5374 Fall '22

Numerical Linear Algebra — Lecture 19

Midterm: * Will be available Friday morning at 8:00 am. Lasts 72 hours.

* What will be on it?

* Basics of linear algebra
  - The rank-nullity theorem
  - Norms and inner products
  - Matrix norms
  - Orthogonal transformation
  - Projectors
* The QR decomposition
* The Normal equations for least square problem

* Algorithm and practical problem
  - The backward substitution algorithm
  - Gram-Schmidt as an algorithm for QR
  - Least squares and parametric regression
  - Discrete Fourier Transform
* Sensitivity analysis (how condition numbers bound errors, types of errors)
* Familiarity with numpy, pyplot, and scipy.sparse

Allowable materials: All Notebooks and lecture notes on the repository, the Solomon book, and the Trefethen-Bau book.

Today: The Householder algorithm for the QR decomposition

When we apply the Gram-Schmidt process to compute the QR decomposition of a matrix A what we are doing is equivalent to repeated multiplication of A on the right by a lower triangular matrix

$$A L_1 L_2 \cdots L_n$$

until the result (after n steps) is an orthogonal matrix Q.

$$Q = A\underbrace{(L_1 L_2 \cdots L_n)}_{\text{Lower Triangular Matrix}}$$

$$\Rightarrow A = Q \boxed{(L_1 \cdots L_n)^{-1}}$$

Upper Triangular Matrix

Householder (about 70 years ago) introduced an algorithm where one repeatedly multiplies A from the left by orthogonal matrices ("Householder reflectors") until we have an upper triangular matrix R.

$$(Q_n \cdots Q_2 Q_1)A = R$$

Householder reflectors

This will produce (after inverting the Q's) the QR decomposition of A:

$$A = \underbrace{(Q_n \cdots Q_1)^{-1}}_{= Q} R$$

Let's see how the reflectors $Q_1, \dots, Q_n$ are constructed. Write:

$$A = \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & \quad & a_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & & a_{1n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & & a_{nn} \end{pmatrix} \end{array}$$

We want Q_1 to produce something that looks like this:

$$Q_1 A = \left(\begin{array}{c|cccc} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ \hline 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & a_{32}^{(1)} & & & \vdots \\ & \vdots & & & \\ 0 & a_{n2}^{(1)} & \cdots & \cdots & a_{nn}^{(1)} \end{array}\right)$$

If we find such a Q_1, we can repeat the process and find Q_2 such that

$$\underbrace{\begin{pmatrix} 1 & 0 \cdots & 0 \\ 0 & & \\ \vdots & \boxed{?} & \\ 0 & & \end{pmatrix}}_{Q_2} (Q_1 A) = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots \; a_{2n}^{(2)} \\ \vdots & 0 & a_{33}^{(2)} & \cdots \; a_{3n}^{(2)} \\ & \vdots & a_{43}^{(2)} & \vdots \\ 0 & 0 & a_{n3}^{(2)} & a_{nn}^{(2)} \end{pmatrix}$$

$$\cdots \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & \boxed{?} & \\ 0 & 0 & & & \end{pmatrix} (Q_2 Q_1 A) = \cdots$$ and so on.

We see that if we have understood how to do the first step then it seems reasonable we can repeat that step n times and obtain orthogonal matrices $Q_1, \dots, Q_n$ such that

$$Q_n Q_{n-1} \cdots Q_1 A = R$$

where R is an upper triangular matrix.

What does this crucial first step look like?

$$Q_1 A = Q_1 \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix}$$

$$= \begin{array}{c} \begin{array}{cc} & \overbrace{\hspace{8em}}^{n-1 \text{ columns}} \end{array} \\ \begin{pmatrix} x & x & x & x & \cdots & x \\ 0 & x & \cdots & & & x \\ 0 & x & & & & \\ \vdots & \vdots & & & & \vdots \\ 0 & x & & & & x \end{pmatrix} \end{array} \begin{array}{l} \\ \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right\} \leftarrow n-1 \text{ rows} \end{array}$$

So, we want

$$Q_1 a_1 = \begin{pmatrix} x \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$ for some x to be determined.

Now Q_1 preserves length, so x is largely constrained:

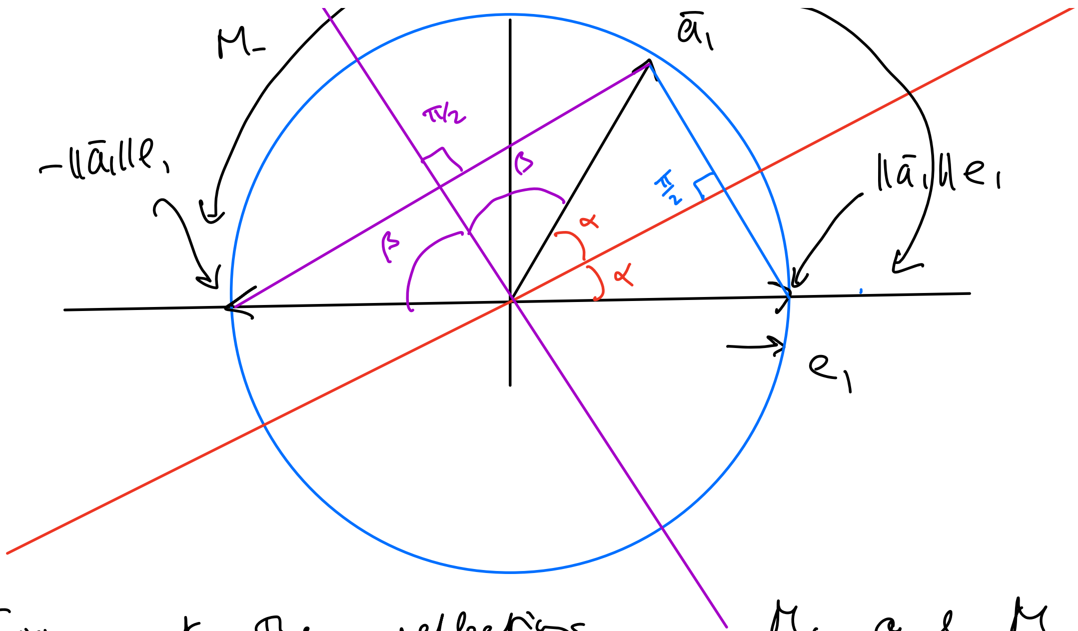
$$\|a_1\|_2 = \|Q_1 a_1\|_2 = |x|$$

so $$x = \pm \|a_1\|_2 \qquad \left(a_{11}^{(1)} = \pm \|a_1\|_2 \right)$$

This means is we want a Q such that

$$Q a_1 = \pm \|a_1\|_2 e_1 , \quad \text{where } e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

In $\mathbb{R}^n$, consider the plane spanned by a_1 and e_1 (if a_1 and e_1 are parallel, we can take $Q_1 = I$ and move on)



Turns out the reflections M_+ and M_- have a simple algebraic expression:

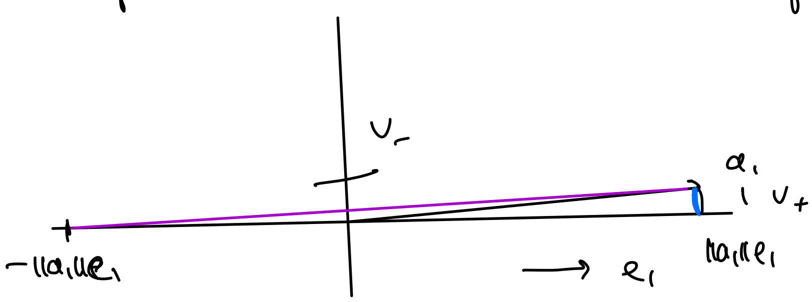
$$V_{+} = \frac{+\|a_1\|_2 e_1 - a_1}{\| +\|a_1\|_2 e_1 - a_1 \|}$$

$$V_{-} = \frac{-\|a_1\|_2 e_1 - a_1}{\| -\|a_1\|_2 e_1 - a_1 \|}$$

Then,

$$M_+ = I - 2 V_+ \otimes V_+$$
$$M_- = I - 2 V_- \otimes V_-$$

We have two choices of Q_1, is there a difference? Observe the following situation could happen:



(or viceversa: a_1 could be almost equal to $-\|a_1\| e_1$)

Since we will divide by $\| \|a_1\| e_1 - a_1 \|_2$ or $\| -\|a_1\| e_1 - a_1 \|_2$ we run the risk of dividing by a very large number.

So to prevent this, we let Q_1 be given by

$$Q_1 = I - 2 V_1 \otimes V_1$$

where

$$V_1 = V_{\text{sign}(\langle a_1, e_1 \rangle)}$$

Now to do the other steps we repeat this formula for smaller and smaller matrices, e.g.

$$Q_2 = \begin{pmatrix} 1 & 0 \;\cdots\; 0 \\ 0 & \\ \vdots & \boxed{I' - 2 V_2' \otimes V_2'} \\ 0 & \end{pmatrix}$$

$$= I - 2 V_2 \otimes V_2$$

where $V_2 = \begin{pmatrix} 0 \\ \hline V_2' \end{pmatrix} \begin{matrix} \} \; 1 \text{ row} \\ \} \; n-1 \text{ rows} \end{matrix}$

Iterating this results in n unit vectors $v_1, \dots, v_n$ such that if

$$Q_k := I - 2 v_k \otimes v_k$$

Then

$$(Q_n Q_{n-1} \cdots Q_1) A = R \quad , R \text{ an upper triangular matrix}$$

$$\Rightarrow \quad A = QR, \quad Q = (Q_n Q_{n-1} \cdots Q_1)^{-1}$$

$$= Q_1^{-1} Q_2^{-1} \cdots Q_n^{-1}$$

$$\left(\begin{array}{c}\text{since the}\\ Q_k \text{ are}\\ \text{reflections}\end{array}\right) \rightarrow \quad = Q_1 Q_2 \cdots Q_n$$

Remark: It is tempting to think of the output of the Householder algorithm as

$$Q = Q_1 \cdots Q_n$$

This would entail in practice to compute n $n \times n$ matrix multiplication, which takes $O(n^3)$ FLOP's.

One should think of the output as being the n unit vectors $v_1, \dots, v_n$. Why?

Well, given a vector b computing Qb is equivalent to running the following algorithm

Qb:

for $k = 1, \dots, n$

$\qquad z = Q_{n-k+1} z$

return z

$$\left(z = Q_n Q_{n-1} \cdots Q_1 b = Q b \right)$$

Each application of the loop amounts:

1. Computing (v_{n-k+1}, z) ($O(n)$ FLOP's)
2. Computing $z - 2(v_{n-k+1}, z)\, v_{n-k+1}$ ($O(n)$ FLOP's)

So you have n steps with $O(n)$ FLOP's each, for a total of $O(n^2)$ FLOP's (which is what $n \times n$ matrix-vector product takes anyway)