**Numerical Recipes in Fortran 77** PhL 12.15.04

I have this book in a large set of PDF files, downloaded from Cornell. Have printed a few sections.

**11.1 The Jacobi Transformation of a Symmetric Matrix ( to full diagonal form) [ jacobi ]**

I read about this in Scheid, and here are the details I misunderstood the first time. The general planar rotation is shown on page 456, and those affected cells are *only on the four lines* shown in 11.1.3. His picture was at first a little unclear on this because it shows some dots that are *not* on the four lines. The picture would be clearer without these extra dots.

Now look at the equations. As I now know, elements that are *only one line* are mixed with their partners in the usual simple fashion. Points on *both* lines get the more complex mix shown because you are hit from both sides at once. In the Given's method, we use the simple one-line equations to kill off elements, but here he is going to kill off a pair of two-line elements. He wants to set 11.1.7 = 0 to do this, and we are going to kill off $a_{pq}$ . Setting 11.1.7 = 0 gives the rightmost equality in 11.1.8. The combination of sines and cosines is in fact cot(2ϕ) but this fact is not used. Instead, regard θ as given by the extreme right side of this equation. Divide the trig ratio by $\cos^2$ to get $(1-t^2)/2t$ where t = tan(ϕ). Then having a number for θ, you get the quadratic 11.1.9 for tan and you choose the smaller root because it is supposed to give an angle less than 45 degrees. Equation 11.1.10 is exactly right, giving the smaller root whichever sign that θ has.

Once we have found our value of t, we use 11.1.11 and 12 to get c and s. Then we need to get τ as in 11.18, and then we use t, τ and s to compute results in 11.1.14-17. We are here computing the results of doing a rotation!

The main claim that I did not prove to myself is 11.1.20 which says that each time you clear out an element, you reduce the sum of the squares of the off-diagonals by exactly those elements you removed. This is not obvious since your rotation has affected many off-diagonal elements. However, look at the picture 11.1.3 again. We have killed $a_{pq}$ and its partner, and the central diagonal elements have been altered. However, all off-diagonal elements have been altered in the simple sense of equations 11.1.4, ie, a simple mix. Each pair so mixed maintain their sum of squares! That is why the affected off-diagonals maintain sum of squares, except you have now removed two of them! Lest the point be lost, whatever p and q you choose, the central square is on the diagonal! So now I have proven equation 11.1.20. This is then a proof that the method converges!! Now you see why Jacobi said to pick the largest $a_{pq}$ in each rotation, because that is the most effective in hammering down the off diagonal sum! Using $P' = R^{-1}PR$, it is trivial to show that the total sum of squares remains constant under any similarity, so we must be increasing the diagonal sum by the same amount we knock down the off diagonal sum.

So there you have it! Now what about eigenvectors ? We are building up the matrix of eigenvectors as shown in 11.1.22, and we can do this iteratively with each rotation as in 11.1.23. Of course when we do this simple right-applied rotation, we only rotate the two rows of the vector as shown, so the cost here is just a few math operations. If you do this extra math, your result has the eigenvectors as well as the eigenvalues!

Let me now estimate the calculation cost per rotation:

compute $\theta^2$ then +1 then sqrt then add then invert, perhaps 40 ops, mostly the square root
compute s and c (another 31 ops say) and then τ (2 more ops).

So perhaps we have a "fixed cost" of 100 ops to get these parameters. Then

11.1.14 4 ops for each non-central pair, there are (N-2)*2 of these.
4 ops for the altered diagonals

So maybe our cost is now 104 + 2(N-2)*4 for doing one rotation, write this as roughly 100 + 8N. The cost of doing the V' is only an extra 6 ops per column vector, or 6N ops.

Now, how many rotations in a sweep? (N-1) + (N-2) + (N-3) + ... (N - N-1) = N(N-1)/2 I think. Then how many sweeps? Perhaps only 6-10.

So my evaluation is this:

each rotation costs about ~8N (14 N if include eigenvector calc)
number of rotations per sweep ~ $N^2/2$
number of sweeps 6-10

So total ops is then: (6-10) * $N^2/2$ * [ 8N or 14 N] = (6-10) * (4 or 7) * $N^3$ ~ 50 $N^3$

Now they refer to multiply and add as one operation, which in a MAC is what happens. The extra cost of the eigenvalues according to me is 7/4 = more than the 50% they claim, but in the ball park.

The main point is that computing the eigenvalues of a N x N symmetric matrix is an order $N^3$ operation.

**11.2 Reduction to Tridiagonal form: Given's and Householder reductions. [ tred2 ]**

The main point here is that if you are willing to stop at tridiagonal, only a finite number of rotations need be applied, again, as Scheid pointed out in his Givens' Method discussion.

The Givens method [ **Wallace Givens 1954** Oak Ridge] is discussed, in particular with regard to the exact order of the rotations and why it works, which I have now fixed up in my Scheid notes because I had it wrong! The claim is that Householder is twice as fast, both now there is a "fast Givens" that competes. In Givens, it seems to me that you have to do about $N^2/2$ little 2x2 rotations to get the job done. Note that Givens only works if matrix A is real symmetric, otherwise you don't get pairwise clearing.

Householder [ **Alston Householder 1958**] gets most of this book's attention. The Householder H is here called P and all the math agrees with our friend Jerry Schultz. The new idea to me is that you do it at the same time from both sides! You do A' = PA tuning your P to the first column to clear it out. However, instead of applying it to the entire first columns, as in Jerry, we do diag(1,P) to make it act only on the lower N-1 elements of the first column, putting k at the top. Now since the matrix is *symmetric*, if you apply the same P from the right, the top row has the same numbers, so the same dig(1,P) acting from the right clears out the first row as well to the right of the second element. The result is shown in 11.2.8. You keep doing this and you get your symmetric tridiagonal form in a total of N-1 such operations, compare to Given's $N^2/2$. But the operations here are more complex, taking square roots of sums of squares.

What does the vector u look like as you progress? The lower part has the elements of the column you are about to clear out. Above that is the β value which is the square root of all below. Then in our scheme here, above that you have all zeros. So u = (0,0,0....0,β, X,X,X...).

Now the only odd thing is they have decided to start at the bottom corner instead of the top corner, so then u looks like 11.2.14 and the order of things is then reversed. We are now cleaning out everything above and to the left, instead of below and to the right.

Rough operation count? Consider doing just the first Householder operation. For first column have to do N-2 squarings, then a square root, then various other things. Getting H requires some overlapping operations (squares). Then p requires N, then K requires N, then q requires 2N, then A' requires $4N^2$ , so call it overall 4 $N^2$. In the next one, N-2 becomes N-3, and so on, so maybe about N/2 required, so overall maybe $2N^3$ is my guess. On page 467 the claim is 1/3rd of my guess, so fine. Another factor of 2 if you want to accumulate the product matrix Q which will be needed later when the tri-diag is taken the rest of the way.

**Comments**: In the previous section we learned that the Jacobi method can take you all the way o eigenvalues and eigenvectors at a cost of about 24 $N^3$ . In this section, we can get to triadiagonal at cost of about only 4/3*$N^3$ .

Now how are we going to finish the job ???

### 11.3 Grinding down the tridiagonal for eigenvalues and eigenvectors. [ tqli ]

The QR Transformation is defined as follows.

(1) Decompose A = QR in the usual way.

(2) Define A' = RQ which then implies that $A' = Q^{-1}AQ = Q^TAQ$.

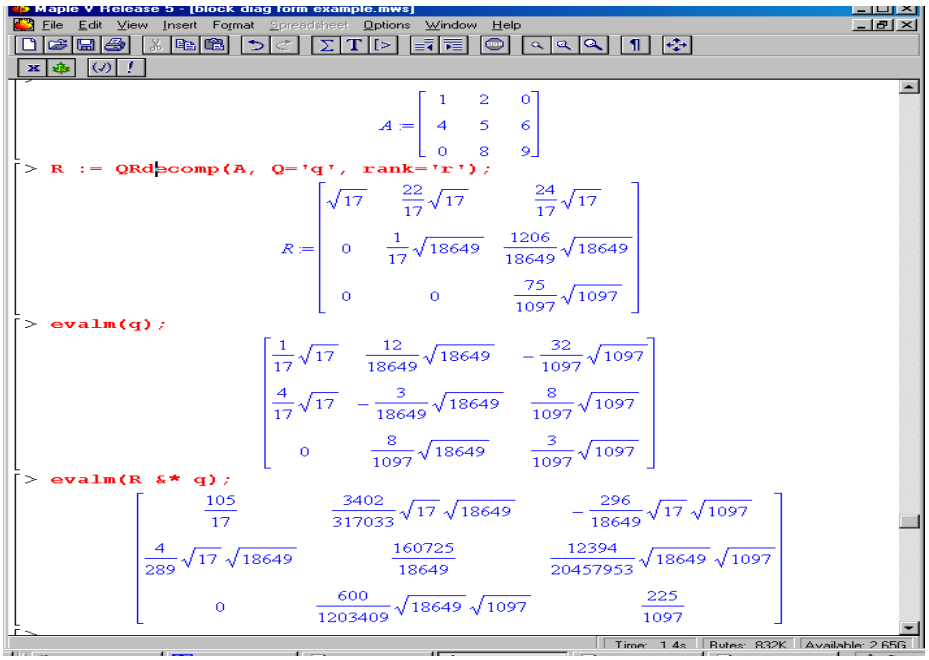
This similarity $A' = Q^{-1}AQ$ is the "QR Transformation".

On page 470 some claims are made, but one is not true as stated:

(1) The QR transformation preserves **symmetry**. Yes, this is true for any congruence.

(2) The QR transformation preserves **Hessenberg**, which is my $UH_1$ form. This is true, and I have proven it to be so in my matrix research doc.

(3) They claim that the QR transformation preserves **tridiagonal** form. This is **not** generally true, and here is an example from Maple.




I started with a tridiagonal A which has det = -75. I do the QR decomposition and we see that R is UT and Q is orthogonal. We then compute RQ and it is UH as predicted, but it is not tridiagonal as claimed!

(4) If A is **tridiagonal AND symmetric**, then A' is tridiagonal and symmetric. We know this because, without symmetry, we know that A' is UH. But if A' is UH and symmetric, then it is tridiagonal.

One other note: I am grateful to Augustin Dubrulle for alerting me to the **1941** dissertation of **Karl Hessenberg** on the unsymmetric eigenvalue problem; in it, he introduced a sequential way to build up an **almost triangular matrix** that is similar to a given full matrix. "Almost" here means that the next-to-diagonal entries will not, in general, be zero. Today, we call this type of matrix the Hessenberg form and refer to the procedure as a **Krylov subspace** method. Hessenberg was German and worked as an engineer during World War II.

Here is the plan. [ Assuming that **A is real symmetric**; non-symmetric is treated in a later section. ]

(1) Start with your original tridiagonal matrix $A_1$ and do whatever you have to do to get it into the factorization $A_1 = Q_1 L_1$ . [ No real difference between QR and QL ]

One way I know to do that is that $Q_1$ is just a sequence of Householders that clear out columns starting at the right side and upside down, instead of the usual left side. We are then building zeros in the top, and we get L = lower triangular. So, yes, we can find $Q_1$ to do this by a set of N-1 Householders, then our result is $L_1$.

On page 471 bottom, however, they say that for something that is already tridiagonal, it is faster to burn off the superdiagonal by doing a set of N rotations $P_{12}$, $P_{23}$ and so on. You end up with a non-symmetric matrix which is of lower diagonal form and which has a diagonal and a subdiagonal only. Since this is in the form L, you have achieved your desired A = QL by doing these N-1 rotations.

(2) Now that you have figured out $A_1 = Q_1 L_1$, create $A_2 = L_1 Q_1 = Q_1^{-1} A_1 Q_1$. Then you repeat the above process to factorize $A_2 = Q_2 L_2$. Then get $A_3 = L_2 Q_2 = Q_2^{-1} A_2 Q_2$ . And so on. So that is the iteration.

Here is the amazing claim of what happens if you keep doing this, and this **theorem** assumes for the moment a completely general A, not necessarily symmetric. [ Whose theorem is this? ]

(a) If all $\lambda_i$ are different, then $A_s \rightarrow A_L$ which is lower triangular, and the diagonal elements are the eigenvalues in fact already sorted by magnitude! This sounds a little like the Pivot and Twist in M&M.

(b) If $\lambda$ repeats p times, then you will have a block that is pxp that sticks out through the diagonal up into the upper right triangle which is normally all 0. The eigenvalues of the pxp matrix are all $\lambda$, but I guess it turns out that this submatrix does not naturally come out diagonalized when you do the algorithm.

Now in the case of a symmetric tridiagonal starting point for A, in case (a) we must be getting a purely diagonal matrix if lower triangular and symmetric. And in case (b) I guess we still have that pxp block sitting there, except we know that it is a symmetric pxp block.

What is the cost of each iteration here? It is ~ N rotations as noted above, so order N per iteration.

**Comments**: OK, I am not going to go look up the proof of this theorem, but it is standard stuff. I'm sure it is the same general idea as doing the Jacobi iteration, there is monotonic convergence. What I don't see is how you know you are done! I guess you just have to see if the diagonal elements are no longer changing. There is a convergence issue that is improved by doing **shifting** in each iteration step, and they give a way to pick a good shift value. If you do this, then convergence is cubic! This is why everyone likes this algorithm. One final improvement is on page 472 called **implicit shifts**, and this reduces numerical error when you have mixed large and small numbers. I did not bother to read these details.

The bottom line is on page 473. I you only want the eigenvalues, cost is 30 $N^2$. So do the Givens' type first stage to get tridiagonal in $1*N^3$ , then do this baby to finish it off in $N^2$ time. HOWEVER, if you want the eigenvectors, you are back up to $N^3$ in this second stage.

### 11.4 Hermitian Matrices.

In this one-page section, we are told that all the previous work can be generalized from real to complex and nothing dramatically new happens, though of course things are slower. This book is not providing code for these routines, however. { Remember, we are here talking about finding the eigenvalues and eigenvectors of a complex Hermitian matrix, instead of a real symmetric matrix. }

Another approach, as was mentioned in Scheid, is to replace the Hermitian matrix problem with a 2n x 2n equivalent real problem which has a symmetric matrix, then just apply what we already have. This would seem to imply a factor of 8 extra speed cost if we are doing now $(2N)^3 = 8^3$.

### 11.5 Reduction of a General Matrix to Hessenberg Form [ balanc, elmhes ]

First, we are reminded that in the world of non-symmetric matrices, there are many new pitfalls. One is that you might not have N distinct eigenvectors, as I will know, because some manifold will have m<k and all that stuff. Such a matrix is called **defective** in this writeup. Also, it is now much more possible to have rounding error problems.

The overall problem is so messy that even in this 1000 page expert's book, they are not going to try to present a scheme for getting all the eigenvectors! But we will do the eigenvalues!

Before proceeding, they recommend that you pre-process your matrix with a balancing operation. What you do here is apply binary similarities to make the $r^{th}$ row and the $r^{th}$ column have the same norm, which is sum of squares, or at least get them close. In a symmetric matrix, these norms are exactly the same! You do this for r = 1 to N, and this is an $N^2$ operation. The claim is simply that this can give a huge reduction in numerical error, but no theory is presented for why this is so. They give the **balanc** routine on page 477 to do this. This reduces the sensitivity of the eigenvalues to rounding errors.

Now, *why* are we going to first reduce our general matrix to Hessenberg form? Recall that in the symmetric case we applied Householder operations to both sides of a matrix to get it down to tridiagonal form. We first did an $H_1$ on the left to clear out the left column. We found that using the that same $H_1$ on the right then cleared out the first column (both clearings are after the diagonal element plus 1, as on page 464-5 ] This is WHY we got the tridiagonal form. Now if the matrix is not symmetric, you clear out the columns exactly before, but now when you apply the $H_i$ to the right side, you do NOT clear the rows. This then is why the best you can do is end up with Hessenberg form. This is what Karl found in 1941!

They then go on to show that, in order to get into Hessenberg form, it is a little better to do the "similarized Gauss elimination" than to do Householder. This method is shown on page 478, fine.

So there is no problem doing this first stage of getting down to Hessenberg form. We just clear out the columns using either method: Householder or similarized Gauss. The code for this second method is given on page 478.

**11.6 The QR Algorithm for Real Hessenberg Matrices [ hqr ]**

Well, we just do the same idea we did in the symmetric case. We start off with 11.6.1. Now in this discussion we have given up on A = QL and we are back to A = QR, but we rename Q to be $Q^T$ and we get right to it by adding the shift $k_s$. So 11.6.2 is our algorithm, and by now I have proven that this algorithm preserves Hessenberg form without shift. When you add in the shift, the conclusion is the same because you are just adding a multiple of E.

So we can just take this to be a repeat of what we did for the tridiagonal grinding down. However, a trick is used here to improve convergence, and I have decided to ignore the rest of this section which talks about a plan based on that trick. The trick is then incorporated into routine hqr on page 484, which sort of means Hessenberg QR algorithm. Earlier on page 473 we had code tqli which meant Tridiagonal QL algorithm with Implicit shifts.

So to summarize: you use a finite set of operations to take A down to Hessenberg form, then you use this modified QR method to get the eigenvalues. As noted, we make no attempt in this book to get the eigenvectors for non-symmetric A.

What Maple does. If you linalg[eigenvalues], and you have a numerical matrix, it does some unspecified numerical method to get those eigenvalues. If you have symbolics, then it does the characteristic poly. However, there is a different call Eigenvals intended only for a numerical matrix. Here is what Maple says: "Eigenvals(A) returns an array of the eigenvalues of A. The eigenvalues are computed by the QR method. The matrix is first balanced and transformed into upper Hessenberg form. Then the eigenvalues (eigenvectors) are computed. " Note that this is the only occurrence of the word Hessenberg in all of Maple under a full text search.

**Comments:**

**Question**: Why don't we just apply "full Householder" directly to general matrix A and make it be triangular? Why do we have to accept Hessenberg or triadiagonal.

**Answer**: Look back at pages 464 and 465. As shown, we don't start with a full N Householder, but we start instead with an N-1 size Householders. Here is the reason. When we apply this N-1 House on the left, we leave the $a_{11}$ untouched, and we then add the column k,0,0... as shown. The left application of $P_1$ gives us a submatrix that is N-1 square and is called "irrelevant". When we now apply $P_1$ from the right, in the one case that A is symmetric, it clears out the rest of the first row. More generally, it does NOT clear out this row. BUT, what about the action of the right-applied $P_1$ on the rest of the rows! All damage done is inside the irrelevant area because we are not doing a full sized Householders. Thus, the first column survives. And as we continue on, all subsequent columns survive. In this way, we either get Hessenberg or tridiagonal for symmetric. That is the main point!

Look what happens if you try to apply a full size Householder from the left,

H A =

| x | |
|---|---|
| 0 | |
| 0 | |
| 0 | |
| 0 | |

That is fine. This House has done weird things to all the columns to the right of the one of interest, but we accept that so far. Now what happens if we apply this same House from the right to make our similarity? Here show an intermediate result where we have processed only the first row,

H A H =

first row only

| * | * * * * |
|---|---|
| 0 | |
| 0 | |
| 0 | |
| 0 | |

If A is non-symmetric, we get unknowns as shown. If A were symmetric, then we know HAH is symmetric, so we could then make this claim

H A H =

first row only,
A was sym

| * | 0 0 0 0 |
|---|---|
| 0 | |
| 0 | |
| 0 | |
| 0 | |

BUT, now let this H acting from the right continue to act on the other rows! Since the scope of this full size H is the full NxN, it is going to mess up the first column we cleared out, and we get this:

H A H =

all rows,
A was sym

| | | | | |
|---|---|---|---|---|
| * | 0 | 0 | 0 | 0 |
| * | | | | |
| * | | | | |
| * | | | | |
| * | | | | |

So the fact is that, with a full size H, applying from the right wrecks what you did on the left. In the A not sym case, we don't have *any* zeros anywhere after doing this!

In contrast, when we start with N-1 house, then we are always one smaller than the square we are working on, and we always avoid damaging the clearing out we have already done. So that is why the best you can do with a finite number of Householders is Hessenberg.

Now it is true that if you don't CARE about getting a similarity, you can apply a full house sequence from the left and get to upper triangular form. This is the A = QR idea. But this is not a similarity, so we don't preserve eigenvalues, etc.