# William Daniels AMS 326 homework two writeup

March 18, 2023

All of my source files can be found in this github repositiory:
https://github.com/William-J-Daniels/DanielsAms326.git

A detailed README on how the repositiory is used is there. The commit that represents the state of the repository upon submission of the assignment is the one at 6:13 AM on March 18 (I can't give the hash before I commit).

## Problem One

### Description

This problem is to find the product of large matrices. In particular, we are asked to find the product of two $2^{10}$ by $2^{10}$ matrices of numbers uniformly distributed between -1 and 1 then the product of two $2^{12}$ by $2^{12}$ matrices populated the same way using both the naive algoritm and Strassen's algorithm.

### Algorithms

#### Naive

The naive matrix multiplication algorithm uses the mathematical definition of matrix multiplication: given two matrices $A$ and $B$, their product $C$ can be found according to $C_i, j = \sum_{k=1}^{n} A_{ik} B_{kj}$.

My threaded implimentation of this algorithm is in `LinearAlgebra/include/matrix.h` and spans the methods `naive_mult` and `mult_helper`.

#### Strassen's

Strassen's algorithm involves breaking the matrices up into four equal slices (note that this reqires that the matrices are square and of a size that is a power of 2). This gives us

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

This grants us

$$C_{11}=A_{11}B_{11}+A_{12}B_{21}$$
$$C_{12}=A_{11}B_{12}+A_{12}B_{22}$$
$$C_{21}=A_{21}B_{11}+A_{22}B_{21}$$
$$C_{22}=A_{21}B_{12}+A_{22}B_{22}$$

In and of itself, this is not enlightening. However, this system can be manipulated to yeild

$$C_{11}=M_1+M_4-M_5+M_7$$
$$C_{12}=M_3+M_5$$
$$C_{21}=M_2+M_4$$
$$C_{22}=M_1-M_2+M_3+M_6$$

where

$$M_1=(S_{11}+A_{22})(B_{11}+B_{22})$$
$$M_2=(A_{21}+A_{22})(B_{11})$$
$$M_3=(A_{11})(B_{12}-B_{22})$$
$$M_4=(A_{22})(B_{21}-B_{11})$$
$$M_5=(A_{11}+A_{12})(B_{22})$$
$$M_6=(A_{21}-A_{11})(B_{11}+B_{12})$$
$$M_7=(A_{12}-A_{22})(B_{21}+B_{22})$$

This has the effect of having to do 7 instead of 8 matrix multiplications at the expense of a greater number of matrix additions. A consequence of this is the existance of a crossover point at which Strassen's method becomes more efficient than the naive method. Since Strassen's method can be applied recursively, this crossover point is where we want our base case to be. Another side effect of Strassen's method is the creation of auxilliary matrices; this reduces the space efficiency of the method.

My implimentation of this algorithm can be found at `LinearAlgebra/include/matrix.h` in the methods `overload *` and `strassen_mult`. I choose $n=512$ as my crossover point per the findings of Huang, Smith, Henry, and van de Geijn (13 Nov 2016) and I create 21 auxilliary matrices of size $n/2$ per level of recursion.

## Results

The naive method took about 7 seconds for $n=2^{10}$ and 24 *minutes* for $n=2^{12}$. Strassen's method took about 6 seconds for $n=2^{10}$ and 6 minutes for $n=2^{12}$.

With the naive algorithm, we used about $2^{30}$ = 1 GFLOP and $2^{36}$ = 69 GFLOPs for the two sizes.

One level of Strassen's algoritms sees 18 additions of $n/2$ matrices and 7 multiplications of $n/2$ matrices. So we have $\left(\frac{n}{2}\right)^2\cdot 18+\left(\frac{n}{4}\right)^2\cdot 18\cdot 4+\left(\frac{n}{8}\right)^2\cdot 18\cdot 4\cdot 4+7^3\cdot\left(\frac{n}{16}\right)^3$, which for $2^{10}$ and $2^{12}$ means we use 104 MFLOPs and 6 GFLOPs respectively.

## Problem Two

The problem is to approximate the Airy function for different values of $x$ to at least 3 decimal places of precision.

### Algorithms

I choose to use the midpoint method for this assignment because it was the simpliest to implement. It approximates the integral as the sum of the area of some number of boxes whose heights are determined by the value of the integrand at the midpoint of the division. It is also threaded.

My implementation header is `Integration/include/midpoint.h` and implementation is `Integration/src/midpoint.cpp`. This class inherits from the abstract class `Integrator`, detailed in `Integration/include/integrator.h` and `Integration/src/integrator.cpp`, so I can add the more sophisticated methods with ease.

### Results

The following table details my results. They are accurate to `1.0e-8`.

| x | -3 | -2 | -1 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|
| Ai(x) given | -0.378814 | 0.227407 | 0.535561 | 0.355028 | 0.135292 | 0.034924 | 0.006591 |
| Ai(x) found | -0.37883 | 0.227325 | 0.535412 | 0.354816 | 0.135022 | 0.0345991 | 0.0062178 |