CS 4220: Numerical Analysis

From power methods to QR iteration

David Bindel

2026-03-06

Orthogonal iteration to QR

The *QR iteration* is the workhorse for solving the nonsymmetric eigenvalue problem. Unfortunately, while the iteration itself is simple to write, the derivation sometimes appears to be a work of black magic. In fact, the QR iteration is essentially the subspace iteration we have already seen, re-cast in a different form.

1. The orthogonal iteration $\underline{Q}^{(k+1)}R^{(k)} = A\underline{Q}^{(k)}$ is a generalization of the power method. In fact, the first column of this iteration is *exactly* the power iteration. In general, the first p columns of $\underline{Q}^{(k)}$ are converging to an orthonormal basis for a p-dimensional invariant subspace associated with the p eigenvalues of A with largest modulus (assuming that there aren't several eigenvalues with the same modulus to make this ambiguous).
2. If all the eigenvalues have different modulus, orthogonal iteration ultimately converges to the orthogonal factor in a Schur form
$$AU = UT$$
What about the T factor? Note that $T = U^* A U$, so a natural approximation to T at step k would be
$$A^{(k)} = (\underline{Q}^{(k)})^* A \underline{Q}^{(k)},$$
and from the definition of the subspace iteration, we have
$$A^{(k)} = (\underline{Q}^{(k)})^* \underline{Q}^{(k+1)} R^{(k)} = Q^{(k)} R^{(k)},$$
where $Q^{(k)} \equiv (\underline{Q}^{(k)})^* \underline{Q}^{(k+1)}$ is unitary.
3. Note that
$$A^{(k+1)} = (\underline{Q}^{(k+1)})^* A^{(k)} \underline{Q}^{(k+1)} = (Q^{(k)})^* A^{(k)} Q^{(k)} = R^{(k)} Q^{(k)}.$$

Thus, we can go from $A^{(k)}$ to $A^{(k+1)}$ directly without the orthogonal factors from subspace iteration, simply by computing

$$\begin{aligned} A^{(k)} &= Q^{(k)}R^{(k)} \\ A^{(k+1)} &= R^{(k)}Q^{(k)}. \end{aligned}$$

This is the QR iteration.

Practical problems

There are two major problems with the basic QR iteration:

1. Each step of the QR iteration requires a QR factorization, which is an $O(n^3)$ operation. This is rather expensive, and even in the happy case where we might be able to get each eigenvalue with a constant number of steps, $O(n)$ total steps at a cost of $O(n^3)$ each gives us an $O(n^4)$ algorithm. Given that everything else we have done so far costs only $O(n^3)$, an $O(n^4)$ cost for eigenvalue computation seems excessive.
2. Like the power iteration upon which it is based, the basic iteration converges linearly, and the rate of convergence is related to the ratios of the moduli of eigenvalues. Convergence is slow when there are eigenvalues of nearly the same modulus, and nonexistent when there are eigenvalues with the same modulus.

The latter problem is dealt with by introducting shifts into QR, much like we did with inverse iteration. It is an interesting topic, but not one we will take up here. Instead, we will focus on the first problem, which leads us to several interesting intermediate factorizations.

Woah, We're Halfway There

The QR iteration maps upper Hessenberg matrices to upper Hessenberg matrices, and this fact allows us to do one QR sweep in $O(n^2)$ time. So how do we reduce to upper Hessenberg in the first place? Or, for that matter, to to other "halfway there" forms:

$$\begin{aligned} &A = QHQ^T && \text{where } Q \text{ is orthogonal and } H \text{ upper Hessenberg} \\ &A = QTQ^T && \text{where } Q \text{ is orthogonal, } T \text{ symmetric tridiagonal, and } A \text{ symmetric} \\ &A = UBV^T && \text{where } U \text{ and } V \text{ are orthogonal and } B \text{ is upper bidiagonal} \end{aligned}$$

Unlike the SVD or various eigendecompositions, these forms can be computed directly. And in many cases, as we will discuss, they are just as useful!

Hessenberg via Householder

Let's start with an over-ambitious goal[1]: can we directly compute the Schur form by applying Householder transformations on the left and right of a matrix? We already secretly know that the answer is no, but trying and failing will set us up neatly for computing a *Hessenberg form* — and for understanding why the Hessenberg form is in fact a natural target. A matrix H is *upper Hessenberg* if it has nonzeros only in the upper triangle and the first subdiagonal. For example, the nonzero structure of a 5-by-5 Hessenberg matrix is

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}.$$

For any square matrix A, we can find a unitarily similar Hessenberg matrix $H = Q^* A Q$ in $O(n^3)$ time (a topic for next time). Because H is similar to A, they have the same eigenvalues; but as it turns out, the special structure of the Hessenberg matrix makes it possible to run QR in $O(n^2)$ per iteration.

How might we go about trying to compute a Schur form directly? A natural first step is to apply an orthogonal transformation from the left in order to introduce zeros in the first column, and then apply the same transformation on the right. Let's try this for the 4×4 matrix case (we'll mark with a star each nonzero element that is updated by the previous transformation):

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

Oops! Applying the transformation from the right scrambles the columns, and undoes the zeros we started just introduced. The problem is that we got too greedy. What if instead of trying to zero out everything below the diagonal, we instead zero out everything below the first subdiagonal? Then we affect rows 2 through n, and a subsequent transform affecting columns 2 through n does not kill the zeros we introduced:

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

[1]Ah, but a man's reach should exceed his grasp...

A second step to introduce zeros in the second column finishes the reduction to Hessenberg form:

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

Let's make this concrete:

```
function my_hessenberg(A)
    H = copy(A)
    n = size(A)[1]
    Q = Matrix{Float64}(I,n,n)

    for j = 1:n-2
        u = householder(H[j+1:n,j])
        H[j+1:n,j:n] -= 2*u*(u'*H[j+1:n,j:n])
        H[j+2:n,j]   .= 0.0
        H[:,j+1:n] -= 2*(H[:,j+1:n]*u)*u'
        Q[:,j+1:n] -= 2*(Q[:,j+1:n]*u)*u'
    end

    Q, H
end
```

Hessenberg matrices and $O(n^2)$ QR steps

The special structure of the Hessenberg matrix makes the Householder QR routine very economical. The Householder reflection computed in order to introduce a zero in the $(j+1, j)$ entry needs only to operate on rows j and $j+1$. Therefore, we have

$$Q^* H = W_{n-1} W_{n-2} \dots W_1 H = R,$$

where W_j is a Householder reflection that operates only on rows j and $j+1$. Computing R costs $O(n^2)$ time, since each W_j only affects two rows ($O(n)$ data). Now, note that

$$RQ = R(W_1 W_2 \dots W_{n-1});$$

that is, RQ is computed by an operation that first mixes the first two columns, then the second two columns, and so on. The only subdiagonal entries that can be introduced in this process lie on the first subdiagonal, and so RQ is again a Hessenberg matrix. Therefore, one step of QR iteration on a Hessenberg matrix results in another Hessenberg matrix, and a Hessenberg QR iteration step can be performed in $O(n^2)$ time.

As it happens, the Hessenberg QR step can be written in terms of *bulge chasing*. The picture (in the 5-by-5 case) is as follows. We start with the original Hessemberg matrix, and first apply an orthogonal transformation (the first in a QR factorization) to the first two rows and then symmetrically to the first two columns. This introduces a nonzero (a "bulge") in the $(3, 1)$ position. Marking the elements modified in the first step with a star, we have:

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

From here, our goal is to "chase" the bulge out of the Hessenberg structure. We start by applying an orthogonal transformation to rows 2 and 3 to remove the $(3, 1)$ element; applying the same transformation to columns 2 and 3 introduces a bulge element in the $(4, 2)$ position; marking the newly modified elements with stars, we have the new structure

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

Continuing for two more step in a similar fashion, we have a nonzero in the $(5, 3)$ position, and then can restore the structure the rest of the way to a Hessenberg form.

Terrific Tridiagonals and Busy Bidiagonals

When A is a symmetric matrix, the corresponding Hessenberg form is symmetric as well. This means that all the entries below the first subdiagonal are zero (by upper Hessenberg structure), and so are all the entries above the first superdiagonal (by symmetry).

What about bidiagonalization? In bidiagonalization, we again interleave transformations on the left and right, but now we allow ourselves to use different transformations. The key is that we want to keep introducing new zero elements through these transformations without destroying zeros we already created. In the 4×4 case, the first two steps look like

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

The code is

```
function my_bidiagonalization(A)
    B = copy(A)
    n = size(A)[1]
    U = Matrix{Float64}(I,n,n)
    V = Matrix{Float64}(I,n,n)

    for j = 1:n-1

        # Transform from the left to introduce zeros in column j
        u = householder(B[j:n,j])
        B[j:n,j:n] -= 2*u*(u'*B[j:n,j:n])
        B[j+1:n,j] .= 0.0
        U[:,j:n] -= 2*(U[:,j:n]*u)*u'

        # Transform from the right to introduce zeros in row j
        v = householder(B[j,j+1:n])
        B[j:n,j+1:n] -= 2*(B[j:n,j+1:n]*v)*v'
        B[j,j+2:n] .= 0.0
        V[:,j+1:n] -= 2*(V[:,j+1:n]*v)*v'

    end

    B, U, V
end
```

Using the Factorizations

Let's look concretely at two cases where these factorizations are useful. The first is in computing *transfer functions* that occur in control theory, where we have the form

$$T(s) = c^T(sI - A)^{-1}b + d$$

for various values of s. Naively, it looks like we might have to spend $O(n^3)$ per value of s where we want the transfer function; but we can instead use the Hessenberg reduction $A = QHQ^T$ to get

$$T(s) = (c^TQ)(sI - H)^{-1}(Q^Tb) + d$$

Solving a Hessenberg linear system like $(sI - H)$ can be done in $O(n^2)$ time rather than $O(n^3)$ time (why?); in fact, this is one of the structures that MATLAB's sparse solver checks for.

A second application again deals with parameter-dependent systems, but in a different setting. Suppose $A = UBV^T$ is a bidiagonal reduction of A, and we are interested in computing the

Tikhonov-regularized solution for different values of the regularization parameter. How can we do this efficiently? Note that

$$\left\|\begin{bmatrix} A \\ \lambda I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix}\right\| = \left\|\begin{bmatrix} B \\ \lambda I \end{bmatrix} (V^T x) - \begin{bmatrix} U^T b \\ 0 \end{bmatrix}\right\|$$

The latter equations can be reduced back to bidiagonal form in $O(n)$ time (left as an exercise for the student!). Therefore, after the initial bidiagonal reduction, we can solve the least squares problem for each new value of the regularization parameter λ in $O(n)$ additional work.

Enter Arnoldi

When we discussed QR factorization, we started with Gram-Schmidt and then moved to the Householder-based method. In talking about reduction to Hessenberg form, we started with the Householder approach — but there is something akin to Gram-Schmidt as well. We can read the basic idea by looking at the columns of the Hessenberg matrix equation $AQ = QH$:

$$Aq_j = \sum_{k=1}^{j} q_j h_{jk} + h_{j,j+1} q_{j+1}.$$

Rearranging, we have

$$q_{j+1} = \frac{1}{h_{j,j+1}} \left(Aq_j - \sum_{k=1}^{j} q_k h_{kj} \right)$$

where $h_{kj} = q_k^T A q_j$. That is, the coefficients H can be exactly derived from using Gram-Schmidt orthgonalization to orthonormalize Aq_j against $\{q_1, \dots, q_j\}$ for each successive j!

The beauty about using the Arnoldi procedure to reduce a matrix to upper Hessenberg form is two fold: we can readily apply the method to sparse matrices; and we can *stop early!*. This latter fact is true of the Householder-based methods as well. But in the case of the Arnoldi procedure, we will end up making use of partial Hessenberg reduction to turn Arnoldi into a method of approximating linear system solutions (GMRES) and a method of approximating eigenpairs for large systems (the Arnoldi method).

When A is symmetric, the Arnoldi procedure becomes the *Lanczos* procedure. Note that in this case, we *tridiagonalize* the original matrix, which means at each step we need (in exact arithmetic) to orthogonalize each successive Aq_j against only two other vectors. This rather remarkable fact allows us to parley the Lanczos iteration into an eigenvalue iteration as well as two truly remarkable iterations for solving linear systems: MINRES and the famous method of conjugate gradients (CG). We will return to these methods later.