AMATH 583 Homework 4

AMATH 583 Course Overview at the University of Washington
Spring Quarter 2025 Section A with Professor Prof. Kenneth J. Roche

Assignment PDF

pdf_amath583_hw4.pdf

Problem 1

$A = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}$

acts on each unit ball in $\mathbb{R}^2$ under different norms.

Step 1: General Form of Transformation

Let $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$. Then:

$$Ax = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + 2x_2 \\ 2x_2 \end{bmatrix} = y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

So the transformation is:

- $y_1 = x_1 + 2x_2$
- $y_2 = 2x_2$

1-Norm Unit Ball: $\|x\|_1 \leq 1$

This is the **diamond-shaped** region bounded by:
$|x_1| + |x_2| \leq 1$

Vertices of this diamond are:

- $(1, 0), (-1, 0), (0, 1), (0, -1)$

Apply A to each vertex:

- $A(1,0) = (1,0)$
- $A(-1,0) = (-1,0)$
- $A(0,1) = (2,2)$
- $A(0,-1) = (-2,-2)$

These transformed points define a **skewed diamond** (parallelogram). It is **elongated and sheared**, not symmetric

So, geometrically, the unit diamond becomes a **parallelogram**, stretched more in the x_2 direction and skewed by mixing with x_1

2-Norm Unit Ball: $\|x\|_2 \leq 1$

This is the **circular region** of radius 1 centered at the origin. To understand the transformation, look at how it affects radial vectors.

Let:
$x = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \Rightarrow Ax = \begin{bmatrix} \cos\theta + 2\sin\theta \\ 2\sin\theta \end{bmatrix}$

This describes a **parametric curve** — the image of the unit circle under a linear map is an **ellipse**.

More formally:

- Since A is linear, the image of the unit circle is an ellipse with axes aligned to the **singular vectors** of A
- The ellipse's shape is defined by the **singular values** of A, which are the square roots of the eigenvalues of A^TA

Compute A^TA:

> ⚠ **Warning**
>
> Make sure I did this one right, suspicious

$A^T = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}, \quad A^TA = \begin{bmatrix} 1^2+0^2 & 1\cdot 2 + 0\cdot 2 \\ 2\cdot 1 + 2\cdot 0 & 2^2+2^2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 8 \end{bmatrix}$

$\det(A^TA - \lambda I) = (1-\lambda)(8-\lambda) - 4 = \lambda^2 - 9\lambda + 4 \Rightarrow \lambda = \frac{9 \pm \sqrt{81-16}}{2} = \frac{9 \pm \sqrt{65}}{2}$

So singular values of A: $\sqrt{\lambda_1}, \sqrt{\lambda_2}$

This tells us the image is an **ellipse** with:

- **Principal axes** along the singular vectors
- **Stretched** according to singular values $\approx \sqrt{(9 \pm \sqrt{65})/2}$

∞-Norm Unit Ball: $\|x\|_\infty \leq 1$

This is the **axis-aligned square** with corners:

- $(\pm 1, \pm 1)$

Apply A to the four corners:

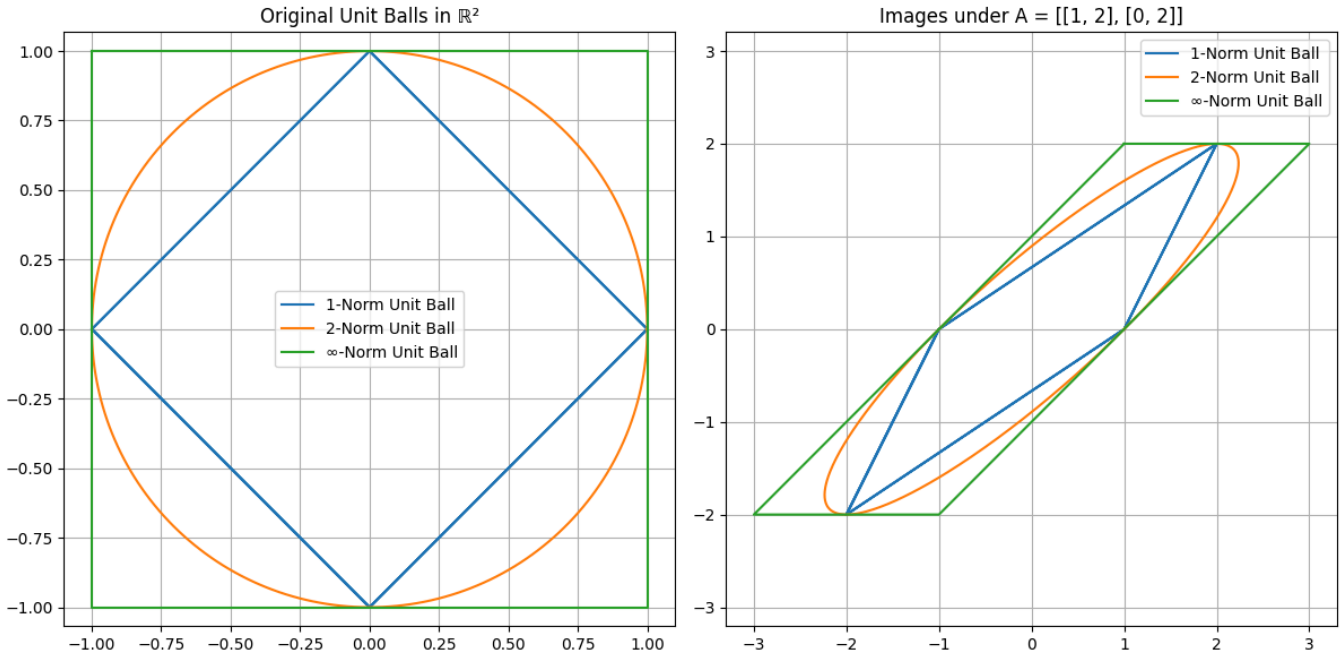
- $A(1,1) = (3,2)$
- $A(1,-1) = (-1,-2)$
- $A(-1,1) = (1,2)$
- $A(-1,-1) = (-3,-2)$

These four points form a **parallelogram**, much like in the 1-norm case, but aligned with the square input rather than diamond input.

Again, the output is **sheared and scaled** — but with sharper corners than the ellipse.

Summary of Image Shapes

Norm	Input Shape	Output under A	Description
1-norm	Diamond	Parallelogram	Sheared and scaled
2-norm	Circle	Ellipse	Aligned to singular vectors of A
∞-norm	Square	Parallelogram	Axis-aligned square → sheared



Problem 2

> **Note**
>
> Compiler Optimization of Matrix Multiplication Loop Permutations. Implement C++ templated gemm, $C \leftarrow \alpha AB + \beta C\,(A \in \mathbb{T}^{m\times p}, B \in \mathbb{T}^{p\times n}, \alpha, \beta \in \mathbb{T})$ for kij and jki loop permutations using the specifications provided here:

```
template<typename T>
void mm_jki(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
std::vector<T>& C, int m, int p, int n);

template<typename T>
void mm_kij(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
std::vector<T>& C, int m, int p, int n);
```

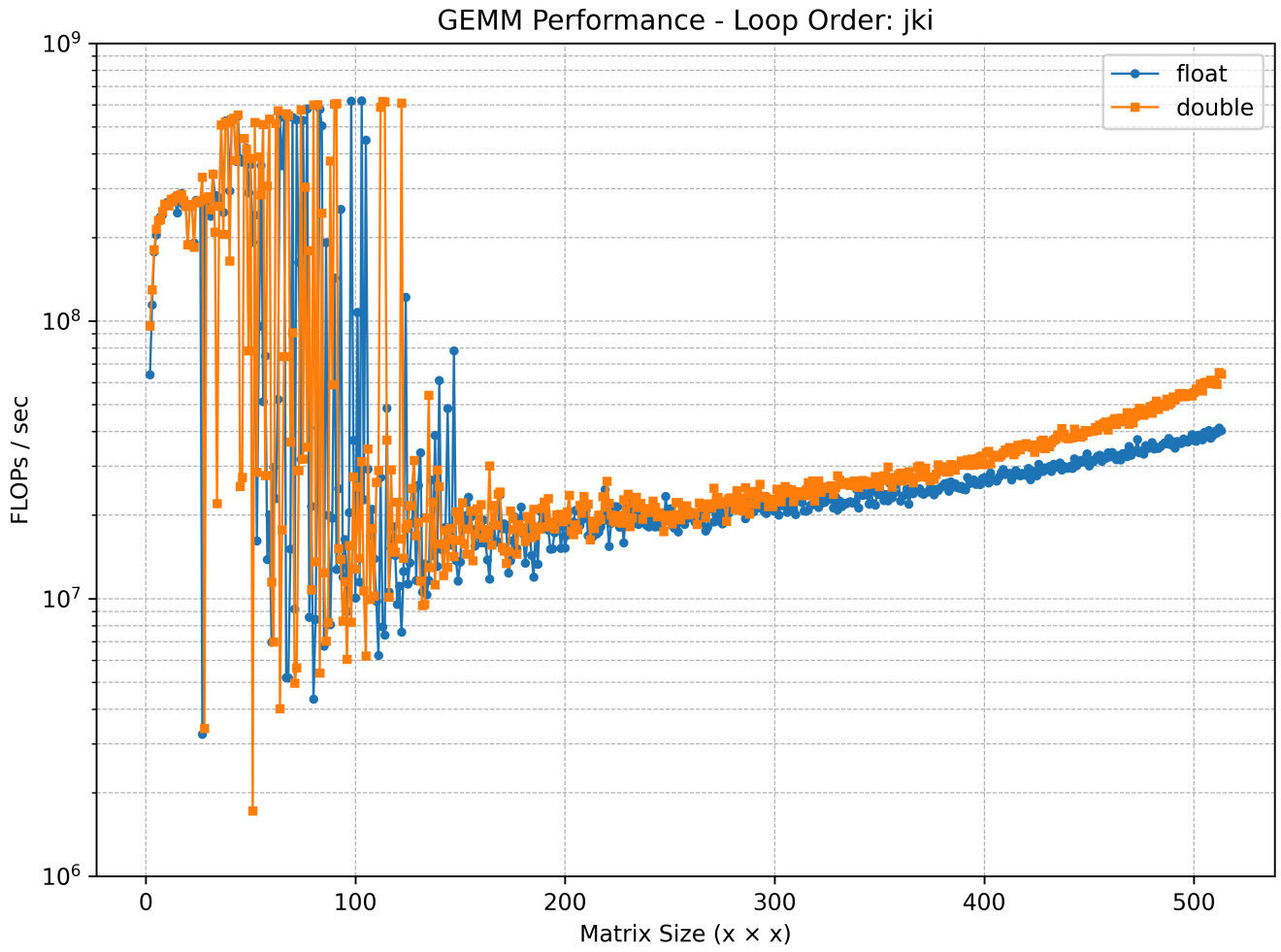
> **Note**
>
> You will explore matrix multiply performance applying compiler optimization levels $-O0$ and $-O3$ for square matrices of dimension $n = 2$ to $n = 512$, strid one to these functions. For reference, recall the loop order for

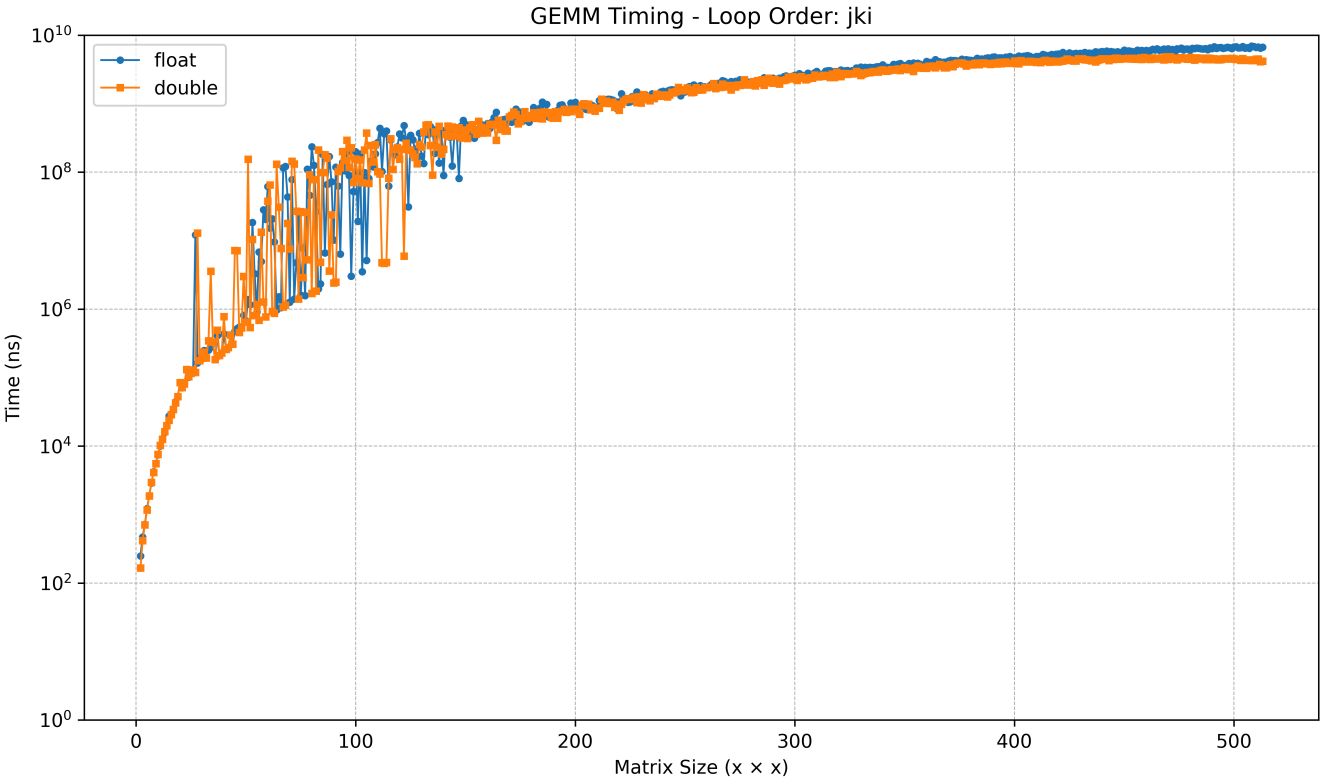
$$C_{i,j} \leftarrow \sum_{k=0}^{p-1} A_{i,k} * B_{k,j} \quad \forall [i = 0, \ldots, m-1][j = 0, \ldots, n-1]$$

is $\{ijk\}$, i outer loop, j middle loop, k inner loop. Let each n be measured $ntrial$ times and plot the average performance in FLOPs (flop count / time (seconds)) for each case versus $n, ntrial \geq 3$. Submit plots for both permutation variants that include both FP32 (*float*) and FP64 (*double*) compiler optimization results.

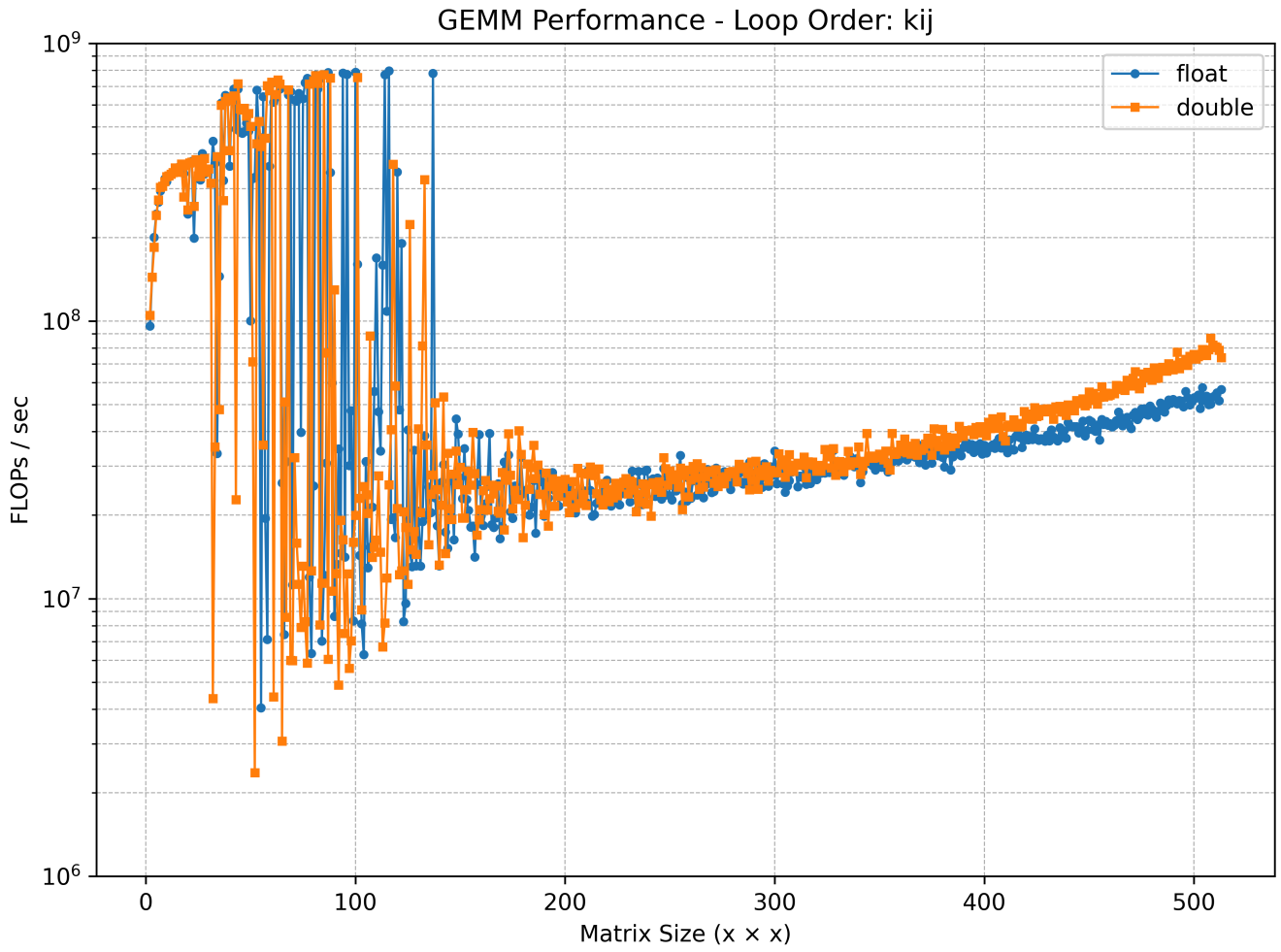
Results

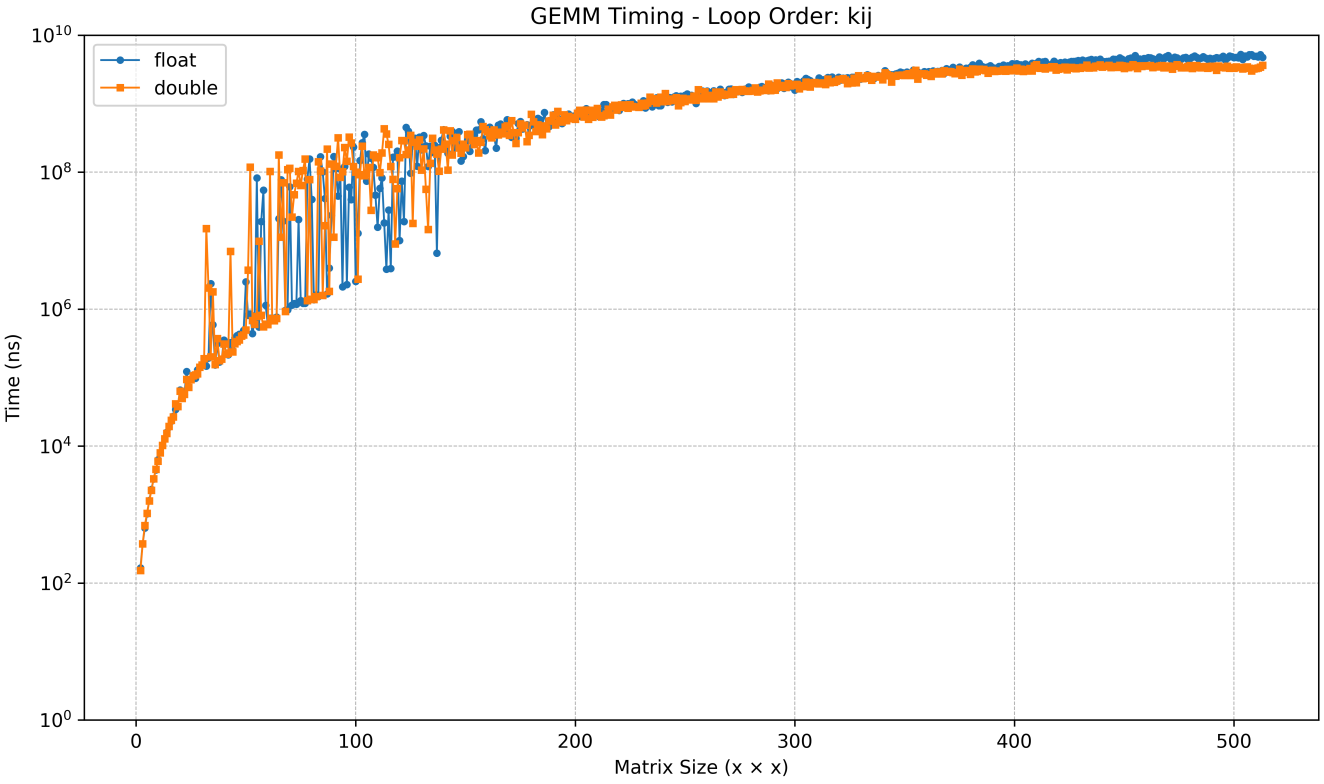
-O0 Permutation JKI



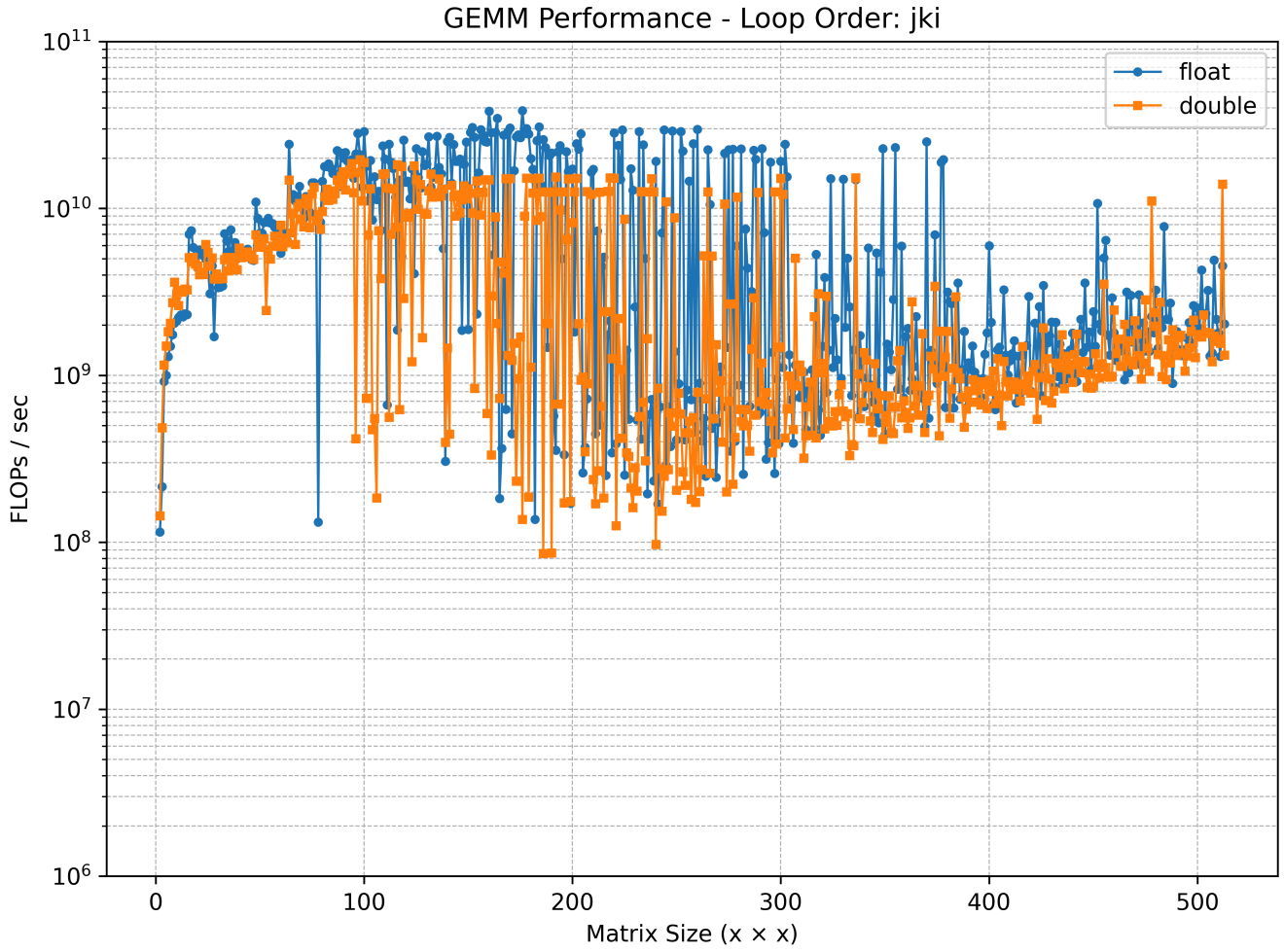


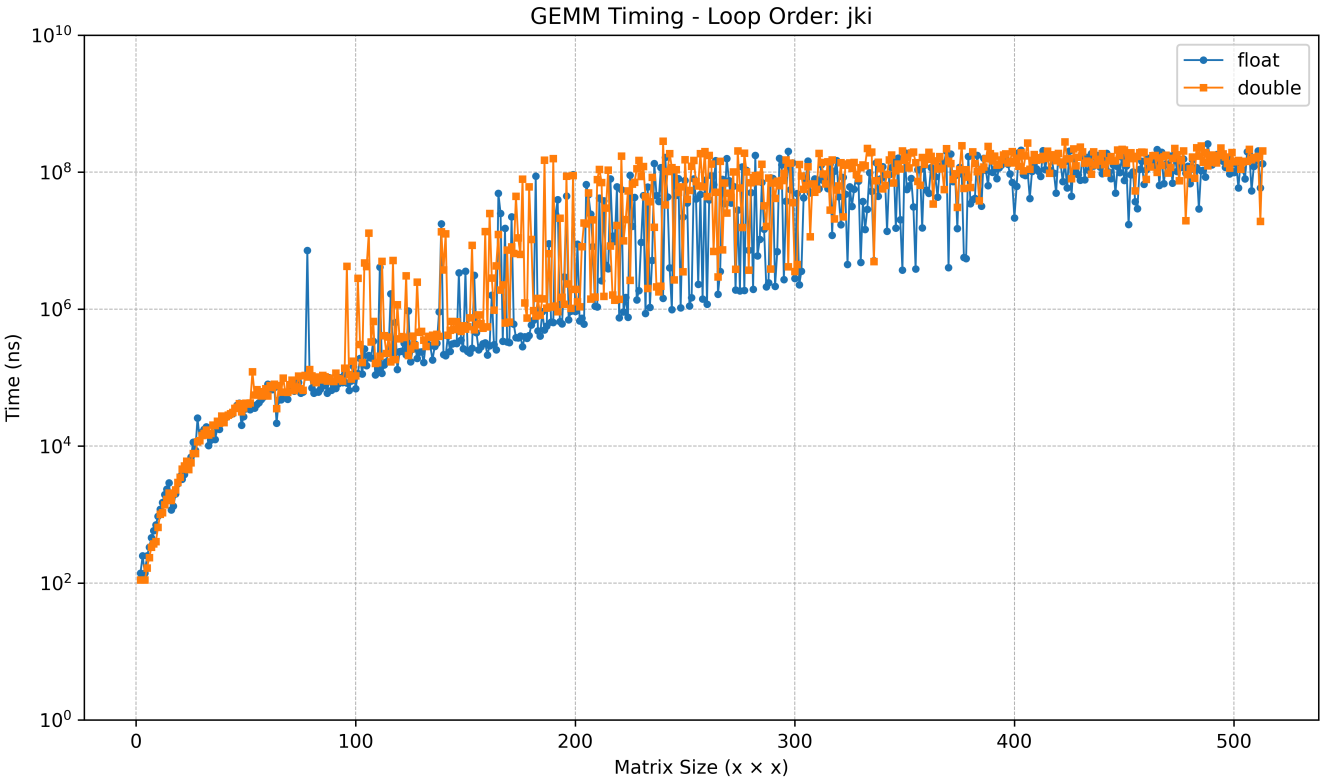
-O0 Permutation KIJ



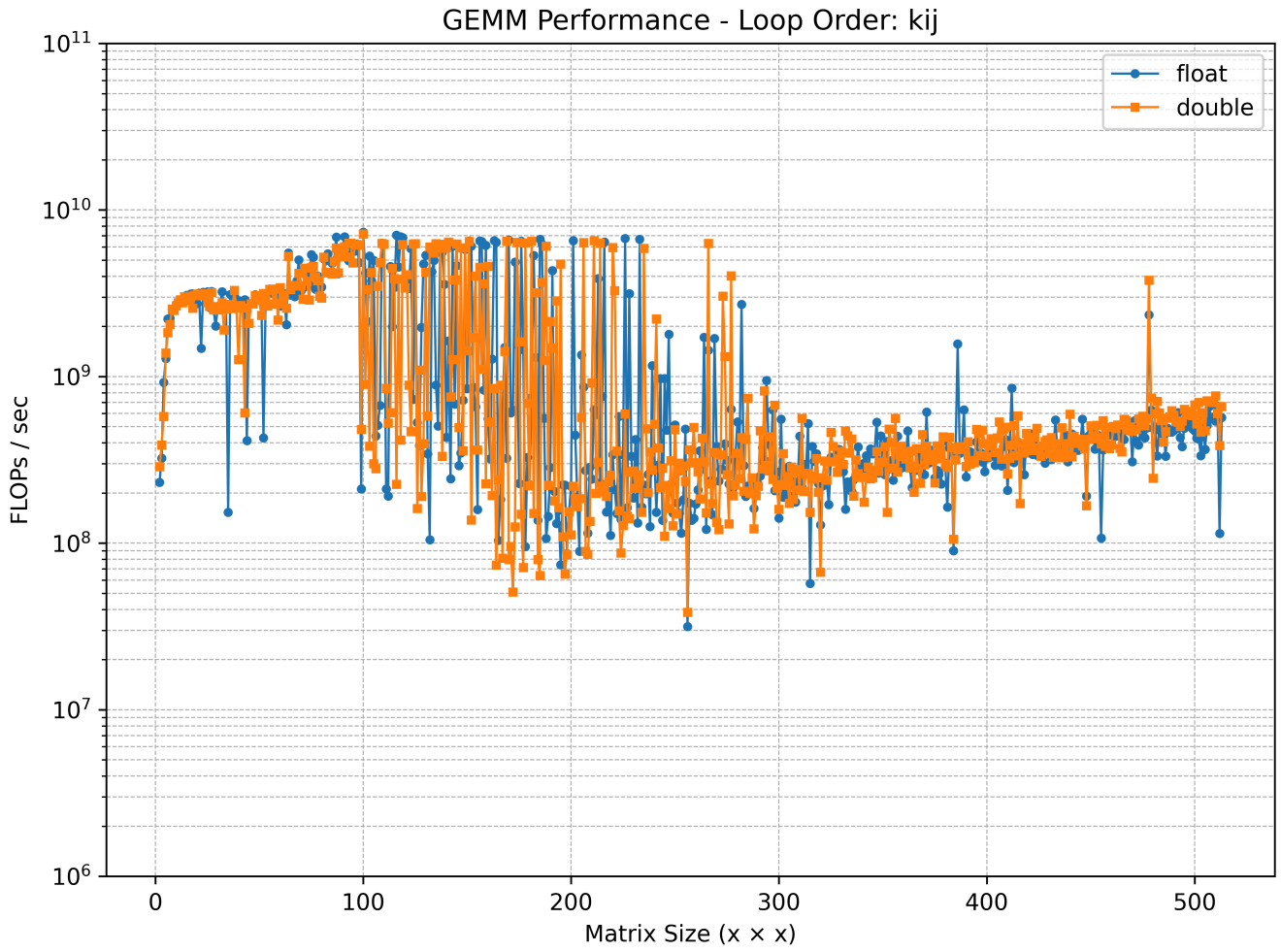


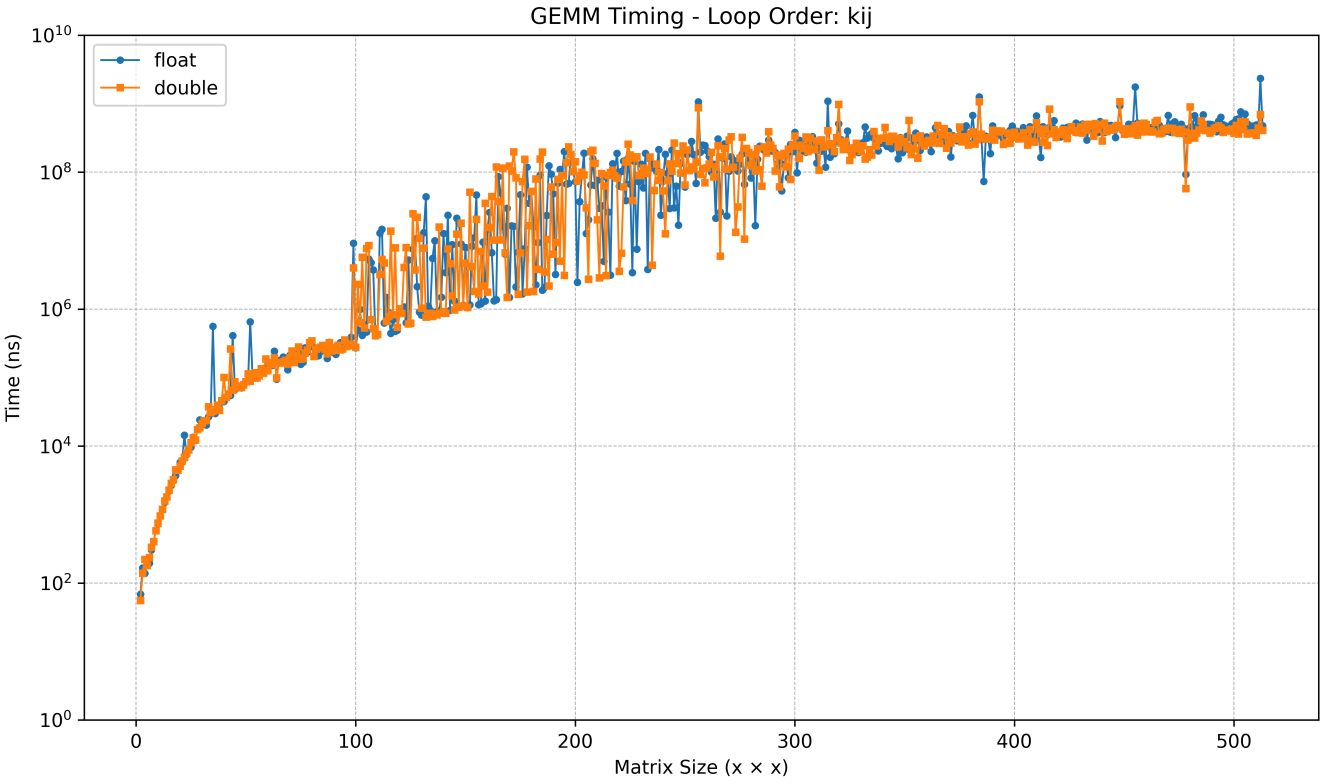
-O3 Permutation JKI





-O3 Permutation KIJ





The compiler `-O3` optimized $\{kij\}$ permutation appears more "time-stable" than the $\{jki\}$ permutation, as in, for large matrices, it performs more consistently at the same speed.

However, permutation $\{jki\}$ shows computes at a greater FLOP rate than $\{kij\}$ for `-O3` optimized compiled code - particularly for floats. Both are slightly faster than the `-O0` compiler setting and the variance between all cases is much smaller than I expected.

Problem 3

✓ **matrix_class.hpp**

problem3.zip

```cpp
// matrix_class.hpp
#ifndef AM583_SP25_HW4_P3_MATRIX_CLASS_HPP
#define AM583_SP25_HW4_P3_MATRIX_CLASS_HPP

#include <vector>
#include <stdexcept>
#include <cmath>
#include <iostream>
#include <iomanip>

// Row-major matrix class template
template <typename T>
class Matrix
{
private:
    std::vector<T> data_;
    int rows_;
    int cols_;

public:
    // Constructors
    Matrix(int rows, int cols, T val = T())
        : data_(rows * cols, val), rows_(rows), cols_(cols) {}

    // matrix_class.hpp (inside class Matrix<T>)
    Matrix(std::initializer_list<std::vector<T>> init)
    {
        rows_ = init.size();
        cols_ = init.begin()->size();
        data_.resize(rows_ * cols_);
        int i = 0;
```

```
        for (const auto &row : init)
        {
            if (row.size() != cols_)
                throw std::invalid_argument("Inconsistent row size in
initializer list.");
            for (int j = 0; j < cols_; ++j)
            {
                data_[i * cols_ + j] = row[j];
            }
            ++i;
        }
    }

    // Accessors
    int rows() const { return rows_; }
    int cols() const { return cols_; }

    T &operator()(int i, int j)
    {
        return data_[i * cols_ + j];
    }

    const T &operator()(int i, int j) const
    {
        return data_[i * cols_ + j];
    }

    // Transpose
    Matrix<T> transpose() const
    {
        Matrix<T> result(cols_, rows_);
        for (int i = 0; i < rows_; ++i)
        {
            for (int j = 0; j < cols_; ++j)
            {
                result(j, i) = (*this)(i, j);
            }
        }
        return result;
    }

    // Infinity norm
    T infinityNorm() const
    {
        T maxSum = T();
        for (int i = 0; i < rows_; ++i)
```

```
        {
            T rowSum = T();
            for (int j = 0; j < cols_; ++j)
            {
                rowSum += std::abs((*this)(i, j));
            }
            if (rowSum > maxSum)
            {
                maxSum = rowSum;
            }
        }
        return maxSum;
    }

    // Matrix addition
    Matrix<T> operator+(const Matrix<T> &other) const
    {
        if (rows_ != other.rows_ || cols_ != other.cols_)
        {
            throw std::invalid_argument("Matrix dimensions must match for
addition.");
        }
        Matrix<T> result(rows_, cols_);
        for (int i = 0; i < rows_ * cols_; ++i)
        {
            result.data_[i] = data_[i] + other.data_[i];
        }
        return result;
    }

    // Matrix multiplication
    Matrix<T> operator*(const Matrix<T> &other) const
    {
        if (cols_ != other.rows_)
        {
            throw std::invalid_argument("Matrix dimensions must be
compatible for multiplication.");
        }
        Matrix<T> result(rows_, other.cols_);
        for (int i = 0; i < rows_; ++i)
        {
            for (int j = 0; j < other.cols_; ++j)
            {
                T sum = T();
                for (int k = 0; k < cols_; ++k)
                {
```

```
                    sum += (*this)(i, k) * other(k, j);
                }
                result(i, j) = sum;
            }
        }
        return result;
    }

    // Print matrix
    void print(int width = 10) const
    {
        for (int i = 0; i < rows_; ++i)
        {
            for (int j = 0; j < cols_; ++j)
            {
                std::cout << std::setw(width) << (*this)(i, j);
            }
            std::cout << "\n";
        }
    }
};

#endif // matrix_class.hpp
```

✓ **EOF**

Problem 4

Find the **minimum distance from the origin to a surface** defined by:

$$xy + 2xz = 5\sqrt{5}, \quad (x, y, z) \in \mathbb{R}^3$$

Let the **distance squared** from a point (x, y, z) on the surface to the origin be:

$$D^2 = x^2 + y^2 + z^2$$

We minimize this subject to the constraint:

$$g(x, y, z) = xy + 2xz - 5\sqrt{5} = 0$$

Step 1: Use Lagrange Multipliers

We introduce a Lagrange multiplier λ, and define:

$\mathcal{L}(x, y, z, \lambda) = x^2 + y^2 + z^2 - \lambda(xy + 2xz - 5\sqrt{5})$

Step 2: Compute the gradient

Take partial derivatives and set equal to zero:

$$\frac{\partial \mathcal{L}}{\partial x} = 2x - \lambda(y + 2z) = 0 \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial y} = 2y - \lambda x = 0 \tag{2}$$

$$\frac{\partial \mathcal{L}}{\partial z} = 2z - 2\lambda x = 0 \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -(xy + 2xz - 5\sqrt{5}) = 0 \tag{4}$$

Step 3: Solve the system

From (2):

$\lambda = \frac{2y}{x}$

From (3):

$\lambda = \frac{z}{x}$

Equating:

$$\frac{2y}{x} = \frac{z}{x} \Rightarrow z = 2y \tag{5}$$

Now from (1):

$2x = \lambda(y + 2z)$

Substitute $\lambda = \frac{2y}{x}$, $z = 2y$:

$2x = \frac{2y}{x}(y + 2(2y)) = \frac{2y}{x}(y + 4y) = \frac{2y}{x}(5y) = \frac{10y^2}{x}$

Multiply both sides by x:

$$2x^2 = 10y^2 \Rightarrow x^2 = 5y^2 \Rightarrow x = \pm\sqrt{5}y \tag{6}$$

Step 4: Plug into constraint

Use the original constraint:

$xy + 2xz = 5\sqrt{5}$

Use (5): $z = 2y$, and (6): $x = \sqrt{5}y$

$\text{LHS} = (\sqrt{5}y)(y) + 2(\sqrt{5}y)(2y) = \sqrt{5}y^2 + 4\sqrt{5}y^2 = 5\sqrt{5}y^2$

So:

$5\sqrt{5}y^2 = 5\sqrt{5} \Rightarrow y^2 = 1 \Rightarrow y = \pm 1$

Then:

- If $y = 1$: $x = \pm\sqrt{5}$, $z = 2$
- If $y = -1$: $x = \mp\sqrt{5}$, $z = -2$

So candidate points are:

$(\sqrt{5}, 1, 2), \quad (-\sqrt{5}, -1, -2)$

Step 5: Evaluate distance squared

$D^2 = x^2 + y^2 + z^2 = 5 + 1 + 4 = 10 \Rightarrow D = \sqrt{10}$

Final Answer:

(a) The points on the surface closest to the origin are:

$(\sqrt{5}, 1, 2) \quad \text{and} \quad (-\sqrt{5}, -1, -2)$

(b) The minimum distance is:

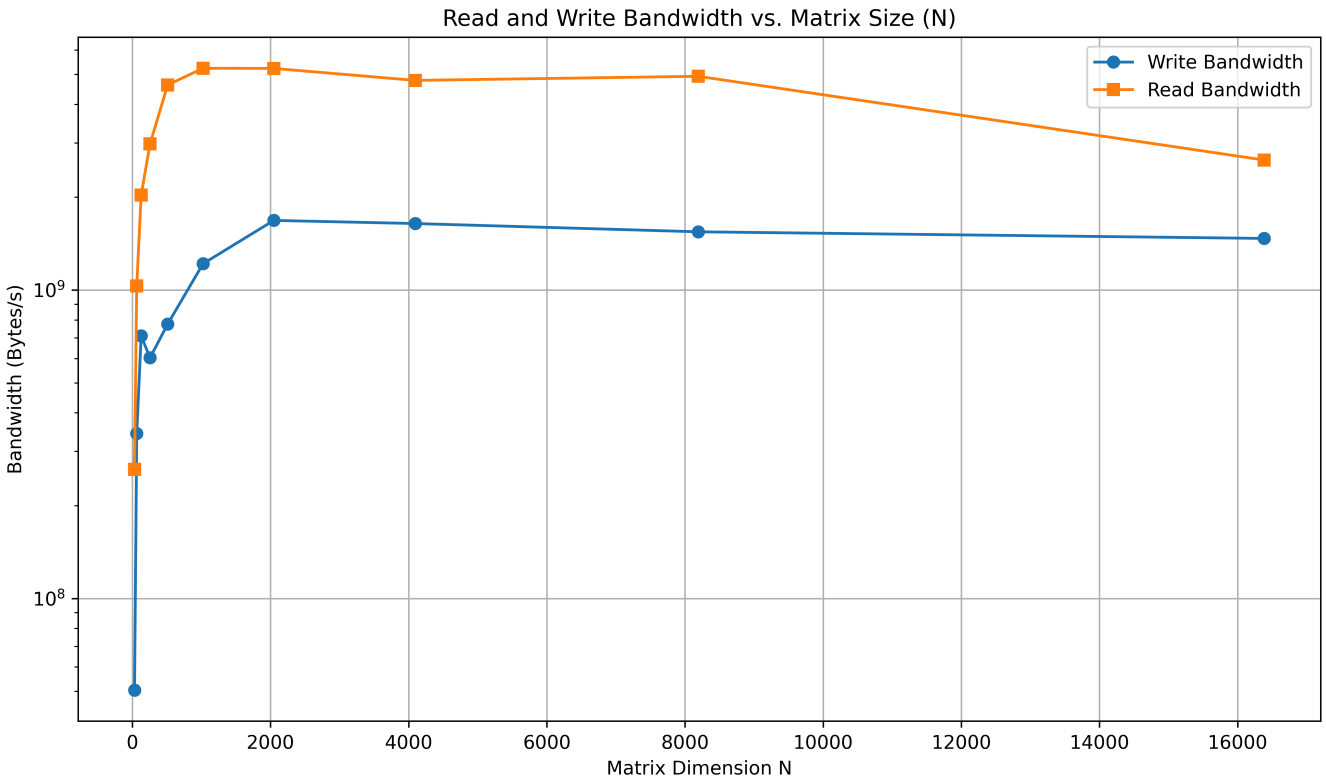
$\boxed{\sqrt{10}}$

Problem 5

Write a C++ function that writes type double square matrices in column major order to a file in binary. Measure the time required to complete the write for matrices of dimension 32, 64, 128, ..., 16384.

Write a C++ function that reads binary matrices from file to type double matrices in memory. Measure the time required to complete each read for the same dimensions.

Make a single plot of the read and write measurements with the bandwidth (bytes per second) on the y-axis and the dimension on the x-axis.



Problem 6

Write C++ functions for the given function declarations that perform row and column swap operations on a type double matrix stored in a file in column major index order. Test the swapping capabilities for correctness. Put the functions you write in the file file_swaps.hpp.

Conduct a performance test fore square matrix dimensions 16, 32, 64, 128, ..., 8192, measuring the time required to conduct file-based row and column swaps separately. Let each operation be measured ntrial times, ntrial >= 3. Make a single plot of the row and column swap average times on the y-axis ($\log_{10}(time)$) and the problem dimension on the x-axis. Submit your header file file_swaps.hpp and plot.

✓ **file_swaps.hpp**

file_swaps.hpp