Assignment 6

SE2324: Mathematical Foundation of Computer Sciences(Spring 2021)

School of Software, Shanghai Jiao Tong University

June 1, 2021

1 Goals and Rules

In this exercise, you need to complete three tasks. In these tasks, you need to apply what you learned about the eigenproblems, SVD and nonlinear system.

Setup Please setup the Python environment and install the following packages: NumPy, SciPy, Matplotlib.

Submission Compress your Python scripts and the document(.pdf), and name it in form *NAME_ID_ex2.zip*, then submit it in Canvas. If you use Jupyter Notebook, it's still necessary to export a PDF of a clearly documented Jupyter Notebook that shows your work.

2 Eigenvectors (40 points)

Guitar String Vibrations In this question, we will study vibrations on a string.

The Helmholtz equation is useful for modeling the standing wave vibrations on a guitar string:

$$\frac{d^2}{dx^2}y(x) + k^2 y(x) = 0. \tag{1}$$

In this equation, x is the location along the string, y is the perpendicular displacement, and $k > 0$ can be any positive real. Our string has length 1, i.e. $x \in [0, 1]$. Both ends of the string are clamped, with boundary conditions $y(0) = y(1) = 0$, which should simplify our problem.

Analytical Spectral Properties of the Laplacian operator Let C be the set of twice-differentiable real functions $f(x)$ for $x \in (0, 1)$. We can consider C a vector space: for functions $f(x)$ and $g(x)$, their sum $h(x) = f(x) + g(x)$ is in this space, and so is any scalar multiple $h(x) = af(x)$. We also define an inner product,

$$\langle f, g\rangle = \int_0^1 f(x)g(x)dx.$$

The Laplacian operator is therefore a linear transformation M, which takes a function $f \in C$ as input and outputs its second derivative:

$$Mf = \frac{d^2}{dx^2}f$$

2.1 (8 points) Verify that functions of the form $\psi_n(x) = \sqrt{2}\sin(\pi n x)$, for positive integer n, are eigenvectors of the Laplacian with eigenvalue $\lambda_n = -n^2\pi^2$. (I.e. verify $M\psi_n(x) = \lambda_n\psi_n(x)$)

2.2 (8 points) Verify that these eigenvectors $\{\psi_n\}$ are orthonormal, i.e.

$$\int_0^1 \psi_n(x)\psi_m(x)dx = \begin{cases} 1 & \text{if } n = m \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Hint: Please answer Question 2.1 and 2.2 with the mathematical induction. You can use the trigonometric identity: $2\sin u \sin v = \cos(u - v) - \cos(u + v)$.

Discrete Solutions Similar to the previous assignment, we can discretize $y(x)$ over a grid of $N = 99$ interior values, with the first value $x_1 = 0.01$ and the last value $_{99} = 0.99$. As before, $y(0) = y(1) = 0$. Denote the spacing as $h = \frac{1}{N+1}$. For any function $f(x)$, we will use the notation $\vec{f}$ to denote $(f(x_1), f(x_2), ..., f(x_N))^T$

Given the discrete derivative of a function using central difference as

$$f'(x) = \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h}, \tag{3}$$

we can get the discrete form of the Laplacian operator by applying this equation twice

$$\begin{aligned} f''(x) &= \frac{f'\left(x + \frac{h}{2}\right) - f'\left(x - \frac{h}{2}\right)}{h} \\ &= \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \end{aligned} \tag{4}$$

With the 1D Laplacian matrix A of dimension $N \times N$

$$A_{ij} = \begin{cases} -2 & \text{if } i = j \\ 1 & \text{if } |i-j| = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

we have $\overrightarrow{f''} = h^{-2}A\vec{f}$. We will now use M to denote $h^{-2}A$.

2.3 (12 points) Construct M and $\vec{y}$ in Python for $y(x) = \psi_1(x)$ and verify that $\vec{y}$ satisfies the Helmholtz equation 1.

2.4 (12 points) Use the NumPy function *linalg.eig*() to find the eigenvectors of the Laplacian, print the 3 eigenvalues with the smallest magnitude, and plot the eigenvectors corresponding to them. The eigenvalues should be close to $-\pi^2, -4\pi^2$, and $-9\pi^2$, and the eigenvectors should look like ψ_1, ψ_2, and ψ_3, as you've shown in Question 2.1.

3 SVD Decomposition (30 points)

Image Compression In this question, we will use SVD in the context of image processing. To simplify the problem, we focus on the grey-scale image. The basic idea is to decomposite the image matrix using SVD and discard those components with small magnitudes. With only the large components, the image can be saved in lower cost and recovered without losing too much detail.

Let A be any $m \times n$ image matrix. Each element represents the grey-scale of one pixel in the original image. A singular value decomposition of A is a factorization of the following form:

$$A = U\Sigma V^T$$

where

- U is an $m \times m$ orthogonal matrix,
- V is an $n \times n$ orthogonal matrix,
- Σ is an $m \times n$ diagonal matrix.

Moreover, it is assumed that diagonal entries of Σ are non-negative and we have $\Sigma_{ii} = \sigma_i$ with

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_p \geq 0$$

where $p = \min(m, n)$. We call σ_i singular values of A. Let $r \leq p$ be the number of non-zero singular values of A. Then, $\text{rank}(A) = r$.

Then, the non-zero $m \times n$ matrix A of rank r can be constructed from its singular values $\{\sigma_1, \ldots, \sigma_r\}$, left singular vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$, and right singular vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ as

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

Further, we can approximate A with only a part of singular values and singular vectors. For $k \leq r$, denote A_k by

$$A_k := \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

Note that when σ_i is very small, the term $\sigma_i \mathbf{u}_i \mathbf{v}_i^T$ in the above sum becomes negligible. Moreover, since singular values of A are in descending order (stored as diagonal entries of Σ in $A = U\Sigma V^T$), once σ_k becomes sufficiently small, we can stop at that value of k and use A_k as a reasonable approximation to A.

3.1 (10 points) Load the sample image in Appendix A using Python Imaging Library (See details in Appendix B). Compute the singular value decomposition of A using NumPy API *linalg.svd*().

3.2 (10 points) For $k = 2, 4, 8, 16, 32, 64, 128, p$, construct A_k and display the corresponding (approximate) image; you will need to report the rank k in the title of the corresponding figure. Organize your images in a 4×2 grid in document.

3.3 (10 points) Make a plot of singular values of A to see how the size of singular values drop at those point (this gives you an idea of for what k, A_k is a reasonably close approximation to A).

4 Nonlinear System (30 points)

Implement the Newton's method and Secant method to find the roots of the following polynomial:

$$p(x) = x^5 - \frac{29x^4}{20} + \frac{29x^3}{36} - \frac{31x^2}{144} + \frac{x}{36} - \frac{1}{720}$$

on the interval $x \in [0, 1]$. You must implement Newton's method and Secant method in Python by yourself. (Calling the existing functions in package is not allowed.)

4.1 (10 points) Make a plot of the polynomial on the interval $[0, 1]$. How many unique roots are there?

4.2 (10 points) Search the root using Newton's method with the initial guess $x_0 = 0.45$. What is the root of the polynomial at least 10 digits of accuracy?

4.3 (10 points) Search the root using Secant method with the initial guess $x_0 = 0.45$ and $x_{-1} = 0$. What is the root at least 10 digits of accuracy? Is it same with the Newton's method?

Appendix A Sample Image

Figure 1 *Sample image http://ndevilla.free.fr/lena/*

Appendix B Read and write image in Python

```
# import
import numpy as np
import PIL
import PIL.Image

# open image in grey-scale
img = PIL.Image.open("lena.jpg").convert('L')
img_seq = img.getdata()
# reshape and normalize to [0,1]
img_arr = np.array(img_seq).reshape(256,256)*(1.0/255)

# do sth...

# new_arr is a 256x256 matrix with the values in [0,1]
y=np.asarray(new_arr*255,dtype=np.uint8)
w=PIL.Image.fromarray(y,mode='L')
w.save("out.png")
```