CS 111: Homework 6: Due by 11:59 pm Sunday, February 26, 2023

Submit your paper as one PDF file, and tell GradeScope which page(s) each problem is on. If you worked with a partner, you must each turn in your own homework paper, and report the name and perm number of your partner. No groups of more than two allowed.

Background: In this homework you'll learn how to solve least squares problems using a different matrix factorization instead of the SVD. Given a matrix A with m rows and n columns (where $m \geq n$), the *QR factorization* is

$$A = QR,$$

where Q is m-by-m and orthogonal, and R is m-by-n and upper triangular. When $m > n$, there is also an "economy size" QR factorization, in which Q is m-by-n with columns orthogonal to each other, and R is square and upper triangular. Please read NCM Section 5.5 to learn more about the QR factorization.

You can use QR factorization to solve $Ax = b$ when A is square, because $QRx = b$ is the same as $Rx = Q^T b$. That's Problem 1 below. As you'll see in Problems 2 and 3, you can also use QR factorization to solve the least squares problem $x = \arg\min ||Ax - b||_2$ when $m > n$. In practice QR is a little less expensive than SVD for dense matrices (by maybe a factor of 2), and it can be a lot less expensive when A is sparse. Numpy's `npla.lstsq()` uses SVD, but most large-scale least squares computations use QR.

Note: When we ask you to "compare" matrix A with matrix B, we mean that you should compute and print the relative norm of their difference,

```
npla.norm(A - B) / npla.norm(B)
```

This uses the Frobenius matrix norm, which is fine for this purpose.

1. Let

$$A = \begin{pmatrix} 4 & -1 & -1 \\ -1 & 4 & -1 \\ -1 & -1 & 4 \end{pmatrix}$$

and let $b = (15, -3, 12)^T$.

1.1. Use the `scipy` QR factorization routine `scipy.linalg.qr()` to compute the two matrices (orthogonal and upper triangular) that constitute the QR factorization of A. Print Q and R, and verify by inspection that R is upper triangular. Verify that Q is orthogonal by comparing $Q^T Q$ to an identity matrix. Verify that the factorization is correct by multiplying the factors and comparing the result to A.

1.2. Use `cs111.Usolve()` and/or `cs111.Lsolve()` to compute the solution x to $Ax = b$ from the QR factors, without calling any other factorization or solve routine. (You are allowed to transpose any matrix if you want.) Verify that x is correct by computing (and showing) the relative residual norm. Show all the Jupyter input and output for your computations.

2. In this problem you will delve into the similarities and differences between the "'full-size" and "economy-size" QR factorizations of a matrix with more rows than columns. Start by generating a random 9-by-5 matrix A, using `np.random.rand()`. Show all your work in Jupyter.

2.1. Use `Q1, R1 = scipy.linalg.qr(A)` to generate the full-size QR factorization of A. What are the dimensions of Q_1? Of R_1? Verify that Q_1 is orthogonal by comparing some matrix to an identity matrix. Verify by inspection that R_1 is upper triangular; note what "triangular" means for a non-square matrix. Verify by comparing matrices that in fact $Q_1 R_1 = A$.

2.2. Use `Q2, R2 = scipy.linalg.qr(A, mode='economic')` to generate the economy-size QR factorization of A. What are the dimensions of Q_2? Of R_2? What is $Q_2^T Q_2$? Is Q_2 orthogonal? Why or why not? Verify by inspection that R_2 is upper triangular. Verify by comparing matrices that in fact $Q_2 R_2 = A$.

2.3. In English words, what is the relationship between Q_1 and Q_2? Use python to compute the relative norm of a difference of some two matrices to demonstrate that relationship.

In English words, what is the relationship between R_1 and R_2? Use python to compute the relative norm of a difference of some two matrices to demonstrate that relationship.

3. Here you will solve a least squares problem with your 9-by-5 matrix A from Problem 2 above. Use `b = np.random.rand(9)` to generate a random right-hand side b. (It's important that b is random here.) The least-squares problem is

$$x = \arg\min ||Ax - b||_2.$$

Since the system is overdetermined, we do not expect there to be any x that makes the residual norm zero. It is a theorem, though, that at the least-squares solution the residual vector is perpendicular to every column of A. That is, if x minimizes the 2-norm of the residual $r = Ax - b$, then $A^T r = 0$ is the vector of all zeros.

To see how QR factorization can be used to solve this problem, think first about the full-size QR factorization from Problem 2.1, where $A = Q_1 R_1$, with Q_1 orthogonal and R_1 rectangular and upper triangular. Multiplying by an orthogonal matrix doesn't change the 2-norm of a vector, so

$$||Ax - b||_2 = ||Q_1^T(Ax - b)||_2 = ||Q_1^T Q_1 R_1 x - Q_1^T b||_2 = ||R_1 x - Q_1^T b||_2.$$

Much like the SVD method we saw in class, we can now solve the first n equations of $R_1 x = Q_1^T b$ exactly, and nothing we do to x will make any difference in the last $m - n$ equations because R_1 is zero in those rows.

Now the trick is to notice that we can do the same thing with the economy-size factorization. You'll see how in Problem 3.2 below.

3.1 Use `npla.lstsq()` to compute the least-squares solution x. Print x and the relative residual norm $||b - Ax||_2/||b||_2$. Verify that the residual is orthogonal to the columns of A by computing (and printing) $||A^T r||_2$.

3.2 Use the economy-size factorization $Q_2 R_2 = A$ from Problem 2.2 to solve for x as follows, showing your work in Jupyter. First compute $y = Q_2^T b$. Then solve $R_2 x = y$ for x, using an appropriate routine from the `cs111` module. As above, print x and the relative residual norm, and verify that the residual is orthogonal to the columns of A.

4. Consider each of the following python loops. For each loop, answer: How many iterations does it do before halting? What are the last two values of x it prints (both as decimals printed by python, and as IEEE standard 16-hex-digit representations as printed by `cs111.print_float64`)?

For each loop, explain in one English sentence what property of the floating-point system the loop's behavior demonstrates.

4.1.

```
x = 1.0
while 1.0 + x > 1.0:
    x = x / 2.0
    print(x)
```

4.2.

```
x = 1.0
while x + x > x:
    x = 2.0 * x
    print(x)
```

4.3.

```
x = 1.0
while x + x > x:
    x = x / 2.0
    print(x)
```