of 2014 is ceiling of (Length of Sample - 1024) / 512 + 1. So we are going to have a (Number of Frames * Frame size) by Length of Sample matrix which is going to consist of mostly zeros but copy of diagnolized hanning window of size 1024 shifting by 512 columns every 1024 row. Multiplying our sample with this matrix would effectively split the sample by frames and apply window.
Once that's done, we just apply DFT by multiplying $\mathbf{I_{NumberofFrames}} \otimes$ DFT matrix of size of 1024 by 1024. with our previous Hanning matrix.

To get the spectogram, we just vec-transpose our matrix by 1024. Throw away rows above 1024/2 + 1 since they are repeated and plot their absolute value.

Following is my implementation of this idea with the complete DFT matrix and a sample spectogram of dense mixture of detuned saw waves, i.e. supersaw wave.

```
% Setting up
x = audioread("supersaw.wav");

frame_sz = 1024;
hop_sz = 512;
```

Generate windowing matrix

```
n_frames = ceil((length(x) - frame_sz) / hop_sz) + 1;

hann_matrix = diag(hann(frame_sz));
w_matrix = zeros(n_frames*frame_sz, length(x));

for i=0:n_frames-1
    col_start = i * hop_sz + 1;
    col_end = i * hop_sz + frame_sz;

    row_start = i * frame_sz + 1;
    row_end = i * frame_sz + frame_sz;

    w_matrix(row_start : row_end, col_start : col_end) = hann_matrix;
end
```

Generate DFT matrix

```
n_frames = ceil((length(x) - frame_sz) / hop_sz) + 1;
dft_matrix = kron(eye(n_frames), dftmtx(frame_sz));
```

Apply DFT

```
final = dft_matrix * w_matrix* x;
final = reshape(final, frame_sz, []);
final = final(1:(frame_sz/2)+1, :);
imagesc(abs(final));
```

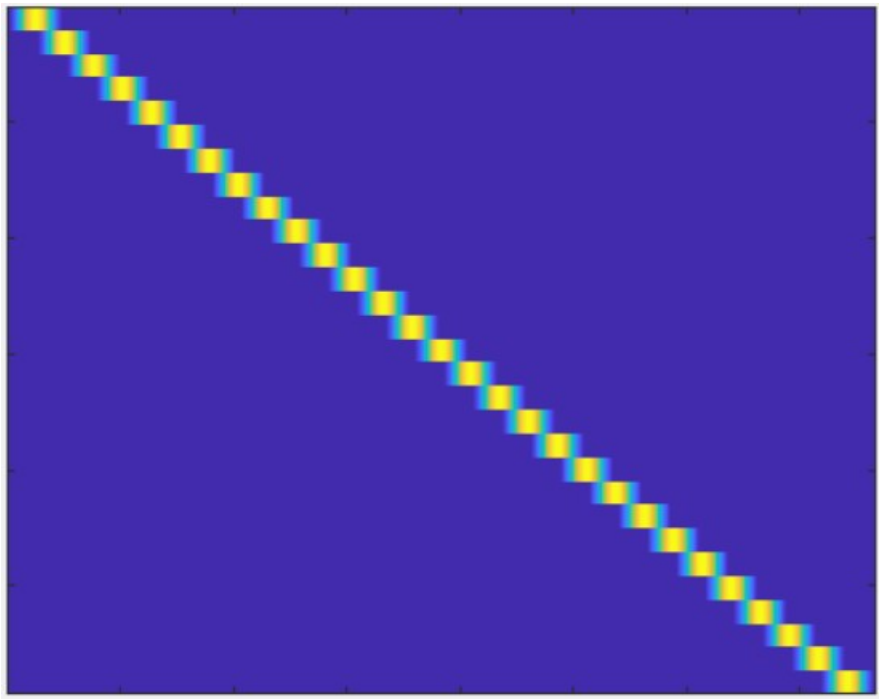

Figure 1: Our A matrix

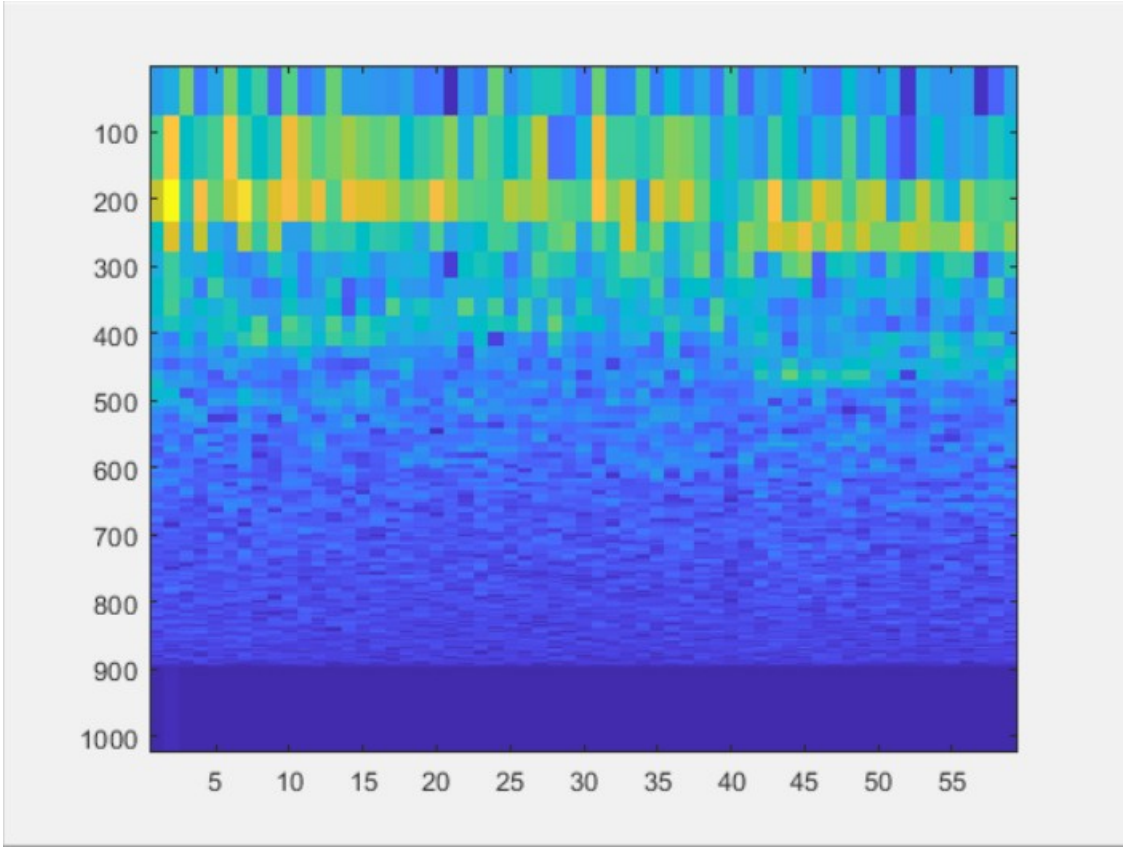

Figure 2: Supersaw Spectogram