ALGORITHM 85
JACOBI

THOMAS G. EVANS
Bolt, Beranek, and Newman*, Cambridge, Mass.

* This work has been sponsored by the Air Force Cambridge Research Laboratories, OAR (USAF), Detection Physics Laboratory, under contract AF 19(628)-227.

```
procedure JACOBI (A, S, n, rho);
value n, rho; integer n; real rho; real array A, S;
comment This procedure finds all eigenvalues and eigenvectors
  of a given square symmetric matrix by a modified Jacobi (itera-
  tive) method (cf. J. Greenstadt, "The determination of the charac-
  teristic roots of a matrix by the Jacobi method," in Mathematical
  Methods for Digital Computers, A. Ralston and H. S. Wilf, eds.).
  JACOBI is given a square symmetric matrix of order n stored in
  the array A. The initial contents of the array S are immaterial,
  as S is initialized by the procedure. At exit the kth column of the
  array S contains the kth of the n eigenvectors of the given matrix,
  and the diagonal element A[k, k] of the array A is the corre-
  sponing kth eigenvalue. The parameter rho is the "accuracy
  requirement" introduced in the above reference, where a de-
  tailed flow chart of the method is given. The significance of rho is
  that the iteration terminates when, for every off-diagonal ele-
  ment A[i, j], abs (A[i, j]) < (rho/n) × norm1, where norm1 is a
  function only of the off-diagonal elements of the original matrix;
begin real norm1, norm2, thr, mu, omega, sint, cost, int1, v1,
  v2, v3;
  integer i, j, p, q, ind;
  comment Set array S = n × n identity matrix;
  for i := 1 step 1 until n do
  for j := 1 step 1 until i do
     if i = j then S[i, j] := 1.0
     else S[i, j] := S[j, i] := 0.0;
  comment Calculate initial norm (norm1), final norm (norm2),
  and threshold (thr);
  int1 := 0.0;
  for i :=2 step 1 until n do
     for j :=1 step 1 until i-1 do
        int1 := int1 + 2.0 × A[i, j] ↑ 2;
  norm1 := sqrt (int1); norm2 := (rho/n) × norm1;
  thr := norm1; ind := 0;
main: thr := thr/n;
  comment The sweep through the off-diagonal elements be-
    gins here;
main1: for q := 2 step 1 until n do
         for p := 1 step 1 until q-1 do
           if abs (A[p, q]) ≧ thr then
           begin ind := 1; v1 := A[p, p]; v2 := A[p, q];
             v3 := A[q, q]; mu := 0.5 × (v1-v3);
             omega := (if mu = 0.0 then 1 else sign (mu)) ×
               (-v2)/sqrt(v2↑2 + mu↑2);
             sint := omega/sqrt(2.0 × (1.0 + sqrt(1.0 -
               omega↑2)));
             cost := sqrt (1.0 - sint↑2);
             for i := 1 step 1 until n do
             begin int1 := A[i, p] × cost - A[i, q] × sint;
                 A[i, q] := A[i, p] × sint + A[i, q] × cost;
                 A[i, p] := int1;
                 int1 := S[i, p] × cost - S[i, q] × sint;
                 S[i, q] := S[i, p] × sint + S[i, q] × cost;
                 S[i, p] := int1
             end;
             for i :=1 step 1 until n do
               begin A[p, i] := A[i, p]; A[q, i] := A[i q] end;
             A[p, p] := v1 × cost↑2 + v3 × sint↑2 - 2.0 ×
               v2 × sint × cost;
             A[q, q] := v1 × sint↑2 + v3 × cost↑2 + 2.0 ×
               v2 × sint × cost;
             A[p, q] := A[q, p] := (v1 - v3) × sint × cost +
               v2 × (cost↑2 - sint↑2)
           end;
  comment Now test to see if current tolerance exceeded and,
    if not, whether final tolerance reached;
  if ind = 1 then begin ind := 0; go to main1 end
  else if thr > norm2 then go to main
end JACOBI
```

CERTIFICATION OF ALGORITHM 85
JACOBI [T. G. Evans, *Comm. ACM*, Apr. 1962]

J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts., England

The statement

omega := (**if** *mu* = 0.0 **then** 1 **else** *sign* (*mu*)) × (−*V*2)/*sqrt*(*V*2 ↑ 2+*mu* ↑ 2);

was changed to

omega := **if** *mu* = 0.0 **then** −1.0 **else** − *sign* (*mu*) × *V*2/*sqrt* (*V*2 ↑ 2+*mu* ↑ 2);

When *mu* = 0, the original statement reduces to

omega := −*V*2/*sqrt* (*V*2 ↑ 2);

and a truncation error in the evaluation of the square root can make the magnitude of omega slightly greater than unity. As a result, an error stop occurs during execution of the next statement when an attempt is made to evaluate *sqrt* (1 − *omega* ↑ 2).

In its modified form the algorithm has been successfully run using the Elliott ALGOL translator on the National-Elliott 803. Matrices of order up to fifteen have been solved, yielding eigenvalues and eigenvectors with an overall accuracy of seven decimal places.

CERTIFICATION OF ALGORITHM 85
JACOBI [Thomas G. Evans, *Comm. ACM* (Apr. 1962), 208]

P. NAUR
Regnecentralen, Copenhagen, Denmark

We have first run this algorithm in the GIER ALGOL system with the following corrections included:

1. The change given by J. S. Hillmore [*Comm. ACM 5* (Aug. 1962), 440] with capital *V* changed to *v*.

2. The 4th **for** clause corrected to read:

for j := 1 **step** 1 **until** $i - 1$ **do**

3. The last **for** clause corrected to read:

for i := 1 **step** 1 **until** n **do**

On closer examination we have found, however, that a significant number of superfluous operations could be eliminated in the innermost loop by rewriting the two **for** statements at the center of the algorithm as a single **for** statement, to read as follows:

```
.....
cost := sqrt (1−sint ↑ 2);
for i := 1 step 1 until n do
  begin if i ≠ p ∧ i ≠ q then
    begin intl := A[i,p];  mu := A[i,q];
    A[q,i] := A[i,q] := intl × sint + mu × cost;
    A[p,i] := A[i,p] := intl × cost − mu + sint
    end;
  intl := S[i,p];  mu := S[i,q];
  S[i,q] := intl × sint + mu × cost;
  S[i,p] := intl × cost − mu × sint
  end;
A[p,p] := v1 × cost ↑ 2 + v3 × sint ↑ 2 − 2 × v2 × sint × cost;
.....
```

This revision is particularly advantageous in systems having a comparatively slow subscript mechanism, such as GIER ALGOL, because it eliminates more than 3 out of 8 references to subscripted variables.

JACOBI has been tried with two different sets of matrices having known eigenvalues. In both cases a test program was set up to find the range of errors of the eigenvalues computed by JACOBI. In addition, the relations $Av - \lambda v = 0$ (A is the given matrix, v an eigenvector, and λ the corresponding eigenvalue) and $A - (ST)$ LAMBDA $S = 0$ (S is the matrix having the eigenvectors as columns and ST its transpose, and LAMBDA is the diagonal matrix of the eigenvalues) were used as checks. The test matrices were TESTMATRIX calculated by the revised algorithm 52 given in *Comm. ACM 6* (Jan. 1963), 39, and the following matrix suggested by Mr. H. B. Hansen:

$$\text{HBH TESTMATRIX } [j,i] = \text{HBH TESTMATRIX } [i,j] = n + 1 - j \qquad j \geqq i$$

having the eigenvalues $0.5/(1 - \cos((2\times i-1)\times pi/(2\times n+1)))$.

The results were as shown in Table 1 (GIER ALGOL works with floating numbers of 29 significant bits).

The compile time for the program which produced one of these tables was about 40 seconds. Run times were as follows:

Rho	*n*	*Original algorithm* TESTMATRIX ALG. 52 (seconds)	*Original algorithm* HBH TESTMATRIX (seconds)	*Revised algorithm* HBH TESTMATRIX (seconds)
$_{10}-3$	5			3
	10			22
	15			70
$_{10}-5$	5	3		5
	10	5	41	29
	15	13	148	99
$_{10}-8$	5	4	7	6
	6	5	12	
	7	5	18	
	8	5	25	
	10	13		38
	15	22		116

From these figures it looks as if TESTMATRIX, Algorithm 52, is atypical as far as solution by means of JACOBI is concerned. The much higher accuracy obtained for this matrix as compared with the HBH matrix points in the same direction.

For further comparison it may be mentioned that the algorithms published by J. H. Wilkinson [*Num. Math. 4* (1962), 354–376] also have been tested successfully with GIER ALGOL. Wilkinson's algorithms reduce the matrix to tridiagonal form by means of Householder's method and use Sturm sequences to find the eigenvalues and inverse iteration to find the eigenvectors. In GIER ALGOL this method is about 1.3 times as fast as JACOBI for the range of matrices considered here. JACOBI has the advantage that the eigenvectors are properly orthogonal, even in the case of multiple eigenvalues, and also has a much simpler logic. On the other hand if only some of the eigenvalues and/or eigenvectors are sought Wilkinson's algorithms will often offer much higher speed than JACOBI, which always finds them all.

TABLE 1

HBH TESTMATRIX

	Range of true errors of eigenvalues				Range of deviations from relation Av − lambda $v = 0$						Range of deviations from relation $A - (ST)$ LAMBDA $S = 0$					
Order	*j*	*error*[*j*]	*j*	*error*[*j*]	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*
							rho = $1.0_{10}-3$									
5	1	$-1.1_{10}-6$	3	$5.2_{10}-8$	1	1	$-1.7_{10}-4$	1	3	$2.0_{10}-4$	1	1	$-2.5_{10}-4$	5	5	$1.0_{10}-4$
10	9	$-7.9_{10}-5$	8	$3.5_{10}-5$	7	2	$-3.3_{10}-3$	6	6	$3.0_{10}-3$	1	1	$-4.2_{10}-3$	6	7	$3.2_{10}-3$
15	15	$-9.2_{10}-5$	12	$3.7_{10}-5$	6	3	$-1.7_{10}-3$	11	13	$1.7_{10}-3$	9	15	$-1.5_{10}-3$	8	9	$1.8_{10}-3$
							rho = $1.0_{10}-5$									
5	1	$-1.1_{10}-6$	3	$6.0_{10}-8$	2	5	$-1.3_{10}-7$	5	2	$4.1_{10}-8$	1	2	$-1.6_{10}-7$	4	5	$4.5_{10}-8$
10	1	$-1.2_{10}-5$	2	$2.2_{10}-7$	7	3	$-2.7_{10}-5$	2	8	$2.2_{10}-5$	7	7	$-2.4_{10}-5$	2	8	$2.3_{10}-5$
15	1	$-3.5_{10}-5$	4	$3.9_{10}-7$	11	9	$-6.4_{10}-6$	7	2	$4.8_{10}-6$	11	12	$-5.3_{10}-6$	12	12	$4.7_{10}-6$
							rho = $1.0_{10}-8$									
5	1	$-1.1_{10}-6$	3	$6.0_{10}-8$	2	5	$-1.3_{10}-7$	4	2	$6.5_{10}-9$	2	2	$-1.3_{10}-7$	4	4	$3.0_{10}-8$
10	1	$-1.2_{10}-5$	2	$2.2_{10}-7$	1	10	$-1.1_{10}-6$	4	2	$6.4_{10}-8$	1	2	$-5.7_{10}-7$	9	9	$8.2_{10}-8$
15	1	$-3.5_{10}-5$	4	$3.9_{10}-7$	1	14	$-3.4_{10}-6$	4	2	$3.9_{10}-7$	2	2	$-1.3_{10}-6$	15	15	$8.9_{10}-8$

TESTMATRIX, Algorithm 52

	Range of true errors of eigenvalues				Range of deviations from relation Av − lambda $v = 0$						Range of deviations from relation $A - (ST)$ LAMBDA $S = 0$					
Order	*j*	*error*[*j*]	*j*	*error*[*j*]	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*	*Element*	*Vector*	*Error*
							rho = $1.0_{10}-5$									
5	4	$-1.0_{10}-8$	1	.0	5	5	$-3.3_{10}-8$	5	4	$4.3_{10}-8$	5	5	$-5.1_{10}-8$	4	4	$3.9_{10}-8$
10	8	$-1.1_{10}-8$	4	.0	7	7	$-1.2_{10}-8$	9	6	$1.3_{10}-8$	7	8	$-5.1_{10}-9$	6	6	$2.0_{10}-8$
15	13	$-1.1_{10}-8$	6	.0	14	14	$-9.3_{10}-9$	10	10	$9.4_{10}-9$	8	9	$-1.9_{10}-9$	10	10	$1.3_{10}-8$
							rho = $1.0_{10}-8$									
3	3	$-7.5_{10}-9$	1	$3.7_{10}-9$	3	1	$-2.8_{10}-9$	2	2	$9.3_{10}-9$	1	3	.0	1	2	$1.9_{10}-8$
4	4	$-5.6_{10}-9$	3	.0	2	2	$-4.5_{10}-9$	3	4	$3.3_{10}-9$	2	2	.0	2	3	$9.3_{10}-9$
5	4	$-1.0_{10}-8$	1	.0	5	4	$-4.9_{10}-9$	4	4	$5.8_{10}-9$	1	1	$-7.5_{10}-9$	3	4	$7.5_{10}-9$
6	4	$-4.7_{10}-9$	4	.0	4	3	$-2.8_{10}-9$	5	4	$3.6_{10}-9$	1	6	$-2.3_{10}-10$	4	5	$9.3_{10}-9$
7	4	$-5.1_{10}-9$	5	.0	6	6	$-2.8_{10}-9$	4	4	$3.4_{10}-9$	5	7	$-1.2_{10}-10$	5	6	$7.5_{10}-9$
8	7	$-7.5_{10}-9$	5	.0	5	5	$-6.0_{10}-9$	5	6	$3.2_{10}-9$	8	8	$-1.2_{10}-10$	7	7	$9.3_{10}-9$
9	6	$-4.4_{10}-9$	7	.0	6	5	$-5.1_{10}-9$	7	6	$3.2_{10}-9$	5	5	$-7.5_{10}-9$	8	8	$1.5_{10}-8$
10	8	$-1.5_{10}-8$	8	.0	8	9	$-9.3_{10}-9$	9	7	$7.2_{10}-9$	6	7	$-2.3_{10}-9$	9	9	$2.0_{10}-8$
11	10	$-7.5_{10}-9$	1	.0	9	10	$-6.5_{10}-9$	8	11	$3.0_{10}-9$	1	1	$-3.1_{10}-9$	8	8	$7.5_{10}-9$
12	8	$-5.0_{10}-9$	11	.0	10	6	$-7.6_{10}-9$	10	8	$2.4_{10}-9$	6	6	$-1.7_{10}-8$	4	4	$1.3_{10}-8$
13	12	$-1.1_{10}-8$	10	.0	10	11	$-6.9_{10}-9$	12	10	$9.1_{10}-9$	7	7	$-3.0_{10}-8$	12	12	$3.2_{10}-8$
14	10	$-1.5_{10}-8$	4	.0	13	13	$-1.1_{10}-8$	10	10	$6.7_{10}-9$	9	10	$-3.5_{10}-9$	6	6	$1.7_{10}-8$
15	13	$-1.1_{10}-8$	6	.0	14	14	$-1.1_{10}-8$	11	10	$3.5_{10}-9$	8	9	$-3.0_{10}-9$	6	11	$7.5_{10}-9$