Assignment #4

Due: 23:55pm March 4, 2020

Upload at: https://www.gradescope.com/courses/75501/assignments/369262

Remember to append your Colab PDF as explained in the first homework, with all outputs visible.
When you print to PDF it may be helpful to scale at 95% or so to get everything on the page.

Problem 1 (13pts) (A) Compute an orthonormal basis of the kernel of

$$A = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

(B) Write down an orthonormal basis for the image of A.

Problem 2 (20pts)

You've encountered power series before in other classes, but one thing you may not've realized is that you can construct *matrix functions* from *matrix power series*. That is, if you have a function $f(\cdot)$ that has a convergent power series representation:

$$f(x) = \sum_{i=0}^{\infty} a_i x^i$$

then you can generally write a similar matrix version for square symmetric matrices $\boldsymbol{X}$ using the same a_i:

$$F(\boldsymbol{X}) = \sum_{i=0}^{\infty} a_i \boldsymbol{X}^i$$

(A) The matrix version F turns out to just apply the scalar f to each eigenvalue independently. Explain why. (Hint: How would a diagonalized version of $\boldsymbol{X}$ interact with the power series?)

(B) In power series there is a notion of radius of convergence. How would you expect this concept to generalize to square symmetric matrices?

(C) One important example is where the function $f(x)$ is the exponential function. I can take any square symmetric matrix and if I compute its matrix exponential, I get a positive definite matrix. Explain why.

(D) These kinds of matrix functions lead to some interesting computational tricks. For example: if I have a positive definite matrix $\boldsymbol{A}$ and I take the *trace* of the *matrix logarithm* (assuming it exists), what quantity have I computed?

Problem 3 (20pts)

In general, computing the determinant of an $n \times n$ matrix scales as n^3 in computational cost. When the matrix is highly structured, however, it can sometimes be possible to take advantage of that structure to save computation for quantities such as the determinant. One example of such structure is in *tridiagonal* matrices, which look like this:

$$\boldsymbol{T} = \begin{bmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ c_1 & a_2 & b_2 & 0 & & 0 \\ 0 & c_2 & a_3 & \ddots & & \vdots \\ 0 & 0 & \ddots & \ddots & b_{n-2} & 0 \\ \vdots & & & c_{n-2} & a_{n-1} & b_{n-1} \\ 0 & 0 & \cdots & 0 & c_{n-1} & a_n \end{bmatrix}$$

Such matrices can come up when simulating a physical system with local structure, e.g., a spring-mass system.

(A) We would like to compute the determinant of the matrix $\boldsymbol{T}$ above, which we are assuming is invertible. Let d_m denote the determinant of the upper left $m \times m$ submatrix. So, $d_1 = a_1$ and $d_n = |\boldsymbol{T}|$ (computing the whole determinant). Use expansion by minors to compute d_n in terms of d_{n-1}, d_{n-2}, and any of the a_i, b_i, or c_i. If necessary you can take $d_0 = 1$ and $d_i = 0$ for $i < 0$. Carefully show how you arrived at your answer.

(B) Based on this recurrence relation, how would you expect the computational cost to scale for computing the determinant of a tridiagonal matrix?

Problem 4 (25pts)

One of the single most important algorithms in data analysis is principal component analysis or PCA. PCA tries to find a way to represent high-dimensional data in a low-dimensional way so that human brains can reason about it. It tries to identify the "important" directions in a data set and represent the data just in that basis. PCA does this by computing the empirical covariance matrix of the data (we'll learn more about that in a couple of weeks), and then looking at the eigenvectors of it that correspond to the largest eigenvalues.

(A) Load `mnist2000.pkl` into a Colab notebook. Take the $2000 \times 28 \times 28$ tensor of training data and reshape it so that it is a 2000×784 matrix, where the rows are "unrolled" image vectors. Typically in PCA, one first centers the data. Center the data by subtracting off the mean image; you did a very similar procedure in HW2.

(B) Now compute the "scatter matrix" which is the 784×784 matrix you get from multiplying data matrix by its transpose, making sure that you get it so the data dimension is the one being summed over.

(C) This scatter matrix is square and symmetric, so use the `eigh` function in the `numpy.linalg` package to compute the eigenvalues and eigenvectors. Plot the eigenvalues in decreasing order.

(D) Read the documentation for `eigh` and figure out how to get the "big" eigenvectors. For each of the top five eigenvectors, reshape them into 28×28 images and use `imshow` to render them.

(E) Now, create a low-dimensional representation of the data. Take the 2000×784 matrix and multiply it by each of the top two eigenvectors. This takes all 2000 data, each of which are 784-dimensional, and gives them two-dimensional coordinates. Make a scatter plot of these two-dimensional coordinates.

(F) That scatter plot doesn't really give you much of a visualization. Here's some starter code to build a more interesting figure. It takes the two-dimensional projection and builds a "scatter plot" where the images themselves are rendered instead of dots. Here I have the projections in a 2000×2 matrix called `proj`, which I modify so that all the values are in $[0, 1]$.

```
# Make the projections into [0,1]
proj = proj - np.min(proj, axis=0)
proj = proj / np.max(proj, axis=0)

# Create a 12" x 12" figure.
viz_fig = pl.figure(figsize=(12.,12.))

# Get the figure width and height in pixels.
width, height = viz_fig.get_size_inches()*viz_fig.dpi

pl.plot() # Colab seems to require this to render.

# Loop over images.  Could do all 2000 but it's crowded.
for ii in range(400):
  # Render each image in a location on the figure.
  pl.figimage(train_images[ii,:,:],
              xo=proj[ii,1]*width,
              yo=(proj[ii,0]*height-150), # hack to make visible
              origin='upper')
```

Modify this code to work with your projections and make a visualization of the MNIST digits. Do you see any interesting structure?

Problem 5 (20pts)

One of the themes in COS 302 has been the idea that vector spaces can be about more than $\mathbb{R}^n$. An example of a more complicated vector space defined by the set of Chebyshev polynomials. These polynomials are often defined by the following recurrence relation:

$$T_0(x) = 1 \qquad T_1(x) = x \qquad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)\,.$$

The Chebyshev polynomials provide a basis for functions in $[-1, 1]$. The inner product in this space is defined as:

$$\langle f(x), g(x) \rangle = \int_{-1}^{1} f(x)\,g(x)\,\frac{dx}{\sqrt{1-x^2}}\,.$$

(A) Use Matplotlib to plot the first 6 Chebyshev polynomials on $[-1, 1]$.

(B) Show that $T_0(x)$ and $T_1(x)$ are orthogonal.

(C) The Chebyshev polynomials can also be written as $T_n(x) = \cos(n\arccos(x))$, without directly requiring the recurrence relation. Use this representation to argue that all of the Chebyshev polynomials are orthogonal to each other.

(D) What is $\langle T_0(x), T_0(x) \rangle$? What is $\langle T_n(x), T_n(x) \rangle$ for $n > 0$?

(E) What are the coordinates of the function $|x|$ in this basis on $[-1, 1]$? (You'll need to take a bit of care to get the constants right using the result of (D) above.)

(F) The representation in this (infinite) basis provides a powerful tool for approximation by giving an infinite sum representation of $|x|$, much like Taylor or Fourier series. Plot three different truncations of the sum out to $n = 10, 25, 50$. That is, plot the resulting "vector" represented with only 10, 25, and 50 components of the $T_n(x)$ "basis".

Problem 6 (2pts)

Approximately how many hours did this assignment take you to complete?

My notebook URL: `https://colab.research.google.com/XXXXXXXXXXXXXXXXXXXXXXX`

Changelog

- 23 February 2020 – Initial version.
- 24 February 2020 – Clarified problem two.