# MATH170A HW1

Tyler Barbero

Due: 12 October 2020

## Problem 1:

*Describe 3 examples or applications that use large matrices and need a computer to be solved.*

1. This is an ODE(PDE) problem but different application

1. Numerical Weather Prediction: The primitive equations are a set of non-linear differential equations that describe the thermodynamic properties (e.g., temperature, viscosity, density, compressibility) and motion of fluid (momentum force-balance equations) in the atmosphere. These equations are solved numerically with input data taken from in-situ ground, satellite-based, and upper-air from weather balloons and flight missions. Millions of data points are input into the equations and the model runs. Current resolution and computing power can only run models with fair accuracy 6 days in advance, with more error as time goes on (NOAA Numerical Weather Prediction).

2. One of the solutions

2. Least Squares Problems: (Idea from Math 102 notes) We have $Ax = b$, such that there is no unique solution. We can find a solution that is closest to b. Let A be a $mxn$ matrix of rank n and then $Ax = b$ can be rewritten into the normal equation $A^TAx = A^Tb$, and thus $\hat{x} = (A^TA)^{-1}A^Tb$, the unique least squares solution of the system $Ax = b$. I guess this equation is derived analytically but we can solve this using a computer when matrices are exceptionally big and doing it by hand would be unfeasible.

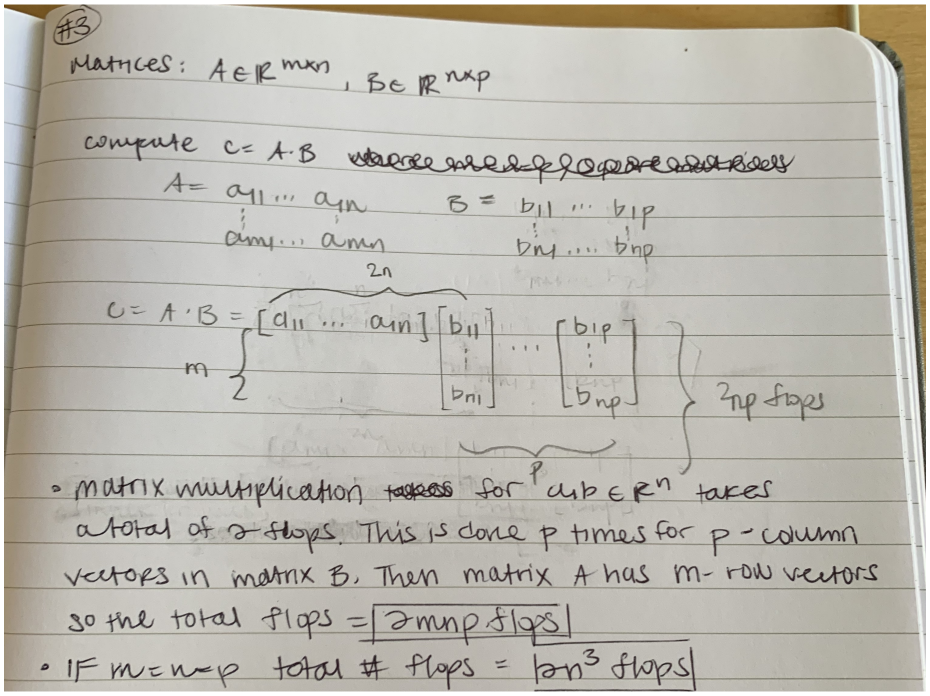
3. I think this is correct because the Equations that govern Ocean Waves are essentially a system of ODEs that are approximated ( so still ODEs but this is a different application ).

3. Iterative methods (Newton's Method): From my understanding iterative methods is an approximation where one inputs a guess and the output from this guess is re-input into the same equation, and as the amount of iterations increases towards infinity, the approximation becomes closer to the actual answer. I did this in a course called SIO 111 (Introduction to Ocean Waves) where I input the results from a first-order Taylor expansion into a second-order Taylor expansion to receive a more accurate answer. I imagine this could be done on a computer numerous times with large matrices to obtain an accurate solution (Taylor-Series).

## Problem 2:

There are two flops for iteration of the loop, one from the addition and one from the squaring of v(i). The loop is run n times, so the total number of flops is 2n. For each real element i in the vector v of dimension n, the i-th component is squared and 0.3 is added to it.

## Problem 3:



#3

Matrices: $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$

compute $C = A \cdot B$ ~~[illegible]~~

$A = \begin{matrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{matrix}$ $B = \begin{matrix} b_{11} & \cdots & b_{1p} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{np} \end{matrix}$

$$C = A \cdot B = m \left\{ \overbrace{\begin{bmatrix} a_{11} & \cdots & a_{1n} \end{bmatrix}}^{2n} \underbrace{\begin{bmatrix} b_{11} \\ \vdots \\ b_{n1} \end{bmatrix} \cdots \begin{bmatrix} b_{1p} \\ \vdots \\ b_{np} \end{bmatrix}}_{p} \right\} 2np \text{ flops}$$

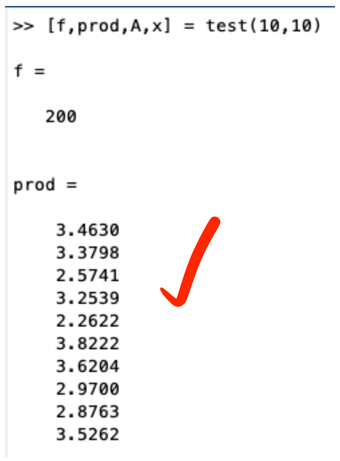
- matrix multiplication ~~takes~~ for $a, b \in \mathbb{R}^n$ takes a total of 2 flops. This is done p times for p-column vectors in matrix B. Then matrix A has m-row vectors so the total flops = $\boxed{2mnp \text{ flops}}$
- If $m = n = p$ total # flops = $\boxed{2n^3 \text{ flops}}$

## Problem 4: Matrix * Vector multiplication

```
% write a function that does matrix multiplication for a matrix A of size
% mxn and a vector of size nx1. First check the sizes of the inputs are
% correct then do the multiplication uses two nested for loops.
```

Prod is the Product of "manually" doing the inner product of row of A with x. This matches with the matrix multiplication A*x when I call A*x outside the program.

```matlab
function [f,prod,A,x] = test(m,n)
A = rand(m,n);
x = rand(n,1);
if size(A,2)~=size(x,1)
    return % stops script
    disp('Matrices are not the same size')
end
f=0;
prod=NaN(m,1);
for i=1:m
    tmp=0;
    for j=1:n
       tmp = tmp + A(i,j).*x(j); % inner prod for each row of A*x
       f = f + 2;
    end
    prod(i,1) = tmp;
end
clearvars tmp i j m n
end
```



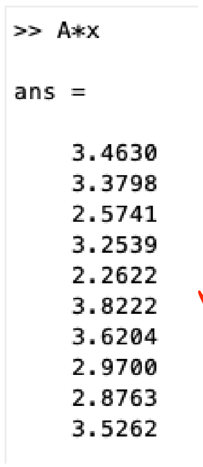
```
>> [f,prod,A,x] = test(10,10)

f =

   200


prod =

    3.4630
    3.3798
    2.5741
    3.2539
    2.2622
    3.8222
    3.6204
    2.9700
    2.8763
    3.5262
```



```
>> A*x

ans =

    3.4630
    3.3798
    2.5741
    3.2539
    2.2622
    3.8222
    3.6204
    2.9700
    2.8763
    3.5262
```

Inner product function and matrix multiplication return the same result.

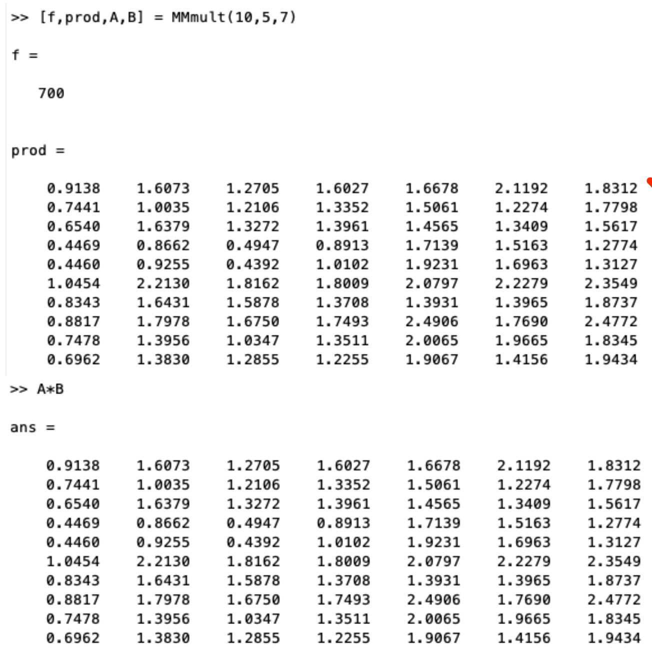
## Problem 5: Matrix * Matrix multiplication

I ask for inputs in the function rather than asking for the entire matrices, I think this still should be fine because I end up defining the matrices inside the program using the function inputs

```matlab
% matrix multiplication function
function [f,prod,A,B] = MMmult(m,n,p)
```

```
A = rand(m,n)           ✓
B = rand(n,p)

if size(A,2)~=size(B,1);return;disp('Error, matrix sizes');end   ✓
f=0;
for i=1:m
    for k=1:p           ✓
        tmp=0;
        for j=1:n
            tmp = tmp + A(i,j)*B(j,k);   ✓
            f=f+2;
            % each tmp is a scalar for inner product of each rowxcolumn
        end
            prod(i,k) = tmp;   ✓
    end
end
```



```
>> [f,prod,A,B] = MMmult(10,5,7)

f =

   700


prod =

    0.9138    1.6073    1.2705    1.6027    1.6678    2.1192    1.8312   ✓
    0.7441    1.0035    1.2106    1.3352    1.5061    1.2274    1.7798
    0.6540    1.6379    1.3272    1.3961    1.4565    1.3409    1.5617
    0.4469    0.8662    0.4947    0.8913    1.7139    1.5163    1.2774
    0.4460    0.9255    0.4392    1.0102    1.9231    1.6963    1.3127
    1.0454    2.2130    1.8162    1.8009    2.0797    2.2279    2.3549
    0.8343    1.6431    1.5878    1.3708    1.3931    1.3965    1.8737
    0.8817    1.7978    1.6750    1.7493    2.4906    1.7690    2.4772
    0.7478    1.3956    1.0347    1.3511    2.0065    1.9665    1.8345
    0.6962    1.3830    1.2855    1.2255    1.9067    1.4156    1.9434

>> A*B

ans =

    0.9138    1.6073    1.2705    1.6027    1.6678    2.1192    1.8312   ✓
    0.7441    1.0035    1.2106    1.3352    1.5061    1.2274    1.7798
    0.6540    1.6379    1.3272    1.3961    1.4565    1.3409    1.5617
    0.4469    0.8662    0.4947    0.8913    1.7139    1.5163    1.2774
    0.4460    0.9255    0.4392    1.0102    1.9231    1.6963    1.3127
    1.0454    2.2130    1.8162    1.8009    2.0797    2.2279    2.3549
    0.8343    1.6431    1.5878    1.3708    1.3931    1.3965    1.8737
    0.8817    1.7978    1.6750    1.7493    2.4906    1.7690    2.4772
    0.7478    1.3956    1.0347    1.3511    2.0065    1.9665    1.8345
    0.6962    1.3830    1.2855    1.2255    1.9067    1.4156    1.9434
```

Returns same result. ⟶