Homework 6 Numerical Analysis Fall 2024

Instructions:

- Due 11/25 at 6:00pm on Gradescope.
- You must follow the submission policy in the syllabus

Problem 1. Suppose we have time-series data $(t_1, y_1), \dots, (t_n, y_n)$. We can try to fit the data with a polynomial of degree k. I.e. find a polynomial

$$p(x) = c_0 + c_1 x + \cdots + c_k x^k$$

so that at each time t_i, we have

$$p(t_i) \approx y_i.$$

To do this, we can solve a least squares problem

$$\min_{c_0,\dots,c_k} \sum_{i=1}^{n} (y_i - p(t_i))^2 = \min_{c_0,\dots,c_k} \sum_{i=1}^{n} (y_i - (c_0 + c_1 t_i + \cdot + c_k t_i^k))^2.$$

As we saw in class, this can be written as a linear algebra problem:

$$\min_{\mathbf{c}\in\mathbb{R}^{k+1}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2^2$$

where

$$\mathbf{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^k \\ 1 & t_2 & t_2^2 & \cdots & t_2^k \\ \vdots & \vdots & & \vdots & \\ 1 & t_n & t_n^2 & \cdots & t_n^k \end{bmatrix}.$$

Get the data `temp.npy` from the website. Load the data:

```
temp = np.load('gdrive/MyDrive/na_f2024/hw files/temp.npy')
t = np.arange(190)/24 # time in days
```

We can plot the data:

```
plt.subplots(1,1,figsize=(12,4))
plt.plot(t,temp,marker='.',ls='None',label='data')
plt.ylabel('temperature (F)')
plt.xlabel('time since Oct 29 (days)')
plt.legend()
```

(a) For each $k = 5, 10, 15$, set up the least squares problem for and and solve it (either using `np.linalg.lstsq` or using a QR factorization followed by a triangular solve).

Add each of the polynomials (evaluated at a finer grid of t values) to the plot. Make sure they are labeled.

(b) Note that we could represent our polynomial in terms of a different basis. I.e. instead of $1, x, x^2, \ldots, x^k$, we could use any family $p_0(x), p_1(x), \ldots, p_k(x)$, where $p_i(x)$ has degree i.

One common choice is the Chebyshev polynomials. On $[-1, 1]$, these are defined by

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{j+1}(x) = 2xT_j(x) - T_j{-}1(x).$$

More generally, we can define them on an interval $[a, b]$ by

$$p_i(x) = T_i\left(\frac{2x - (a+b)}{b-a}\right).$$

Repeat the above process with the scaled Chebyshev polynomials (here $a = 0$ and $b = 8$); i.e. using

$$\mathbf{A} = \begin{bmatrix} p_0(t_1) & p_1(t_1) & \cdots & p_k(t_1) \\ T_0(t_2) & p_1(t_2) & \cdots & p_k(t_2) \\ \vdots & \vdots & & \vdots \\ p_0(t_n) & p_1(t_n) & \cdots & p_k(t_n) \end{bmatrix}.$$

We can make a function to evaluate the Chebyshev polynomials on $[a, b]$ as follows: [1]

```
def chebyshev_polynomail(j,x,a,b):
    return np.cos(j*np.arccos((2*x-(a+b))/(b-a)))
```

If we want to evaluate this for $j = 3$ at all the t values we can do:

```
chebyshev_polynomail(3,t,0,8)
```

Make a plot with $k = 5, 10, 15$.

(c) Explain why the plots should look the same if we were doing th e computations exactly.

(d) Look at the condition numbers of all of the matrices you use in the least squares problems. How does this explain why the plots with different polynomial families look different?

[1] It's not obvious how to get this formula, but you could prove it satisfies the recurrence formula! You can look at the wikipedia page for more info

Problem 2. Suppose $\mathbf{A}$ is symmetric with eigenvalues $\lambda_1, \ldots, \lambda_n$ (so that $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$) and corresponding orthonormal eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Our analysis of the power-method involved writing the starting vector $\mathbf{x}$ in terms of $\mathbf{V}$; i.e. writing

$$\mathbf{x} = \mathbf{V} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}.$$

As long as $|c_1| > 0$, then we got convergence to $\mathbf{v}_1$.

Of course, in practice we don't know the eigenvectors or the c_is. However, it turns out if we choose $\mathbf{x}$ randomly, then c_1 will never be zero (and in fact will never be that small).

(a) Suppose we have $\mathbf{x}$ and $\mathbf{V}$. How we compute c_1?

(b) Make any 5×5 symmetric matrix $\mathbf{A}$ sample a length 5 vector $\mathbf{x}$ whose entries are independent Gaussians. You can do this by using `np.random.randn(5)`.

(c) Use numpy's `np.linalg.eigh` to compute its eigendecompsition. Use (a) to compute c_1 and report it's value.

(d) Repeat (b) 1000 times with a new $\mathbf{x}$ each time. Make a histogram of the value of c_1 over these trials.[2]

Was c_1 ever zero?

(e) In the above process, we needed to compute $\mathbf{V}$ in order to find c_1. But this would be expensive for large matrices.

Explain why we do not need to compute $\mathbf{V}$ in practice in order to use the power method to find $\mathbf{v}_1$.

Problem 3. Suppose $\mathbf{A}$ is symmetric with eigenvalue decomposition $\mathbf{V}\mathbf{\Lambda}\mathbf{V}^\mathsf{T}$, where $\mathbf{\Lambda}$ is diagonal with entries $\lambda_1, \ldots, \lambda_n$.

(a) Find the eigenvalue decomposition of:
- $\mathbf{A}^k$
- $\mathbf{A}^3 - 2\mathbf{A}$
- $\mathbf{A}^{-1}$
- $(\mathbf{A} + \lambda)^{-1}$

(b) What is the largest eigenvalue of $(\mathbf{A} + \lambda)^{-1}$ in terms of the λ_is?

[2]It turns out the distribution of c_1 does not depend on how you generated the matrix $\mathbf{A}$! This is because of something called the orthogonal invariance of the Gaussian distribution.

Problem 4. Suppose $\mathbf{A}$ is symmetric with eigenvalues $\lambda_1, \dots, \lambda_n$ (so that $|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots > |\lambda_n|$) and corresponding orthonormal eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$.

Define $\mathbf{B} = \mathbf{A}(\mathbf{I} - \mathbf{v}_1\mathbf{v}_1^\top)$.

(a) What are the eigenvalues of $\mathbf{B}$?

(b) Explain how to use the observation in (a) and the power-method to find $\mathbf{v}_2$.