L4c Symmetric Eigenvalue Decomposition - Jacobi Method and High Relative Accuracy

Ivan Slapničar

April 9, 2018

1 Symmetric Eigenvalue Decomposition - Jacobi Method and High Relative Accuracy

The Jacobi method is the oldest method for EVD computations, dating back from 1864. The method does not require tridiagonalization. Instead, the method computes a sequence of orthogonally similar matrices which converge to a diagonal matrix of eigenvalues. In each step a simple plane rotation which sets one off-diagonal element to zero is performed.

For positive definite matrices, the method computes eigenvalues with high relative accuracy.

For more details, see Section [Sla14] and Section [Drm14] and the references therein.

References

[Sla14] I. Slapničar, Symmetric Matrix Eigenvalue Techniques, in: L. Hogben, ed., 'Handbook of Linear Algebra', pp. 55.1-55.25, CRC Press, Boca Raton, 2014.

[Drm14] Z. Drmač, Computing Eigenvalues and Singular Values to High Relative Accuracy, in: L. Hogben, ed., 'Handbook of Linear Algebra', pp. 59.1-59.21, CRC Press, Boca Raton, 2014.

1.1 Prerequisites

The reader should be familiar with concepts of eigenvalues and eigenvectors, related perturbation theory, and algorithms.

1.2 Competences

The reader should be able to recognise matrices which warrant high relative accuracy and to apply Jacobi method to them.

1.3 Jacobi method

A is a real symmetric matrix of order n and $A = U\Lambda U^T$ is its EVD.

1.3.1 Definitions

The **Jacobi method** forms a sequence of matrices,

$$A_0 = A, \qquad A_{k+1} = G(c, s, i_k, j_k) A_k G(c, s, i_k, j_k)^T, \qquad k = 1, 2, \ldots,$$

where $G(c, s, i_k, j_k)$ is the orthogonal **plane rotation matrix**. The parameters c and s are chosen such that

$$[A_{k+1}]_{i_k j_k} = [A_{k+1}]_{j_k i_k} = 0.$$

The plane rotation is also called **Jacobi rotation**.

The **off-norm** of A is

$$\|A\|_{\text{off}} = \Big(\sum_i \sum_{j \neq i} a_{ij}^2 \Big)^{1/2},$$

that is, off-norm is the Frobenius norm of the matrix consisting of all off-diagonal elements of A.

The choice of **pivot elements** $[A_k]_{i_k j_k}$ is called the **pivoting strategy**.

The **optimal pivoting strategy**, originally used by Jacobi, chooses pivoting elements such that

$$|[A_k]_{i_k j_k}| = \max_{i<j} |[A_k]_{ij}|.$$

The **row-cyclic** pivoting strategy chooses pivot elements in the systematic row-wise order,

$$(1,2), (1,3), \ldots, (1,n), (2,3), (2,4), \ldots, (2,n), (3,4), \ldots, (n-1,n).$$

Similarly, the column-cyclic strategy chooses pivot elements column-wise.

One pass through all matrix elements is called **cycle** or **sweep**.

1.3.2 Facts

1. The Jacobi rotations parameters c and s are computed as follows: If $[A_k]_{i_k j_k} = 0$, then $c = 1$ and $s = 0$, otherwise

$$\tau = \frac{[A_k]_{i_k i_k} - [A_k]_{j_k j_k}}{2[A_k]_{i_k j_k}}, \qquad t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1 + \tau^2}},$$
$$c = \frac{1}{\sqrt{1 + t^2}}, \qquad s = c \cdot t.$$

2. After each rotation, the off-norm decreases,

$$\|A_{k+1}\|_{\text{off}}^2 = \|A_k\|_{\text{off}}^2 - 2[A_k]_{i_k j_k}^2.$$

With the appropriate pivoting strategy, the method converges in the sense that

$$\|A_k\|_{\text{off}} \to 0, \qquad A_k \to \Lambda, \qquad \prod_{k=1}^{\infty} G(i_k, j_k, c, s)^T \to U.$$

3. For the optimal pivoting strategy the square of the pivot element is greater than the average squared element,

$$[A_k]_{i_k j_k}^2 \geq \frac{1}{n(n-1)} \|A_k\|_{\text{off}}^2.$$

Thus,

$$\|A_{k+1}\|_{\text{off}}^2 \leq \left(1 - \frac{2}{n(n-1)}\right) \|A_k\|_{\text{off}}^2$$

and the method converges.

4. For the row cyclic and the column cyclic pivoting strategies, the method converges. The convergence is ultimately **quadratic** in the sense that

$$\|A_{k+n(n-1)/2}\|_{\text{off}} \leq \ const \cdot \|A_k\|_{\text{off}}^2,$$

provided $\|A_k\|_{\text{off}}$ is sufficiently small.

5. The EVD computed by the Jacobi method satisfies the standard error bounds.

6. The Jacobi method is suitable for parallel computation. There exist convergent parallel strategies which enable simultaneous execution of several rotations.

7. The Jacobi method is simple, but it is slower than the methods based on tridiagonalization. It is conjectured that standard implementations require $O(n^3 \log n)$ operations. More precisely, each cycle clearly requires $O(n^3)$ operations and it is conjectured that $\log n$ cycles are needed until convergence.

8. If A is positive definite, the method can be modified such that it reaches the speed of the methods based on tridiagonalization and at the same time computes the EVD with high relative accuracy.

1.3.3 Examples

```
In [1]: function myJacobi{T}(A::Array{T})
            n=size(A,1)
            U=eye(T,n)
            # Tolerance for rotation
            tol=sqrt(n)*eps(T)
            # Counters
            p=n*(n-1)/2
            sweep=0
            pcurrent=0
            # First criterion is for standard accuracy, second one is for relative accuracy
            # while sweep<30 && vecnorm(A-diagm(diag(A)))>tol
            while sweep<30 && pcurrent<p
```

```
            sweep+=1
            # Row-cyclic strategy
            for i = 1 : n-1
                for j = i+1 : n
                    # Check for the tolerance - the first criterion is standard,
                    # the second one is for relative accuracy for PD matrices
                    # if A[i,j]!=zero(T)
                    if abs(A[i,j])>tol*sqrt(abs(A[i,i]*A[j,j]))
                        # Compute c and s
                        τ=(A[i,i]-A[j,j])/(2*A[i,j])
                        t=sign(τ)/(abs(τ)+sqrt(1+τ^2))
                        c=1/sqrt(1+t^2)
                        s=c*t
                        G=LinAlg.Givens(i,j,c,s)
                        A=G*A
                        A*=G'
                        A[i,j]=zero(T)
                        A[j,i]=zero(T)
                        U*=G'
                        pcurrent=0
                    else
                        pcurrent+=1
                    end
                end
            end
        end
        diag(A), U
    end
```

```
Out[1]: myJacobi (generic function with 1 method)
```

```
In [2]: n=4
        s=srand(421)
        A=full(Symmetric(rand(n,n)))
```

```
Out[2]: 4×4 Array{Float64,2}:
         0.345443  0.105135  0.785847  0.608612
         0.105135  0.77247   0.135538  0.346561
         0.785847  0.135538  0.958365  0.561248
         0.608612  0.346561  0.561248  0.915812
```

```
In [3]: λ,U=myJacobi(A)
```

```
Out[3]: ([-0.239024, 0.76775, 2.15297, 0.310394], [0.855237 -0.20096 0.476545 -0.0329968; 0.0566
```

```
In [4]: # Orthogonality
        U'*U
```

```
Out[4]: 4×4 Array{Float64,2}:
          1.0           6.93889e-17  -8.32667e-17  -5.55112e-17
          6.93889e-17   1.0          -1.38778e-17   5.55112e-17
         -8.32667e-17  -1.38778e-17   1.0           5.55112e-17
         -5.55112e-17   5.55112e-17   5.55112e-17   1.0
```

```
In [5]: # Residual
        A*U-U*diagm(λ)
```

```
Out[5]: 4×4 Array{Float64,2}:
          2.77556e-17  -3.05311e-16  -1.11022e-15  -2.42861e-17
          2.77556e-17   6.66134e-16  -7.77156e-16  -5.55112e-17
          8.32667e-17  -3.33067e-16  -1.55431e-15   2.77556e-17
         -1.17961e-16   8.32667e-17  -1.33227e-15   3.33067e-16
```

```
In [6]: # Positive definite matrix
        n=100
        A=rand(n,n)
        A=full(Symmetric(A'*A));
```

```
In [7]: λ,U=myJacobi(A)
        vecnorm(U'*U-I),vecnorm(A*U-U*diagm(λ))
```

```
Out[7]: (1.7934291007986255e-13, 2.796882256667019e-11)
```

```
In [8]: λ
```

```
Out[8]: 100-element Array{Float64,1}:
         2501.5
            0.166124
           31.7579
            7.50007e-6
           19.7288
            1.96576
            0.0600245
            0.0834034
           28.8219
            0.00403152
           30.5573
           27.315
           19.1515
            ⋮
            5.44234
            2.24968
            2.86735
            2.69984
```

```
3.2681
4.89029
3.72714
4.39552
4.6202
3.12447
3.98454
4.7994
```

```
In [9]: cond(A)
```

```
Out[9]: 3.335308268422186e8
```

```
In [10]: # Now the standard QR method
         λs,Us=eig(A);
```

```
In [11]: vecnorm(Us'*Us-I),vecnorm(A*Us-Us*diagm(λs))
```

```
Out[11]: (5.221244590058073e-14, 2.3983515770923427e-12)
```

1.4 Relative perturbation theory

A is a real symmetric PD matrix of order n and $A = U\Lambda U^T$ is its EVD.

1.4.1 Definition

The **scaled matrix** of the matrix A is the matrix

$$A_S = D^{-1}AD^{-1}, \quad D = \text{diag}(\sqrt{A_{11}}, \sqrt{A_{22}}, \ldots, \sqrt{A_{nn}}).$$

1.4.2 Facts

1. The above diagonal scaling is nearly optimal (van der Sluis):

$$\kappa_2(A_S) \leq n \min_{D=\text{diag}} \kappa(DHD) \leq n\kappa_2(H).$$

2. Let A and $\tilde{A} = A + \Delta A$ both be positive definite, and let their eigenvalues have the same ordering. Then

$$\frac{|\lambda_i - \tilde{\lambda}_i|}{\lambda_i} \leq \frac{\|D^{-1}(\Delta A)D^{-1}\|_2}{\lambda_{\min}(A_S)} \equiv \|A_S^{-1}\|_2 \|\Delta A_S\|_2.$$

   If λ_i and $\tilde{\lambda}_i$ are simple, then

$$\|U_{:,i} - \tilde{U}_{:,i}\|_2 \leq \frac{\|A_S^{-1}\|_2 \|\Delta A_S\|_2}{\min\limits_{j \neq i} \dfrac{|\lambda_i - \lambda_j|}{\sqrt{\lambda_i \lambda_j}}}.$$

   These bounds are much sharper than the standard bounds for matrices for which $\kappa_2(A_S) \ll \kappa_2(A)$.

3. The Jacobi method with the relative stopping criterion

$$|A_{ij}| \leq tol\sqrt{A_{ii}A_{jj}}, \quad \forall i \neq j,$$

and some user defined tolerance tol (usually $tol = n\varepsilon$), computes the EVD with small scaled backward error

$$\|\Delta A_S\| \leq \varepsilon\, O(\|A_S\|_2) \leq O(n)\varepsilon,$$

provided that $\kappa_2([A_k]_S)$ does not grow much during the iterations. There is overwhelming numerical evidence that the scaled condition does not grow much, and the growth can be monitored, as well.

1.4.3 Example - Scaled matrix

```
In [12]: n=6
         A=rand(n,n)
         A=full(Symmetric(A'*A));
         As=[A[i,j]/sqrt(A[i,i]*A[j,j]) for i=1:n, j=1:n]
```

```
Out[12]: 6×6 Array{Float64,2}:
          1.0       0.832247  0.975987  0.70636   0.736069  0.460841
          0.832247  1.0       0.908833  0.70454   0.651929  0.715246
          0.975987  0.908833  1.0       0.731738  0.704186  0.519369
          0.70636   0.70454   0.731738  1.0       0.939989  0.574457
          0.736069  0.651929  0.704186  0.939989  1.0       0.472141
          0.460841  0.715246  0.519369  0.574457  0.472141  1.0
```

```
In [13]: cond(As), cond(A)
```

```
Out[13]: (4961.858340517916, 5172.032232802419)
```

```
In [14]: # We add a strong scaling
         D=exp.(50*(rand(n)-0.5))
```

```
Out[14]: 6-element Array{Float64,1}:
              7.14869e-11
              6.07424e10
           33301.5
             141.604
               6.65053e-8
               6.86988e-9
```

```
In [15]: H=diagm(D)*As*diagm(D)
```

```
Out[15]: 6×6 Array{Float64,2}:
          5.11037e-21     3.61385     2.32346e-6  ...     3.49946e-18    2.26322e-19
          3.61385         3.68964e21  1.8384e15       2633.59            298.467
```

```
 2.32346e-6      1.8384e15   1.10899e9         0.00155958     0.00011882
 7.15035e-9      6.06e12     3.4506e6          8.85225e-6     5.58833e-7
 3.49946e-18  2633.59        0.00155958        4.42295e-15    2.15713e-16
 2.26322e-19   298.467       0.00011882  ...     2.15713e-16    4.71952e-17
```

In [16]:
```
# Now we scale again
Hs=[H[i,j]/sqrt(H[i,i]*H[j,j]) for i=1:n, j=1:n]
```

Out[16]:
```
6×6 Array{Float64,2}:
 1.0       0.832247  0.975987  0.70636   0.736069  0.460841
 0.832247  1.0       0.908833  0.70454   0.651929  0.715246
 0.975987  0.908833  1.0       0.731738  0.704186  0.519369
 0.70636   0.70454   0.731738  1.0       0.939989  0.574457
 0.736069  0.651929  0.704186  0.939989  1.0       0.472141
 0.460841  0.715246  0.519369  0.574457  0.472141  1.0
```

In [17]:
```
cond(Hs),cond(H)
```

Out[17]:
```
(4961.858340517929, 1.0166023132674682e41)
```

In [18]:
```
# Jacobi method
λ,U=myJacobi(H)
```

Out[18]:
```
([1.60578e-23, 3.68964e21, 1.92991e8, 9135.26, 4.97889e-16, 1.57472e-17], [0.999999 9.7
```

In [19]:
```
# Standard QR method
λ₁,U₁=eig(H)
```

Out[19]:
```
([5.11037e-21, 3.68964e21, 1.92991e8, 9135.26, 1.57472e-17, 4.97889e-16], [1.0 9.7946e-
```

In [20]:
```
# Compare
[sort(λ) sort(λ₁)]
```

Out[20]:
```
6×2 Array{Float64,2}:
    1.60578e-23     5.11037e-21
    1.57472e-17     1.57472e-17
    4.97889e-16     4.97889e-16
 9135.26         9135.26
    1.92991e8       1.92991e8
    3.68964e21      3.68964e21
```

In [21]:
```
# Check with BigFloat
λ₂,U₂=myJacobi(map(BigFloat,H))
λ₂
```

```
Out[21]: 6-element Array{BigFloat,1}:
          1.60578064521779425081170004653247131435369069352312229522267436825139102107138 1e-23
          3.68964009959306672383378470489228283885017960914055196375076928078030525245256 4e+21
          1.92990917589727066742905351526221965980096276302175497436617634763437096691490 1e+08
          9.13525745130292512095177344681432552879916755679410511626830486882607155532157 7e+03
          4.97889051100701760537669880361640341003427651332816834695734174938008130842977 8e-16
          1.57471978563043692844042236571714209590804534674014142853578746547691383634594 4e-17

In [22]: # Relative error is eps()*cond(AS)
         map(Float64,(sort(λ2)-sort(λ))./sort(λ2))

Out[22]: 6-element Array{Float64,1}:
           1.15064e-13
          -4.32799e-16
           2.04755e-15
           4.23305e-16
          -9.63666e-16
           1.63882e-16
```

1.5 Indefinite matrices

1.5.1 Definition

Spectral absolute value of the matrix A is the matrix

$$|A|_{\text{spr}} = (A^2)^{1/2}.$$

This is positive definite part of the polar decomposition of A.

1.5.2 Facts

1. The above perturbation bounds for positive definite matrices essentially hold with A_S replaced by $[|A|_{\text{spr}}]_S$.
2. Jacobi method can be modified to compute the EVD with small backward error $\|\Delta[|A|_{\text{spr}}]_S\|_2$.

The details of the indefinite case are beyond the scope of this course, and the reader should consider references.