```
{ local
'A def
A(0)(0) 'a def
A(0)(1) 'b def
A(1)(1) 'c def
b 0 eq {                            # b = 0?
[[1 0][0 1]] [a c]                   # if so, do essentially nothing
} {
c a - b 2 * / 'z def                 # z = (c-a)/2b
z 0 gt {-1}{1} ifelse 's def         # s = -sign(z)
s z abs 1 z dup * + sqrt + / 'y def  # y = -sign/(|z| + sqrt(1 + z^2))
1 1 y dup * + sqrt / 'p def          # p = 1/sqrt(1 + y^2)
p y * 'q def
[[p q neg][q p]]                     # V
[a b y * +
c b y * -]                           # [x1 x2]
} ifelse
endlocal }
'twoeigs def

# A shift

{
local
's def
'A def
A rowNo 'n def

A n identity s * -
inverse 'B def      # B = (A -sI)^{-1}

[1 n 1 - {0} repeat] 'v def          # v = [1 0 ...  0]

3 {
B v * dup length / 'v def              # v := Bv/|Bv|
v ==
} repeat

endlocal
} 'isi def

# isi stands for "shift, invert, iterate"

# ------------------------------------------------

# A i j

{ local
'j def
'i def
'A def                        # define A, i, j as the arguments
[[A(i)(i) A(i)(j)]
[A(j)(i) A(j)(j)]] 'B def     # set B = the correct 2 x 2 matrix
B twoeigs                     # find its eigenvalues etc.
'C def                        # save the eigenvalue pair
'V def                        # save the eigenvector matrix for B
C(0) 'x1 def
```

```
C(1) 'x2 def                    # save the eigenvalues
A rowNo 'n def                  # A = n x n matrix
n identity 'X def
V(0)(0) 'X(i)(i) def
V(0)(1) 'X(i)(j) def
V(1)(0) 'X(j)(i) def
V(1)(1) 'X(j)(j) def            # build X by modifying the n x n identity matrix

X transpose A * X *             # semi-diagonalize A

endlocal } 'jac def

{
local
'A def
A rowNo 'n def
0 's def
1 'i def
n 1 - {
0 'j def
i {
s A(i)(j) dup * + 's def
j 1 + 'j def
} repeat
i 1 + 'i def
} repeat
s
endlocal
} 'sumOfSquares def

# ------------------------------------------------

4 prec

[[2 1 3]
 [1 2 1]
 [3 1 3]] 'A def

A copy 'origA def

A sumOfSquares ==
A 0 2 jac 'A def
A sumOfSquares ==
A ==

A 1 2 jac 'A def
A sumOfSquares ==
A ==

A 0 1 jac 'A def
A sumOfSquares ==
A ==

A 0 2 jac 'A def
A sumOfSquares ==
A ==
```

```
A 1 2 jac 'A def
A sumOfSquares ==
A ==

"" ==
origA A(0)(0) isi
"" ==
origA A(1)(1) isi
"" ==
origA A(2)(2) isi
```

```
// input i, j; fills in x = [c s e0 e1]

            m[k][j] = y;

            m[i][k] = v1;


            m[j][k] = y;

            m[k][i] = v1;

        }

        for (int k=i+1;k<r;k++) {

            double v0 = m[k][j], v1 = m[k][i];

            double y;

            y = c*v0 + s*v1;

            v1 = -s*v0 + c*v1;

            m[k][j] = y;

            m[k][i] = v1;


            m[j][k] = y;

            m[i][k] = v1;

        }

    }


    // diagonalizes this; returns X


    public static double[][] jacobi(double m[][]) {

        double eps = 0.0000000000000001;


        int r = rowNo(m), c = columnNo(m);
        // initialize the eigenvector matrix
to the identity matrix
        double x[][] = new double[r][r];

        for (int i=0;i<r;i++) x[i][i] = 1.0;
```

```
// initialize storage:
double R[] = new double[2];
double v[] = new double[2];
double ell[] = new double[2];

double N = ((double) r*(r - 1))/2.0;
// choose initial threshold
double s = 0.0;
for (int i=0;i<r;i++) {
    for (int j=0;j<i;j++) {
        double y = m[i][j];
        s += y*y;
    }
}
double t = Math.sqrt(s/N);

while (t > eps) {
    System.out.println("t = " + t);
    for (int i=0;i<r;i++) {
        for (int j=0;j<i;j++) {
            double C = m[i][j];
            if (Math.abs(C) > t) {
                s -= C*C;
                t = Math.sqrt(s/N);
                double te[] = twoEigs(m, j, i);
                jacobiRotate(j, i, te[0], te[1]);
                m[i][j] = 0.0; m[j][i] = 0.0;
                m[j][j] = te[2]; m[i][i] = te[3];
```

```
                    for (int k=0;k<r;k++) {
                        double v0 = x[k][j];
                        double v1 = x[k][i];
                        double y = R[0]*v0 + R[1]*v1;
                        x[k][i] = -R[1]*v0 + R[0]*v1;
                        x[k][j] = y;
                    }
                }
            }
        }
        // recompute the threshold
        // double olds = s;
        s = 0.0;
        for (int i=0;i<r;i++) {
            for (int j=0;j<i;j++) {
                double y = m[i][j];
                s += y*y;
            }
        }
        // System.out.println("s, old s = " + s + ", " + olds);
        t = Math.sqrt(s/N);
    }
    return(new RealMatrix(x));
}
```