The Householder transformation in numerical linear algebra

John Kerl

February 3, 2008

Abstract

In this paper I define the Householder transformation, then put it to work in several ways:

- To illustrate the usefulness of geometry to elegantly derive and prove seemingly algebraic properties of the transform;
- To demonstrate an application to numerical linear algebra — specifically, for matrix determinants and inverses;
- To show how geometric notions of determinant and matrix norm can be used to easily understand round-off error in Householder and Gaussian-elimination methods.

These are notes to accompany a talk given to graduate students in mathematics. However, most of the content (except references to the orthogonal group) should be accessible to an undergraduate with a course in introductory linear algebra.

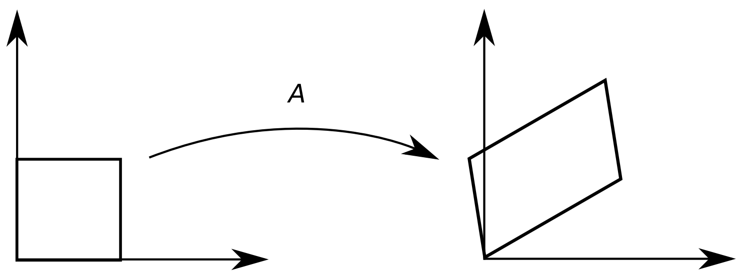
Contents

1 Linear algebra

In this paper I compare and contrast two techniques for computation of determinants and inverses of square matrices: the more-familiar Gaussian-elimination method, and the less-familiar Householder method. I primarily make use of geometric methods to do so. This requires some preliminaries from linear algebra, including geometric interpretations of determinant, matrix norm, and error propagation.

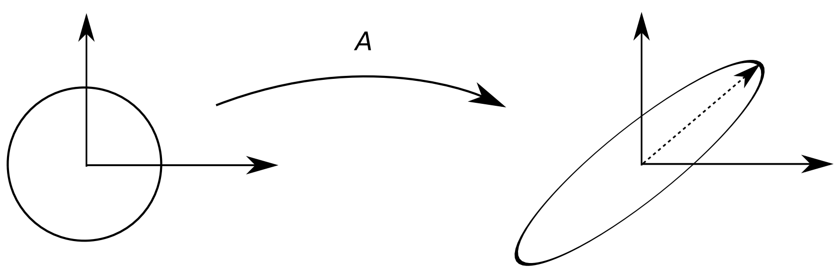
1.1 Geometric meanings of determinant and matrix norm

Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear transformation (which I will always think of as a matrix with respect to the standard basis). The **determinant** of A can be thought of as the **signed volume** of the image of the **unit cube**:



det(A) = volume of image of unit cube

The **matrix norm** is, by definition, the **maximum extent** of the image of the **unit ball**:



||A|| = maximum extent of image of unit ball

Definition 1.1. We define the matrix norm of A either by

$$\|A\| = \sup_{\|x\|=1} \|Ax\|$$

or equivalently by

$$\|A\| = \sup_{\|x\|\neq 0} \frac{\|Ax\|}{\|x\|}.$$

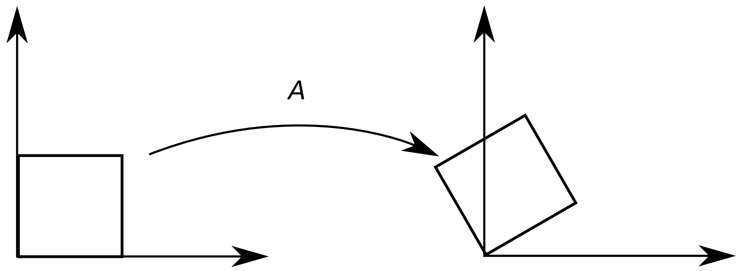
That is, either we consider vectors in the unit ball and find the maximum extent of their images, or we consider *all* non-zero vectors and find the maximum amount by which they are scaled. The equivalence of these two characterizations is due to the linearity of A.

A matrix with determinant ± 1 preserves (unsigned) volume, but does not necessarily preserve norm. Of particular interest for this paper are three kinds of matrices:

A 2×2 **rotation matrix** is of the form

$$A = \begin{bmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{bmatrix},$$

and has determinant 1:

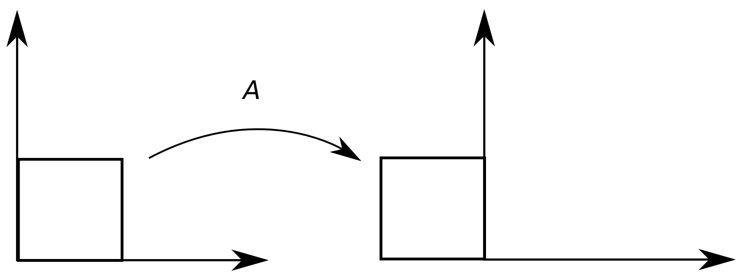


Rotation has determinant 1 and norm 1

An example of a 2×2 **reflection matrix**, reflecting about the y axis, is

$$A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix},$$

which has determinant -1:

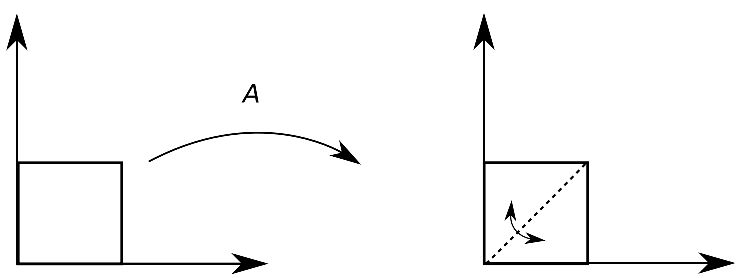


Reflection has determinant -1 and norm 1

Another example of a reflection is a **permutation matrix**:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

which has determinant -1:



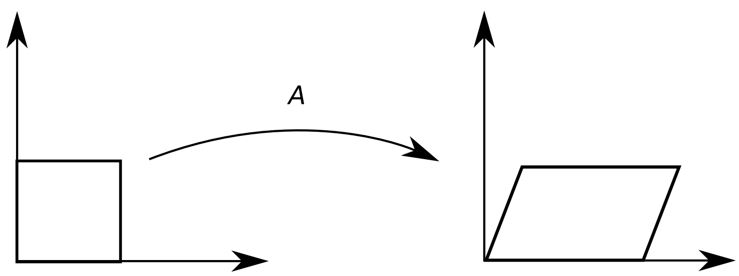
Permutation has determinant -1 and norm 1

This reflection is about the $45°$ line $x = y$. (Construction of a reflection matrix about an arbitrary axis is accomplished using Householder transformations, as discussed in section 3.)

An example of a 2×2 **shear matrix** is

$$A = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix},$$

which has determinant 1:



Shear has determinant 1 but not norm 1

1.2 Computation of determinants

In elementary linear algebra (see perhaps [**FIS**]), we are first taught to compute determinants using **cofactor expansion**. This is fine for computing determinants of 2×2's or 3×3's. However, it is ineffective for larger matrices.

- The determinant of a 2×2 matrix
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$
is $ad - bc$: two multiplies and an add.
- To compute the determinant of a 3×3 using cofactor expansion, we work down a row or column. There are 3 determinants of 2×2's.
- To do a 4×4, there are 4 determinants of 3×3's, each of which takes (recursively) 3 determinants of 2×2's.
- Continuing this pattern, we see that there are on the order of $n!$ multiplies and adds for an $n \times n$. For example, $25! \approx 10^{25}$. Even at a billion operations per second, this requires 10^{16} seconds, which is hundreds of millions of years. We can do better.

* * *

If a matrix is **upper-triangular** or **lower-triangular**, then its determinant is the product of its diagonal entries. There are n of these and we only need to multiply them, so there are n multiplies.

The question is, how efficiently can we *put* a given (square) matrix into upper-triangular form, and how does this modification affect the determinant?

Upper-triangularization is the process of putting zeroes in certain elements of a matrix, while modifying other entries. Recall that when a matrix Q acts by premultiplication on a matrix A, we can think of Q acting on each **column vector** of A. That is, the jth column of QA is simply Q times the jth column of A. And certainly we can transform column vectors to put zeroes in various locations.

How does this affect the determinant? If we apply n transformations, M_1 through M_m, then

$$\det(M_m \cdots M_2 M_1 A) = \det(M_m) \cdots \det(M_2)\det(M_1)\det(A)$$

i.e.

$$\det(A) = \frac{\det(M_m \cdots M_2 M_1 A)}{\det(M_m) \cdots \det(M_2)\det(M_1)} = \frac{\text{product along diagonal of upper-triangular matrix}}{\det(M_m) \cdots \det(M_2)\det(M_1)}.$$

That is, all we need to do is keep track of the determinants of our transformations, multiply them up, and compute the determinant of the resulting upper-triangular matrix.

How long does it take to compute and apply all these M_i's, though? It can be shown (but I will not show in this paper) that the Gaussian-elimination and Householder methods for upper-triangularization are on the order of n^3. Thus, these methods are far more efficient than naive cofactor expansion.

1.3 Computation of matrix inverses

In elementary linear algebra, we are taught to compute inverses using **cofactor expansion**. This also can be shown to require on the order of $n!$ operations.

* * *

A more efficient method, which we are also taught in elementary linear algebra, is to use an **augmented matrix**. That is, we paste our $n \times n$ input matrix A next to an $n \times n$ identity matrix:

$$[\, A \mid I \,]$$

and put the augmented matrix into reduced row echelon form. If it is possible to get the identity matrix on the left-hand side, then the inverse will be found in the right-hand side:

$$[\, I \mid A^{-1} \,].$$

Proposition 1.2. *This works.*

Proof. When we row-reduce the augmented matrix, we are applying a sequence $M_1, \ldots, M_m$ of linear transformations to the augmented matrix. Let their product be M:

$$M = M_n \cdots M_1.$$

Then, if the row-reduction works, i.e. if we get I on the left-hand side, then

$$M[\, A \mid I \,] = [\, I \mid * \,],$$

i.e. $MA = I$. But this forces $M = A^{-1}$. Thus

$$* = MI = A^{-1}I = A^{-1}.$$

□

Note that putting a matrix into reduced row echelon form is a two-step process:

- Putting it into row echelon form. This simply means putting it into **upper-triangular** form and diving each row by its leading coefficient:

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} 1 & * & * & * & * & * \\ 0 & 1 & * & * & * & * \\ 0 & 0 & 1 & * & * & * \end{bmatrix}.$$

- Clearing the above-diagonal entries:

$$\begin{bmatrix} 1 & * & * & * & * & * \\ 0 & 1 & * & * & * & * \\ 0 & 0 & 1 & * & * & * \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & * & * & * \\ 0 & 1 & 0 & * & * & * \\ 0 & 0 & 1 & * & * & * \end{bmatrix}.$$

The remaining question (to be addressed below) is what particular sequence of M_i's to apply.

1.4 Error propagation

Whenever we use numerical techniques, we need to ask about error. A poorly designed algorithm can permit error to grow ridiculously, producing nonsense answers. This paper is about well-designed algorithms. The study of error is **numerical analysis** and is discussed, for example, in [**BF**] and [**NR**]. Here I simply make a few points.

Consider a true matrix A at the kth step of an upper-triangularization process, along with an actual matrix $A + \varepsilon$ where ε is a matrix. We want to know how error **propagates** when we apply a transformation Q. Since we use linear transformations, we have

$$Q(A + \varepsilon) = Q(A) + Q(\varepsilon).$$

Also, since the jth column of QA is Q times the jth column of A, we have

$$Q(\mathbf{v} + \eta) = Q(\mathbf{v}) + Q(\eta)$$

where $\mathbf{v}$ and η are column vectors. The **vector norm** of $Q(\eta)$ is related to the vector norm of η by the **matrix norm** of Q: since (from definition 1.1)

$$\|Q\| \geq \frac{\|Q(\eta)\|}{\|\eta\|},$$

we have

$$\|Q(\eta)\| \leq \|Q\| \, \|\eta\|.$$

In particular, if a transformation matrix Q is norm-preserving, it does not increase error.

Next we consider the norms of the matrices which are used in the Gaussian-elimination and Householder methods.

2 Gaussian elimination

In this section, Gaussian elimination is discussed from the viewpoint of concatenated linear transformations, with an emphasis on geometrical interpretation of those transformations.

2.1 Row reduction using Gaussian elimination

For clarity, consider matrices of height 2. Let's row-reduce an example matrix:

$$\begin{bmatrix} 0 & 3 \\ 1 & 2 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

This example illustrates the three matrices which we use in Gaussian elimination:

The **row-swap matrix** (a permutation matrix from section 1.1) has determinant -1 and norm 1:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The **row-scaling matrix** has determinant m (in the example, $m = 1/3$) and norm $\max(1, m)$:

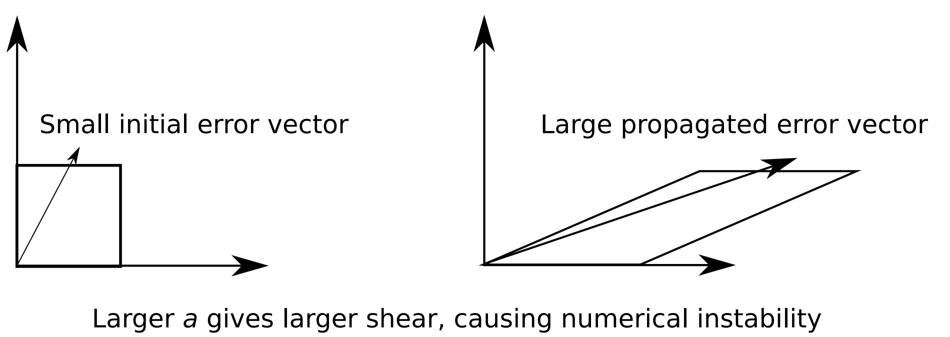
$$\begin{bmatrix} 1 & 0 \\ 0 & m \end{bmatrix}$$

The **row-update matrix** (a shear matrix from section 1.1) has determinant 1 and a norm which we can most easily visualize using the diagrams in section 1.1:

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}.$$

(In the example, $a = -2$.)

It's clear that Gaussian elimination magnifies error when a row-scaling matrix has large $|m|$, and/or when a row-update matrix has a large $|a|$.



2.2 Gaussian elimination without pivoting

Here is an example of the error which can accumulate when we naively use Gaussian elimination. First let's solve the linear system

$$\begin{aligned} y &= 1 \\ x - y &= 0 \end{aligned}$$

which is, in matrix form,

$$\begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

or in augmented form

$$\left[\begin{array}{cc|c} 0 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right].$$

Clearly, the solution is

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Now let's perturb it a bit, solving the system

$$\left[\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right].$$

We expect a solution close to $(1, 1)$. Here is Gaussian elimination, rounded to 4 and 3 decimal places, respectively, with no pivoting operations. (3 decimal places means that, in particular, 1001 rounds to 1000.) At each step I include the transformation matrix which takes us to the next step, along with the norm of that transformation matrix.

<table>
<tr><th colspan="3">4 places</th><th colspan="3">3 places</th></tr>
<tr><td>(norm 1000)</td><td>$\begin{bmatrix} 1000 & 0 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right]$</td><td>(norm 1000)</td><td>$\begin{bmatrix} 1000 & 0 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right]$</td></tr>
<tr><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 1 & -1 & 0 \end{array}\right]$</td><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 1 & -1 & 0 \end{array}\right]$</td></tr>
<tr><td>(norm $= 1$)</td><td>$\begin{bmatrix} 1 & 0 \\ 0 & -1/1001 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 0 & -1001 & -1000 \end{array}\right]$</td><td>(norm $= 1$)</td><td>$\begin{bmatrix} 1 & 0 \\ 0 & -1/1000 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 0 & -1000 & -1000 \end{array}\right]$</td></tr>
<tr><td>(norm $= 1000$)</td><td>$\begin{bmatrix} 1 & -1000 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 0 & 1 & 0.999 \end{array}\right]$</td><td>(norm $= 1000$)</td><td>$\begin{bmatrix} 1 & -1000 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{cc|c} 1 & 1000 & 1000 \\ 0 & 1 & 1 \end{array}\right]$</td></tr>
<tr><td></td><td></td><td>$\left[\begin{array}{cc|c} 1 & 0 & 0.999 \\ 0 & 1 & 0.999 \end{array}\right]$</td><td></td><td></td><td>$\left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 1 \end{array}\right]$</td></tr>
</table>

Here, the roundoff error dominates: the right-hand solution is very wrong. You can check that the left-hand solution is nearly correct. There were two norm-1000 operations, contributing to the error.

2.3 Gaussian elimination with pivoting

Now let's do the same example, but with pivoting. This means that we use a permutation matrix to place the largest absolute-value column head at the top of a column.

Again using 4 and 3 decimal places, respectively, we have the following.

<table>
<tr><th colspan="3">4 places</th><th colspan="3">3 places</th></tr>
<tr><td>(norm 1)</td><td>$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 0.001 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right]$</td><td>(norm 1)</td><td>$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 0.001 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right]$</td></tr>
<tr><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 0 \\ -1/1000 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 1 & -1 & 0 \\ 0.001 & 1 & 1 \end{array}\right]$</td><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 0 \\ -1/1000 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 1 & -1 & 0 \\ 0.001 & 1 & 1 \end{array}\right]$</td></tr>
<tr><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 1/1.001 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 1 & -1 & 0 \\ 0 & 1.001 & 1 \end{array}\right]$</td><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 1 & -1 & 0 \\ 0 & 1 & 1 \end{array}\right]$</td></tr>
<tr><td>(norm ≈ 1)</td><td>$\begin{bmatrix} 1 & 0 \\ 0 & 1/1.001 \end{bmatrix}$</td><td>$\left[\begin{array}{rr|r} 1 & 0 & 0.999 \\ 0 & 1.001 & 1 \end{array}\right]$</td><td></td><td></td><td>$\left[\begin{array}{rr|r} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right]$</td></tr>
<tr><td></td><td></td><td>$\left[\begin{array}{rr|r} 1 & 0 & 0.999 \\ 0 & 1 & 0.999 \end{array}\right]$</td><td></td><td></td><td></td></tr>
</table>

Here, both solutions are approximately equal and approximately correct. All the transformations have norm on the order of 1, which in turn is the case because of the pivoting operation.

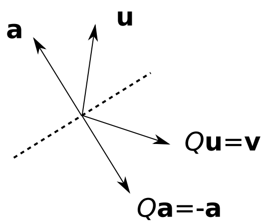
The point is that successful Gaussian elimination *requires* pivoting, which is a **data-dependent decision**. Also, we must store the **permutations**. Neither of these is a serious detriment in a carefully implemented software system. We will next look at Householder transformations not as a *necessity*, but as an *alternative*. It will turn out that they will permit somewhat simpler software implementations.

3 Householder transformations

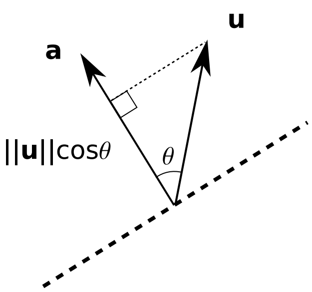
In this section the Householder transformation is derived, then put to use in applications.

3.1 Geometric construction

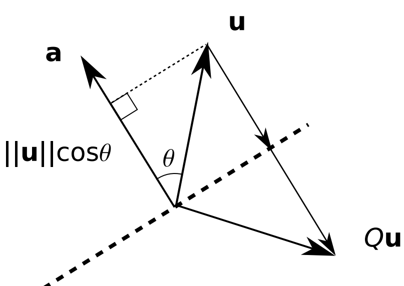
Here we construct a **reflection matrix** Q acting on $\mathbb{R}^n$ which sends a chosen **axis vector**, $\mathbf{a}$, to its negative, and reflects all other vectors through the hyperplane perpendicular to $\mathbf{a}$:



How do we do this? We can decompose a given matrix $\mathbf{u}$ into its components which are parallel and perpendicular to $\mathbf{a}$:



Then, if we can subtract off from $\mathbf{u}$ twice its parallel component $\mathbf{u}_{||}$, then we'll have obtained the reflection:



That is, we want

$$Q\mathbf{u} = \mathbf{u} - 2\mathbf{u}_{||}.$$

Moreover, we would like a single matrix Q which would transform all $\mathbf{u}$'s.

What is the projection vector $\mathbf{u}_{||}$? It must be in the **direction** of $\mathbf{a}$, and it must have **magnitude** $\|\mathbf{u}\| \cos\theta$:

$$\mathbf{u}_{||} = \hat{\mathbf{a}}\|\mathbf{u}\| \cos\theta = \frac{\mathbf{a}}{\|\mathbf{a}\|}\|\mathbf{u}\| \cos\theta.$$

I use the notation

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|}.$$

It turns out (this is the central trick of the Householder transformation) that we can reformulate this expression in terms of dot products, eliminating $\cos\theta$. To do this, recall that

$$\mathbf{a} \cdot \mathbf{u} = \|\mathbf{a}\| \, \|\mathbf{u}\| \cos\theta.$$

So,

$$\mathbf{u}_{||} = \frac{\mathbf{a}}{\|\mathbf{a}\|^2} \|\mathbf{a}\| \, \|\mathbf{u}\| \cos\theta = \frac{\mathbf{a}}{\|\mathbf{a}\|^2} \mathbf{a} \cdot \mathbf{u}.$$

Also recall that

$$\|\mathbf{a}\|^2 = \mathbf{a} \cdot \mathbf{a} \qquad \text{and} \qquad \|\mathbf{u}\|^2 = \mathbf{u} \cdot \mathbf{u}.$$

Now we have

$$\mathbf{u}_{||} = \mathbf{a} \frac{\mathbf{a} \cdot \mathbf{u}}{\mathbf{a} \cdot \mathbf{a}}.$$

Now we can use this to rewrite the reflection of $\mathbf{u}$:

$$\begin{aligned} Q\mathbf{u} &= \mathbf{u} - 2\mathbf{u}_{||} \\ &= \mathbf{u} - 2\mathbf{a} \frac{\mathbf{a} \cdot \mathbf{u}}{\mathbf{a} \cdot \mathbf{a}}. \end{aligned}$$

Now (here's the other trick of Householder) remember that we can write the dot product, or **inner product**, as a matrix product:

$$\mathbf{a} \cdot \mathbf{a} = \mathbf{a}^t \mathbf{a}.$$

Here, we think of a column vector as an $n \times 1$ matrix, while we think of a row vector as a $1 \times n$ matrix which is the transpose of the column vector. So,

$$Q\mathbf{u} = \mathbf{u} - 2\mathbf{a} \frac{\mathbf{a}^t \mathbf{u}}{\mathbf{a}^t \mathbf{a}}.$$

But now that we've recast the dot products in terms of matrix multiplication, we can use the associativity of matrix multiplication:

$$\mathbf{a}(\mathbf{a}^t \mathbf{u}) = (\mathbf{a}\mathbf{a}^t)\mathbf{u}.$$

The product $\mathbf{a}\mathbf{a}^t$ is the **outer product** of $\mathbf{a}$ with itself. It is an $n \times n$ matrix with ijth entry $a_i a_j$. Now we have

$$Q\mathbf{u} = \left(I - 2\frac{\mathbf{a}\mathbf{a}^t}{\mathbf{a}^t \mathbf{a}} \right) \mathbf{u}$$

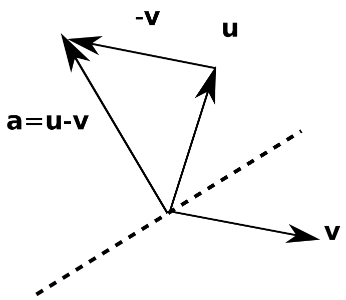
from which

$$Q = I - 2\frac{\mathbf{a}\mathbf{a}^t}{\mathbf{a}^t \mathbf{a}}.$$

3.2 Construction with specified source and destination

In section 3.1, we saw how to construct a Householder matrix when given an axis of reflection $\mathbf{a}$. This would send any other vector $\mathbf{u}$ to its image $\mathbf{v}$ through the hyperplane perpendicular to $\mathbf{a}$.

However, what if we were given $\mathbf{u}$ and $\mathbf{v}$, and wanted to find the axis of reflection $\mathbf{a}$ through which $\mathbf{v}$ is $\mathbf{u}$'s mirror image? First note that since Householder transformations are norm-preserving, $\mathbf{u}$ and $\mathbf{v}$ must have the same norm. (Normalize first them if not.) Drawing a picture, and using the head-to-tail method, it looks like $\mathbf{a} = \mathbf{u} - \mathbf{v}$ ought to work:



We can check that this is correct. If $\mathbf{a}$ is indeed the reflection axis, then it ought to get reflected to its own negative:

$$\begin{aligned} Q(\mathbf{a}) &= Q(\mathbf{u} - \mathbf{v}) \\ &= Q(\mathbf{u}) - Q(\mathbf{v}) \\ &= \mathbf{v} - \mathbf{u} \\ &= -\mathbf{a}. \end{aligned}$$

3.3 Properties of Q, obtained algebraically

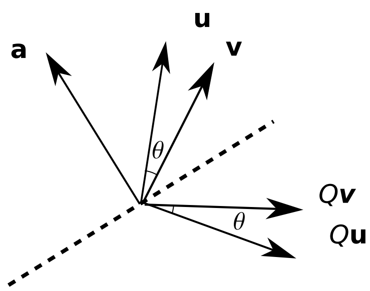
Here we attempt to prove algebraically that the Householder matrix has certain properties. In the next section, we do the same proofs geometrically.

- Q is **orthogonal**, i.e. $Q^t Q = I$: Compose $(I - 2\mathbf{a}\mathbf{a}^t/\mathbf{a}^t\mathbf{a})$ with itself and FOIL it out.
- Q is **symmetric**, i.e. $Q = Q^t$. This is easy since $Q = I - 2\mathbf{a}\mathbf{a}^t/\mathbf{a}^t\mathbf{a}$. The identity is symmetric; $\mathbf{a}\mathbf{a}^t$ has ijth entry $a_i a_j = a_j a_i$ and so is symmetric as well.
- Q is **involutary**, i.e. $Q^2 = I$: Same as orthogonality, due to symmetry, since $Q = Q^t$.
- **Determinant**: I don't see a nice way to find this algebraically. It seems like it would be a mess.
- **Matrix norm**: Likewise.

3.4 Properties of Q, obtained geometrically

In this section, we prove certain properties of the Householder matrix using geometric methods. Remember that dot products, via $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \, \|\mathbf{v}\| \cos\theta$, give us norms as well as angles.

- Q is **involutary**, i.e. $Q^2 = I$: Q reflects $\mathbf{u}$ to its mirror image $\mathbf{v}$; a second application of Q sends it back again.
- Q is **orthogonal**, i.e. $(Q\mathbf{u} \cdot Q\mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$ for all vectors $\mathbf{u}, \mathbf{v}$: Since Q is a reflection, it preserves norms; also, from the picture, it's clear that it preserves angles:



- Q is **symmetric**, i.e. $(Q\mathbf{u}) \cdot \mathbf{v} = \mathbf{u} \cdot (Q\mathbf{v})$ for all vectors $\mathbf{u}, \mathbf{v}$: This is the same geometric argument as for orthogonality, making use of involutarity.
- **Determinant**: Since Q is a reflection about the $\mathbf{a}$ axis, leaving all the axes orthogonal to $\mathbf{a}$ fixed, Q must have determinant -1. That is, it turns the unit cube inside out along one axis.
- **Matrix norm**: Since Q is a reflection, it is length-preserving.

3.5 Repeated Householders for upper-triangularization

The goal of upper triangularization is to put a matrix of the form

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

into the form

$$\begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix}.$$

We can do this one step at a time: from

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

to

$$\begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$$

to

$$\begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix}.$$

The first step (we will use a Householder transformation for this) is on all of the $m \times n$ input matrix, with an operation which transforms the first column. The second step is on the $(m-1) \times (n-1)$ submatrix obtained by omitting the top row and left column:

$$\left[\begin{array}{c|ccccc} * & * & * & * & * & * \\ \hline 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{array}\right].$$

We can keep operating on submatrices until there are no more of them. So, the problem of upper-triangularization reduces to that of modifying the first column of a matrix to have all non-zero entries except the entry at the top of the column.

3.6 Householders for column-zeroing

The goal is to put a matrix

$$A = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

into the form

$$QA = \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$$

The key insight is that when Q acts on A by left multiplication, the jth column of QA is the matrix-times-vector product of Q times the jth column of A.

Let $\mathbf{u}$ = column 0 of A, and $\mathbf{v}$ = column 0 of QA. At this point all we know is that we want $\mathbf{v}$ to have all zeroes except the first entry. But if we're going to use Householder transformations, which are norm-preserving, we know that $\|\mathbf{v}\| = \|\mathbf{u}\|$. So, $v_1 = \pm\|\mathbf{u}\|$. (We'll see in the next paragraph how to choose the postive or negative sign.) We want a Householder matrix Q which sends $\mathbf{u}$ to $\mathbf{v}$, and which can do what it will with the rest of the matrix as a side effect.

What is the reflection axis $\mathbf{a}$? This is just as in section 3.2: $\mathbf{a} = \mathbf{u} - \mathbf{v}$. Above we had $v_1 = \pm\|\mathbf{u}\|$. Since what we're going to do with that is to compute $\mathbf{a} = \mathbf{u} - \mathbf{v}$, we can choose v_1 to have the opposite sign of u_1, to avoid the cancellation error which can happen when we subtract two nearly equal numbers.

Error analysis: Since these Q's are orthogonal, they're norm-preserving, and so

$$Q(\mathbf{u} + \varepsilon) = Q(\mathbf{u}) + Q(\varepsilon)$$

but

$$\|Q(\varepsilon)\| \leq \|\varepsilon\|$$

as discussed in section 1.4.

Also, there is no pivoting involved, and thus (other than the choice of the sign of v_1) no **data-dependent decisions**.

3.7 Computation of determinants

Given a square matrix A, we can use repeated Householder transformations to turn it into an upper-triangular matrix U. As discussed in section 1.2, $\det(A)$ is the product of diagonal entries of U, divided by the product of the determinants of the transformation matrices. However, as seen in sections 3.3 and 3.4, Householder matrices have determinant -1. So, we just have to remember whether the number of Householders applied was odd or even. But it takes $n-1$ Householders to upper-triangularize an $n \times n$ matrix:

$$\begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix}$$

So,

$$\det(A) = (-1)^{n-1} \det(U).$$

3.8 Computation of matrix inverses

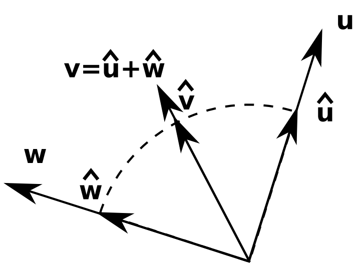
Given an invertible (square) matrix A, we can make an augmented matrix just as in section 1.3. We can use Householder upper-triangularization to put the augmented matrix into upper-triangular form, i.e. row echelon form. The rest of the work is in clearing out the above-diagonal entries of the left-hand side of the augmented matrix, for which we can use scale and shear matrices as in section 1.1.

3.9 Rotation matrices

$O(n)$ has two connected components: $SO(n)$ and $O^-(n)$. The former are (rigid) **rotations** (determinant $+1$); the latter are **reflections** (determinant -1). (See [**Fra**] or [**Simon**] for more information on $SO(n)$.) One might ask: If Householder transformations provide easily understandable, readily computable elements of $O^-(n)$, then is there another, equally pleasant technique to compute elements of $SO(n)$? In fact there is. One can consult the literature for the Rodrigues formula, and modify that.

Alternatively, one can use the fact that since determinants multiply, the product of two reflections, each having determinant -1, will have determinant 1. Thus the product will be a rotation. If the same reflection is composed with itself, the product will be the identity, as seen in sections 3.3 and 3.4. But the product of two *distinct* reflections will be a non-trivial rotation.

Let $\mathbf{u}$ and $\mathbf{w}$ be given. Since rotations are norm-preserving, we can only hope to rotate $\mathbf{u}$ into $\mathbf{w}$ if they both have the same norm. We can help this by computing $\hat{\mathbf{u}}$ and $\hat{\mathbf{w}}$. Now note that $\hat{\mathbf{u}} + \hat{\mathbf{w}}$ will lie halfway between $\hat{\mathbf{u}}$ and $\hat{\mathbf{w}}$:



Let $\mathbf{v} = \hat{\mathbf{u}} + \hat{\mathbf{w}}$, and compute $\hat{\mathbf{v}}$. Then let Q_1 reflect $\hat{\mathbf{u}}$ to $\hat{\mathbf{v}}$. Just as in section 3.2, this is a reflection about an axis $\mathbf{a}_1 = \hat{\mathbf{u}} - \hat{\mathbf{v}}$. Likewise for the second reflection. We then have

$$R = Q_2 Q_1 = \left(I - 2\frac{\mathbf{a}_2\mathbf{a}_2^t}{\mathbf{a}_2^t\mathbf{a}_2} \right) \left(I - 2\frac{\mathbf{a}_1\mathbf{a}_1^t}{\mathbf{a}_1^t\mathbf{a}_1} \right).$$

3.10 Software

The algorithms described in this document have been implemented in the Python programming language in the files

- `http://math.arizona.edu/~kerl/python/sackmat_m.py`
- `http://math.arizona.edu/~kerl/python/pydet`
- `http://math.arizona.edu/~kerl/python/pymatinv`
- `http://math.arizona.edu/~kerl/python/pyhh`

- `http://math.arizona.edu/~kerl/python/pyrefl`
- `http://math.arizona.edu/~kerl/python/pyrot`

The first is library file containing most of the logic; the remaining four handle I/O and command-line arguments, and call the library routines.

- The library routine `householder_vector_to_Q` takes a vector **a** and produces a matrix Q, as described in section 3.1.
- The program `pyhh` is a direct interface to this routine.
- The library routine `householder_UT_pass_on_submatrix` takes a matrix and a specified row/column number, applying a Householder transformation on a submatrix, as described in section 3.5.
- The library routine `householder_UT` puts a matrix into upper-triangular form, using repeated calls to `householder_UT_pass_on_submatrix`.
- The program `pydet` computes determinants as described in section 3.7.
- The program `pymatinv` computes matrix inverses as described in section 3.8.
- The program `pyrefl` reads two vectors **u** and **v**, normalizes them to $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$, and then uses the technique described in section 3.2 to compute a reflection matrix sending $\hat{\mathbf{u}}$ to $\hat{\mathbf{v}}$.
- The program `pyrot` reads two vectors **u** and **v**, normalizes them to $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$, and then uses the technique described in section 3.9 to compute a rotation matrix sending $\hat{\mathbf{u}}$ to $\hat{\mathbf{v}}$.

The Python programming language is clear and intuitive. (Whether my coding style is equally clear and intuitive is left to the judgment of the reader). As a result, these algorithms should be readily translatable to, say, Matlab.

4 Acknowledgments

Thanks to Dr. Ross Martin of Lockheed Martin Corporation and Dr. John Palmer of the University of Arizona for many productive conversations.

References

[**BF**] Burden, R. and Faires, J. *Numerical Analysis* (4th ed.). PWS-KENT, 1989.

[**Fra**] Frankel, T. *The Geometry of Physics: An Introduction* (2nd ed). Cambridge University Press, 2004.

[**FIS**] H. Friedberg, A. Insel, and L. Spence. *Linear Algebra* (3rd ed). Prentice Hall, 1997.

[**NR**] Press, W. et al. *Numerical Recipes* (2nd ed.). Cambridge, 1992.

[**Simon**] B. Simon *Representations of Finite and Compact Groups*. American Mathematical Society, 1996.

Index