L4b Symmetric Eigenvalue Decomposition - Algorithms for Tridiagonal Matrices

Ivan Slapničar

April 9, 2018

1 Symmetric Eigenvalue Decomposition - Algorithms for Tridiagonal Matrices

Due to their importance, there is plethora of excellent algorithms for symmetric tridiagonal matrices.

For more details, see Section [Sla14] 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.

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 apply adequate algorithm to a given symmetric tridiagonal matrix and to assess the speed of the algorithm and the accuracy of the solution.

1.3 Bisection and inverse iteration

The bisection method is convenient if only part of the spectrum is needed. If the eigenvectors are needed, as well, they can be efficiently computed by the inverse iteration method.

1.3.1 Facts

A is a real symmetric $n \times n$ matrix and T is a real symmetric tridiagonal $n \times n$ matrix.

1. **Application of Sylvester's Theorem.** Let $\alpha, \beta \in \mathbb{R}$ with $\alpha < \beta$. The number of eigenvalues of A in the interval $[\alpha, \beta)$ is equal to $\nu(A - \beta I) - \nu(A - \alpha I)$. By systematically choosing the intervals $[\alpha, \beta)$, the bisection method pinpoints each eigenvalue of A to any desired accuracy.

2. The factorization $T - \mu I = LDL^T$, where $D = \operatorname{diag}(d_1, \ldots, d_n)$ and L is the unit lower bidiagonal matrix, is computed as:

$$d_1 = T_{11} - \mu, \quad d_i = (T_{ii} - \mu) - \frac{T_{i,i-1}^2}{d_{i-1}}, \quad i = 2, \ldots n,$$
$$L_{i+1,i} = \frac{T_{i+1,i}}{d_i}, \quad i = 1, \ldots, n-1.$$

   Since the matrices T and D have the same inertia, this recursion enables an efficient implementation of the bisection method for T.

3. The factorization from Fact 2 is essentially Gaussian elimination without pivoting. Nevertheless, if $d_i \neq 0$ for all i, the above recursion is very stable. Even when $d_{i-1} = 0$ for some i, if the IEEE arithmetic is used, the computation will continue and the inertia will be computed correctly. Namely, in that case, we would have $d_i = -\infty$, $l_{i+1,i} = 0$, and $d_{i+1} = t_{i+1.i+1} - \mu$.

4. Computing one eigenvalue of T by using the recursion from Fact 2 and bisection requires $O(n)$ operations. The corresponding eigenvector is computed by inverse iteration. The convergence is very fast, so the cost of computing each eigenvector is also $O(n)$ operations. Therefore, the overall cost for computing all eigenvalues and eigenvectors is $O(n^2)$ operations.

5. Both, bisection and inverse iteration are highly parallel since each eigenvalue and eigenvector can be computed independently.

6. If some of the eigenvalues are too close, the corresponding eigenvectors computed by inverse iteration may not be sufficiently orthogonal. In this case, it is necessary to orthogonalize these eigenvectors (for example, by the modified Gram--Schmidt procedure). If the number of close eigenvalues is too large, the overall operation count can increase to $O(n^3)$.

7. The EVD computed by bisection and inverse iteration satisfies the error bounds from previous notebook.

8. The bisection method for tridiagonal matrices is implemented in the LAPACK subroutine DSTEBZ. This routine can compute all eigenvalues in a given interval or the eigenvalues from λ_l to λ_k, where $l < k$, and the eigenvalues are ordered from smallest to largest. Inverse iteration (with reorthogonalization) is implemented in the LAPACK subroutine DSTEIN.

```
In [1]: n=6
        s=srand(421)
        T=SymTridiagonal(rand(n),rand(n-1))
```

```
Out[1]: 6×6 SymTridiagonal{Float64}:
         0.345443  0.17008    ·         ·         ·         ·
         0.17008   0.68487   0.525208   ·         ·         ·
```

```
 ⋅         0.525208   0.650991   0.785847    ⋅          ⋅
 ⋅          ⋅         0.785847   0.973053   0.135538    ⋅
 ⋅          ⋅          ⋅         0.135538   0.105135   0.958365
 ⋅          ⋅          ⋅          ⋅         0.958365   0.77247
```

In [2]:
```
λ,U=eig(T)
```

Out[2]:
```
([-0.588807, -0.197104, 0.311314, 0.814277, 1.4474, 1.74488], [0.00681508 0.143821 ... -
```

In [3]:
```
methods(LAPACK.stebz!);
```

In [4]:
```
λ₁, rest=LAPACK.stebz!('A','E',1.0,1.0,1,1,2*eps(),T.dv,T.ev)
```

Out[4]:
```
([-0.588807, -0.197104, 0.311314, 0.814277, 1.4474, 1.74488], [1, 1, 1, 1, 1, 1], [6])
```

In [5]:
```
λ-λ₁
```

Out[5]:
```
6-element Array{Float64,1}:
 -1.11022e-15
  2.55351e-15
  3.88578e-16
  1.11022e-15
 -4.44089e-16
  0.0
```

In [6]:
```
U₁=LAPACK.stein!(T.dv,T.ev,λ₁)
```

Out[6]:
```
6×6 Array{Float64,2}:
  0.00681508   0.143821    0.945295    0.289652    -0.0158664  0.0392057
 -0.0374353   -0.458782   -0.189687    0.79844     -0.1028     0.322589
  0.0885769    0.723851   -0.171203    0.102931    -0.144113   0.638373
 -0.114725    -0.474569    0.200775   -0.512237    -0.0773456  0.67301
  0.808452    -0.0997173   0.0123816   0.00326567   0.564874   0.131208
 -0.569166     0.0985644  -0.0257312   0.0748602    0.802089   0.129313
```

In [7]:
```
λ
```

Out[7]:
```
6-element Array{Float64,1}:
 -0.588807
 -0.197104
  0.311314
  0.814277
  1.4474
  1.74488
```

In [8]:
```
# Let us compute just some eigenvalues - from 2nd to 4th
λ₂,rest=LAPACK.stebz!('I','E',1.0,1.0,2,4,2*eps(),T.dv,T.ev)
```

```
Out[8]: ([-0.197104, 0.311314, 0.814277], [1, 1, 1], [6])

In [9]: # And the corresponding eigenvectors
        U₂=LAPACK.stein!(T.dv,T.ev,λ₂)

Out[9]: 6×3 Array{Float64,2}:
          0.143821    0.945295    0.289652
         -0.458782   -0.189687    0.79844
          0.723851   -0.171203    0.102931
         -0.474569    0.200775   -0.512237
         -0.0997173   0.0123816   0.00326567
          0.0985644  -0.0257312   0.0748602
```

1.4 Divide-and-conquer

This is currently the fastest method for computing the EVD of a real symmetric tridiagonal matrix T. It is based on splitting the given tridiagonal matrix into two matrices, then computing the EVDs of the smaller matrices and computing the final EVD from the two EVDs.

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

1.4.1 Facts

1. Let T be partitioned as
$$T = \begin{bmatrix} T_1 & \alpha_k e_k e_1^T \\ \alpha_k e_1 e_k^T & T_2 \end{bmatrix}.$$
We assume that T is unreduced, that is, $\alpha_i \neq 0$ for all i. Further, we assume that $\alpha_i > 0$ for all i, which can be easily be attained by diagonal similarity with a diagonal matrix of signs. Let
$$\hat{T}_1 = T_1 - \alpha_k e_k e_k^T, \qquad \hat{T}_2 = T_2 - \alpha_k e_1 e_1^T.$$
In other words, $\hat{T}_1$ is equal to T_1 except that T_{kk} is replaced by $T_{kk} - \alpha_k$, and $\hat{T}_2$ is equal to T_2 except that $T_{k+1,k+1}$ is replaced by $T_{k+1,k+1} - \alpha_k$. Let $\hat{T}_i = \hat{U}_i \hat{\Lambda}_i \hat{U}_i^T$, $i = 1, 2$, be the respective EVDs and let
$$v = \begin{bmatrix} \hat{U}_1^T e_k \\ \hat{U}_2^T e_1 \end{bmatrix}$$
(v consists of the last column of $\hat{U}_1^T$ and the first column of $\hat{U}_2^T$). Set $\hat{U} = \hat{U}_1 \oplus \hat{U}_2$ and $\hat{\Lambda} = \hat{\Lambda}_1 \oplus \hat{\Lambda}_2$. Then
$$T = \begin{bmatrix} \hat{U}_1 & \\ & \hat{U}_2 \end{bmatrix} \left[\begin{bmatrix} \hat{\Lambda}_1 & \\ & \hat{\Lambda}_2 \end{bmatrix} + \alpha_k v v^T \right] \begin{bmatrix} \hat{U}_1^T & \\ & \hat{U}_2^T \end{bmatrix} = \hat{U}(\hat{\Lambda} + \alpha_k v v^T)\hat{U}^T.$$
If $\hat{\Lambda} + \alpha_k v v^T = X \Lambda X^T$ is the EVD of the rank-one modification of the diagonal matrix $\hat{\Lambda}$, then $T = U\Lambda U^T$, where $U = \hat{U}X$ is the EVD of T. Thus, the original tridiagonal eigenvalue problem is reduced to two smaller tridiagonal eigenvalue problems and one eigenvalue problem for the diagonal-plus-rank-one matrix.

2. If all $\hat{\lambda}_i$ are different, then the eigenvalues λ_i of $\hat{\Lambda} + \alpha_k v v^T$ are solutions of the so-called secular equation,

$$1 + e_k \sum_{i=1}^{n} \frac{v_i^2}{\hat{\lambda}_i - \lambda} = 0.$$

   The eigenvalues can be computed by bisection, or by some faster zero finder of the Newton type, and they need to be computed as accurately as possible. The corresponding eigenvectors are

$$x_i = (\hat{\Lambda} - \lambda_i I)^{-1} v.$$

3. Each λ_i and x_i in $O(n)$ operations, respectively, so the overall computational cost for computing the EVD of $\hat{\Lambda} + \alpha_k v v^T$ is $O(n^2)$ operations.

4. The method can be implemented so that the accuracy of the computed EVD is given by the bound from the previous notebook.

5. Tridiagonal Divide-and-conquer method is implemented in the LAPACK subroutine DSTEDC. This routine can compute just the eigenvalues or both, eigenvalues and eigenvectors.

The file lapack.jl contains wrappers for a selection of LAPACK routines needed in the current Julia `Base`. However, *all* LAPACK routines are in the compiled library, so additional wrappers can be easily written. Notice that arrays are passed directly and scalars as passed as pointers. The wrapper for `DSTEDC`, similar to the ones from the file `lapack.jl` follows.

```
In [10]: # Part of the preamble of lapack.jl
         const liblapack = Base.liblapack_name
         import Base.LinAlg.BLAS.@blasfunc
         # import ..LinAlg: BlasFloat, Char, BlasInt, LAPACKException,
             # DimensionMismatch, SingularException, PosDefException, chkstride1, chksquare
         import Base.LinAlg.BlasInt
         function chklapackerror(ret::BlasInt)
             if ret == 0
                 return
             elseif ret < 0
                 throw(ArgumentError("invalid argument #$(-ret) to LAPACK call"))
             else # ret > 0
                 throw(LAPACKException(ret))
             end
         end

Out[10]: chklapackerror (generic function with 1 method)

In [11]: for (stedc, elty) in
             ((:dstedc_,:Float64),
             (:sstedc_,:Float32))
             @eval begin
                 """
                 COMPZ is CHARACTER*1
```

```
                  = 'N':  Compute eigenvalues only.
                  = 'I':  Compute eigenvectors of tridiagonal matrix also.
                  = 'V':  Compute eigenvectors of original dense symmetric
                          matrix also.  On entry, Z contains the orthogonal
                          matrix used to reduce the original matrix to
                          tridiagonal form.
                """
                function stedc!(compz::Char, dv::Vector{$elty}, ev::Vector{$elty}, Z::Array{$el
                    n = length(dv)
                    ldz=n
                    if length(ev) != n - 1
                        throw(DimensionMismatch("ev has length $(length(ev)) but needs one less
                    end
                    w = deepcopy(dv)
                    u = deepcopy(ev)
                    lwork=5*n^2
                    work = Array{$elty}(lwork)
                    liwork=6+6*n+5*n*round(Int,ceil(log(n)/log(2)))
                    iwork = Array{BlasInt}(liwork)
                    info = Array{BlasInt}(5)
                    ccall((@blasfunc($stedc), liblapack), Void,
                        (Ptr{UInt8}, Ptr{BlasInt}, Ptr{$elty},
                        Ptr{$elty}, Ptr{$elty}, Ptr{BlasInt}, Ptr{$elty}, Ptr{BlasInt},
                        Ptr{BlasInt}, Ptr{BlasInt}, Ptr{BlasInt}),
                        &compz, &n, w,
                        u, Z, &ldz, work, &lwork,
                        iwork, &liwork, info)
                        chklapackerror(info[])
                    w,Z
                end
            end
        end
```

In [12]:
```
μ,Q=stedc!('I',T.dv,T.ev,eye(n))
```

Out[12]:
```
([-0.588807, -0.197104, 0.311314, 0.814277, 1.4474, 1.74488], [-0.00681508 -0.143821 ..
```

In [13]:
```
λ-μ
```

Out[13]:
```
6-element Array{Float64,1}:
 -1.11022e-15
  2.35922e-15
  5.55112e-16
  8.88178e-16
 -2.22045e-16
 -4.44089e-16
```

In [14]:
```
Q'*(T*Q)
```

```
Out[14]: 6×6 Array{Float64,2}:
          -0.588807     -5.55112e-17  -7.80626e-18  ...  -4.44089e-16   5.55112e-16
          -4.16334e-17  -0.197104     -2.72135e-17       0.0            6.59195e-17
          -6.93889e-18  -6.17995e-18   0.311314          4.51028e-17  -9.1073e-17
          -1.04083e-17  -2.18575e-16   5.52943e-17      -1.38778e-16   2.04697e-16
          -3.33067e-16  -3.29597e-17   3.03577e-17       1.4474        9.15934e-16
           5.68989e-16   7.24247e-17  -7.32921e-17  ...   8.74301e-16   1.74488
```

```
In [15]: # Timings
         n=3000
         Tbig=SymTridiagonal(rand(n),rand(n-1));
```

```
In [16]: @time eig(Tbig);
         @time stedc!('I',Tbig.dv,Tbig.ev,eye(n));
```

```
  1.486532 seconds (151 allocations: 69.451 MiB, 3.95% gc time)
  0.682184 seconds (20 allocations: 413.545 MiB, 24.59% gc time)
```

1.5 MRRR

The method of Multiple Relatively Robust Representations

The computation of the tridiagonal EVD which satisfies the error stabdard error bounds such that the eigenvectors are orthogonal to working precision, all in $O(n^2)$ operations, has been the *holy grail* of numerical linear algebra for a long time. The method of Multiple Relatively Robust Representations does the job, except in some exceptional cases. The key idea is to implement inverse iteration more carefully. The practical algorithm is quite elaborate and the reader is advised to consider references.

The MRRR method is implemented in the LAPACK subroutine DSTEGR. This routine can compute just the eigenvalues, or both eigenvalues and eigenvectors.

```
In [17]: methods(LAPACK.stegr!);
```

```
In [18]: LAPACK.stegr!('V',T.dv,T.ev)
```

```
Out[18]: ([-0.588807, -0.197104, 0.311314, 0.814277, 1.4474, 1.74488], [0.00681508 0.143821 ...
```

```
In [19]: # Timings
         @time LAPACK.stegr!('V',Tbig.dv,Tbig.ev);
```

```
  1.581996 seconds (65 allocations: 69.399 MiB, 10.16% gc time)
```

```
In [20]: n=1500
         Tbig=SymTridiagonal(rand(n),rand(n-1));
```

```
In [21]: @time LAPACK.stegr!('V',Tbig.dv,Tbig.ev);
```

```
  0.373838 seconds (63 allocations: 17.535 MiB)
```