

CS / MATH 4334 : Numerical Analysis

Homework Assignment 3

Matthew McMillian
mgm160130@utdallas.edu

October 17, 2018

MatLab Problems

```

1  function [A,P] = gaelpp(A)
2
3  %Gaussian elimination
4  %inputs:
5  %nXn matrix A, and and nX1 vector b
6  %outputs:
7  %nXn matrix m of multipliers
8  %nXn upper triangular matrix A
9  %nX1 vector b
10
11 %get n
12 [n,n] = size(A);
13
14
15 % Create P
16 P = eye(n);
17
18 %set up matrix of zeros that will store multipliers
19 M = zeros(n,n);
20
21 % This initially swaps the top row by finding the maximum
   value 'd
22 % row index inside this row and then swapping it with the
   initial row
23 max = 0;
24 pos = 1;
25 for p = 1:n
26     if abs(A(p,1)) > max
27         max = abs(A(p,1));
28         pos = p;
29     end
30 end
31
32
33 A([1 pos], :) = A([pos 1], :);
34 P([1 pos], :) = P([pos 1], :);
35
36 %Gaussian elimination
37 for j = 1:n-1 %j is pivot row
38
39     if A(j,j) == 0 %avoid div. by 0
40         break

```

```

41     end
42
43     % If the value at row j is less than the value
44     % at row j+1, swap the rows and update A, P, and M
45     if abs(A(j, j)) < abs(A(j+1, j))
46         A([j j+1], :) = A([j+1 j], :);
47         P([j j+1], :) = P([j+1 j], :);
48         M([j j+1], :) = M([j+1 j], :);
49     end
50
51     for i = j+1:n %elim. col. j from row i = j+1 to i = n
52
53         M(i, j) = A(i, j)/A(j, j); %multiplier to elim. A(i, j),
            store in matrix m
54
55         for k = j+1:n % add mult of row j to row i
56             A(i, k) = A(i, k) - M(i, j)*A(j, k);
57         end
58     end
59 end
60
61 % Correctly sets the pivots in A to return properly (modified
    matrix)
62 for y=2:n
63     for x=1:y-1
64         A(y, x) = M(y, x);
65     end
66 end

```

Problem 1 : gaelppscript.m

```

1  format long e
2
3  % Clears console for clean testing
4  clc
5
6  % Given A Matrix
7  A = [3.03  -12.1  14;
8        -3.03  12.1  -7;
9        6.11  -14.2  21];
10
11 % Vector of values we want to solve for
12 b = [-119; 120; -139];
13

```

```

14 % Calling Gaussian Elimination with Partial Pivoting
15 [A, P] = gaelpp(A);
16
17 A
18 P
19
20 % This checks to make sure that our PA=LU worked
21 % Forms L Matrix
22 L = [1 0 0
23       A(2,1) 1 0
24       A(3,1) A(3,2) 1];
25 % Forms U Matrix
26 U = [A(1,1) A(1,2) A(1,3)
27       0      A(2,2) A(2,3)
28       0      0      A(3,3)];
29
30 % Forward Sub and Backward Sub Output
31 y = forsub(L,b)
32 s = backsub(U, y)

```

Problem 1 : gaelscript.m _____

```

1 format long e
2
3 %matrix for circuit problem
4 A = [3.03 -12.1 14;
5       -3.03 12.1 -7;
6       6.11 -14.2 21];
7
8 %rhs for circuit problem - ' makes it a column
9 b = [-119; 120; -139];
10
11
12 %now, call function that does GE on A and b to get U and c (
   modified b)
13 [U,c,M] = gael(A,b);
14
15 U
16
17 c
18
19 %call the backsub function to find solution x
20 x = backsub(U,c)
21

```

```

22 %check to see if original Ax equals original b
23  $A*x - b$ 
24
25 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 =
```

```
-1.190000000000000e+02  
6.098690671031097e+01  
-1.899999999999999e+01
```

```
(In gaelscript.m)
```

```
s =
```

```
2.213234104513831e+01  
1.388933829477430e+01  
-2.714285714285713e+00
```

```
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.

```

1  format long e
2
3  % Clear console for easy testing
4  clc
5
6  % Setup the Vandermonde matrix of size n, as well as the
       solution vector
7  n = 10;
8  A = zeros(n,n);
9  x = ones(n,1);
10
11 % Loop creating of the Vandermonde matrix
12 for i=1:n
13     for j=1:n
14         A(i,j) = i^(j-1);
15     end
16 end
17
18 % Performing GEPP via MatLab built-in
19 A;
20 b = A*x;
21 xa = A\b;
22
23 % Calculating the errors based upon the actual solutiopn
24 % x, and the approx solution xa
25 relBkwdErr = (norm(A*x - A*xa))/norm(b) f
26 condA = cond(A)
27 boundOnFwrError = condA*relBkwdErr
28 relFwrError = (norm(x - xa))/norm(x)
29
30 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
```

```
boundOnFwrError =  
1.333079215223059e-04
```

```
relFwrErr =  
6.074920350417704e-06
```

Since we are using double precision, our answer is only accurate to 16 digits


```

1  format long e
2
3  % Clear console for easy testing
4  clc
5
6  %relative error tolerance
7  TOL = .000001;
8
9  %initial guess; x is "current" iterate
10 x = [1; 1; 1];
11
12 %store as previous iterate xp
13 xp = x;
14
15 %call function to generate right-hand-side, -f
16 rhs = genrhs(x);
17
18 %call function to generate matrix, DF
19 DF = genmatrix(x);
20
21 %Use MATLAB function to do PA=LU factorization
22 [L,U,P] = lu(DF);
23
24 %Use our functions in elearning, forsub and backsub, as
   appropriate
25 y = forsub(L, P*rhs);
26 s = backsub(U, y);
27
28 % Calculating the next iterate
29 x = xp + s;
30
31 %x is the current iterate
32 %s is the difference between current and previous iterate
33 while norm(s)/norm(x) >= TOL
34
35     %store previous iterate
36     xp = x;
37
38     %generate right-hand-side, -f
39     rhs = genrhs(x);
40
41     %generate matrix, DF

```

```

42     DF = genmatrix(x);
43
44     %PA=LU factorization
45     [L,U,P] = lu(DF);
46
47     %Use our functions forsub and backsub as appropriate
48     y = forsub(L, P*rhs);
49     s = backsub(U, y);
50
51     x = xp + s;
52
53 end
54
55 %display last iterate and relerr estimate
56 X
57
58 norm(s)/norm(x)

```

Problem 3: genmatrix.m _____

```

1  function [A] = genmatrix(x)
2  % Generates a matrix of the derivatives of the
3  % log-transformed equation given in the HW.
4  % [Overview] Takes in a column vector X and returns a matrix A
5  % (the jacobian / gradient) the original function
6  % with the values of x plugged into it.
7  A = [1/x(1) 1 1/(10-x(3)*1);
8       1/x(1) 2 2/(12-x(3)*2);
9       1/x(1) 3 3/(15-x(3)*3)];
10 end

```

Problem 3: genrhs.m _____

```

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

```

```
>> newt3d.m
```

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

```
ans =  
3.901377355679192e-07
```