Assignment 5

SE2324: Mathematical Foundation of Computer Sciences(Spring 2021)

School of Software, Shanghai Jiao Tong University

May 21, 2021

1 Goals and Rules

In this exercise, you need to complete three tasks. The first one helps you get familiar with the NumPy and Matplotlib package in Python. In the following two tasks, you need to apply what you learned about the linear system and decomposition method.

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_ex1.zip*, then submit it in Canvas.

2 NumPy Warm-up (15 points)

Consider the following one-dimensional *Gaussian probability distribution*, also known as the Normal distribution or bell curve distribution:

$$G(x \mid \mu, \sigma) = \frac{1}{Z} \exp\left[-\frac{1}{2}\frac{1}{\sigma^2}(x-\mu)^2\right] \quad \text{where } Z = \sqrt{\frac{\pi}{\frac{1}{2}\frac{1}{\sigma^2}}}$$

Here, the function $G : \mathbb{R} \to \mathbb{R}$ takes as input a scalar x, and produces as output a scalar. The particular bell shape of G is determined by two parameters: the mean $\mu \in \mathbb{R}$ and the variance $\sigma^2 \in \mathbb{R}$

Numerically verify the following identity in Python:

$$\int_{\mathbb{R}} G(x)dx = 1$$

2.1 (10 points) Choose a few different one-dimensional Gaussian functions (by choosing different mean and variance values), plot them.

2.2 (10 points) Verify the above identity for each Gaussian function.

Hint: You do not actually need to numerically integrate over all the real numbers. A Gaussian function is close to zero almost everywhere (except near its mean), so just choose a finite integration domain you think is reasonable.

Hint: NumPy is a fundamental package for scientific computing in Python. It provides multidimensional array(NumPy array), various derived objects and an assortment of routines for fast operations on NumPy arrays, including mathematical, logical, shape manipulation, basic linear algebra and much more. Most of linear algebra operators are vectorized in NumPy, which means a single operator can be applied to a scalar, a vector or even a matrix. Vectorization not only helps you implement your algorithm in a short and elegant way, but also speed up the Python code with the underlying optimizations. We've list some of them in Appendix A. More details about NumPy can be found in [2].

Hint: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. [1] is a basic tutorial to help you get started with Matplotlib quickly.

3 Numerics and Linear Algebra (60 points)

In this question, you will explore how the condition number of a matrix can have practical influence on which algorithms (e.g. LU, Cholesky) you can use to solve linear systems of the form: $A\vec{x} = \vec{b}$ where $A \in \mathbb{R}^{n\times n}$. We will study the Vandermonde matrix, as well as a matrix constructed from a finite Fourier Series basis. In this question, you will interpolate samples of the analytical function

$$f(x) = \frac{1}{1+x^2}$$

for $x \in [0, 1]$. Your monomial interpolants (with $N = n + 1$ terms) are given by

$$g_V(x) = \sum_{j=0}^{N} c_j x^j$$

and

$$g_F(x) = \sum_{j=1}^{N/2} c_j \sin(j\pi x) + \sum_{j=N/2+1}^{N} c_j \cos((j - N/2)\pi x).$$

Use $M = m + 1$ uniformly sampled positions

$$x_i = ih,\ i = 0 \ldots m,$$

with spacing $h = 1/m$, to generate M samples of the test function f,

$$f_i = f(x_i),\ i = 0 \ldots m.$$

To estimate the polynomial coefficients $\vec{c}$, you will assemble and solve the linear systems

$$V\vec{c} = \vec{f}$$

and

$$F\vec{c} = \vec{f}$$

where V is the M-by-N Vandermonde matrix with entries

$$V_{ij} = (x_i)^j$$

and F is the finite Fourier Series basis matrix with entries

$$F_{i,j-1} = \begin{cases} \sin(j\pi x_i), & \text{if } 1 \le j \le N/2 \\ \cos((j - N/2)\pi x_i), & N/2 + 1 \le j \le N \end{cases}$$

For simplicity, assume $M = N$ for the entire problem.

3.1 (25 points) Use an LU solve (*scipy.linalg.lu* from SciPy package) to estimate the monomial coefficients $\vec{c}$. Report the residual L2 norm for both linear systems when $N = 8$ and $N = 16$.

Hint: You can solve a linear system $A\vec{x} = \vec{b}$ using NumPy API *numpy.linalg.solve*().

Hint: Given a problem $A\vec{x} = \vec{b}$, the residual L2 norm is $||r||_2 = ||A\vec{x} - \vec{b}||_2$.

3.2 (10 points) Using the *numpy.linalg.cond* function in NumPy, plot N vs. *cond*(V) and N vs. *cond*(F) for $N = 4, 6, 8, \ldots 32$. Write a couple of sentences explaining the reasons for the trends in these two plots. Hints: Use a logarithmic scale in y axis for better clarity. Also, try wrapping the creation of V and F into functions that you can call repeatedly to generate the required output data. These functions will be helpful for the next part.

3.3 (15 points) A necessary condition for being able to use Cholesky factorization is that the matrix must be positive definite. Construct $A_V = V^T V$ and $A_F = F^T F$ for $N = 4, 6, \ldots 32$ (a total of 30 matrices). Mathematically, when would these matrices be positive definite? Explain. Using the *isposdef*() function introduced in Appendix B, check to which matrices NumPy reports as positive definite. Create a table of values that includes the following columns: N, *isposdef*(A_V), *isposdef*(A_F), cond(V), cond(F). What is the largest value of N where A_V is positive definite, and what is the condition number of that V? What is the largest value of N where A_F is positive definite, and what is the condition number of that F? Are these condition numbers connected in some way? If so, how?

3.4 (10 points) For $N = 8$, transform the linear systems above into positive definite systems according to Question 3.3, and use Cholesky factorization (*numpy.linalg.cholesky*()) to solve them. Report the residual L2 norm for each solution. Compare the residuals to Question 3.1: how does Cholesky compare to LU?

4 Least Squares Problems and QR (25 points)

In the question above, a $M \times M$ square linear system is solved for the interpolation coeffcients. The resulting interpolation function can provide a solution that pass through all the sample points. However, in some applications, finding such a solution could be too time-consuming and unnecessary, or even impossible (Consider two sample with the same x value and different y values). In such cases, we usually reduce the number of the interpolants (let $M > N$), and solve the resulting over-determined linear system using least square method. Instead of looking for the exact solution to an over-determined system, we look for the solution with the smallest error vector $V\vec{c} - \vec{f}$ (or $F\vec{c} - \vec{f}$) by minimizing the overall backward error:

$$\text{minimize } \|V\vec{c} - \vec{f}\|_2^2$$

4.1 (15 points) Solve the least square system with QR decomposition(*numpy.linalg.qr*()) when $M = 16$, $N = 4, 8$.

4.2 (10 points) Plot the g_V, g_F when $M = 16$, $N = 4, 8$, compare them with the analytical function $f(x)$ and the interpolation function obtained in Question 3.1.

References

[1] Matplotlib. Usage guide. `https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py`, 2021.

[2] NumPy. Numpy documentation. `https://matplotlib.org/stable/contents.html#`, 2021.

Appendix A NumPy Vectorized Operator

Array Creation

```
# https://numpy.org/doc/stable/reference/routines.array-creation.html
arr=np.arange(start=0,end=1,step=0.25)
# [0, 0.25, 0.5, 0.75]

arr=np.linspace(start=0,end=1,num=5)
# [0, 0.25, 0.5, 0.75, 1]

arr=np.ones((3))
# [0, 0, 0]

arr=np.zeros((2,2))
# [[0, 0],
# [0, 0]]

arr=np.eye(2)
# [[1, 0],
# [0, 1]]
```

Indexing

```
# https://numpy.org/doc/stable/reference/arrays.indexing.html
# A = [[1, 2, 3],
#      [4, 5, 6],
#      [7, 8, 9]]

a01 = A[0,1] # a01=2
c0 = A[:,1]  # c0=[2,5,8]
x_idxs = np.asarray([0,1,2])
y_idxs = np.asarray([0,1,2])
diag = A[x_idxs,y_idxs] # diag=[1,5,9]

A[x_idxs,y_idxs] = 0
# A = [[0 2 3]
#       [4 0 6]
#       [7 8 0]]
```

Linear Algebra

```
# https://numpy.org/doc/stable/reference/routines.linalg.html
# A = [[1 2]
#       [3 4]]
# x = [0.1 0.2]
# y = [0.3 0.4]

z = np.sum(A)       # z = 10
z = x*y             # z = [0.03 0.08]
```

```
z = np.dot(x,y)     # z = 0.11
z = A*x             # z = [[0.1 0.4] [0.3 0.8]]
z = np.matmul(A,x)  # z = [0.5 1.1]
```

Appendix B Positive Definite Matrix

```
def is_pos_def(A):
    return np.all(np.linalg.eigvals(A) > 0)
```