
MATH96023/MATH97032/MATH97140 - Computational Linear Algebra

Edition 2020.0

Colin J. Cotter

Jul 09, 2020

CONTENTS

1	Preliminaries	1
1.1	Matrices, vectors and matrix-vector multiplication	1
1.2	Range, nullspace and rank	3
1.3	Invertibility and inverses	3
1.4	Orthogonal vectors and orthogonal matrices	4
1.5	Inner products and orthogonality	4
1.6	Orthogonal components of a vector	5
1.7	Unitary matrices	5
1.8	Vector norms	6
1.9	Projectors and projections	6
1.10	Constructing orthogonal projectors from sets of orthonormal vectors	7
2	QR Factorisation	9
2.1	What is the QR factorisation?	9
2.2	QR factorisation by classical Gram-Schmidt algorithm	9
2.3	Projector interpretation of Gram-Schmidt	10
2.4	Modified Gram-Schmidt	11
2.5	Modified Gram-Schmidt as triangular orthogonalisation	12
2.6	Householder triangulation	13
2.7	Application: Least squares problems	16
3	Analysing algorithms	17
3.1	Operation count	17
3.2	Operation count for modified Gram-Schmidt	17
3.3	Operation count for Householder	18

PRELIMINARIES

In this preliminary section we revise a few key linear algebra concepts that will be used in the rest of the course, emphasising the column space of matrices. We will quote some standard results that should be found in an undergraduate linear algebra course.

1.1 Matrices, vectors and matrix-vector multiplication

We will consider the multiplication of a vector

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad x_i \in \mathbb{C}, i = 1, 2, \dots, n, \text{ i.e. } x \in \mathbb{C}^n,$$

by a matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix},$$

i.e. $A \in \mathbb{C}^{m \times n}$. A has m rows and n columns so that the product

$$b = Ax$$

produces $b \in \mathbb{C}^m$, defined by

$$b_i = \sum_{j=1}^n a_{ij}x_j, \quad i = 1, 2, \dots, m.$$

In this course it is important to consider the general case where $m \neq n$, which has many applications in data analysis, curve fitting etc. We will usually state generalities in this course for vectors over the field \mathbb{C} , noting where things specialise to \mathbb{R} .

We can quickly check that the map $x \rightarrow Ax$ given by matrix multiplication is a linear map from $\mathbb{C}^n \rightarrow \mathbb{C}^m$, since it is straightforward to check from the definition that

$$A(\alpha x + y) = \alpha Ax + Ay,$$

for all $x, y \in \mathbb{C}^n$ and $\alpha \in \mathbb{C}$. (Exercise: show this for yourself.)

It is very useful to interpret matrix-vector multiplication as a linear combination of the columns of A with coefficients taken from the entries of x . If we write A in terms of the columns,

$$A = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix},$$

where

$$a_i \in \mathbb{C}^m, i = 1, 2, \dots, n,$$

then

$$b = \sum_{j=1}^n x_j a_j,$$

i.e. a linear combination of the columns of A as described above.

We can extend this idea to matrix-matrix multiplication. Taking $A \in \mathbb{C}^{l \times m}$, $C \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{l \times n}$, with $B = AC$, then the components of B are given by

$$b_{ij} = \sum_{k=1}^m a_{ik} c_{kj}, \quad 1 \leq i \leq l, 1 \leq j \leq n.$$

Writing $b_j \in \mathbb{C}^m$ as the j th column of B , for $1 \leq j \leq n$, and c_j as the j th column of C , we see that

$$b_j = A c_j.$$

This means that the j th column of B is the matrix-vector product of A with the j th column of C . This kind of “column thinking” is very useful in understanding computational linear algebra algorithms.

An important example is the outer product of two vectors, $u \in \mathbb{C}^m$ and $v \in \mathbb{C}^n$. Here it is useful to see these vectors as matrices with one column, i.e. $u \in \mathbb{C}^{m \times 1}$ and $v \in \mathbb{C}^{n \times 1}$. The outer product is $uv^T \in \mathbb{C}^{m \times n}$. The columns of v^T are just single numbers (i.e. vectors of length 1), so viewing this as a matrix multiplication we see

$$uv^T = \begin{pmatrix} uv_1 & uv_2 & \dots & uv_n \end{pmatrix},$$

which means that all the columns of uv^T are multiples of u . We will see in the next section that this matrix has rank 1.

1.2 Range, nullspace and rank

In this section we'll quickly rattle through some definitions and results.

Definition 1 (Range) The range of A , $\text{range}(A)$, is the set of vectors that can be expressed as Ax for some x .

The next theorem follows as a result of the column space interpretation of matrix-vector multiplication.

Theorem 2 $\text{range}(A)$ is the vector space spanned by the columns of A .

Definition 3 (Nullspace) The nullspace $\text{null}(A)$ of A (or kernel) is the set of vectors x satisfying $Ax = 0$, i.e.

$$\text{null}(A) = \{x \in \mathbb{C}^n : Ax = 0\}.$$

Definition 4 (Rank) The rank $\text{rank}(A)$ of A is the dimension of the column space of A .

If

$$A = (a_1 \ a_2 \ \dots \ a_n),$$

the column space of A is $\text{span}(a_1, a_2, \dots, a_n)$.

Definition 5 An $m \times n$ matrix A is full rank if it has maximum possible rank i.e. rank equal to $\min(m, n)$.

If $m \geq n$ then A must have n linearly independent columns to be full rank. The next theorem is then a consequence of the column space interpretation of matrix-vector multiplication.

Theorem 6 An $m \times n$ matrix A is full rank if and only if it maps no two distinct vectors to the same vector.

Definition 7 A matrix A is called nonsingular, or invertible, if it is a square matrix ($m = n$) of full rank.

1.3 Invertibility and inverses

This means that an invertible matrix has columns that form a basis for \mathbb{C}^m . Given the canonical basis vectors defined by

$$e_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

i.e. e_j has all entries zero except for the j th entry which is 1, we can write

$$e_j = \sum_{k=1}^m z_{jk} a_k, \quad 1 \leq j \leq m.$$

In other words,

$$\begin{aligned} I &= (e_1 \ e_2 \ \dots \ e_m) \\ &= ZA. \end{aligned}$$

We call Z a (left) inverse of A . (Exercises: show that Z is the unique left inverse of A , and show that Z is also the unique right inverse of A , satisfying $I = AZ$.) We write $Z = A^{-1}$.

The first four parts of the next theorem are a consequence of what we have so far, and we shall quote the rest (see a linear algebra course).

Theorem 8 *Let $A \in \mathbb{C}^{m \times m}$. Then the following are equivalent.*

1. A has an inverse.
2. $\text{rank}(A) = m$.
3. $\text{range}(A) = \mathbb{C}^m$.
4. $\text{null}(A) = \{0\}$.
5. 0 is not an eigenvalue of A .
6. 0 is not a singular value of A .
7. The determinant $\det(A) \neq 0$.

Finding the inverse of a matrix can be seen as a change of basis. Considering the equation $Ax = b$, we have $x = A^{-1}b$ for invertible A . We have seen already that b can be written as

$$b = \sum_{j=1}^m x_j a_j.$$

Since the columns of A span \mathbb{C}^m , the entries of x thus provide the unique expansion of b in the columns of A which form a basis. Hence, whilst the entries of b give basis coefficients for b in the canonical basis (e_1, e_2, \dots, e_m) , the entries of x give basis coefficients for b in the basis given by the columns of A .

1.4 Orthogonal vectors and orthogonal matrices

Definition 9 (Adjoint) *The adjoint (or Hermitian conjugate) of $A \in \mathbb{C}^{m \times n}$ is a matrix $A^* \in \mathbb{C}^{n \times m}$ (sometimes written A^\dagger or A'), with*

$$a_{ij}^* = \bar{a}_{ji},$$

where the bar denotes the complex conjugate of a complex number. If $A^* = A$ then we say that A is Hermitian.

For real matrices, $A^* = A^T$. If $A = A^T$, then we say that the matrix is symmetric.

The following identity is very important when dealing with adjoints.

Theorem 10 *For matrices A, B with compatible dimensions (so that they can be multiplied),*

$$(AB)^* = B^* A^*.$$

1.5 Inner products and orthogonality

The inner product is a critical tool in computational linear algebra.

Definition 11 (Inner product) *Let $x, y \in \mathbb{C}^m$. Then the inner product of x and y is*

$$x^* y = \sum_{i=1}^m \bar{x}_i y_i.$$

(Exercise: check that the inner product is bilinear, i.e. linear in both of the arguments.)

We will frequently use the natural norm derived from the inner product to define size of vectors.

Definition 12 (2-Norm) Let $x \in \mathbb{C}^m$. Then the 2-norm of x is

$$\|x\| = \sqrt{\sum_{i=1}^m x_i^2} = \sqrt{x^*x}.$$

Orthogonality will emerge as an early key concept in this course.

Definition 13 (Orthogonal vectors) Let $x, y \in \mathbb{C}^m$. The two vectors are orthogonal if $x^*y = 0$.

Similarly, let X, Y be two sets of vectors. The two sets are orthogonal if

$$x^*y = 0 \forall x \in X, y \in Y.$$

A set S of vectors is itself orthogonal if

$$x^*y = 0 \forall x, y \in S.$$

We say that S is orthonormal if we also have $\|x\| = 1$ for all $x \in S$.

1.6 Orthogonal components of a vector

Let $S = \{q_1, q_2, \dots, q_n\}$ be an orthonormal set of vectors in \mathbb{C}^m , and take another arbitrary vector $v \in \mathbb{C}^m$. Now take

$$r = v - (q_1^*v)q_1 - (q_2^*v)q_2 - \dots - (q_n^*v)q_n.$$

Then, we can check that r is orthogonal to S , by calculating for each $1 \leq i \leq n$,

$$q_i^*r = q_i^*v - (q_1^*v)(q_i^*q_1) - \dots - (q_n^*v)(q_i^*q_n)$$

$$= q_i^*v - q_i^*v = 0,$$

since $q_i^*q_j = 0$ if $i \neq j$, and 1 if $i = j$. Thus,

$$v = r + \sum_{i=1}^n (q_i^*v)q_i = r + \sum_{i=1}^n \underbrace{(q_i q_i^*)}_{\text{rank-1 matrix}} v.$$

If S is a basis for \mathbb{C}^m , then $n = m$ and $r = 0$, and we have

$$v = \sum_{i=1}^m (q_i q_i^*)v.$$

1.7 Unitary matrices

Definition 14 (Unitary matrices) A matrix $Q \in \mathbb{C}^{m \times m}$ is unitary if $Q^* = Q^{-1}$.

For real matrices, a matrix Q is orthogonal if $Q^T = Q^{-1}$.

Theorem 15 The columns of a unitary matrix Q are orthonormal.

Proof 16 We have $I = Q^*Q$. Then using the column space interpretation of matrix-matrix multiplication,

$$e_j = Q^*q_j,$$

where q_j is the j th column of Q . Taking row i of e_j , we have

$$\delta_{ij} = q_i^*q_j, \text{ where } \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise} \end{cases}.$$

Extending a theme from earlier, we can interpret $Q^* = Q^{-1}$ as representing a change of orthogonal basis. If $Qx = b$, then $x = Q^*b$ contains the coefficients of b expanded in the basis given by the orthonormal columns of Q .

1.8 Vector norms

Various vector norms are useful to measure the size of a vector. In computational linear algebra we need them for quantifying errors etc.

Definition 17 (Norms) A norm is a function $\|\cdot\| : \mathbb{C}^m \rightarrow \mathbb{R}$, such that

1. $\|x\| \geq 0$, and $\|x\| = 0 \implies x = 0$.
2. $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality).
3. $\|\alpha x\| = |\alpha|\|x\|$ for all $x \in \mathbb{C}^m$ and $\alpha \in \mathbb{C}$.

We have already seen the 2-norm, or Euclidean norm, which is part of a larger class of norms called p-norms, with

$$\|x\|_p = \left(\sum_{i=1}^m |x_i|^p \right)^{1/p},$$

for real ' $p > 0$ '. We will also consider weighted norms

$$\|x\|_{W,p} = \|Wx\|_p,$$

where W is a matrix.

1.9 Projectors and projections

Definition 18 (Projector) A projector P is a square matrix that satisfies $P^2 = P$.

If $v \in \text{range}(P)$, then there exists x such that $Pv = x$. Then,

$$Pv = P(Px) = P^2x = Px = v,$$

and hence multiplying by P does not change v .

Now suppose that $Pv \neq v$ (so that $v \notin \text{range}(P)$). Then,

$$P(Pv - v) = P^2v - Pv = Pv - Pv = 0,$$

which means that $Pv - v$ is the nullspace of P . We have

$$Pv - v = -(I - P)v.$$

Definition 19 (Complementary projector) Let P be a projector. Then we call $I - P$ the complementary projector.

To see that $I - P$ is also a projector, we just calculate,

$$(I - P)^2 = I^2 - 2P + P^2 = I - 2P + P = I - P.$$

If $Pu = 0$, then $(I - P)u = u$.

In other words, the nullspace of P is contained in the range of $I - P$.

On the other hand, if v is in the range of $I - P$, then there exists some w such that

$$v = (I - P)w = w - Pw.$$

We have

$$Pv = P(w - Pw) = Pw - P^2w = Pw - Pw = 0.$$

Hence, the range of $I - P$ is contained in the nullspace of P . Combining these two results we see that the range of $I - P$ is equal to the nullspace of P . Since P is the complementary projector to $I - P$, we can repeat the same argument to show that the range of P is equal to the nullspace of $I - P$.

We see that a projector P separates \mathbb{C}^m into two subspaces, the nullspace of P and the range of P . In fact the converse is also true: given two subspaces S_1 and S_2 of \mathbb{C}^m with $S_1 \cap S_2 = \{0\}$, then there exists a projector P whose range is S_1 and whose nullspace is S_2 .

Now we introduce orthogonality into the concept of projectors.

Definition 20 (Orthogonal projector) P is an orthogonal projector if

$$(Pv)^*(Pv - v) = 0, \forall v \in \mathbb{C}^m.$$

In this case, P separates the space into two orthogonal subspaces.

1.10 Constructing orthogonal projectors from sets of orthonormal vectors

Let $\{q_1, \dots, q_n\}$ be an orthonormal set of vectors in \mathbb{C}^m . We write

$$\hat{Q} = (q_1 \quad q_2 \quad \dots \quad q_n).$$

Previously we showed that for any $v \in \mathbb{C}^m$, we have

$$v = \underbrace{\quad}_\text{Orthogonal to column space of } \hat{Q} + \underbrace{\sum_{i=1}^n (q_i q_i^*) v}_{\text{in the column space of } \hat{Q}}.$$

Hence, the map

$$v \mapsto Pv = \underbrace{\sum_{i=1}^n (q_i q_i^*)}_{=P} v,$$

is an orthogonal projector. In fact, P has very simple form.

Theorem 21 *The orthogonal projector P takes the form*

$$P = \hat{Q}\hat{Q}^*.$$

Proof 22 *From the change of basis interpretation of multiplication by \hat{Q}^* , the entries in \hat{Q}^*v gives coefficients of the projection of v onto the column space of \hat{Q} when expanded using the columns as a basis. Then, multiplication by \hat{Q} gives the projection of v expanded again in the canonical basis. Hence, multiplication by $\hat{Q}\hat{Q}^*$ gives exactly the same result as multiplication by the formula for P above.*

This means that $\hat{Q}\hat{Q}^*$ is an orthogonal projection onto the range of \hat{Q} . The complementary projector is $P_\perp = I - \hat{Q}\hat{Q}^*$ is an orthogonal projection onto the nullspace of \hat{Q} .

An important special case is when \hat{Q} has just one column, and then

$$P = q_1 q_1^*, \quad P_\perp = I - q_1 q_1^*.$$

We notice that $P^* = (\hat{Q}\hat{Q}^*)^* = \hat{Q}\hat{Q}^* = P$. In fact the following is true.

Theorem 23 *$P = P^*$ if and only if Q is an orthogonal projector.*

QR FACTORISATION

A common theme in computational linear algebra is transformations of matrices and algorithms to implement them. A transformation is only useful if it can be computed efficiently and sufficiently free of pollution from truncation errors (either due to finishing an iterative algorithm early, or due to round-off errors). A particularly powerful and insightful transformation is the QR factorisation. In this section we will introduce the QR factorisation and some good and bad algorithms to compute it.

2.1 What is the QR factorisation?

We start with another definition.

Definition 24 (Upper triangular matrix) An $m \times n$ upper triangular matrix R has coefficients satisfying $r_{ij} = 0$ when $i \geq j$.

It is called upper triangular because the nonzero rows form a triangle on and above the main diagonal of R .

Now we can describe the QR factorisation.

Definition 25 (QR factorisation) A QR factorisation of an $m \times n$ matrix A consists of an $m \times m$ unitary matrix Q and an $m \times n$ upper triangular matrix R such that $A = QR$.

The QR factorisation is a key tool in analysis of datasets, and polynomial fitting. It is also at the core of one of the most widely used algorithms for finding eigenvalues of matrices. We shall discuss all of this later during this course.

When $m > n$, R must have all zero rows after the n block of R consisting of the first n rows, which we call \hat{R} . Similarly, in the matrix vector product QR , all columns of Q beyond the n , so it makes sense to only work with the first n columns of Q , which we call \hat{Q} . We then have the reduced QR factorisation, $\hat{Q}\hat{R}$.

In the rest of this section we will examine some algorithms for computing the QR factorisation, before discussing the application to least squares problems. We will start with a bad algorithm, before moving on to some better ones.

2.2 QR factorisation by classical Gram-Schmidt algorithm

The classical Gram-Schmidt algorithm for QR factorisation is motivated by the column space interpretation of the matrix-matrix multiplication $A = QR$, namely that the j th column a_j of A is a linear combination of the orthonormal columns of Q , with the coefficients given by the j th column r_j of R .

The first column of R only has a non-zero entry in the first row, so the first column of Q must be proportional to A , but normalised (i.e. rescaled to have length 1). The scaling factor is this first row of the first column of R . The second column of R has only non-zero entries in the first two rows, so the second column of A must be writeable as a linear combination of the first two columns of Q . Hence, the second column of Q must be the second column of A with the first column of Q projected out, and then normalised. The first row of the second column of R is then the coefficient for this projection, and the second row is the normalisation scaling factor. The third row of Q is then the third row of A with the first two columns of Q projected out, and so on.

Hence, finding a QR factorisation is equivalent to finding an orthonormal spanning set for the columns of A , where the span of the first j elements of the spanning set and of the first j columns of A is the same, for $j = 1, \dots, n$.

Hence we have to find R coefficients such that

$$\begin{aligned} q_1 &= \frac{a_1}{r_{11}}, \\ q_2 &= \frac{a_2 - r_{12}q_1}{r_{22}} \\ &\vdots \\ q_n &= \frac{a_n - \sum_{i=1}^{n-1} r_{in}q_i}{r_{nn}}, \end{aligned}$$

with (q_1, q_2, \dots, q_n) an orthonormal set. The non-diagonal entries of R are found by inner products, i.e.,

$$r_{ij} = q_i^* a_j, \quad i > j,$$

and the diagonal entries are chosen so that $\|q_i\| = 1$, for $i = 1, 2, \dots, n$, i.e.

$$|r_{jj}| = \left\| a_j - \sum_{i=1}^{j-1} r_{ij}q_i \right\|.$$

Note that this absolute value does leave a degree of nonuniqueness in the definition of R . It is standard to choose the diagonal entries to be real and non-negative.

We now present the classical Gram-Schmidt algorithm as pseudo-code.

- FOR $j = 1$ TO n
 - $v_j \leftarrow a_j$
 - FOR $i = 1$ TO $j - 1$
 - * $r_{ij} \leftarrow q_i^* a_j$
 - * $v_j \leftarrow v_j - r_{ij}q_i$
 - END FOR
 - $r_{jj} \leftarrow \|v_j\|_2$
 - $q_j \leftarrow v_j / r_{jj}$
- END FOR

(Remember that Python doesn't have END FOR statements, but instead uses indentation to terminate code blocks. We'll write END statements for code blocks in pseudo-code in these notes.)

2.3 Projector interpretation of Gram-Schmidt

At each step of the Gram-Schmidt algorithm, a projector is applied to a column of A . We have

$$\begin{aligned} q_1 &= \frac{P_1 a_1}{\|P_1 a_1\|}, \\ q_2 &= \frac{P_2 a_2}{\|P_2 a_2\|}, \\ &\vdots \\ q_n &= \frac{P_n a_n}{\|P_n a_n\|}, \end{aligned}$$

where P_j are orthogonal projectors that project out the first $j - 1$ columns (q_1, \dots, q_{j-1}) (P_1 is the identity as this set is empty when $j = 1$). The orthogonal projector onto the first $j - 1$ columns is $\hat{Q}_{j-1}\hat{Q}_{j-1}^*$, where

$$\hat{Q}_{j-1} = (q_1 \quad q_2 \quad \dots \quad q_{j-1}).$$

Hence, P_j is the complementary projector, $P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^*$.

2.4 Modified Gram-Schmidt

There is a big problem with the classical Gram-Schmidt algorithm. It is unstable, which means that when it is implemented in inexact arithmetic on a computer, round-off error unacceptably pollutes the entries of Q and R , and the algorithm is not useable in practice. What happens is that the columns of Q are not quite orthogonal, and this loss of orthogonality spoils everything. We will discuss stability later in the course, but right now we will just discuss the fix for the classical Gram-Schmidt algorithm, which is based upon the projector interpretation which we just discussed.

To reorganise Gram-Schmidt to avoid instability, we decompose P_j into a sequence of $j - 1$ projectors of rank $m - 1$, that each project out one column of Q , i.e.

$$P_j = P_{\perp q_{j-1}} \dots P_{\perp q_2} P_{\perp q_1},$$

where

$$P_{\perp q_j} = I - q_j q_j^*.$$

Then,

$$v_j = P_j a_j = P_{\perp q_{j-1}} \dots P_{\perp q_2} P_{\perp q_1} a_j.$$

Here we notice that we must apply $P_{\perp q_1}$ to all but one columns of A , and $P_{\perp q_2}$ to all but two columns of A , $P_{\perp q_3}$ to all but three columns of A , and so on.

By doing this, we gradually transform A to a unitary matrix, as follows.

$$\begin{aligned} A &= (a_1 \quad a_2 \quad a_3 \quad \dots \quad a_n) \\ &\quad (q_1 \quad v_2^1 \quad v_3^1 \quad \dots \quad v_n^1) \\ &\rightarrow (q_1 \quad q_2 \quad v_3^2 \quad \dots \quad v_n^2) \\ &\dots \rightarrow (q_1 \quad q_2 \quad q_3 \quad \dots \quad q_n). \end{aligned}$$

Then it is just a matter of keeping a record of the coefficients of the projections and normalisation scaling factors and storing them in R .

This process is mathematically equivalent to the classical Gram-Schmidt algorithm, but the arithmetic operations happen in a different order, in a way that turns out to reduce accumulation of round-off errors.

We now present this modified Gram-Schmidt algorithm as pseudo-code.

- FOR $i = 1$ TO n
 - $v_i \leftarrow a_i$
- END FOR
- FOR $i = 1$ TO n
 - $r_{ii} \leftarrow \|v_i\|_2$
 - $q_i = v_i / r_{ii}$ * FOR $j = i + 1$ TO n
 - * $r_{ij} \leftarrow q_i^* a_j$

```

*  $v_j \leftarrow v_j - r_{ij}q_i$ 
* END FOR
• END FOR
    
```

This algorithm can be applied “in place”, overwriting the entries in A with the v s and eventually the q s.

2.5 Modified Gram-Schmidt as triangular orthogonalisation

This iterative transformation process can be written as right-multiplication by an upper triangular matrix. For example, at the first iteration,

$$\underbrace{(v_1^0 \ v_2^0 \ \dots \ v_n^0)}_A \underbrace{\begin{pmatrix} \frac{1}{r_{11}} & -\frac{r_{12}}{r_{11}} & \dots & \dots & -\frac{r_{1n}}{r_{11}} \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}}_{A_1} = \underbrace{(q_1 \ v_2^1 \ \dots \ v_n^1)}_{A_1}.$$

To understand this equation, we can use the column space interpretation of matrix-matrix multiplication. The columns of A_1 are linear combinations of the columns of A with coefficients given by the columns of R_1 . Hence, q_1 only depends on v_1^0 , scaled to have length 1, and v_i^1 is a linear combination of (v_1^0, v_i^0) such that v_i^1 is orthogonal to q_1 , for $1 < i \leq n$.

Similarly, the second iteration may be written as

$$\underbrace{(v_1^1 \ v_2^1 \ \dots \ v_n^1)}_{A_1} \underbrace{\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & r_{22} & -\frac{r_{23}}{r_{22}} & \dots & -\frac{r_{2n}}{r_{22}} \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}}_{R_2} = \underbrace{(q_1 \ q_2 \ v_3^2 \ \dots \ v_n^2)}_{A_2}.$$

It should become clear that each transformation from A_i to A_{i+1} takes place by right multiplication by an upper triangular matrix R_{i+1} , which is an identity matrix plus entries in row i . By combining these transformations together, we obtain

$$A \underbrace{R_1 R_2 \dots R_n}_{\hat{R}^{-1}} = \hat{Q}.$$

Since upper triangular matrices form a group, the product of the R_i matrices is upper triangular. Further, all the R_i matrices have non-zero determinant, so the product is invertible, and we can write this as \hat{R}^{-1} . Right multiplication by \hat{R} produces the usual reduced QR factorisation. We say that modified Gram-Schmidt implements triangular orthogonalisation: the transformation of A to an orthogonal matrix by right multiplication of upper triangular matrices.

This is a powerful way to view the modified Gram-Schmidt process from the point of view of understanding and analysis, but of course we do not form the matrices R_i explicitly (we just follow the pseudo-code given above).

2.6 Householder triangulation

This view of the modified Gram-Schmidt process as triangular orthogonalisation gives an idea to build an alternative algorithm. Instead of right multiplying by upper triangular matrices to transform A to \hat{Q} , we can consider left multiplying by unitary matrices to transform A to R ,

$$\underbrace{Q_n \dots Q_2 Q_1}_{=Q^*} A = R.$$

Multiplying unitary matrices produces unitary matrices, so we obtain $A = QR$ as a full factorisation of A .

To do this, we need to work on the columns of A , from left to right, transforming them so that each column has zeros below the diagonal. These unitary transformations need to be designed so that they don't spoil the structure created in previous columns. The easiest way to ensure this is construct a unitary matrix Q_k with an identity matrix as the $(k-1) \times (k-1)$ submatrix,

$$Q_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & F \end{pmatrix}.$$

This means that multiplication by Q_k won't change the first $k-1$ rows, leaving the previous work to remove zeros below the diagonal undisturbed. For Q_k to be unitary and to transform all below diagonal entries in column k to zero, we need the $(n-k+1) \times (n-k+1)$ submatrix F to also be unitary, since

$$Q_k^* = \begin{pmatrix} I_{k-1} & 0 \\ 0 & F^* \end{pmatrix}, \quad Q_k^{-1} = \begin{pmatrix} I_{k-1} & 0 \\ 0 & F^{-1} \end{pmatrix}.$$

We write the k of A_k as

$$v_k^k = \begin{pmatrix} \hat{v}_k^k \\ x \end{pmatrix},$$

where \hat{v}_k^k contains the first $k-1$ entries of v_k^k . The column gets transformed according to

$$Q_k v_k^k = \begin{pmatrix} \hat{v}_k^k \\ Fx \end{pmatrix}.$$

and our goal is that Fx is zero, except for the first entry (which becomes the diagonal entry of $Q_k v_k^k$). Since F is unitary, we must have $\|Fx\| = \|x\|$, so we choose to have

$$Fx = \pm \|x\| e_1,$$

where we shall consider the sign later. Here we have chosen for Fx to be real. We could alternatively pick another value of the same magnitude but different complex argument (but that would lead to a slightly different QR factorisation).

We can achieve this by using a Householder reflector for F , which is a unitary transformation that does precisely what we need. Geometrically, the idea is that we consider a line joining x and $Fx = \|x\| e_1$, which points in the direction $v = \|x\| e_1 - x$. We can transform x to Fx by a reflection in the hyperplane H that is orthogonal to v . Since reflections are norm preserving, F must be unitary. Applying the projector P given by

$$Px = \left(I - \frac{vv^*}{v^*v} \right) x,$$

does half the job, producing a vector in H . To do a reflection we need to go twice as far,

$$Fx = \left(I - 2 \frac{vv^*}{v^*v} \right) x.$$

We can check that this does what we want,

$$\begin{aligned} Fx &= \left(I - 2 \frac{vv^*}{v^*v} \right) x, \\ &= x - 2 \frac{(\pm \|x\| e_1 - x)}{\|\pm \|x\| e_1 - x\|^2} (\pm \|x\| e_1 - x)^* x, \\ &= x - 2 \frac{(\pm \|x\| e_1 - x)}{\|\pm \|x\| e_1 - x\|^2} \|x\| (\pm x_1 - \|x\|), \\ &= x + (\pm \|x\| e_1 - x) = \pm \|x\| e_1, \end{aligned}$$

as required, having checked that

$$\|\pm \|x\| e_1 - x\|^2 = -2\|x\|(\pm x_1 - \|x\|).$$

We can also check that F is unitary. First we check that F is Hermitian,

$$\begin{aligned} \left(I - 2 \frac{vv^*}{v^*v} \right)^* &= I - 2 \frac{(vv^*)^*}{v^*v}, \\ &= I - 2 \frac{(v^*)^* v^*}{v^*v}, \\ &= I - 2 \frac{vv^*}{v^*v} = F. \end{aligned}$$

Now we use this to show that F is unitary,

$$\begin{aligned} F^* F &= \left(I - 2 \frac{vv^*}{v^*v} \right) \left(I - 2 \frac{vv^*}{v^*v} \right) \\ &= I - 4 \frac{vv^*}{v^*v} \frac{vv^*}{v^*v} + 4 \frac{vv^*}{v^*v} \frac{vv^*}{v^*v} = I, \end{aligned}$$

so $F^* = F^{-1}$. In summary, we have constructed a unitary matrix Q_k that transforms the entries below the diagonal of the k th column of A_k to zero, and leaves the previous $k - 1$ columns alone.

Earlier, we mentioned that there is a choice of sign in v . This choice gives us the opportunity to improve the numerical stability of the algorithm. To avoid unnecessary numerical round off, we choose the sign that makes v furthest from x , i.e.

$$v = \text{sign}(x_1) \|x\| e_1 + x.$$

(Exercise, show that this choice of sign achieves this.)

We are now in a position to describe the algorithm in pseudo-code. Here it is described an “in-place” algorithm, where the successive transformations to the columns of A are implemented as replacements of the values in A . This means that we can allocate memory on the computer for A which is eventually replaced with the values for R . To present the algorithm, we will use the “slice” notation to describe submatrices of A , with $A_{k:l,r:s}$ being the submatrix of A consisting of the rows from k to l and columns from r to s . Be careful with Python implementations, where the numbering of rows and columns is from 0, and not 1.

- FOR $k = 1$ TO n
 - $x = A_{k:m,k}$
 - $v_k \leftarrow \text{sign}(x_1)\|x\|_2 e_1 + x$
 - $v_k \leftarrow v_k / \|v_k\|$
 - $A_{k:m,k:n} \leftarrow A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$.
- END FOR

Note that we have not explicitly formed the matrix Q or the product matrices Q_i . In some applications, such as solving least squares problems, we don’t explicitly need Q , just the matrix-vector product Q^*b with some vector b . To compute this product, we can just apply the same operations to b that are applied to the columns of A . This can be expressed in the following pseudo-code, working “in place” in the storage of b .

- FOR $k = 1$ TO n
 - $b_{k:m} \leftarrow b_{k:m} - 2v_k(v_k^* b_{k:m})$
- END FOR

If we really need Q , we can get it by matrix-vector products with each element of the canonical basis (e_1, e_2, \dots, e_n) . This means that first we need to compute a matrix-vector product Qx with a vector x . This is just done by applying the Householder reflections in reverse, since

$$Q = (Q_n \dots Q_2 Q_1)^* = Q_1 Q_2 \dots Q_n,$$

having made use of the fact that the Householder reflections are Hermitian. This can be expressed in the following pseudo-code.

- FOR $k = n$ TO 1 (DOWNWARDS)
 - $x_{k:m} \leftarrow x_{k:m} - 2v_k(v_k^* x_{k:m})$
- END FOR

Note that this requires to record all of the history of the v vectors, whilst the Q^* application algorithm above can be interlaced with the steps of the Householder algorithm, using the v values as they are needed and throwing them away. Then we can compute Q via

$$Q = (Qe_1 \quad Qe_2 \quad \dots \quad Qe_n),$$

with each column using the Q application algorithm described above.

2.7 Application: Least squares problems

Least square problems are relevant in data fitting problems, optimisation and control, and are also a crucial ingredient of modern massively parallel linear system solver algorithms. They are a way of solving “long thin” matrix vector problems $Ax = b$ where we want to obtain $x \in \mathbb{C}^m$ from $b \in \mathbb{C}^n$ with A an $n \times m$ matrix. Often the problem does not have a solution as it is overdetermined for $n > m$. Instead we just seek x that minimises the 2-norm of the residual $r = b - Ax$, i.e. x is the minimiser of

$$\min_x \|Ax - b\|^2.$$

This residual will not be zero in general, when b is not in the range of A . The nearest point in the range of A to b is Pb , where P is the orthogonal projector onto the range of A . From *Theoremnumber < dualcondition >*, we know that $P = \hat{Q}\hat{Q}^*$, where \hat{Q} from the reduced QR factorisation has the same column space as A (but with orthogonal columns).

Then, we just have to solve

$$Ax = Pb,$$

which is now solveable since Pb is in the column space of A (and hence can be written as a linear combination of the columns of A i.e. as a matrix-vector product Ax for some unknown x).

Now we have the reduced QR factorisation of A , and we can write

$$\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^*b.$$

Left multiplication by \hat{Q}^* then gives

$$\hat{R}x = \hat{Q}^*b.$$

This is an upper triangular system that can be solved efficiently using back-substitution (which we shall come to later.)

ANALYSING ALGORITHMS

In the previous section we saw three algorithms to compute the QR factorisation of a matrix. They have a beautiful mathematical structure based on orthogonal projectors. But are they useful? To answer this we need to know:

1. Is one faster than others?
2. Is one more sensitive than others to small perturbations due to early truncation of the algorithm or due to round-off errors?

In this course we will characterise answers to the first question by operation count (acknowledging that this is an incomplete evaluation of speed), and answers to the second question by analysing stability.

In this section we will discuss both of these questions by introducing some general concepts but also looking at the examples of the QR algorithms that we have seen so far.

3.1 Operation count

Operation count is one aspect of evaluating how long algorithms take. Here we just note that this is not the only aspect, since transferring data between different levels of memory on chips can be a serious (and often dominant) consideration, even more so when we consider algorithms that make use of large numbers of processors running in parallel. However, operation count is what we shall focus on here.

In this course, a floating point operation (FLOP) will be any arithmetic unary or binary operation acting on single numbers (such as $+$, $-$, \times , \div , $\sqrt{}$). Of course, in reality, these different operations have different relative costs, and codes can be made more efficient by blending multiplications and additions (fused multiply-adds) for example. Here we shall simply apologise to computer scientists in the class, and proceed with this interpretation, since we are just making relative comparisons between schemes. We shall also concentrate on asymptotic results in the limit of large n and/or m .

3.2 Operation count for modified Gram-Schmidt

We shall discuss operation counts through the example of the modified Gram-Schmidt algorithm. We shall find that the operation count is $\sim mn^2$ to compute the QR factorisation, where the \sim symbol means

$$\lim_{m,n \rightarrow \infty} \frac{N_{\text{FLOPS}}}{2mn^2} = 1.$$

To get this result, we return to the pseudocode for the modified Gram-Schmidt algorithm, and concentrate on the operations that are happening inside the inner j loop. Inside that loop there are two operations,

1. $r_{ij} \leftarrow q_i^* v_j$. This is the inner product of two vectors in \mathbb{R}^m , which requires m multiplications and $m - 1$ additions, so we count $2m - 1$ FLOPS per inner iteration.
2. $v_j \leftarrow v_j - r_{ij} q_i$. This requires m multiplications and m subtractions, so we count $2m$ FLOPS per inner iteration.

At each iteration we require a combined operation count of $\sim 4m$ FLOPS. There are n outer iterations over i , and $n - i - 1$ inner iterations over j , which we can estimate by approximating the sum as an integral,

$$N_{\text{FLOPS}} \sim \sum_{i=1}^n \sum_{j=i+1}^n 4m \sim 4m \sum_{i=1}^n i \int_1^n x dx \sim 4m \frac{n^2}{2} = 2mn^2,$$

as suggested above.

3.3 Operation count for Householder

In the Householder algorithm, the computation is dominated by the transformation

$$A_{k:m,k:n} \leftarrow A_{k:m,k:n} - \underbrace{2v_k \underbrace{(v_k^* A_{k:m,k:n})}_1}_{2},$$

which must be done for each ‘ k ’ iteration. To evaluate the part marked 1 requires $n - k$ inner products of vectors in \mathbb{C}^{m-k} , at a total cost of $\sim 2(n - k)(m - k)$ (we already examined inner products in the previous example). To evaluate the part marked 2 then requires the outer product of two vectors in \mathbb{C}^{m-k} and \mathbb{C}^{n-k} respectively, at a total cost of $(m - k)(n - k)$ FLOPS. Finally two $(k - m) \times (n - k)$ matrices are subtracted, at cost $(k - m)(n - k)$. Putting all this together gives $\sim 4(n - k)(m - k)$ FLOPS per k iteration.

Now we have to sum this over k , so the total operation count is

$$\begin{aligned} 4 \sum_{k=1}^n (n - k)(m - k) &= 4 \sum_{k=1}^n (nm - k(n + m) + k^2) \\ &\sim 4n^2m - 4(n + m) \frac{n^2}{2} + 4 \frac{n^3}{3} = 2mn^2 - \frac{2n^3}{3}. \end{aligned}$$