Computational Linear Algebra – Programming Assignment 2

(points: 60, due on January 25)

1 Important points

- Submit crisp codes in a zip folder with folder name '*FIRSTNAME_LASTNAME_pga2*.zip'.
- Add adequate comments in the code to explain the algorithms.
- The code will be checked for correctness and plagiarism.
- Inbuilt commands should not be used unless specified.
- The programs have to be coded in MATLAB.
- Error between two vectors x and $y \in \mathbb{R}^n$ is the euclidean norm of their difference, i.e. square root of $\sum_{i=1}^{n} (x_i - y_i)^2$.
- Error between two matrices A and $B \in \mathbb{R}^{n \times n}$ is the frobenius norm of their difference.

$$\text{Error}(A, B) = \sqrt{\sum_{i,j=1}^{n} (A_{ij} - B_{ij})^2}.$$

- Error between two 3D arrays A and B in Matlab (having the same shape) is defined in a similar way:

$$\text{Error}(A, B) = \sqrt{\sum_{i,j,k} (A_{ijk} - B_{ijk})^2}.$$

2 Questions

2.1 Singular values of a matrix (6+8+5)

a) Write a program (*sym2tri.m*) to convert a symmetric matrix to tridiagonal matrix using a similarity transformation.

b) Write a program (*trieig.m*) to implement QR algorithm and thereby find eigenvalues of tridiagonal matrix. Note that QR decomposition should be performed using Householder transformation or method proposed in PGA1. Also the subroutine function to implement QR decomposition should be tailor-made for tridiagonal matrix. Print per iteration complexity for QR algorithm in this specific case.

Let $\hat{\sigma}(t)$ be maximum eigenvalue computed by above mentioned procedure in iteration t and σ_0 be maximum eigenvalue computed by inbuilt MATLAB function. Write a program (*Q2_1b.m*) to plot $\|\hat{\sigma}(t) - \sigma_0\|$ with iteration t for the matrix $A = \begin{bmatrix} 3 & 3 & 4 \\ 3 & 7 & 6 \\ 4 & 6 & 10 \end{bmatrix}$

c) Given a matrix of size $n \times n$, write a program (*findsing.m*) to compute its singular values using a) and b).

2.2 Application of Cholesky decomposition (6+10)

a) Write a program (*cholprog.m*) to find Cholesky decomposition of a symmetric positive definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. Hint: Cholesky decomposition is $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix.

b) We need to generate correlated random vectors, sampled from distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for given mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$. Write a program (*corrNRV.m*) to first generate M uncorrelated random vectors sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ (using *MATLAB* command *randn*) followed by a procedure to generate $\boldsymbol{y}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ from $\boldsymbol{x}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for $i \in \{1, 2 \ldots, M\}$ by applying Cholesky decomposition on $\boldsymbol{\Sigma}$. Note that this program is a widely used procedure to generate correlated random vectors from uncorrelated random vectors (not restricted to normal random vectors as well).

For $n = 2$ and 3, write a program (*Q2_2b.m*) to plot 1000 uncorrelated random vectors sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and output correlated vectors for fixed $\mu = \mathbf{0}$ and covariance matrix $\Sigma_1 = \begin{bmatrix} 0.025 & 0.0075 \\ 0.0075 & 0.007 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 0.025 & 0.0075 & 0.00175 \\ 0.0075 & 0.007 & 0.00135 \\ 0.00175 & 0.00135 & 0.00043 \end{bmatrix}$

2.3 Application: Image compression using Principal Component Analysis (25)

Download the image 'watch.bmp' provided with this assignment. Create a file named *pcacomp.m* and write code to implement the following. (Note: The following steps are not separate questions, so you don't need to create separate files for each step.)

1. An image is a rectangular array of pixels. Read the image in Matlab as: `I = im2double(imread('watch.bmp'));`. The image is of size 768×1024 pixels, and contains 3 color channels – red, green and blue. Hence, `I` is an array of size $768 \times 1024 \times 3$. Extract the three color channels separately as follows:

```
R = I(:,:,1);    % Red channel
G = I(:,:,2);    % Green channel
B = I(:,:,3);    % Blue channel
```

   Each of `R`, `G` and `B` is a matrix of size 768×1024.

2. Break each channel matrix into 8×8 contiguous blocks. Thus, you will get a total of $\frac{768\times1024}{8\times8} = 12288$ blocks for each channel. Reshape each 8×8 block into a 64×1 vector. Each such vector is a point in $\mathbb{R}^{64}$. Therefore, you now have $N = 12288$ points in $\mathbb{R}^{64}$ corresponding to each channel. We will call these points as 'data samples'. Let us denote the i-th data sample corresponding to the c-th channel by $\boldsymbol{x}_{i,c}$, where $i \in \{1, \ldots, N\}$ and $c \in \{R, G, B\}$.

3. For each channel, calculate the 'sample mean':

$$\boldsymbol{\mu}_c = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_{i,c}.$$

4. For each channel, calculate the 'sample covariance matrix':

$$\boldsymbol{\Sigma}_c = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c)(\boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c)^\top.$$

   Note that $\boldsymbol{\Sigma}_c$ is a symmetric matrix of size 64×64. Hence, it has a set of 64 orthogonal eigenvectors. Moreover, $\boldsymbol{\Sigma}_c$ is positive semidefinite, so its eigenvalues are non-negative.

5. For each channel, calculate the orthonormal eigenvectors of $\mathbf{\Sigma}_c$, and sort them in descending order of their corresponding eigenvalues. You are free to use the following code snippet for this task.

```
C = ⟨Covariance matrix⟩
% The eig() command returns the normalized eigenvectors and
% eigenvalues of a matrix. The columns of V contain the
% eigenvectors, and Lambda is a diagonal matrix containing the
% corresponding eigenvalues. But these are not sorted by default.
[V,Lambda] = eig(C);
Lambda = diag(Lambda);  % Extract the diagonal elements
% Sort the eigenvalues in descending order, and get the indices.
[Lambda,indices] = sort(Lambda,'descend');
% Reorder the eigenvectors in the order given by the indices.
V = V(:,indices);
% Now you can use V(:,k) to get the k-th largest eigenvector of C.
```

   Let $\boldsymbol{v}_{1,c}, \ldots, \boldsymbol{v}_{64,c}$ denote the orthonormal eigenvectors of $\mathbf{\Sigma}_c$, in descending order.

6. We first explain the concept of Principal Component Analysis (PCA). For each channel c, note that $\boldsymbol{v}_{1,c}, \ldots, \boldsymbol{v}_{64,c}$ forms an orthonormal basis of $\mathbb{R}^{64}$. (These vectors are known as the 'principal components' of the data samples.) PCA refers to the representation of the data samples with respect to this basis, instead of the standard basis of $\mathbb{R}^{64}$. For each $i = 1, \ldots, N$ we know that

$$\boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c = \sum_{k=1}^{64} \langle \boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c, \boldsymbol{v}_{k,c} \rangle \boldsymbol{v}_{k,c},$$

   where $\langle \cdot, \cdot \rangle$ is the standard dot product. Equivalently,

$$\boldsymbol{x}_{i,c} = \boldsymbol{\mu}_c + \sum_{k=1}^{64} \langle \boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c, \boldsymbol{v}_{k,c} \rangle \boldsymbol{v}_{k,c}.$$

   Now let $K = 5$, and approximate $\boldsymbol{x}_{i,c}$ by using only the first K terms of the above summation. That is, let

$$\tilde{\boldsymbol{x}}_{i,c} = \boldsymbol{\mu}_c + \sum_{k=1}^{K} \langle \boldsymbol{x}_{i,c} - \boldsymbol{\mu}_c, \boldsymbol{v}_{k,c} \rangle \boldsymbol{v}_{k,c}.$$

   Calculate $\tilde{\boldsymbol{x}}_{i,c}$ for all i and all c. Reshape each 64×1 vector $\tilde{\boldsymbol{x}}_{i,c}$ into a 8×8 block and create an image `J` of size $768 \times 1024 \times 3$ by putting each block into its appropriate location. Calculate and print the error between `I` and `J`. Display both the images `I` and `J` using the following commands:

```
figure; imshow(I); title('Original image');
figure; imshow(J); title('Compressed image, K=5');
```

   Now repeat this for $K = 10$ and $K = 20$. Note that as the number of principal components used for the approximation (K) increases, the reconstructed image looks more similar to the original image. (You can use the following test to check if your code is correct: If you use $K = 64$ you should get back the original image `I`, upto roundoff error.)

7. Note that the amount of numerical data needed to store `J` depends on the value of K. As K increases, the storage size increases. Therefore, as long as the error between `I` and `J` is small, we can achieve data compression by choosing a low value of K. Plot the error between `I` and `J` against the value of K, for $K = 1, \ldots, 64$. Comment on whether the error monotonically increases/decreases with K, or whether there is no such relation.