```
format long e
```

Question 1.

```
disp('Question 1');
```

```
Question 1
```

```
n = 5;
U = triu(randi(10, n, n));
L = tril(randi(10, n, n));

b_upper = randi(10, n, 1);
b_lower = randi(10, n, 1);

disp('Upper Triangular Matrix U:');
```

```
Upper Triangular Matrix U:
```

```
disp(U);
```

```
     9     2     9     3     5
     0    10     9     1     4
     0     0     1     3     9
     0     0     0     2     5
     0     0     0     0     3
```

```
disp('Right-hand side vector b (for Ux = b):');
```

```
Right-hand side vector b (for Ux = b):
```

```
disp(b_upper);
```

```
     1
     6
     5
    10
     7
```

```

disp('Lower Triangular Matrix L:');
```

```
Lower Triangular Matrix L:
```

```
disp(L);
```

```
     1     0     0     0     0
     3     7     0     0     0
     3     5     6     0     0
     1     3     2     9     0
     8     7     9     7     9
```

```
disp('Right-hand side vector b (for Lx = b):');
```

```
Right-hand side vector b (for Lx = b):
```

```
disp(b_lower);
```

```
    10
     6
     9
     5
     6
```

```
x_upper = colbackward(U, b_upper);

x_lower = rowforward(L, b_lower);

disp('Solution x (for Ux = b):');
```

```
Solution x (for Ux = b):
```

```
disp(x_upper);
```

```
     9.948148148148146e+00
     1.190000000000000e+01
    -1.350000000000000e+01
    -8.333333333333339e-01
     2.333333333333333e+00
```

```
disp('Solution x (for Lx = b):');
```

```
Solution x (for Lx = b):
```

```
disp(x_lower);
```

```
     1.000000000000000e+01
    -3.428571428571428e+00
    -6.428571428571429e-01
     7.301587301587300e-01
    -5.480599647266313e+00
```

```
disp('Verification for Ux = b:');
```

```
Verification for Ux = b:
```

```
disp(U * x_upper);
```

```
     9.999999999999947e-01
     6.000000000000000e+00
     5.000000000000000e+00
     1.000000000000000e+01
     7.000000000000000e+00
```

```
disp('Verification for Lx = b:');
```

```
Verification for Lx = b:
```

```
disp(L * x_lower);
```

```
    10
     6
```

```
     9
     5
     6
```

Question 2.

```
disp('Question 2');
```

```
Question 2
```

```
n = 4;
A = randi(10, n, n);

disp('Randomly generated matrix A:');
```

```
Randomly generated matrix A:
```

```
disp(A);
```

```
     2     5     1     5
     7     8     6     4
    10     1     7     7
     6     9     6    10
```

```
[L, U] = genp(A);

disp('Lower Triangular Matrix L:');
```

```
Lower Triangular Matrix L:
```

```
disp(L);
```

```
     1.000000000000000e+00                         0                         0
     3.500000000000000e+00     1.000000000000000e+00                         0
     5.000000000000000e+00     2.526315789473684e+00     1.000000000000000e+00
     3.000000000000000e+00     6.315789473684210e-01    -3.292682926829268e-01     1.000000000000000e+0
```

```
disp('Upper Triangular Matrix U:');
```

```
Upper Triangular Matrix U:
```

```
disp(U);
```

```
     2.000000000000000e+00     5.000000000000000e+00     1.000000000000000e+00     5.000000000000000e+0
                         0    -9.500000000000000e+00     2.500000000000000e+00    -1.350000000000000e+0
                         0                         0    -4.315789473684211e+00     1.610526315789473e+0
                         0                         0    -2.220446049250313e-16     8.829268292682926e+0
```

```
disp('Verification (A should be equal to L * U):');
```

```
Verification (A should be equal to L * U):
```

```
disp(L * U);
```

```
2.000000000000000e+00     5.000000000000000e+00     1.000000000000000e+00     5.000000000000000e+0
7.000000000000000e+00     8.000000000000000e+00     6.000000000000000e+00     4.000000000000000e+0
1.000000000000000e+01     1.000000000000000e+00     7.000000000000000e+00     7.000000000000000e+0
6.000000000000000e+00     9.000000000000000e+00     6.000000000000001e+00     1.000000000000000e+0
```

Question 3.

```
disp('Question 3');
```

```
Question 3
```

```
disp('part a');
```

```
part a
```

```
%part a
A = [1e-20, 1;
     1, 1];

[L, U] = genp(A);

disp('Lower Triangular Matrix L:');
```

```
Lower Triangular Matrix L:
```

```
disp(L);
```

```
1.000000000000000e+00                         0
1.000000000000000e+20     1.000000000000000e+00
```

```
disp('Upper Triangular Matrix U:');
```

```
Upper Triangular Matrix U:
```

```
disp(U);
```

```
9.999999999999999e-21     1.000000000000000e+00
                    0    -1.000000000000000e+20
```

```
A_minus_LU = A - L * U;

disp('A - LU:');
```

```
A - LU:
```

```
disp(A_minus_LU);
```

```
0     0
0     1
```

```
%part b
disp('part b');
```

part b

```
b = [1; 0];
y = rowforward(L, b);

xc = colbackward(U, y);
disp('Computed solution xc:');
```

```
Computed solution xc:
```

```
disp(xc);
```

```
     0
     1
```

```
x = [-1/(1-10^-20); 1/(1-10^-20)];
error_norm = norm(xc - x) / norm(x);
disp('2-norm difference between the computed and exact solution:');
```

```
2-norm difference between the computed and exact solution:
```

```
disp(error_norm);
```

```
     7.071067811865475e-01
```

Conclusion:

1. **Gaussian Elimination without Pivoting (GENP)** fails when the matrix A has very small or nearly zero pivots. This leads to large rounding errors and numerical instability.
2. The step where things start to go wrong is during the LU decomposition, where the pivot 10^-20 in matrix A causes large errors in the subsequent computations.

Question 4.

```
disp('Question 4');
```

```
Question 4
```

```
num_tests = 5;

for test = 1:num_tests
    n = 4;
    A = randn(n);

    [L_gepp, U_gepp, p_gepp] = gepp(A);
    [L_matlab, U_matlab, P_matlab] = lu(A);

    P_gepp = eye(n);
    P_gepp = P_gepp(p_gepp, :);

    norm_diff_L = norm(L_gepp - L_matlab);
    norm_diff_U = norm(U_gepp - U_matlab);
```

```
    norm_diff_P = norm(P_gepp - P_matlab);

    fprintf('Test %d:\n', test);
    fprintf('Norm of difference in L: %e\n', norm_diff_L);
    fprintf('Norm of difference in U: %e\n', norm_diff_U);
    fprintf('Norm of difference in P: %e\n\n', norm_diff_P);
end
```

```
Test 1:
Norm of difference in L: 0.000000e+00
Norm of difference in U: 0.000000e+00
Norm of difference in P: 0.000000e+00
Test 2:
Norm of difference in L: 0.000000e+00
Norm of difference in U: 2.220446e-16
Norm of difference in P: 0.000000e+00
Test 3:
Norm of difference in L: 0.000000e+00
Norm of difference in U: 0.000000e+00
Norm of difference in P: 0.000000e+00
Test 4:
Norm of difference in L: 0.000000e+00
Norm of difference in U: 1.144392e-16
Norm of difference in P: 0.000000e+00
Test 5:
Norm of difference in L: 0.000000e+00
Norm of difference in U: 1.241267e-16
Norm of difference in P: 0.000000e+00
```

Question 5.

```
disp('Question 5');
```

```
Question 5
```

```
disp('part a');
```

```
part a
```

```
%part a
num_tests = 5;

for test = 1:num_tests
    n = 4;
    A = randn(n);
    b = randn(n, 1);
    x_gepp = geppsolve(A, b);

    x_matlab = A\b;
    error_norm = norm(x_gepp - x_matlab) / norm(x_gepp);

    fprintf('Test %d:\n', test);
    fprintf('Norm of difference between geppsolve and A\\b: %e\n\n',
error_norm);
end
```

```
Test 1:
Norm of difference between geppsolve and A\b: 0.000000e+00
Test 2:
Norm of difference between geppsolve and A\b: 0.000000e+00
Test 3:
Norm of difference between geppsolve and A\b: 0.000000e+00
Test 4:
Norm of difference between geppsolve and A\b: 0.000000e+00
Test 5:
Norm of difference between geppsolve and A\b: 0.000000e+00
```

```
disp('part b');
```

```
part b
```

```
%part b
A = [1e-20, 1;
     1, 1];
b = [1; 0];

xc_gepp = geppsolve(A, b);
[L, U, p] = gepp(A);

P = eye(2);
P = P(p, :);
A_minus_LU = P * A - L * U;

disp('A(p,:) - LU using gepp:');
```

A(p,:)

```
A(p,:) - LU using gepp:
```

```
disp(A_minus_LU);
```

```
     0     0
     0     0
```

```
x_exact =[-1/(1-10^-20); 1/(1-10^-20)];
error_norm_gepp = norm(xc_gepp - x_exact);
disp('Norm of difference between geppsolve and exact solution:');
```

```
Norm of difference between geppsolve and exact solution:
```

```
disp(error_norm_gepp);
```

```
     0
```

Conclusion: After running the code we can confirm that `gepp` provides more accurate results than `genp`, especially for matrices like A in the 2*2 case, where pivoting is crucial. `genp` fails due to numerical instability when the matrix has small pivots, while `gepp` handles such cases more robustly.

Question 6.

```
disp('Question 6');
```

Question 6

```
n=5;
A=randn(n);
disp('random matrix A = ');
```

```
random matrix A =
```

```
disp(A);
```

```
  -4.590747733815575e-01    -3.310675676112584e-01    -1.033819342256185e+00    -7.420983198496416e-0
  -4.609848531962479e-03    -3.846281848135416e-01     6.037454129312170e-01    -8.531496569128730e-0
   1.185861712459795e+00     1.058003564006352e+00     2.491761293026885e-01    -1.284109016934457e+0
   2.237393850810007e-01    -5.316595034394730e-01     4.415437592863519e-01    -3.385957300213092e-0
   8.039214645663924e-02    -4.727048362995947e-02    -2.692619949749908e-02     2.307189740375808e+0
```

```
disp('determinent computed by built in function: ');
```

```
determinent computed by built in function:
```

```
disp(det(A));
```

```
  -2.666452511283450e+00
```

```
disp('determinent computed by mydet function: ');
```

```
determinent computed by mydet function:
```

```
disp(mydet(A));
```

```
  -2.666452511283450e+00
```

Functions

Question 1

what if U(j j) == 0

```
function x = colbackward(U, b)
    n = length(b);
    x = zeros(n, 1);

    for j = n:-1:1
        x(j) = b(j) / U(j, j);
        for i = 1:j-1
            b(i) = b(i) - U(i, j) * x(j);
        end
    end
end

function x = rowforward(L, b)
    n = length(b);
    x = zeros(n, 1);
    for i = 1:n
        x(i) = b(i);
        for j = 1:i-1
```

```
            x(i) = x(i) - L(i, j) * x(j);
        end
        x(i) = x(i) / L(i, i);
    end
end
```

what if L(i,i) = 0

Question 2

```
function [L, U] = genp(A)
    n = size(A, 1);
    L = eye(n);
    U = A;

    for k = 1:n-1
        for i = k+1:n
            L(i, k) = U(i, k) / U(k, k);
            U(i, k:n) = U(i, k:n) - L(i, k) * U(k, k:n);
        end
    end
end
```

Question 4

```
function [L, U, p, s] = gepp(A)
    n = size(A, 1);
    s=1;
    L = eye(n);
    U = A;
    p = (1:n)';

    for k = 1:n-1
        [~, maxIndex] = max(abs(U(k:n, k)));
        maxIndex = maxIndex + k - 1;

        if maxIndex ~= k
            U([k, maxIndex], :) = U([maxIndex, k], :);
            p([k, maxIndex]) = p([maxIndex, k]);
            s=s*-1;
            if k > 1
                L([k, maxIndex], 1:k-1) = L([maxIndex, k], 1:k-1);
            end
        end
        for i = k+1:n
            L(i, k) = U(i, k) / U(k, k);
            U(i, k:n) = U(i, k:n) - L(i, k) * U(k, k:n);
        end
    end
end
```

Question 5

```
function x = geppsolve(A, b)
    [L, U, p] = gepp(A);

    b = b(p);
    y = rowforward(L, b);
    x = colbackward(U, y);
end
```

Question 6

don't use det(L).

```
function val = mydet(A)
    [L, U, p, s] = gepp(A);
    val = det(L)*det(U)*s;
end
```