Replacing Armadillo: Jacobi Diagonalizer

Jeffrey Jacob
Chem 279

What had to be done and Motivation

- Build the infrastructure to be a successful Armadillo replacement
    - Includes custom matrix and vector classes and all operations
- Implement Jacobi algorithm that utilizes the infrastructure to find Eigenvalues and Eigenvectors
- Eigenvalues and eigenvectors are fundamental concepts in linear algebra and have widespread applications in various fields of mathematics, physics, engineering, and computer science. They provide valuable insights into the behavior and properties of linear systems and matrices.

The ‘Support Code’

- Matrix and Vector classes:
    - Custom library: Templated Matrix<T> and Vector<T> classes.
    - Armadillo: arma::mat and arma::vec classes.
        - Element access and resizing
- Sum of all elements:
    - Custom: `sum()` member function for matrix and `sumVector(const Vector<T>& v)` non-member function for vector.
    - Armadillo: `accu()` function for both matrix and vector.
- .Norm calculation:
    - Custom library: `norm()` member function for matrix (only for vector-like matrices).
    - Armadillo: `norm()` member function for both matrix and vector.

.

The ‘Support Code’ (2)

- Dot product:
  - Custom library: `dot(const Vector<T>& other)` member function for vector.
  - Armadillo: `dot()` member function for vector.
- Distance between vectors:
  - Custom library: `distanceTo(const Vector<T>& other)` member function for vector.
  - Armadillo: `norm(vec1 - vec2)` can be used to calculate the distance between two vectors.
- Symmetric matrix check:
  - Custom library: `isSymmetric(const Matrix<T>& matrix)` non-member function.
  - Armadillo: `is_symmetric()` member function for matrix.
- Operator overloading for matrix multiplication etc

The Jacobi Algorithm

- The Jacobi Algorithm allows us to find the eigenvalues and eigenvectors of a symmetric matrix
- Choose a threshold value ε (epsilon) as a stopping criterion for the iteration.
- Repeat until off-diagonal elements of A are smaller than ε:
    - Find largest off-diagonal element A[i][j].
    - Compute rotation angle θ = 0.5 * arctan(2 * A[i][j] / (A[i][i] - A[j][j])).
    - Compute cosine (c) and sine (s) of θ.
    - Create rotation matrix R with c, s at (i,i), (i,j), (j,i), (j,j).
    - Update A := R^T * A * R.
    - Update eigenvector matrix V := V * R (initially V = identity matrix)
- After the iteration converges, the diagonal elements of the resulting matrix A will be the eigenvalues, and the columns of the matrix V will be the corresponding eigenvectors.

Why it works

- The Jacobi Method works by iteratively zeroing out the off-diagonal elements of the matrix A through a series of rotations. Each rotation eliminates the largest off-diagonal element by mixing the corresponding rows and columns. The rotation angles are chosen to minimize the off-diagonal elements while preserving the eigenvalues.
    - The goal is to find a rotation matrix R that, when applied to the matrix A, zeros out the target off-diagonal element while preserving the eigenvalues.
- The Jacobi Method is very stable and guarantees convergence for symmetric matrices

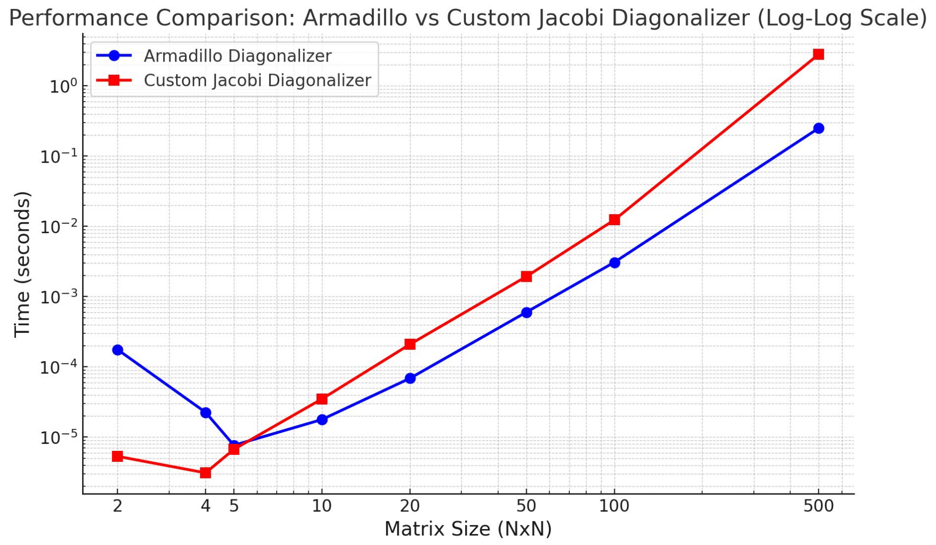
$$\begin{aligned} \boldsymbol{J}^T \boldsymbol{A} \boldsymbol{J} &= \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a & b \\ b & d \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \\ &= \begin{bmatrix} c^2a - 2csb + s^2d & c^2b + cs(a-d) - s^2b \\ c^2b + cs(a-d) - s^2b & c^2d + 2csb + s^2a \end{bmatrix} \end{aligned}$$

My Jacobi Algorithm

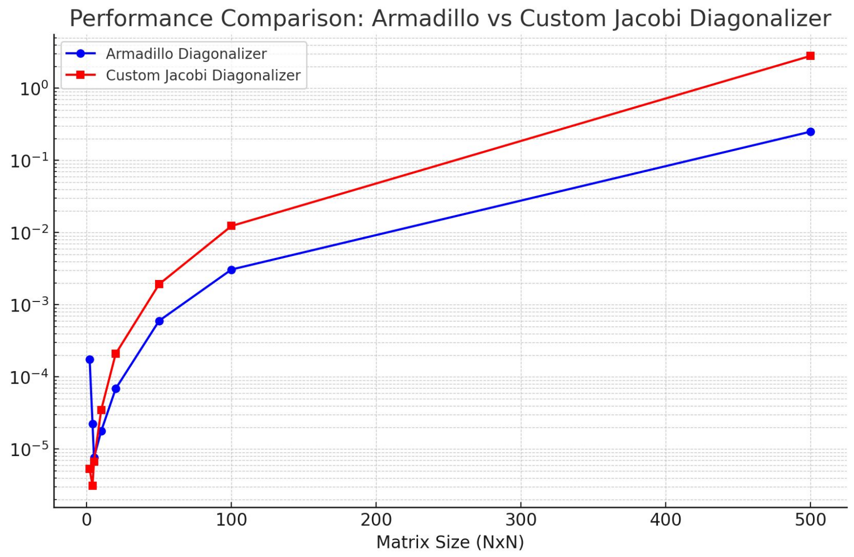
Replacing Armadillo eig_sym

- The algorithm is set to run for 50 iterations
- The threshold value is set differently for the first 4 iterations (0.2 * sum / (size * size)) compared to the later iterations (0.0). This is done to accelerate convergence in the early iterations.
  - Setting to 0 ensures full diagonalization
- If an off-diagonal element is larger than the threshold, the jacobiRotate function is called to perform a Jacobi rotation.
- If the sum of the absolute values of the off-diagonal elements becomes zero during the iteration, the matrix is considered diagonal, and the iteration is stopped.

Demo



Performance Comparison: Armadillo vs Custom Jacobi Diagonalizer (Log-Log Scale)
Armadillo Diagonalizer
Custom Jacobi Diagonalizer
Time (seconds)
Matrix Size (NxN)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
2
4
5
10
20
50
100
500



Performance Comparison: Armadillo vs Custom Jacobi Diagonalizer
Armadillo Diagonalizer
Custom Jacobi Diagonalizer
Matrix Size (NxN)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
100
200
300
400
500

Next Steps

- Can improve by using Blas and Lapack for things like matrix multiplication
- Integrate more armadillo features like decomposition, linear system solver etc
- Improve the Jacobi Algorithm
    - Instead of using the classical Jacobi rotation strategy, which rotates the largest off-diagonal element, A optimal rotation strategy could minimize the sum of the squares of the off-diagonal elements in each iteration.
    - Add a even more dynamic threshold than is already done
    - Early termination
    - Blocking and parallelization
    - Vectorization and SIMD instructions
    - Adaptive precision