Solving the generalized eigenvalue problem

Julie Thiim Gadeberg

June 2019

1 Introduction

The aim of this report is to describe a created algorithm that can solve a generalized eigenvalue problem. The report gives a short introduction to the theory behind the problem provided by [1] and [2]. This is done in section 2. In section 3 the created algorithm is explained, and the reader is provided with a resulting example in section 4.

2 The generalized eigenvalue problem

The generalized eigenvalue equation is defined as:

$$Ax = \lambda Nx, \tag{1}$$

where A is a real symmetric matrix, and N is a real symmetric positive-definite matrix. The problem can represented as an ordinary eigenvalue problem:

$$By = \lambda y, \tag{2}$$

by using a eigenvalue decomposition on the matrix N. The chosen eigenvalue decomposition is the Jacobi diagonalization with cyclic sweeps, in which the off-diagonal elements of the matrix are zeroed, row after row, by each Jacobi rotation. The eigenvectors of the matrix can be calculated by $V = \mathbf{I}J_1 J_2 ...$ with J_n being the successive Jacobi matrices. The remaining result is a diagonalized eigenvalue matrix D, and a matrix of eigenvectors, V. It is then possible to rewrite N as:

$$N = VDV^T. \tag{3}$$

Utilizing $VV^T = \mathbf{I}$ we can rewrite equation 1 as equation 2 by inserting the newly found expression 3:

$$V^T A x = \lambda V D V^T x, \tag{4}$$
$$\sqrt{D}\sqrt{D^{-1}} V^T A V V^T x = \lambda \sqrt{D}\sqrt{D} x, \tag{5}$$
$$\sqrt{D^{-1}} V^T A V V^T \sqrt{D^{-1}} x = \lambda \sqrt{D} x, \tag{6}$$
$$By = \lambda y, \tag{7}$$

with $B = \sqrt{D^{-1}} V^T A V \sqrt{D^{-1}}$ while the eigenvector $y = \sqrt{D} V^T x$. From there, the Jacobi diagonalization with cyclic sweeps is performed again, this time over the matrix B, yielding its eigenvalues and eigenvectors. Solving the linear equation:

$$y = \sqrt{D} V^T x, \tag{8}$$

we thus obtain the original eigenvectors for the generalized system.

3 The algorithm

The overall master algorithm is based on several small subroutines created throughout the course. In the following subsections, these small subroutines will be explained thoroughly. In the last subsection the structure of the master algorithm is described.

Jacobi diagonalization

The first useful routine is the Jacobi eigenvalue decomposition routine with cyclic sweeps. The arguments of the algorithm is a real symmetric matrix A, and a precision that is used as a convergence criterion.

The algorithm firstly sets up the Jacobi matrix, J, and the matrix of eigenvectors as two separate identity matrices. Once this is done, the decomposition can start. The convergence criterion is chosen as $\max |A_{ij}| < \epsilon$, with ϵ being the aforementioned precision argument. A double nested for-loop is inserted inside the while-statement that is `True` until the convergence criterion is met. This ranges firstly over the number of rows (p) subtracted by 1, and then the columns (q) with numbers larger than the row number. While the rest of the matrix J is kept as before, the diagonal elements J_{pp}

and J_{qq} are replaced by $\cos(\phi)$, and the off-diagonal elements J_{pq} and J_{qp} are replaced by $\sin(\phi)$ and $-\sin(\phi)$ respectively. ϕ is given by:

$$\phi = 0.5 \arctan\left(\frac{2 * A_{pq}}{A_{qq} - A_{pp}}\right).$$

The algorithm proceeds by calculating $A \to A' = J^T A J$ while setting $V \to V' = VJ$. The Jacobi matrix is then reset as the identity matrix, and the loop continues. When the convergence criterion is met, the algorithm returns A' which is the diagonal eigenvalue matrix, as well as V being the matrix with eigenvectors as columns.

QR-decomposition

The master algorithm indirectly utilizes a subroutine in order to solve linear equations and calculate inverse matrices. The subroutine implements a QR-decomposition by modified in-place Gram-schmidt orthogonalization, and takes a real matrix A, as argument. The decomposition factorizes A such that $A = QR$ with Q being an orthogonal matrix, and R being an upper right triangular matrix.

The first part of the algorithm allocates memory for the matrices Q and R, and copies the matrix A. Then two parallel for-loops are created inside an outer for-loop ranging over the number of columns. The diagonal elements of R are the length of the separate column vectors in the copied matrix A'. This length is used in order to calculate the elements of Q where $Q_{pq} = {}^{A'_{pq}}/_{R_{qq}}$ which is the aim of the first parallel nested for-loop. The second parallel for-loop is doubly nested and calculates the remaining upper right triangular elements of R by $R_{pq} = \mathbf{q}_i^T \mathbf{a}_q$. Every time a new element of R is found, the matrix A is updated, so $A_{pq} = A_{pq} - Q_{pi} R_{iq}$ with i being the column number from the outer for-loop.

The algorithm returns the factorized matrix A as Q and R.

Solving the linear equation

In order to solve the linear equation $y = \sqrt{D} V^T x$, QR-decomposition subroutine is used. This algorithm takes a matrix A and vector b as arguments.

We use the QR-decomposition routine to firstly factorize the matrix A into Q and R in order to transform the system – moving from the original to an upper triangular system $Rx = Q^Tb$. The system can be solved by using back-substitution given as:

$$y_i = \frac{1}{R_{ii}}\left((Q^Tb)_i - \sum_{k=i+1}^{n} U_{ik}x_k\right), i = n, n-1, ..., 1. \tag{9}$$

The first $x_n = \frac{b_n}{U_{nn}}$. Each element calculated by use of equation 9 is stored in the vector x, which is then returned by the algorithm.

Calculating inverse matrices

In order to calculate the inverse of a given matrix A, we only need to utilize the linear equation solver subroutine, as it is the system $AA^{-1} = I$ that needs to be solved. Space is allocated for the inverse matrix A^{-1}, and the identity matrix I is defined. For every column in I the subroutine is used to solve the equation $Ae_i = a_i$ with a_i being the i'th column of the inverse matrix. The columns are then combined to the single inverse matrix that is returned by this algorithm.

The master algorithm

The master algorithm solves a generalized eigenvalue equation 1. Two real symmetric matrices A and N, with N also being positive definite, are used as arguments.

We firstly calculate the eigenvalue and -vector matrices D and V for the matrix N, and take the square root of the elements in D. By using the inverse subroutine it is possible to finally calculate the matrix B as described in equation 7. Once more the Jacobi eigenvalue decomposition subroutine is used to find the eigenvalues and eigenvectors for B.

As the problem is transformed into an ordinary eigenvalue problem, each eigenvector – y_i – of B can be used to restore the original eigenvectors x. This only requires solving the linear equation $y_i = \sqrt{D}V^Tx$. The original eigenvectors x and eigenvalues stored in D are then returned by the algortihm, and the generalized eigenvalue equation has been solved.

4 Results

The master algorithm is tested on two randomly generated real symmetric matrices A and N, where N is also positive-definite. The latter is ensured by generating a random real symmetric matrix M and multiplying it by its transpose - giving $N = M^T M$. In this report we look at the generalized eigenvalue problem:

$$\begin{bmatrix} 0.242 & 0.683 & 0.997 \\ 0.683 & 0.026 & 0.272 \\ 0.997 & 0.272 & 0.249 \end{bmatrix} x = \lambda \begin{bmatrix} 1.485 & 0.266 & 0.794 \\ 0.266 & 1.545 & 0.748 \\ 0.794 & 0.748 & 1.586 \end{bmatrix} x. \tag{10}$$

The master algorithm returns the eigenvalues and corresponding eigenvectors for the generalized eigenvalue equation one set at a time. The result is:

$$\mathbf{x} = \begin{bmatrix} 0.835 & -0.063 & 0.476 \\ 0.038 & 0.890 & 0.229 \\ -0.831 & -0.544 & 0.325 \end{bmatrix}, \boldsymbol{\lambda} = \begin{bmatrix} -1.017 & 0 & 0 \\ 0 & -0.176 & 0 \\ 0 & 0 & 0.580 \end{bmatrix}, \tag{11}$$

with $\mathbf{x}$ being a matrix with an eigenvector in each column, and $\boldsymbol{\lambda}$ being a diagonal eigenvalue matrix. It is possible to test the result by inserting one of the eigenvectors and its corresponding eigenvalue into the original equation 1 yielding:

$$\begin{bmatrix} 0.242 & 0.683 & 0.997 \\ 0.683 & 0.026 & 0.272 \\ 0.997 & 0.272 & 0.249 \end{bmatrix} \begin{bmatrix} 0.835 \\ 0.266 \\ 0.794 \end{bmatrix} = -1.02 \begin{bmatrix} 1.485 & 0.266 & 0.794 \\ 0.266 & 1.545 & 0.748 \\ 0.794 & 0.748 & 1.586 \end{bmatrix} \begin{bmatrix} 0.835 \\ 0.266 \\ 0.794 \end{bmatrix}$$

$$\begin{bmatrix} -0.600 \\ 0.346 \\ 0.636 \end{bmatrix} = \begin{bmatrix} -0.600 \\ 0.346 \\ -0.636 \end{bmatrix}.$$

In general, all of the sets are checked in order to make sure that the master algorithm works as intended, but only an example is given here. This shows that the algorithm works as intended, and the generalized eigenvalue problem has been solved.

References

[1] D. V. Fedorov. *Eigenvalues and eigenvectors*. `62.107.14.89/~fedorov/prog/numeric/book/eigen.pdf`.

[2] D. V. Fedorov. *Systems of linear equations*. `http://62.107.14.89/~fedorov/prog/numeric/book/lineq.pdf`.