Matrix Library Implementation

Juliet James & Trisha Kholiya

Our Goals

- Build a lightweight C++ matrix library
- Support core matrix operations
- Implement an eigensolver for symmetric matrices
- Validate accuracy + compare performance with Armadillo
- Use our library as a drop-in replacement for HW3

Library Architecture

```
Matrix Class
├── Storage: flat vector
├── Operators: +  −  *  scalar multiply
├── Utilities: transpose, identity, zeros, ones, random
└── Eigensolver Pipeline
     Householder → Tridiagonal T
            ↓
          QL → Eigenvalues
            ↓
      Combine Q_house × Q_ql → eigenvectors
```

Constructors

```
Matrix(int rows, int cols);
Matrix();
Matrix(const vec& values, int rows, int cols); LVALUE
Matrix(vec&& values, int rows, int cols); RVALUE
static Matrix Ones(int rows, int cols);
static Matrix Zeros(int rows, int cols);
static Matrix Random(int rows, int cols);
static Matrix Identity(int n);
```

Operators

```
double& operator()(int x, int y);
const double& operator()(int x, int y) const;
bool operator==(const Matrix& other) const;
Matrix operator+(const Matrix& other) const;
Matrix operator-(const Matrix& other) const;
Matrix operator*(const Matrix& other) const;
Matrix operator*(double s) const;
std::ostream& operator<<(std::ostream & out, const Matrix & M);
```

Optimizing Performance in our Library

- STORAGE: Switched to 1D storage
- Optimized our operators by using raw pointers as much as possible -> direct memory access
- unit-stride vector operations (i, i +1, etc.) optimizes performance

Multiplication Code

```
for (int i = 0; i < rows; ++i) {
 for (int j = 0; j < cols; ++j) {
  double sum = 0.0;
  for (int k = 0; k < my_cols; ++k) {
      sum += m1[i * my_cols + k] * m2[k * cols + j];
      }
      r[i * cols + j] = sum;
    }
}
```

```
for (int i = 0; i < rows; ++i) {
  for (int k = 0; k < my_cols; ++k) {
    double a = m1[i * my_cols + k];
    for (int j = 0; j < my_cols; ++j) {
      r[i * cols + j] += a * m2[k * cols + j];
```

- Removed temporary variable sum
- Decreased direct memory access
- Increased performance!

Householder & QL Overview

Householder tridiagonalization

- Uses reflections to zero out lower entries
- Produces tridiagonal T
- Stores transforms in Q_house
- Makes eigen-solve cheap ($O(n^2)$)

QL on T

- Implicit-shift iterations diagonalize T
- Diagonal → eigenvalues
- Stores transforms in Q_ql

$$\underset{A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1} \underset{Q_1A}{\begin{bmatrix} \mathbf{\times} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} \end{bmatrix}} \xrightarrow{Q_2} \underset{Q_2Q_1A}{\begin{bmatrix} \times & \times & \times \\ & \mathbf{\times} & \mathbf{\times} \\ & \mathbf{0} & \mathbf{\times} \\ & \mathbf{0} & \mathbf{\times} \\ & \mathbf{0} & \mathbf{\times} \end{bmatrix}} \xrightarrow{Q_3} \underset{Q_3Q_2Q_1A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \mathbf{\times} \\ & & \mathbf{0} \\ & & \mathbf{0} \end{bmatrix}} \qquad (10.1)$$

Each reflection wipes out one column of lower entries until only a tridiagonal band remains.

Trefethen, L. N., & Bau, D. (1997). Numerical Linear Algebra. Philadelphia: SIAM.

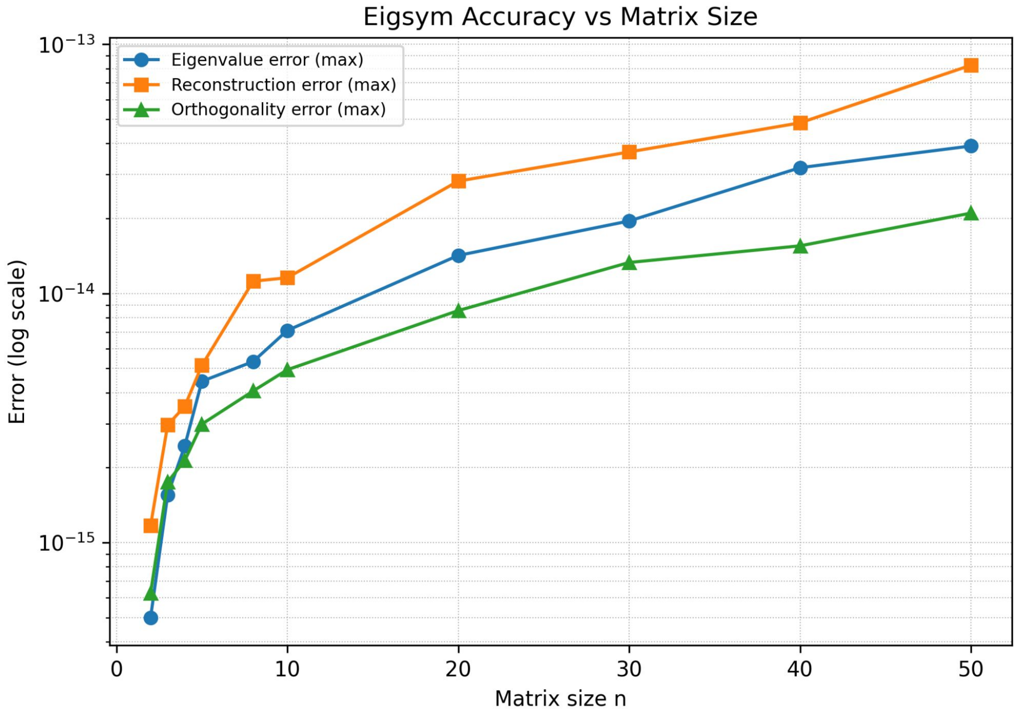
How the Pieces Combine ($A \to T \to D$)

- Householder: $A \to T$
- QL: $T \to$ eigenvalues + local eigenvectors
- Full eigenvectors: P = Q_house × Q_ql
- Validation: $A \approx P \Lambda P^T$

```
Input matrix A
   |
   |  Householder reflections
   v
Tridiagonal matrix T
   |
   |  QL on tridiagonal (fast)
   v
Diagonal eigenvalue matrix D
   |
   |  Combine transforms
   v
Eigenvectors P = Q_house × Q_ql
```

Accuracy Validation & Results

- Reconstruction: $A \approx P \Lambda P^T$
- Orthogonality: $P^T P \approx I$
- Perfect on identity + diagonal tests
- Errors grow with matrix size (floating-point + iterative QL)
- Stable and accurate for small–medium matrices

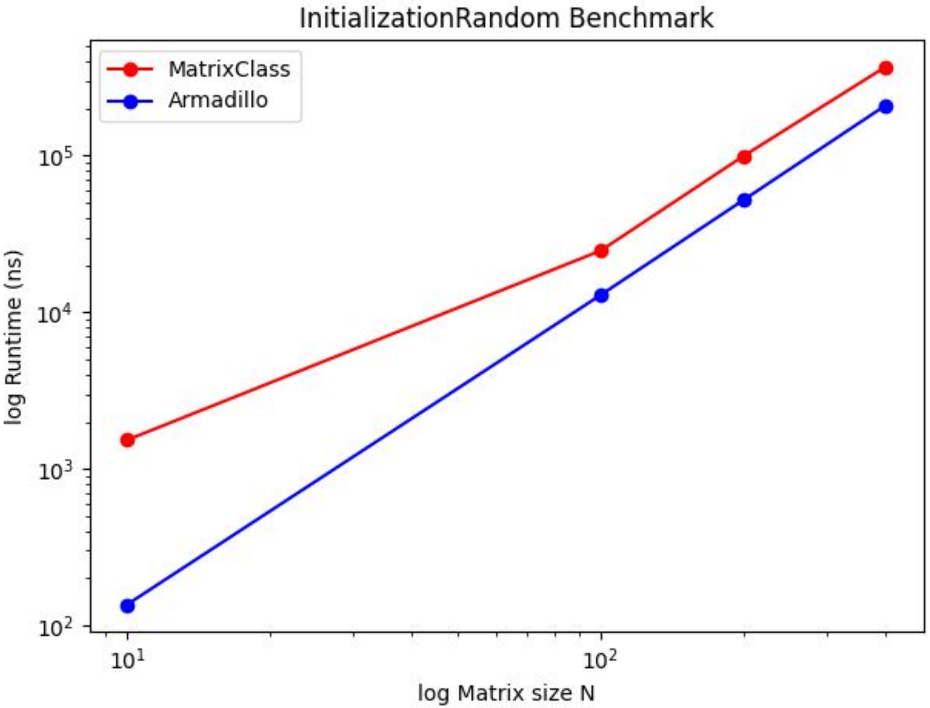


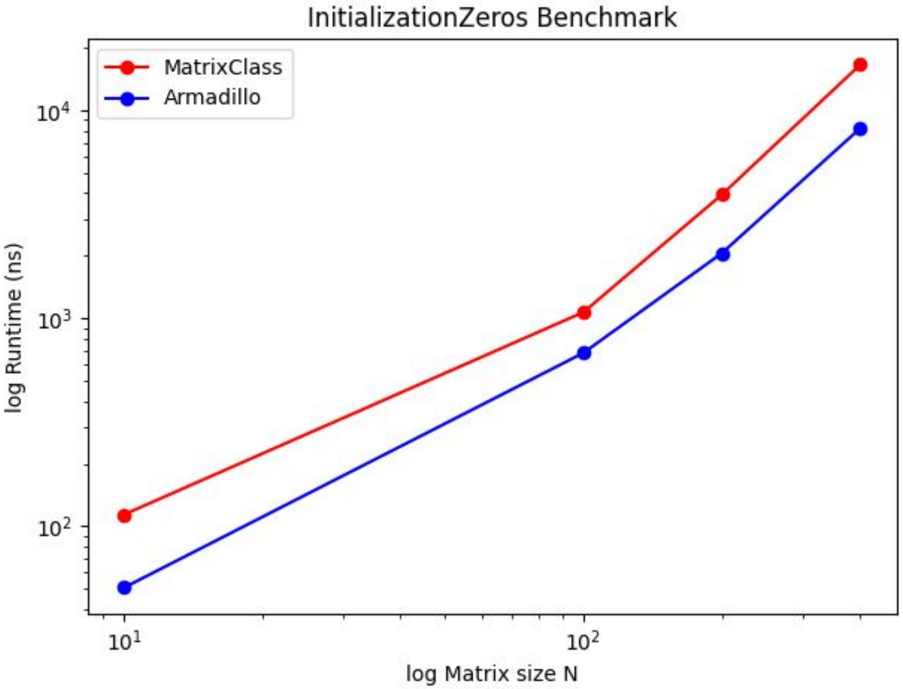
Accuracy vs Armadillo (why ours diverges)

- Armadillo uses highly optimized LAPACK routines
- Better numerical stability + shift strategies
- Our solver works but isn't as optimized
- Produces correct eigenpairs for moderate sizes

Benchmarking: Constructors

Google benchmark



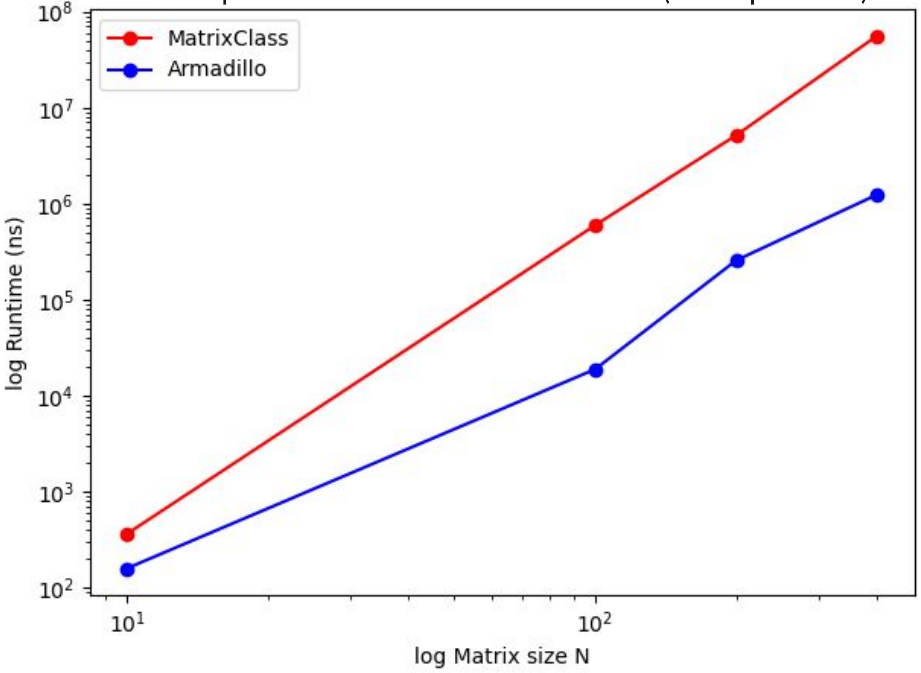


Benchmarking: Multiplication

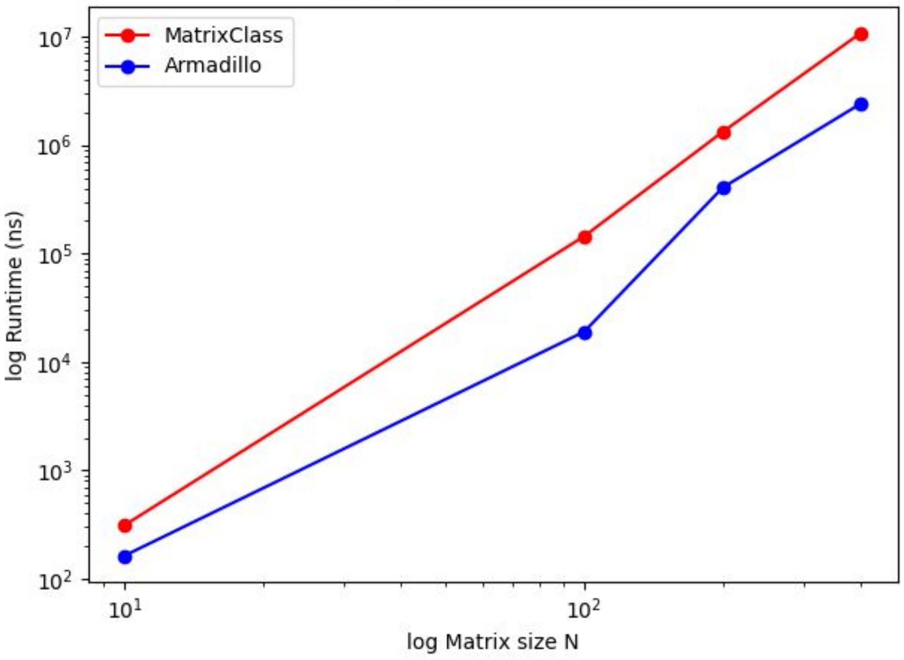
BEFORE

NOW

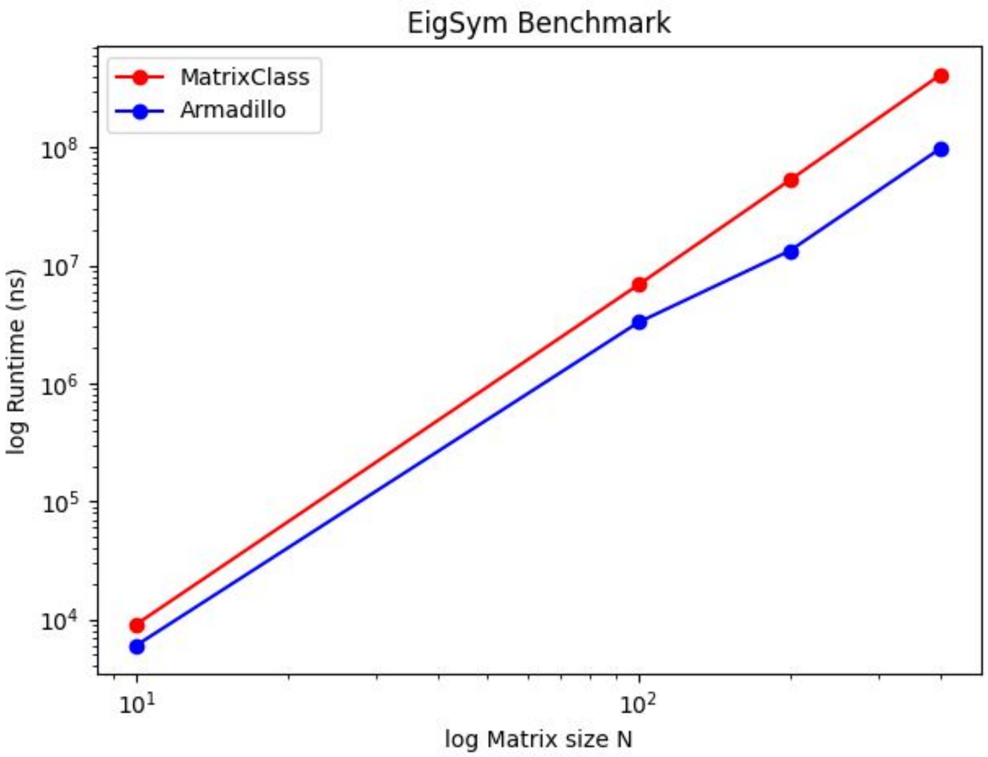
Multiplication Benchmark: Dot Product (Not Optimized)



Multiplication Benchmark: Outer Product (Optimized)



Benchmarking EigSym



Armadillo is Faster...why?

- Armadillo uses high-performance SIMD ->
- Can be used with with NVBLAS to obtain GPU-accelerated matrix multiplication
- Delayed Evaluation: Expression Templates
  - Lightweight marker objects that hold references to matrices and data associated with specific operations.
  - The marker objects can be chained together, leading to the full description of an arbitrary -> reduces temporary variables

Packaging the Library

- CMakeLists includes Install Section and Lib includes a config file so that the package can be installed by users that need Matrix Class
- Custom Exception class to make errors more user readable
- Printing vector class

Replacing Armadillo in HW3

DEMO!

Next Steps

- Implement HPC methods to boost performance
- Explore more optimization opportunities i.e. delayed evaluation
- Extend Matrix class to have more vector-capabilities like Armadillo
- Generalization to non-symmetric matrices is a key next milestone

References

Conrad Sanderson and Ryan Curtin.
Armadillo: An Efficient Framework for Numerical Linear Algebra.
International Conference on Computer and Automation Engineering, 2025.
https://doi.org/10.1109/ICCAE64891.2025.10980539

Conrad Sanderson and Ryan Curtin.
Practical Sparse Matrices in C++ with Hybrid Storage and Template-Based Expression Optimisation.
Mathematical and Computational Applications, Vol. 24, No. 3, 2019.
https://doi.org/10.3390/mca24030070

Trefethen, L. N., & Bau, D. (1997).
Numerical Linear Algebra. Philadelphia: SIAM.
https://epubs.siam.org/doi/book/10.1137/1.9781611977165

John D. McCalpin
A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers
https://www.cs.virginia.edu/~mccalpin/papers/balance/

Daniel Lemire
Rolling your own fast matrix multiplication: loop order and vectorization
https://lemire.me/blog/2024/06/13/rolling-your-own-fast-matrix-multiplication-loop-order-and-vectorization/?utm_source=chatgpt.com

Thank you :)

Please use our Matrix Library for your code!!!