nalgebra

Rust library

Compared against: faer-rs, eigen (C++), ndarray, Julia

Cal Brynestad, Tinie Doan, Cameron Tredoux

About the nalgebra Library

- Linear algebra library for the Rust programming language
- Free and open source
- Heap or stack-allocated vectors and matrices parametrized by their dimensions
- Compute matrix decompositions and solutions to linear systems
- Benefits from efficient Rust implementations and Lapack bindings

Why nalgebra?

- It has a pretty complete set of methods. It’s written in Rust which gave us an opportunity to test the speed of linear algebra operations in a system language
- It has a large community for gaming, graphics, and data science, so there is some overlap with the Julia community
- There were a lot of benchmarks written for this library by the creator of faer-rs, so it made it easier to contribute to those tests
- We wanted to see why this library was better for smaller dimension arrays compared to others that perform much faster for larger dimensions

What and How

- Libraries Compared:
  - faer-rs
  - eigen (C++)
  - Julia’s LinearAlgebra
  - ndarray
  - nalgebra
- Methods compared:
  - Matrix multiplication
  - Triangular solve
  - Cholesky decomposition
  - LU decomposition
  - QR decomposition
  - Square matrix singular value decomposition

Execution time testing

Each method is tested anywhere from 10 to 100,000,000 times depending on the initial few executions (if they meet a speed threshold, they get tested more).

For larger dimensions, since the runtime is longer, only a few thousand iterations.

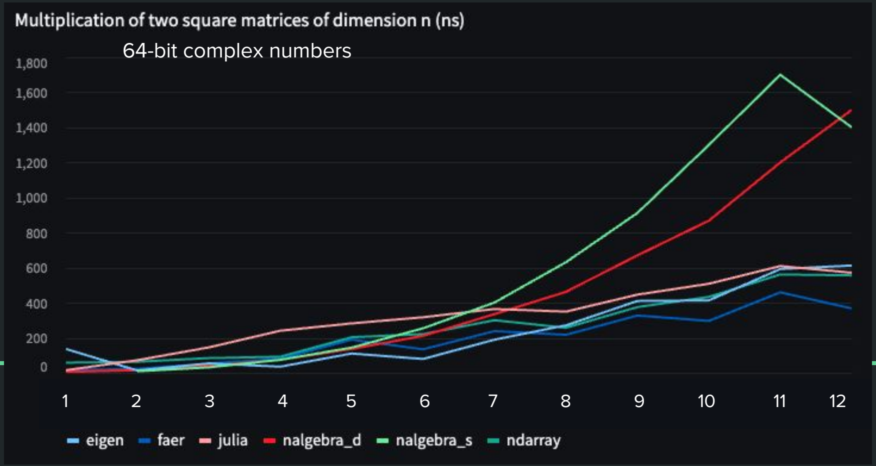
All matrices are square with dimensions from 1x1 to 12x12.

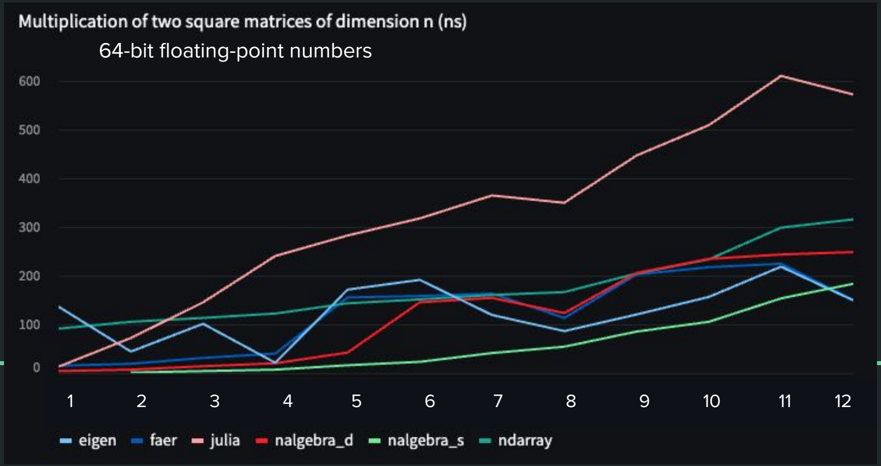
All tests were run on a Ryzen 9 5900x @ 3.7GHz w/ 32GB RAM

Number of tests was proportional to the runtime

Matrix Multiplication

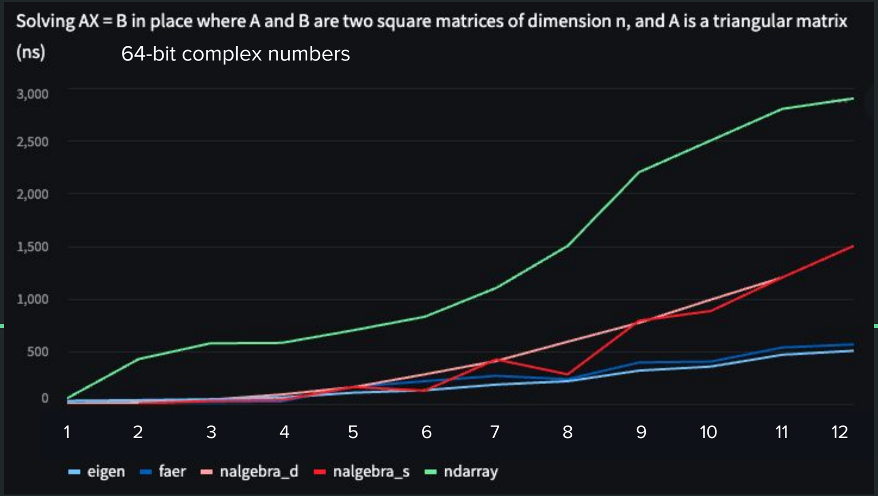
y-axis in nanoseconds, x-axis is matrix dimension

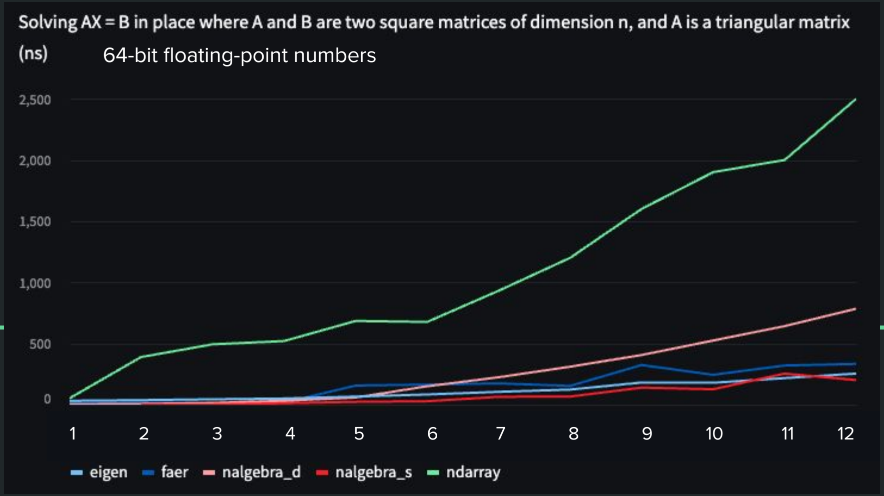




Triangular Solve

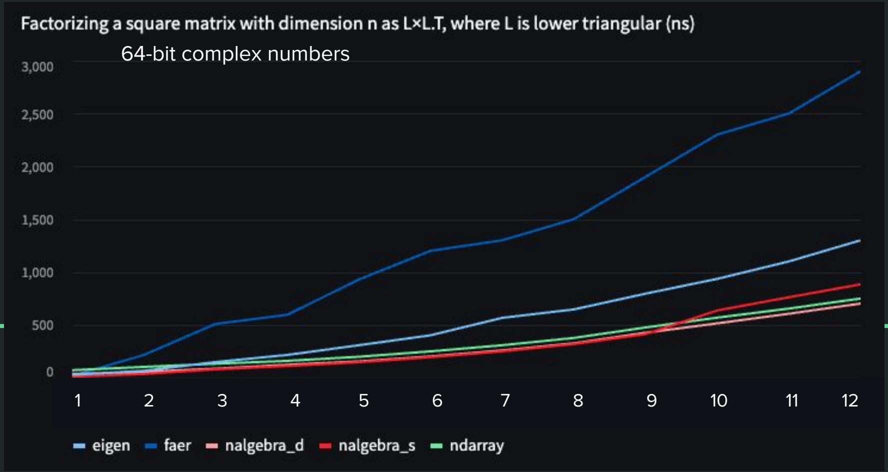
y-axis in nanoseconds, x-axis is matrix dimension

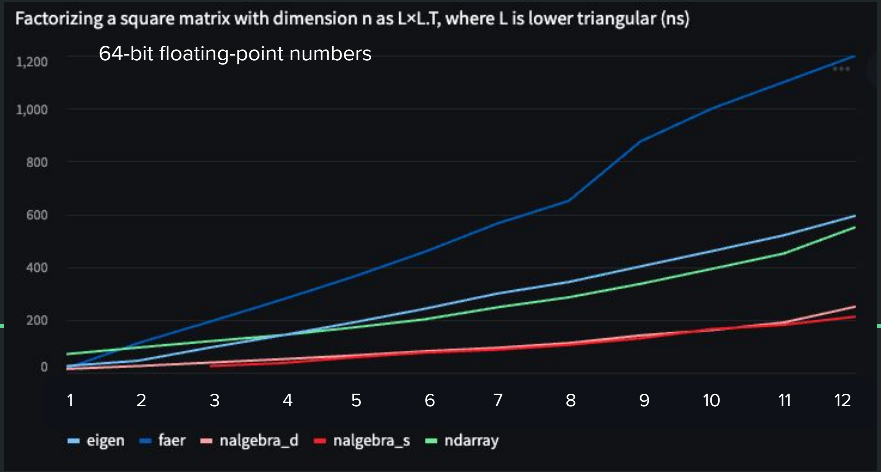




Cholesky Decomposition

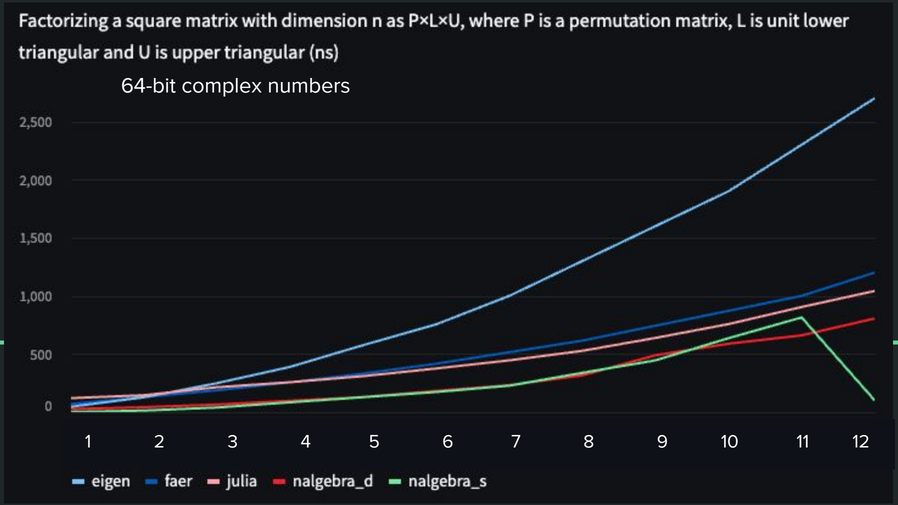
y-axis in nanoseconds, x-axis is matrix dimension

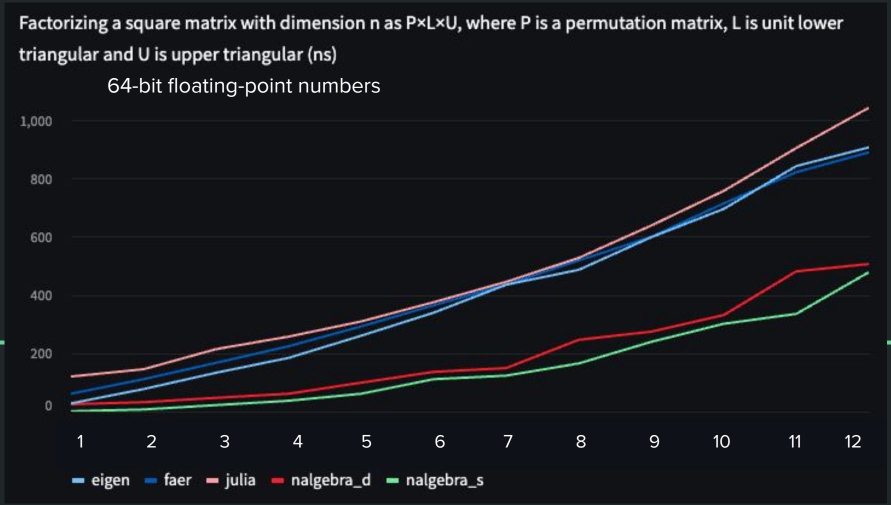




LU decomposition with partial pivoting

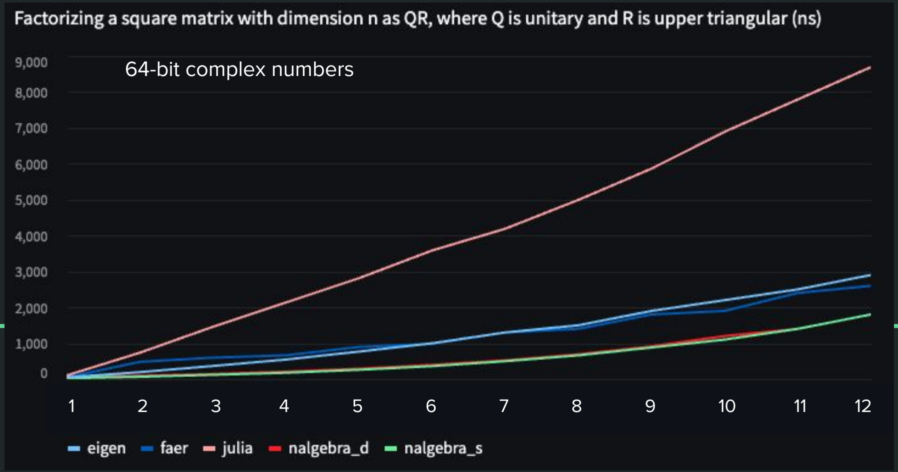
y-axis in nanoseconds, x-axis is matrix dimension

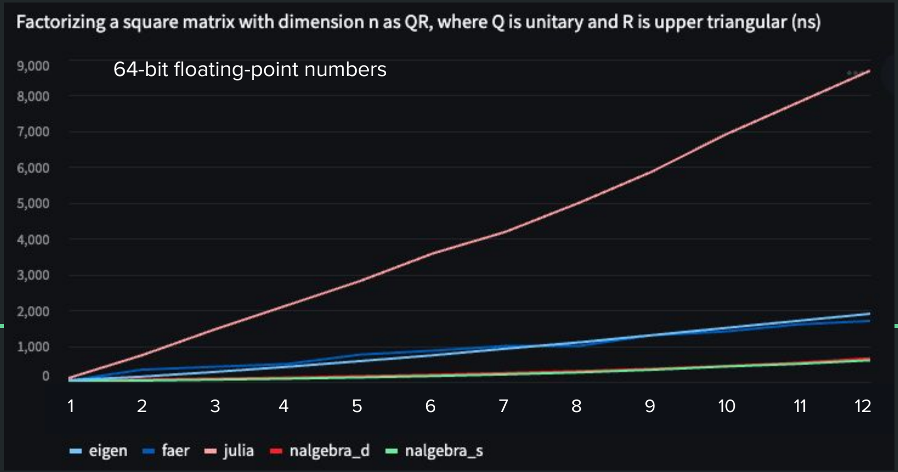




QR decomposition without pivoting

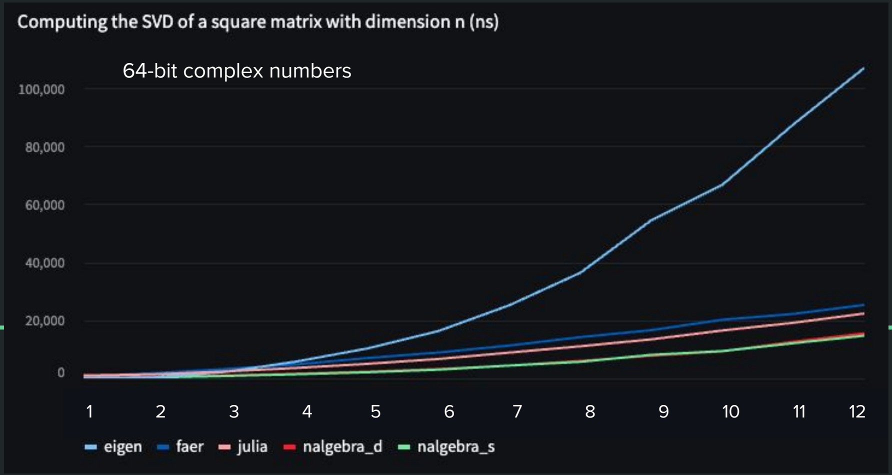
y-axis in nanoseconds, x-axis is matrix dimension

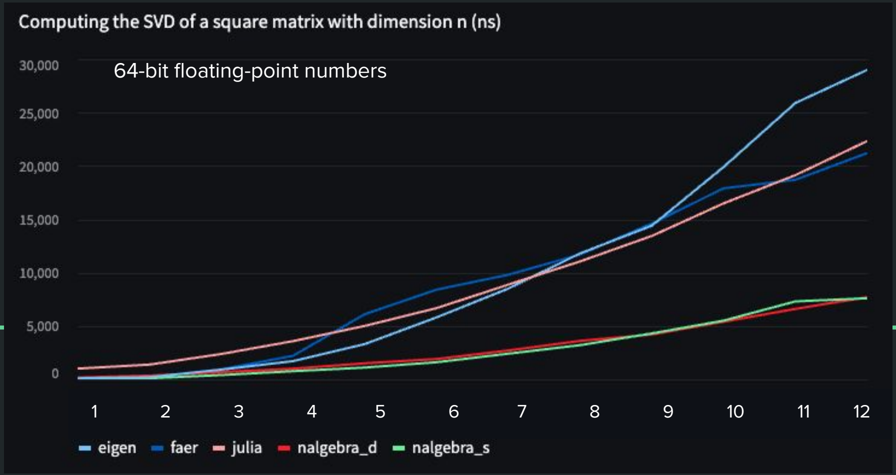




Square matrix singular value decomposition

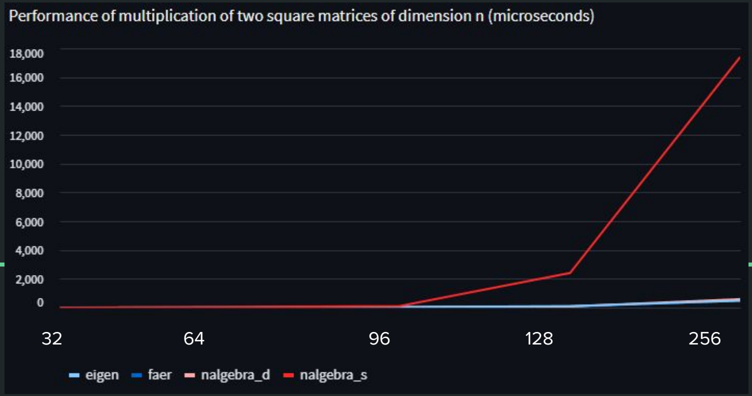
y-axis in nanoseconds, x-axis is matrix dimension



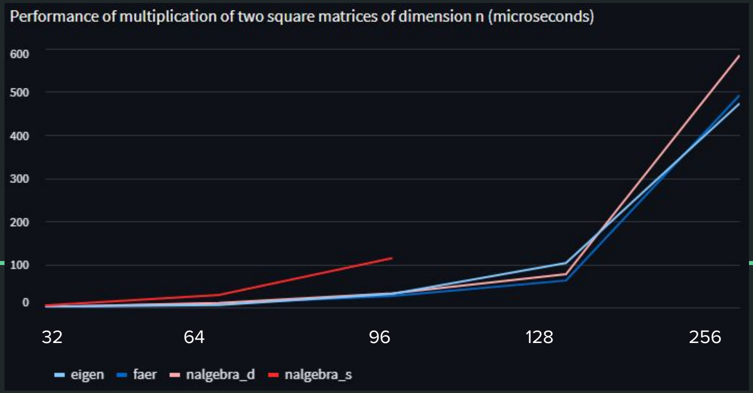


Large dimension matrix mult.

y-axis in microseconds, x-axis is matrix dimension



64-bit floating-point numbers



Not shown in graph: Julia computes a 256x256 in 467 microseconds

Considerations and Interpretations

- Sometimes a 4x4 matrix or a 12x12 matrix computes faster than matrices of smaller dimensions
    - Possibly has to do with SIMD which provides better parallelism for specific matrix dimensions
        - SIMD = Single Instruction Multiple Data
        - Creators of nalgebra have a Rust crate called simba which can perform SIMD algebra
    - Might be more efficient for the computer to pack those matrices into registers compared to other dimensions
        - If a 4x4 can be stored in more registers than a 3x3, then the CPU could perform matrix multiplication using fewer instructions and therefore be faster
- Nalgebra with static allocation is far more efficient
    - Falls off only when computing complex numbers, but is still faster for smaller dimensions
    - Did not always support complex numbers and the way we tested did not use the Complex crate provided by nalgebra

SIMD Speed Improvements

Results from https://www.rustsim.org/blog/2020/03/23/simd-aosoa-in-nalgebra/

benchmark	nalgebra_f32x4	nalgebra
2x2 matrix transpose	6.4984ns	11.0205ns
4x4 matrix mult	0.06897μs	0.1285μs
3x4 matrix mult	0.02883μs	0.09077μs
vec3 norm	15.5892ns	59.1804ns

Summary of Findings/Impacts

- Great for small dimensions and is faster when statically allocated, making it perfect for graphics libraries where a 4x4 matrix is commonly used
- Falls off in terms of performance for larger dimensions, making it weaker for scientific computing or physics simulations
    - This is where eigen or faer really seem to outshine nalgebra
- For data scientists who deal with large sets, nalgebra is not the way to go
- Nalgebra uses stack-allocation which is faster than heap but requires dimensions to be known at compile time
    - Probably why it is better than the other libraries for smaller dimensions since it is optimized specifically for this situation

Summary of Findings/Impacts

- Nalgebra uses OpenBLAS (instead of BLAS)
  - Adds specific optimizations for certain processors
  - Optimized for square matrices
  - Provides improved performance on multi-core CPUs that also support SIMD

Questions & Feedback