Andrew Garwood
Written HW 3

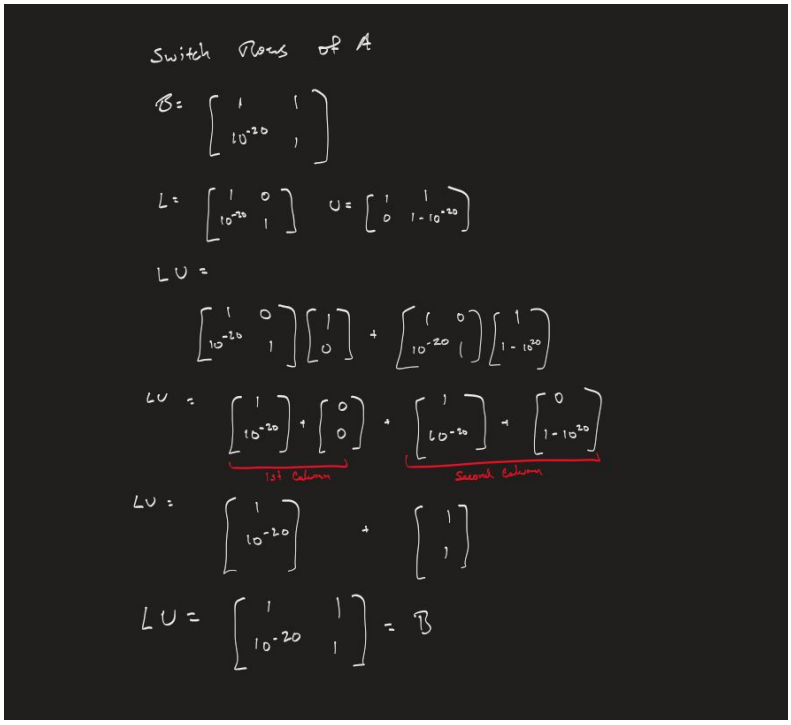
Problem 1
---

a)
% abs(a - b) = 31072. I think this is a significant error; These values should be the same, but
% because of rounding errors. a is around 30 percent off its "expected"
% value.

b)
cond(A) = 2.618033988749895
Hmmm very reasonable

c)
```
% 1c LU decomp (no pivoting)
L = [1    , 0;
    10e20, 1];
U = [10e-20, 1;
    0     , 1 - 10e20];
% LU =
% [1    , 0; * [10e-20, 0]' + [1    , 0; * [1, 1 - 10e20]'
%  10e20, 1]                  10e20, 1 ]
% LU =
% [10e-20, 1]' + ([1, 10e20]' + [0, 1 -10e20]')
% LU =
% [10e-20, 1]' + ([1, 1]')
% LU =
% [10e-20, 1; 1, 1]
%
% This is what I got:
%
%   1.0e+02 *
%
%   0.000000000000000   0.010000000000000
%   1.000000000000000                   0
%
% I do not believe that this is close to A. This demonstrates that
% catastrophic cancellation can drastically change the elements of the
% matrix which will ruin calculations we do when solving problems
%
```

d)



% upon inspection, LU(2, 1) = 1e-19
% So yes this is pretty damn close. we seemed to have avoided the
% catastrophic cancellation by not having 10e20 in our decomposition, this
% eliminated the error that would have arisen when operating with it.

e)
% We end up with the same matrices from part d for both decompositions,
% this demonstrates that the permutation matrix P rearranges the rows such
% that we (hopefully) avoid catastrophic cancellation.

Problem 2
---

a)

```
A = readmatrix('example_matrix.csv');
```

b)

```
r1 = zeros(1, 100);
tic
for i = 1:100
    b = rand(3000, 1);
    x = A\b;
    r = A * x - b;
    r1(i) = max(abs(r));
end
t_b = toc
r1_average = mean(r1)
```

% pre part e) Elapsed time is 29.775509 seconds.
% t_b = 29.775509;

c)

```
% 2c now use LU decomp
r2 = zeros(1, 100);
tic
[L, U, P] = lu(A);
for i = 1:100
    b = rand(3000, 1);
    y = L\(P * b);
    x = U\y;
    r = A * x - b;
    r2 = max(abs(r));
end
t_c = toc
r2_average = mean(r2)
```

% pre part e) t_c = 2.054032800000000

d)

```
% % 2d Ax = b   x = A_inv * b
r3 = zeros(1, 100);
tic
A_inv = inv(A);
for i = 1:100
    b = rand(3000, 1);
```

```
    x = A_inv * b;
    r = A * x - b;
    r3(i) = max(abs(r));
end
t_d = toc
r3_average = mean(r3)

% pre part e) t_d = 0.846639800000000

e)
% 2e
% compare accuracy of algo.
%          time          average residual
%----------------------------------------------
% GE 30.623165000000000  7.865930129469233e-15
% LU  2.587481200000000  3.119726699196690e-14
% INV 1.226056100000000  1.845068542394302e-13

% 2g
% For speed, the inverse method is the fastest. LU is somewhat close, it's
% about twice as long. GE is not close.

% 2h
% for accuracy only, I would choose GE because it has the smallest average
% residual. LU is one magnitude (10 times) larger than GE, Inverse is 100
% times larger than GE.

% 2i
% If I wanted to balance speed and accuracy, I'd choose LU decomposition.

% 2j
% Yes, this is important and will affect the results. LU decomposition has
% O(n^3) runtime and Matrix inversion has O(n^3) runtime. So if we increase
% the size complexity by a lot then it will have more drastic effects on
% performance.

% 2k
% oh dear what to say. Perhaps we could instead time each x = A\b operation
% rather than the whole loop. And then sum the results at the end. I think
% this will be more precise. Why you ask? well my brilliant intuition tells
% me so. Well. My thinking is that there is more opportunity for background
% processes to affect our measurements if we have a single measurement for
% the for loops.
```

Thank u for bearing with me.