```
function q2_b()
    % Solve for X_c
    % AX = B
    % B = A * X_t
    % x_t = rand(10,4)
    % A = hilbert

    iters = 10;
    iterations = (1:iters)';
    epsilon = ones(iters, 1) * eps;
    distances = zeros(iters, 4);
    condition_numbers = zeros(iters, 1);
    relative_residuals = zeros(iters, 4);

    for i = 1:iters
        A = hilb(10);
        X_t = rand(10,4);
        B = A * X_t;

        % I'm using MATLAB's mldivide function here as a placeholder.
        % Replace it with your 'ggepp' function if you have it.
        X_c = A \ B;

        % Part 1
        % distances: computed per column j
        for j = 1:4
            distances(i, j) = norm(X_c(:,j) - X_t(:,j), 2) / norm(X_t(:,j),
2);
        end

        % Part 2
        condition_number = eps * cond(A, 2);
        condition_numbers(i) = condition_number;

        % Part 3
        % compute the relative residual per column j
        for j = 1:4
            residual = B(:,j) - A * X_c(:,j);
            relative_residuals(i, j) = norm(residual, 2) / (norm(A, 2) *
norm(X_c(:,j), 2));
        end
    end

    % Separate tables: one for epsilon and distances, another for the rest.
    T1 = table(iterations, condition_numbers, distances(:,1), distances(:,2),
distances(:,3), distances(:,4), ...
              'VariableNames', {'i', 'cond', 'error(1)', 'error(2)',
'error(3)', 'error(4)'});

    T2 = table(iterations, epsilon, ...
               relative_residuals(:,1), relative_residuals(:,2),
relative_residuals(:,3), relative_residuals(:,4), ...
```

```
            'VariableNames', {'i', 'epsilon', ...
            'RelRes(1)', 'RelRes(2)', 'RelRes(3)', 'RelRes(4)'});

    disp("10 Iterations: Condition Numbers , Euclidiean Error Distances
j=1:4");
    disp(T1)
    disp("10 Iterations: Machine Epsilon , Relative Residual j=1:4");
    disp(T2)




    % Part 3 i)
    % As seen in class the condition number given by the 2-norm of A X A^-1
    % can be used to estimate the relative error in the solution of a linear
    % system. The condition number multiplied by the machine epsilon
    % is greater than the distance between the computed solution and the
    % true solution. This is seen in the table above. Each distance is less
than
    % the condition number multiplied by the machine epsilon.

    % Part 3 ii)
    % we saw in class that
    % norm(r) < (machine epsilon) * norm(A) * norm(X_c)
    %
    %r = B - AX_c
    %
    % Note this can be re arranged to our equation
    % norm( B - AX_c, 2 ) / (norm(A, 2) * norm(x_c, 2)) < (machine epsilon)
    %
    % We can see from our results that for all values of relative residuals
    % they are indeed smaller the machine epsilon
    %



    % Part 4

    % use the code to compute the inverse of a n x n non singular matrix A

    % note that A inverse is deifned as A x A_inv = I
    % where I is the identity matrix
    % this means to find A_inv we need to solve for this equation
    %

    A = hilb(10);
    format;
    disp("A =");
    disp(A);

    n = size(A,1);
    I = eye(n);
```

```
    A_I = ggepp(A,I);

    disp("A_I =");
    disp(A_I);

    % Note the computational cost is the same as for GEPP which is
    % 2/3n^3 + 1/2n^2


end



 function X = ggepp(A, B)
     [n, ~] = size(A);
     [L,U,P] =  lupp(A);
     [~, p] = size(B);
     Y = zeros(n, p);
     X = zeros(n, p);
     for i = 1:p
         y = forward_substitution(L, P * B(:, i));
         x = backward_substitution(U, y);
         Y(:, i) = y;
         X(:, i) = x;
     end
 end

 function [L,U,P] = lupp(A)
    % lupp: LU factorization with partial pivoting
    %
    % input:  A
    % output: L, U and P such that PA = LU
    %
    n = size(A,1);
    P = eye(n);

    for k = 1:n-1
       [maxval, maxindex] = max(abs(A(k:n,k)));
       q = maxindex + k - 1;
       if maxval == 0, error('A is singular'), end
       if q ~= k
           A([k,q],:) = A([q,k],:);
           P([k,q],:) = P([q,k],:);
       end
       i = k+1:n;
       A(i,k) = A(i,k)/A(k,k);
       A(i,i) = A(i,i) - A(i,k)*A(k,i);
    end

    L = tril(A,-1) + eye(n);
    U = triu(A);
 end

 function x = backward_substitution(A, b)
```

```
    n = size(A, 1);
    x = zeros(n, 1);
    for i = n:-1:1
        x(i) = (b(i) - A(i, i+1:end) * x(i+1:end)) / A(i, i);
    end
 end

 function x = forward_substitution(A, b)
    n = size(A, 1);
    x = zeros(n, 1);
    for i = 1:n
        x(i) = (b(i) - A(i, 1:i-1) * x(1:i-1)) / A(i, i);
    end
end
```

10 Iterations: Condition Numbers , Euclidiean Error Distances j=1:4

i	cond	error(1)	error(2)	error(3)	error(4)
1	0.0035582	0.00012106	4.1294e-06	0.00012139	0.00028925
2	0.0035582	0.00023961	0.00019616	0.00012472	0.00010619
3	0.0035582	0.00038274	0.00016712	2.5095e-05	0.00020481
4	0.0035582	0.00048397	0.00014908	6.5387e-05	0.00025672
5	0.0035582	2.3141e-05	0.00027844	3.6521e-05	0.0001835
6	0.0035582	0.00018904	4.2913e-05	0.00012595	0.000103
7	0.0035582	0.00017823	6.9976e-05	0.00013087	0.00037042
8	0.0035582	0.00028516	0.00011794	0.00024204	0.00031481
9	0.0035582	0.00054699	0.0003977	4.2866e-05	0.00039303
10	0.0035582	0.00010305	3.6482e-05	0.00024061	0.00021473

10 Iterations: Machine Epsilon , Relative Residual j=1:4

i	epsilon	RelRes(1)	RelRes(2)	RelRes(3)	RelRes(4)
1	2.2204e-16	1.0567e-16	8.2082e-17	6.6435e-17	0
2	2.2204e-16	6.8141e-17	9.0713e-17	5.5702e-17	7.7366e-17
3	2.2204e-16	4.4733e-17	1.1814e-16	6.8379e-17	1.4123e-16
4	2.2204e-16	6.2514e-17	9.6633e-17	1.1488e-16	1.4402e-16
5	2.2204e-16	1.4353e-16	6.9144e-17	3.739e-17	7.1597e-17
6	2.2204e-16	1.2068e-16	9.6017e-17	7.4815e-17	8.4256e-17
7	2.2204e-16	1.0799e-16	1.5784e-16	6.1567e-17	1.0903e-16
8	2.2204e-16	8.2491e-17	6.0976e-17	8.3778e-17	1.0434e-16
9	2.2204e-16	1.2102e-16	6.8838e-17	6.0437e-17	9.5968e-17
10	2.2204e-16	5.8119e-17	9.1075e-17	5.4699e-17	7.1378e-17

A =

Columns 1 through 7

1.0000	0.5000	0.3333	0.2500	0.2000	0.1667	0.1429
0.5000	0.3333	0.2500	0.2000	0.1667	0.1429	0.1250
0.3333	0.2500	0.2000	0.1667	0.1429	0.1250	0.1111
0.2500	0.2000	0.1667	0.1429	0.1250	0.1111	0.1000
0.2000	0.1667	0.1429	0.1250	0.1111	0.1000	0.0909
0.1667	0.1429	0.1250	0.1111	0.1000	0.0909	0.0833

```
    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769
    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769    0.0714
    0.1111    0.1000    0.0909    0.0833    0.0769    0.0714    0.0667
    0.1000    0.0909    0.0833    0.0769    0.0714    0.0667    0.0625

  Columns 8 through 10

    0.1250    0.1111    0.1000
    0.1111    0.1000    0.0909
    0.1000    0.0909    0.0833
    0.0909    0.0833    0.0769
    0.0833    0.0769    0.0714
    0.0769    0.0714    0.0667
    0.0714    0.0667    0.0625
    0.0667    0.0625    0.0588
    0.0625    0.0588    0.0556
    0.0588    0.0556    0.0526

A_I =
   1.0e+12 *

  Columns 1 through 7

    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000
   -0.0000    0.0000   -0.0000    0.0000   -0.0002    0.0005   -0.0008
    0.0000   -0.0000    0.0001   -0.0010    0.0043   -0.0112    0.0178
   -0.0000    0.0000   -0.0010    0.0082   -0.0379    0.1010   -0.1616
    0.0000   -0.0002    0.0043   -0.0379    0.1767   -0.4772    0.7712
   -0.0000    0.0005   -0.0112    0.1010   -0.4772    1.3014   -2.1208
    0.0000   -0.0008    0.0178   -0.1616    0.7712   -2.1208    3.4803
   -0.0000    0.0008   -0.0166    0.1529   -0.7358    2.0376   -3.3636
    0.0000   -0.0004    0.0085   -0.0788    0.3820   -1.0643    1.7659
   -0.0000    0.0001   -0.0018    0.0171   -0.0832    0.2330   -0.3883

  Columns 8 through 10

   -0.0000    0.0000   -0.0000
    0.0008   -0.0004    0.0001
   -0.0166    0.0085   -0.0018
    0.1529   -0.0788    0.0171
   -0.7358    0.3820   -0.0832
    2.0376   -1.0643    0.2330
   -3.3636    1.7659   -0.3883
    3.2675   -1.7231    0.3804
   -1.7231    0.9122   -0.2021
    0.3804   -0.2021    0.0449
```

Published with MATLAB® R2023b