Homework 4 Numerical Analysis Spring 2023

Instructions:

- Due 03/16 at 11:59pm on Gradescope.
- Write the names of anyone you work with on the top of your assignment. If you worked alone, write that you worked alone.
- Show your work.
- Include all code you use as copyable `monospaced` text in the PDF (i.e. not as a screenshot).
- Do not put the solutions to multiple problems on the same page.
- Tag your responses on gradescope. Each page should have a *single* problem tag. Improperly tagged responses will not receive credit.

Problem 1. (a) Let $x_1, \dots, x_n$ be uniformly space points from -1 to 1. You can generate this in code by `x = np.linspace(-1,1,n)`.

For $n = 100$, construct the matrix

$$\mathbf{A} = \begin{bmatrix} (x_1)^0 & (x_1)^1 & \cdots & (x_1)^4 \\ (x_2)^0 & (x_2)^1 & \cdots & (x_2)^4 \\ \vdots & \vdots & & \vdots \\ (x_n)^0 & (x_n)^1 & \cdots & (x_n)^4 \end{bmatrix}$$

(b) Plot each column of $\mathbf{A}$ against $\mathbf{x} = [x_1, \dots, x_n]$ on a single plot. Label each curve.

If your matrix $\mathbf{A}$ is a stored as a python array `A`, you can plot the i-th column using `plt.plot(x,A[:,i])`.

(c) Apply a QR factorization to $\mathbf{A}$ to obtain $\mathbf{QR}$. You are allowed to use numpy's algorithm or the ones from the lecture.

On a seperate plot from (b), plot each of the the columns of $\mathbf{Q}$.

(d) Explain how each column of $\mathbf{Q}$ relates to $\mathbf{x}$. In particular, write down the polynomials $p_0(x), p_1(x), \dots, p_4(x)$ such that

$$\mathbf{Q} = \begin{bmatrix} p_0(x_1) & p_1(x_1) & \cdots & p_4(x_1) \\ p_0(x_2) & p_1(x_2) & \cdots & p_4(x_2) \\ \vdots & \vdots & & \vdots \\ p_0(x_n) & p_1(x_n) & \cdots & p_4(x_n) \end{bmatrix}.$$

(e) Change n from 100 to 1000. What do you notice about the columns of $\mathbf{Q}$; in particular, about the polynomials $p_0, p_2, \dots, p_4$?

What would happen as $n \to \infty$?

Problem 2. Let

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{9}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ -\frac{1}{2} & -\frac{9}{2} & \frac{5}{2} \end{bmatrix}$$

(a) By hand, compute the QR factorization of $\mathbf{A}$ using the regular Gram-Schmidt algorithm. Show your work at each step.

(b) By handc, compute the QR factorization of $\mathbf{A}$ using the modified Gram-Schmidt algorithm. Show your work at each step.

Problem 3. Recall the regular Gram–Schmidt projection of $\mathbf{a}$ onto the orthogonal compliment of the columns $\mathbf{u}_1, \ldots, \mathbf{u}_j$ of $\mathbf{U}$ is

$$\text{proj}_{\mathbf{U}^\perp}(\mathbf{a}) = \mathbf{a} - \mathbf{u}_1(\mathbf{u}_1^\top \mathbf{a}) - \ldots - \mathbf{u}_1(\mathbf{u}_1^\top \mathbf{a}).$$

(a) Write this in terms of matrix-vector products with $\mathbf{U}$.

(b) Matrix products are associative, so the order you multiply things above does not impact the solution. However, number of flops does depend on the order.

Describe the more efficient ordering, and give the number of floating point operations required.

(c) Implement a new version of `proj_perp_GS(U,a)` using (b).

(d) Compare the original `proj_perp_GS`, the modified `proj_perp_MGS` and your new implementation.

In particular, let us generate an arbitrary orthogonal matrix $\mathbf{U}$ and vector $\mathbf{a}$.

```
n = 2000
U,_ = np.linalg.qr(np.random.randn(n,500))
a = np.random.randn(n)

ks = [1,50,100,150,200,250,300,350,400,450,500]
```

For each of the k values above, time how long it takes to compute the orthogonal projection of $\mathbf{a}$ onto the first k columns of $\mathbf{U}$. The matrix with the first k columns of $\mathbf{U}$ is `U[:,:k]`.

Plot all the timings on the same plot, labeling each curve and the axes.

Note that because of noise, it will help to average together several runs for each value of k.

Optionally, you can repeat this and make new plots for different values of n.

(e) What do you observe? How can you explain this, given that all of the algorithms use rough the same number of floating point operations?