CS / MATH 4334 : Numerical Analysis
Homework Assignment 3

Matthew McMillian
mgm160130@utdallas.edu

October 17, 2018

MatLab Problems

Problem 1 : gaelpp.m

```
function [A,P] = gaelpp(A)

%Gaussian elimination
%inputs:
%nXn matrix A, and and nX1 vector b
%outputs:
%nXn matrix m of multipliers
%nXn upper triangular matrix A
%nX1 vector b

%get n
[n,n] = size(A);


% Create P
P = eye(n);

%set up matrix of zeros that will store multipliers
M = zeros(n,n);

% This initially swaps the top row by finding the maximum value'd
% row index inside this row and then swapping it with the initial row
max = 0;
pos = 1;
for p = 1:n
     if abs(A(p,1)) > max
         max = abs(A(p,1));
         pos = p;
     end
end


A([1 pos],:) = A([pos 1], :);
P([1 pos],:) = P([pos 1], :);

%Gaussian elimination
for j = 1:n-1 %j is pivot row

     if A(j,j) == 0 %avoid div. by 0
             break
```

```
        end

        % If the value at row j is less than the value
        % at row j+1, swap the rows and update A, P, and M
        if abs(A(j, j)) < abs(A(j+1, j))
            A([j j+1],:) = A([j+1 j], :);
            P([j j+1],:) = P([j+1 j], :);
            M([j j+1],:) = M([j+1 j], :);
         end

        for i = j+1:n %elim. col. j from row i = j+1 to i = n

            M(i,j) = A(i,j)/A(j,j); %multiplier to elim. A(i,j), store in matrix m

            for k = j+1:n % add mult of row j to row i
                A(i,k) = A(i,k) - M(i,j)*A(j,k);
            end
        end
end

% Correctly sets the pivots in A to return properly (modified matrix)
for y=2:n
    for x=1:y-1
        A(y,x) = M(y,x);
    end
end
```

Problem 1 : gaelppscript.m

```
format long e

% Clears console for clean testing
clc

% Given A Matrix
A = [3.03 -12.1 14;
     -3.03 12.1 -7;
     6.11 -14.2 21];

% Vector of values we want to solve for
b = [-119; 120; -139;];

```

```
% Calling Gaussian Elimination with Partial Pivoting
[A, P] = gaelpp(A);

A
P

% This checks to make sure that our PA=LU worked
% Forms L Matrix
L = [1 0 0
     A(2,1) 1 0
     A(3,1) A(3,2) 1];
% Forms U Matrix
U = [A(1,1) A(1,2) A(1,3)
     0    A(2,2)  A(2,3)
     0    0    A(3,3)];

% Forward Sub and Backward Sub Output
y = forsub(L,b)
s = backsub(U, y)
```

Problem 1 : gaelscript.m

```
format long e

%matrix for circuit problem
A = [3.03 -12.1 14;
     -3.03 12.1 -7;
     6.11 -14.2 21];

%rhs for circuit problem - ' makes it a column
b = [-119; 120; -139;];


%now, call function that does GE on A and b to get U and c (
    modified b)
[U,c,M] = gael(A,b);

U

c

%call the backsub function to find solution x
x = backsub(U,c)

```

```
%check to see if original Ax equals original b
A*x - b

fprintf("We can tell from our answer that partial pivoting helps avoid swamping errors that regular Gaussian Elimination cannot deal with.\n")
```

>> gaelppscript.m + gaelscript.m

(In gaelppscript.m)
A =
6.110000000000000e+00 -1.420000000000000e+01 2.100000000000000e+01
-4.959083469721767e-01 5.058101472995091e+00 3.414075286415711e+00
4.959083469721767e-01 -1.000000000000000e+00 7.000000000000000e+00

P =
0 0 1
0 1 0
1 0 0

y (forsub) =
-1.190000000000000e+02
6.098690671031097e+01
-1.899999999999999e+01

s (backsub) =
2.213234104513831e+01
1.388933829477430e+01
-2.714285714285713e+00

(In gaelscript.m)
x =
Inf
Inf
-1.396257416704701e+01

ans =
NaN
NaN
NaN

We can tell from our answer that partial pivoting helps avoid swamping errors that regular Gaussian Elimination cannot deal with.

Problem 2 : hw2.m

```
format long e

% Clear console for easy testing
clc

% Setup the Vandermonde matrix of size n, as well as the
    solution vector
n = 10;
A = zeros(n,n);
x = ones(n,1);

% Loop creating of the Vandermonde matrix
for i=1:n
    for j=1:n
        A(i,j) = i^(j-1);
    end
end

% Performing GEPP via MatLab built-in
A;
b = A*x;
xa = A\b;

% Calculating the errors based upon the actual solutiopn
% x, and the approx solution xa
relBkwdErr = (norm(A*x - A*xa))/norm(b)
condA = cond(A)
boundOnFwrdError = condA*relBkwdErr
relFwrdErr = (norm(x - xa))/norm(x)

fprintf("Since we are using double percision, our answer is
    only accurate to 16 digits\n")
```

>> hw2.m

relBkwdErr =
6.329171661699566e-17

condA =
2.106245945721575e+12

boundOnFwrdError =
1.333079215223059e-04

relFwrdErr =
6.074920350417704e-06

Since we are using double percision, our answer is only accurate to 10^{16} - cond(A) digits = 16 - $\log_{10}$(2.106245945721575e+12) $\approx$ 3.67, so we can trust $\approx$ 3 digits.

Problem 3 : newt3d.m

```
format long e

% Clear console for easy testing
clc

%relative error tolerance
TOL = .000001;

%initial guess; x is "current" iterate
x = [1; 1; 1;];

%store as previous iterate xp
xp = x;

%call function to generate right-hand-side, -f
rhs = genrhs(x);

%call function to generate matrix, DF
DF = genmatrix(x);

%Use MATLAB function to do PA=LU factorization
[L,U,P] = lu(DF);

%Use our functions in elearning, forsub and backsub, as
    appropriate
y = forsub(L, P*rhs);
s = backsub(U, y);

% Calculating the next iterate
x = xp + s;

%x is the current iterate
%s is the difference between current and previous iterate
while norm(s)/norm(x) >= TOL

    %store previous iterate
    xp = x;

    %generate right-hand-side, -f
    rhs = genrhs(x);

    %generate matrix, DF
```

```
    DF = genmatrix(x);

    %PA=LU factorization
    [L,U,P] = lu(DF);

    %Use our functions forsub and backsub as appropriate
    y = forsub(L, P*rhs);
    s = backsub(U, y);

    x = xp + s;

end

%display last iterate and relerr estimate
x

norm(s)/norm(x)
```

Problem 3 : genmatrix.m

```
function [A] = genmatrix(x)
% Generates a matrix of the derivatives of the
% log-transformed equation given in the HW.
% [Overview] Takes in a column vector X and returns a matrix A
% (the jacobian / gradient) the original function
% with the values of x plugged into it.
A = [1/x(1) 1 1/(10-x(3)*1);
     1/x(1) 2 2/(12-x(3)*2);
     1/x(1) 3 3/(15-x(3)*3);];
end
```

Problem 3 : genrhs.m

```
function [f] = genrhs(x)
% Generates the right-hand-side of the
% log-transformed equation given in the HW.
% [Overview] Takes in a column vector X and returns a
% column vector f (actually -f) of the original function
% with the values of x plugged into it.
f = [-log(x(1)) - x(2)*1 + log(10-x(3)*1);
        -log(x(1)) - x(2)*2 + log(12-x(3)*2);
        -log(x(1)) - x(2)*3 + log(15-x(3)*3);];
end
```

>> newt3d.m

```
x =
8.771286446121147e+00
2.596954489674528e-01
-1.372281323269016e+00

ans =
3.901377355679192e-07
```

Note that if we do not log-transform the equations before hand we will have an ill-conditioned matrix.