Graded Homework 2 Numerical Linear Algebra 2026

The solutions are due Before Final Exam Week

Programming Exercise

General Tasks

1. Implement the Implicit QR algorithm for symmetric tridiagonal matrices. Explore the convergence behavior for various sizes of n and eigevalue distributions of T. Pay attention to the behavior of the subdiagonal elements in positions other than $n, n-1$, i.e., watch for deflation of the problem. You do not have to exploit this in your algorithm but if you do discuss how it helps your computational complexity.

2. Implement a code that transforms a symmetric matrix to a symmetric tridiagonal matrix using the appropriate similarity transformations based on Householder reflectors. The input to your code should only require storing the diagonal and either the elements in the upper have or the lower half of the symmetric matrix. (You may find it useful to follow the design strategy discussed in the solutions to Program 1, i.e., first design for a dense matrix with all elements storeed, add symmetry to the specification of the algorithm, , then add an efficient symmetric data structure.) Your output should be consistent with the input data structure required by your symmetric tridiagonal eigenvalue problem code.

3. Use the two codes to solve both tridiagonal and dense symmetric eigenvalue problems. Your codes should have the option of producing an orthogonal matrix U that contains the orthonormal eigenvectors corresponding to the eigenvalues computed in an order consistent with the eigenvalues placement in a diagonal matrix Λ in the factorization $A = U\Lambda U^T$. It is recommended that you first handle finding the eigenvalues then add the accumulation of the eigenvectors to your codes.

Comments

Make sure you create efficient computational versions of both codes. For the Householder reduction consult the literature on options such as using a single update form that combines the effect of left and right multiplication by a given Householder reflector.

There are forms of tridiagonal matrices that have known eigenvalues in addition to examples you can find in various text books. For example, if $T \in \mathbb{R}^n$ and the diagonals are constant, i.e., a typical row has only β, α, β as the three nonzero elements then

$$\lambda_k = \alpha + 2\beta \cos\left(\frac{k\pi}{n+1}\right)$$

You can also use Gershgorin's three theorems to generate a matrix with various distributions on eigenvalues but not necessarily the specific values. See for example, the textbook Numerical Mathematics by Quarteroni, Sacco and Saleri for a statement of all three theorems or Golub and Van Loan's book. Another example that is interesting is the use of the tridiagonal eigenvalue problem to find the roots of orthogonal polynomials as the eigenvalues of the associated Jacobi matrix. See for example, Numerical Mathematics by Quarteroni, Sacco and Saleri for the form of the matrix and a related discussion for Gauss quadrature.

You can also generate a symmetric matrix $A = Q\Lambda Q^T$ by choosing the eigenvalues and then constructing an orthogonal matrix Q. In addition to using built-in primitives of MATLAB or similar environments to generate orthogonal matrices for transfer to your code, it is possible to generate them via simple techniques. For example, an $n \times n$ rotation matrix can be easily defined by considering a random index pair (i, j) and random angle θ. The matrix, Z, that is the identity everywhere except positions (i, i), (j, j), (i, j), (j, i), where it is taken to be

$$\cos\theta = e_i^T Z e_i, \quad \cos\theta = e_j^T Z e_j, \sin\theta = e_i^T Z e_j, \quad -\sin\theta = e_i^T Z e_j$$

is a plane rotation and orthogonal. Selecting many, say s, random index pairs (i, j) and random angles θ and multiplying all of the rotations they define together yields an orthogonal matrix

$$Q = Z_1 Z_2 \cdots Z_s.$$

Of course, you need to select enough pairs and angles, s, so that the matrix is dense.

Similarly, one can use reflectors to generate an orthogonal matrix Q. Simply choose several random vectors, v_i, $i = 1, \ldots, s$ for a large s. Then for each v_i form an elementary reflector

$$Q_i = I - 2u_i u_i^T, \quad u_i = \frac{1}{\|v_i\|_2} v_i$$

and $Q = Q_1 Q_2 \cdots Q_s$. Taking $s >> n$ is a good way of scrambling the directions defining Q.

When using either rotations, reflectors or a combination, it is necessary to compute the the matrix-matrix product or matrix-vector products efficiently so as not to take huge amounts of time when creating test problems since you must run many of them. Make sure you exploit all of the structure available to gain computational and storage efficiency. You should of course point out anything you do along these lines in your solution.

Finally, you should verify that the matrix computed has orthonormal columns to at least single precision accuracy. This computation of $Q^T Q$ or $\tilde{Q}^T \tilde{Q}$ and comparison to I_n or I_k should be performed in double precision. Also, note that it is a good idea to do all of the computations creating the test matrix of any type, i.e., orthogonal, symmetric, tridiagonal, in double precision and verify that it satisfies all of the properties required of the matrix up to single precision levels for assessing single precision factorization and solution algorithms.

You may use Matlab's libraries or other well-known libraries such as LAPACK for comparison and correctness evaluations as well as any graphical display you find useful. However, your code must be implemented in a compiled and typed language.