**EP 501 HW01**

Patrick Good

%Project 1  
%Problem 1  
%Parts a -> d  
%This script preformes simple elimination for linear alegrbraic matrixcies  
%Created by: Patrick Good with example code from Dr. Z  
  
%Clear the workspace  
clc  
clear  
  
%Input Matrix  
load('testproblem.mat')  
%A = [80 -20 -20; -20 40 -20; -20 -20 130];  
%b = [20; 20; 20];  
  
%Forward elimination function  
Awork = forwordelim(A,b);  
  
%Back substitution function  
Solution = backsub(Awork);  
  
%Matlab built in function  
matlab = A\b;  
  
%print results  
disp('Initial Matrix A|B = ')  
disp(cat(2,A,b)) %initial matrix  
disp('My\_Results = ')  
disp(Solution) %foward and back sub results  
disp('MatLab\_Results = ')  
disp(matlab) %matlab built in results  
  
% ----------------------------------------------------------------------  
%Lower Triangle Test Problem  
  
%Load lower triangle test problem  
load('lowertriang\_testproblem.mat')  
  
%Forward elimination  
LT\_solution = forwordelim(L,bL);  
LT\_Final = LT\_solution(:,9); %results  
  
%Matlab built in function  
LT\_matlab = L\bL;  
  
%Print results  
disp('-----------------Lower-Triangle System-----------------')  
disp(' ')  
disp('Initial Matrix L|bL = ')  
disp(cat(2,L,bL)) %initial matrix  
disp('LT\_My\_Results = ')  
disp(LT\_Final)  
disp('MatLab\_Results = ')  
disp(LT\_matlab) %matlab built in results  
  
% functions  
function Out = forwordelim(A,b)  
  
% This function preforms forword elimination  
% Created by Patrick Good based on Dr. Zettergren's simple elimination  
%example cdoe  
% Inputs: forwordelim('A matrix', 'b matrix')  
  
%preform forword elimination  
  
%set up system size based on length of b  
nref=length(b);  
  
%concat A and b into 1 matrix  
Awork=cat(2,A,b);  
for ir1=2:nref %loop over rows starting at row 2  
 for ir2=ir1:nref %loop to apply row operations  
 %multiplier of the variable we are attempting to eliminate  
 fact=Awork(ir2,ir1-1); %its ir-1 column of this row  
 %subtract previous row from current row times the factor  
 Awork(ir2,:)=Awork(ir2,:)-fact/Awork(ir1-1,ir1-1).\*Awork(ir1-1,:);  
 end %for  
end %for  
  
%output  
Out = Awork;  
  
end %function  
  
function x=backsub(A)  
  
% This function performs back substitution on an upper triangular matrix that has  
% Note that B is assumed to be upper triangular at this point.  
% Sourced from Dr. Z  
  
  
n=size(A,1); %number of unknowns in the system  
x=zeros(n,1); %space in which to store our solution vector  
x(n)=A(n,n+1)/A(n,n); %finalized solution for last variable, resulting from upper triangular conversion  
  
for ir1=n-1:-1:1  
 x(ir1)=A(ir1,n+1); %assume we're only dealing with a single right-hand side here.  
 fact=A(ir1,ir1); %diagonal element to be divided through doing subs for the ir2 row  
 for ic=ir1+1:n  
 x(ir1)=x(ir1)-A(ir1,ic)\*x(ic);  
 end %for  
 x(ir1)=x(ir1)/fact; %divide once at the end to minimize number of ops  
end %for  
  
end %function

Initial Matrix A|B =   
 Columns 1 through 7  
  
 -1.0149 -2.1321 2.1778 -0.2730 -0.7841 -0.4677 -0.2841  
 -0.4711 1.1454 1.1385 1.5763 -1.8054 -0.1249 -0.0867  
 0.1370 -0.6291 -2.4969 -0.4809 1.8586 1.4790 -1.4694  
 -0.2919 -1.2038 0.4413 0.3275 -0.6045 -0.8608 0.1922  
 0.3018 -0.2539 -1.3981 0.6647 0.1034 0.7847 -0.8223  
 0.3999 -1.4286 -0.2551 0.0852 0.5632 0.3086 -0.0942  
 -0.9300 -0.0209 0.1644 0.8810 0.1136 -0.2339 0.3362  
 -0.1768 -0.5607 0.7477 0.3232 -0.9047 -1.0570 -0.9047  
  
 Columns 8 through 9  
  
 -0.2883 -10.8600  
 0.3501 3.9577  
 -1.8359 -17.3415  
 1.0360 1.3800  
 2.4245 17.1229  
 0.9594 8.8013  
 -0.3158 2.0375  
 0.4286 -11.5312  
  
My\_Results =   
 1.0000  
 2.0000  
 3.0000  
 4.0000  
 5.0000  
 6.0000  
 7.0000  
 8.0000  
  
MatLab\_Results =   
 1.0000  
 2.0000  
 3.0000  
 4.0000  
 5.0000  
 6.0000  
 7.0000  
 8.0000  
  
-----------------Lower-Triangle System-----------------  
   
Initial Matrix L|bL =   
 Columns 1 through 7  
  
 1.0000 0 0 0 0 0 0  
 0.3885 1.0000 0 0 0 0 0  
 0.5040 -0.8461 1.0000 0 0 0 0  
 -1.2484 1.8781 -1.5861 1.0000 0 0 0  
 -0.8977 0.5929 -1.5219 -0.6058 1.0000 0 0  
 -1.7325 1.1953 -3.0180 -0.2596 0.3782 1.0000 0  
 -4.1011 1.6584 -2.8602 0.7259 -1.5369 6.9610 1.0000  
 -2.8207 2.0153 -4.2019 0.0744 0.1021 3.5430 0.3983  
  
 Columns 8 through 9  
  
 0 1.0000  
 0 3.3885  
 0 2.9657  
 0 3.4554  
 0 -1.9692  
 0 -0.6498  
 0 67.3931  
 1.0000 42.8073  
  
LT\_My\_Results =   
 1.0000  
 3.0000  
 5.0000  
 7.0000  
 9.0000  
 11.0000  
 13.0000  
 15.0000  
  
MatLab\_Results =   
 1.0000  
 3.0000  
 5.0000  
 7.0000  
 9.0000  
 11.0000  
 13.0000  
 15.0000

[*Published with MATLAB® R2020a*](https://www.mathworks.com/products/matlab)

%Project 1  
%Problem 2  
%Part a  
%This script demos Forward Elimination with multible right-hand sides  
%Created by: Patrick Good  
clear  
clc  
% Input matrixies  
A = [80 -20 -20; -20 40 -20; -20 -20 130];  
b = [20 20; 20 10; 20 20]; %from ex 1.8 in Hoffman text  
%load('testproblem.mat')  
  
% Measure the size of the b matrix  
[~ ,c] = size(b);  
  
% Send A and b to multible right-hand side forword elimination function  
Awork = forelim\_MRHS(A,b);  
  
% Send forword elimination results to backsub function to uptain the  
%solution  
X = cell(1,c); %reallacating for effeciency  
for n=1:c  
  
 X{n} = backsub(Awork{n});  
  
end  
  
% Concat the solution back to 1 matrix  
% If you use an identity matrix for b this will give you the A inverse  
SOL = X{1};  
for n=2:c  
  
 SOL = cat(2,SOL,X{n});  
  
end  
  
%Matlab Solution  
M\_SOL = A\b;  
  
% Display the results  
disp('Solution = ')  
disp(SOL)  
disp('Matlab Solution =')  
disp(M\_SOL)  
  
%function  
function Out = forelim\_MRHS(A,b)  
  
% This function preforms forward elimination  
% Created by Patrick Good based on Dr. Zettergren's simple elimination  
%example cdoe  
% Inputs: forsub\_MRHS('A matrix', 'b matrix')  
  
% Measure the size of the b matrix  
[~ ,c] = size(b);  
  
%split b into seperate collums for the forword elimination  
B = cell(1,c); %reallacating for effeciency  
for n=1:c  
 B{n} = b(:,n);  
end  
  
%preform forword elimination  
Out = cell(1,c); %reallacating for effeciency  
for n=1:c %loop for each b collum  
  
 %set up system size based on length of b  
 nref=length(B{n});  
  
 %concat A and b into 1 matrix  
 Awork=cat(2,A,B{n});  
 for ir1=2:nref %loop over rows starting at row 2  
 for ir2=ir1:nref %loop to apply row operations  
 %multiplier of the variable we are attempting to eliminate  
 fact=Awork(ir2,ir1-1); %its ir-1 column of this row  
 %subtract previous row from current row times the factor  
 Awork(ir2,:)=Awork(ir2,:)-fact/Awork(ir1-1,ir1-1).\*Awork(ir1-1,:);  
 end %for  
 end %for  
  
 Out{n} = Awork;  
  
end %for  
  
end %function

Solution =   
 0.6000 0.5000  
 1.0000 0.6667  
 0.4000 0.3333  
  
Matlab Solution =  
 0.6000 0.5000  
 1.0000 0.6667  
 0.4000 0.3333

[*Published with MATLAB® R2020a*](https://www.mathworks.com/products/matlab)

%Project 1  
%Problem 2  
%Parts b -> d  
%This script demos Gauss-Jordan Elimination  
%Created by: Patrick Good  
  
clear  
clc  
% Input matrixies  
%A = [80 -20 -20; -20 40 -20; -20 -20 130];  
%b = [20;20;20];  
%b = [1 0 0;0 1 0;0 0 1];  
load('testproblem.mat')  
b = eye(8); %identity matrix for test problem  
  
% Measure the size of the b matrix  
[~ ,c] = size(b);  
  
% Send A and b to Gauss-Jordan elimination function  
Awork = gaussjordan(A,b);  
  
% Pull solution from cell data type  
X = cell(1,c); %reallacating for effeciency  
for n=1:c  
  
 X{n} = Awork{n};  
  
end  
  
% Concat the solution back to 1 matrix  
% If you use an identity matrix for b this will give you the A inverse  
SOL = X{1};  
for n=2:c  
  
 SOL = cat(2,SOL,X{n});  
  
end  
  
%Matlab Inverse Solution  
M\_SOL = inv(A);  
  
% Display the results  
disp('Solution = ')  
disp(SOL)  
disp('MatLab Inverse Solution =')  
disp(M\_SOL)  
  
%function  
function Out = gaussjordan(A,b)  
  
% This function preforms Gauss-Jordan elimination  
% Created by Patrick Good based on Dr. Zettergren's simple elimination  
%example cdoe  
% Inputs: gaussjordan('A matrix', 'b matrix', 'size of b matrix')  
  
% Measure the size of the b matrix  
[~ ,c] = size(b);  
  
%split b into seperate collums for the elimination  
B = cell(1,c); %reallacating for effeciency  
for n=1:c  
 B{n} = b(:,n);  
end  
  
%preform Gauss-Jordan elimination  
Out = cell(1,c); %reallacating for effeciency  
for n=1:c %loop for each b collum  
  
 %set up system size based on length of b  
 nref=length(B{n});  
  
 %concat A and b into 1 matrix  
 Awork=cat(2,A,B{n});  
  
 for r=1:nref %loop over rows  
  
 Awork(r,:)=Awork(r,:)./Awork(r,r); %reference row scaling  
 for k=1:nref %loop to preform elimination on rows  
  
 if k==r %check if on reference row  
 Awork(k,:)=Awork(k,:); %do not eliminate reference row  
 else  
 Awork(k,:)=Awork(k,:)-(Awork(k,r).\*Awork(r,:)); %elimination  
 end %if  
  
 end %for  
 end %for  
 %Output  
 Out{n} = Awork(:,nref+1);  
end %for  
end %function

Solution =   
 Columns 1 through 7  
  
 -0.4480 0.3835 0.0281 -0.0881 -0.5795 1.0474 -0.5356  
 -0.0540 -0.1948 -0.2456 -0.6264 0.1978 -0.2692 0.2222  
 0.2062 -0.1064 -0.3766 -1.1154 -0.0220 0.5605 0.2837  
 -0.3250 0.4251 0.0724 -0.1670 -0.3128 0.8816 0.4305  
 -0.0697 -0.5582 -0.4000 -1.3059 0.0704 0.6537 0.8908  
 0.3565 0.3345 0.1079 -0.1491 0.2014 0.0363 -0.2920  
 -0.1222 0.1436 0.0008 0.7677 -0.2421 -0.0132 -0.1231  
 0.1043 -0.2818 -0.2839 -0.2878 0.4281 -0.1212 0.1503  
  
 Column 8  
  
 0.2581  
 0.2324  
 0.3873  
 0.2608  
 0.6467  
 -0.6463  
 -0.7433  
 0.0735  
  
MatLab Inverse Solution =  
 Columns 1 through 7  
  
 -0.4480 0.3835 0.0281 -0.0881 -0.5795 1.0474 -0.5356  
 -0.0540 -0.1948 -0.2456 -0.6264 0.1978 -0.2692 0.2222  
 0.2062 -0.1064 -0.3766 -1.1154 -0.0220 0.5605 0.2837  
 -0.3250 0.4251 0.0724 -0.1670 -0.3128 0.8816 0.4305  
 -0.0697 -0.5582 -0.4000 -1.3059 0.0704 0.6537 0.8908  
 0.3565 0.3345 0.1079 -0.1491 0.2014 0.0363 -0.2920  
 -0.1222 0.1436 0.0008 0.7677 -0.2421 -0.0132 -0.1231  
 0.1043 -0.2818 -0.2839 -0.2878 0.4281 -0.1212 0.1503  
  
 Column 8  
  
 0.2581  
 0.2324  
 0.3873  
 0.2608  
 0.6467  
 -0.6463  
 -0.7433  
 0.0735

[*Published with MATLAB® R2020a*](https://www.mathworks.com/products/matlab)

%Project 1  
%Problem 3  
%Parts a -> b  
%This script demos Gaussian Elimination and Determinants  
%Created by: Patrick Good based on Gauss\_elim\_example from Dr. Z  
clc  
clear  
  
%Input Matrix  
load('testproblem.mat')  
%A = [80 -20 -20; -20 40 -20; -20 -20 130]  
%b = [1 0 0; 0 1 0; 0 0 1]  
  
% Use the Gaussian elimination function to solve the same system (include scaled pivoting)  
[Amod,ord,detm]=Gauss\_elim(A,b);  
  
% Compare against built in MATLAB solution  
xmat=A\b;  
det\_matlab = det(A);  
  
%Outputs  
disp('Elimination with scaled pivoting on matrix: ');  
disp(Amod(ord,:));  
xgauss=backsub(Amod(ord,:));  
disp('Gaussian elimination solution = ');  
disp(xgauss);  
disp('Determinate = ')  
disp(detm)  
disp('Built-in MATLAB solution = ');  
disp(xmat);  
disp('Matlab Determinate = ')  
disp(det\_matlab)  
  
%function  
function [Amod,ord,det]=Gauss\_elim(A,b,verbose)  
  
%Modefied from Dr. Z's Gauss\_elim function to calculate the determinate  
% This function perform elimination with partial pivoting and scaling as  
% described in Section 1.3.2 in the Hoffman textbook (viz. it does Gaussian  
% elimination). Note that the ordering which preserves upper triangularity  
% is stored in the ord output variable, such that the upper triangular output  
% is given by row-permuted matrix Amod(ord,:). The verbose flag can be set to  
% true or false (or omitted, default=false) in order to print out what the algirthm  
% is doing for each elimination step.  
  
% Parse the inputs, throw an error if something is obviously wrong with input data  
narginchk(2,3);  
if (nargin<3)  
 verbose=false;  
end %if  
  
% Need to error check for square input.  
% Allocation of space and setup  
Amod=cat(2,A,b); %make a copy of A and modify with RHS of system  
n=size(A,1); %number of unknowns  
ord=(1:n)'; %ord is a mapping from input row being operated upon to the actual row that represents in the matrix ordering  
count = 0; %initialize interchange counter for correct det sighn  
% Elimination with scaled, partial pivoting for matrix Amod; note all row  
%indices must be screen through ord mapping.  
for ir1=1:n-1  
 if (verbose)  
 disp('Starting Gauss elimination from row: ');  
 disp(ir1);  
 disp('Current state of matrix: ');  
 disp(Amod(ord,:));  
 end %if  
  
 %check scaled pivot elements to see if reordering should be done  
 pivmax=0;  
 ipivmax=ir1; %max pivot element should never be higher than my current position  
 for ipiv=ir1:n %look only below my current position in the matrix  
 pivcurr=abs(Amod(ord(ipiv),ir1))/max(abs(Amod(ord(ipiv),:))); %note that columns never get reordered...  
 if (pivcurr>pivmax)  
 pivmax=pivcurr;  
 ipivmax=ipiv; %this stores the index into ord for row having largest pivot element  
 end %if  
 end %for  
  
 %reorder if situation calls for it  
 if (ipivmax ~= ir1)  
 itmp=ord(ir1);  
 ord(ir1)=ord(ipivmax);  
 ord(ipivmax)=itmp;  
  
 if (verbose)  
 disp('Interchanging rows: ');  
 disp(itmp);  
 disp(' and: ');  
 disp(ord(ir1));  
 disp('Current matrix state after interchange: ');  
 disp(Amod(ord,:));  
 count = count + 1; %for correct det sighn counts row interchanges  
 end %if  
 end %if  
  
 %perform the elimination for this row, former references to ir1 are now  
 %mapped through the ord array  
 for ir2=ir1+1:n  
 fact=Amod(ord(ir2),ir1);  
 Amod(ord(ir2),ir1:n+1)=Amod(ord(ir2),ir1:n+1)-fact/Amod(ord(ir1),ir1).\*Amod(ord(ir1),ir1:n+1); %only need columns ahead of where we are in matrix  
 end %for  
  
 if (verbose)  
 disp('Following elimination for row: ');  
 disp(ir1);  
 disp(' matrix state: ');  
 disp(Amod(ord,:));  
 end %if  
end %for  
  
%Determinate eq 1.119 from Hoffman textbook  
A\_final = Amod(ord,:);  
det = A\_final(1,1);  
for k = 2:n %loop through rows  
 det = det\*A\_final(k,k);  
end %for  
%disp(count)  
  
%Correct the Determinate sighn for row interchanges  
if rem(count,2)==0  
 %det=det so do nothing for even row interchanges  
else  
 det =det\*-1; %flip the sighn  
end %if  
  
end %function

Elimination with scaled pivoting on matrix:   
 Columns 1 through 7  
  
 -0.9300 -0.0209 0.1644 0.8810 0.1136 -0.2339 0.3362  
 0 -1.1973 0.3897 0.0510 -0.6402 -0.7874 0.0867  
 -0.0000 0 1.4314 1.1793 -2.4810 -0.7666 -0.1733  
 0 0 0 1.8286 -2.4289 0.4258 -1.7899  
 0 -0.0000 0 0.0000 1.4990 0.5853 0.7876  
 0 0 0 0 0 2.7112 -2.6306  
 0 0 0 0 0 0 -1.4250  
 0 0 0 0 0 0 0  
  
 Columns 8 through 9  
  
 -0.3158 2.0375  
 1.1351 0.7405  
 1.6059 3.6406  
 0.5231 -10.6202  
 -0.0765 15.9078  
 -2.7627 -24.2488  
 -0.8058 -16.4214  
 2.3358 18.6867  
  
Gaussian elimination solution =   
 1.0000  
 2.0000  
 3.0000  
 4.0000  
 5.0000  
 6.0000  
 7.0000  
 8.0000  
  
Determinate =   
 -39.4247  
  
Built-in MATLAB solution =   
 1.0000  
 2.0000  
 3.0000  
 4.0000  
 5.0000  
 6.0000  
 7.0000  
 8.0000  
  
Matlab Determinate =   
 -39.4247

[*Published with MATLAB® R2020a*](https://www.mathworks.com/products/matlab)