```
function x = forwsub1(L,B)
L = [1, 0, 0; 2, 1, 0; 9, 2, 1];
B = [1,2;1];
[n,~] = size(L);
x = zeros(n,1);
for i = 1:n
    sum = 0;
        for k = 1:i
        sum = sum+L(i,k)*x(k);
        end
    x(i) = (B(i)-sum)/L(i,i);
end
end
```

```
ans =

     1
     0
    -8
```

Published with MATLAB® R2021b

```
function X = forwsub(L,B)
L = [1, 0, 0; 2, 1, 0; 9, 2, 1];
B = [1,2,1]';
[n,~] = size(L);
[~,m] = size(B);
X = zeros(n,m);
for j = 1:m
    y = B(:,j);
    forwsub1(L,y);
    X(:,j) = forwsub1(L,y);
end
end
```

```
ans =

     1
     0
    -8
```

Published with MATLAB® R2021b

```
function X = backsub(U,B)
U = [2, 1, 2; 0, -3, 1; 0, 0, 5];
B = [1,2,1]';
[n,~] = size(U);
[~,m] = size(B);
x = zeros(n,1);
X = zeros(n,m);
for j = 1:m
    y = B(:,j);
    for i = n:-1:1
    sum = 0;
        for k = i+1:n
        sum = sum+U(i,k)*x(k);
        end
    x(i) = (y(i)-sum)/U(i,i);
    end
X(:,j) = x;
end
end
```

```
ans =

    0.6000
   -0.6000
    0.2000
```

Published with MATLAB® R2021b

```
function X = backsub(U,B)
U = [2, 1, 2; 0, -3, 1; 0, 0, 5];
B = eye(3);
[n,~] = size(U);
[~,m] = size(B);
x = zeros(n,1);
X = zeros(n,m);
for j = 1:m
    y = B(:,j);
    for i = n:-1:1
    sum = 0;
        for k = i+1:n
        sum = sum+U(i,k)*x(k);
        end
    x(i) = (y(i)-sum)/U(i,i);
    end
X(:,j) = x;
end
end
```

```
ans =

    0.5000    0.1667   -0.2333
         0   -0.3333    0.0667
         0         0    0.2000
```

Published with MATLAB® R2021b

```
function [X,row] = mygauss(A,B)
A = [0, 1, 4, 1; 1, 4, 1, 0; 0, 0, 1, 4; 4, 1, 0, 0];
B = [-6, 9, 39, 11]';
adj = [A,B];
[n,m] = size(adj);
row = (1:n)';
Scales = zeros(n,1);
for k = 1:n
    Scales(k) = max(abs(A(k,:)));
end
for j = 1:n-1
    d = zeros(n-j+1,1);
    for i = j:n
    d(i) = abs(A(i,j))/Scales(i);
    end
    [~,p] = max(d(j:n));
    P = p+j-1;
        temp = adj(j,:);
        adj(j,:) = adj(P,:);
        adj(P,:) = temp;
        t = row(j);
        row(j) = row(P);
        row(P) = t;
    for i = j+1:n
        s = adj(i,j)/adj(j,j);
        adj(i,:) = adj(i,:) - s*adj(j,:);
    end
end
U = adj(:,1:m-1);
b = adj(:,m);
X = backsub(U,b);
end
```

```
ans =

    2.0000
    3.0000
   -5.0000
   11.0000
```

Published with MATLAB® R2021b

```
function [X,row] = mygausspp(A,b)
A = [0, 1, 4, 1; 1, 4, 1, 0; 0, 0, 1, 4; 4, 1, 0, 0];
b = [-6, 9, 39, 11]';
adj = [A,b];
[n,m] = size(adj);
row = (1:n)';
Scales = zeros(n,1);
for k = 1:n
    Scales(k) = max(abs(A(k,:)));
end
for j = 1:n-1
    d = zeros(n-j+1,1);
    for i = j:n
    d(i) = abs(A(i,j))/Scales(i);
    end
    [~,p] = max(d(j:n));
    P = p+j-1;
        temp = adj(j,:);
        adj(j,:) = adj(P,:);
        adj(P,:) = temp;
        t = row(j);
        row(j) = row(P);
        row(P) = t;
    for i = j+1:n
        s = adj(i,j)/adj(j,j);
        adj(i,:) = adj(i,:) - s*adj(j,:);
    end
end
U = adj(:,1:m-1);
b = adj(:,m);
X = backsub(U,b);
end
```

```
ans =

    2.0000
    3.0000
   -5.0000
   11.0000
```

Published with MATLAB® R2021b

```
function [A,b] = Hilbert(n)
for n = 5,10,20
A = zeros(n,n);
b = zeros(n,1);
for i = 1:n
    sum = 0;
for j = 1:n
A(i,j) = 1/(i+j-1);
sum = sum + 1/(i+j-1);
end
b(i) = sum;
end
end
```

```
ans =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

Published with MATLAB® R2021b

Project 1 Discussion:

Problem 1: I struggled to get forwsub to solve for multiple right-hand sides with only one script. I ended up having to write forwsub1 to solve for individual right-hand sides, then I wrote the forwsub script to loop that process. For some reason I didn't have this problem with backsub.

My forwsub and backsub scripts gave the same answer as the MATLAB command "\" even when using multiple right-hand sides and finding the inverse of U as asked in part 4.

Problem 2: This problem gave me a lot more trouble. I was able to write an algorithm that works for most matrices, but there were still some problems that I couldn't solve. I had some trouble with the row swapping, it seems that the last row change operation doesn't work every time, which clearly can lead to incorrect solutions. For the Hilbert equation, I was able to write a script that returns both the Hilbert matrix and the corresponding right-hand side. Using mygauss and mygausspp both give the proper results of $x_i = 1$ for i = 1:n. For n = 5,10, and 20, the computational cost of doing standard gaussian elimination with row exchanges versus using pointers is not very high relative to the total cost of gaussian elimination. It takes three operations to swap two rows using pointers, while it takes 3n operations to swap two rows in in the original matrix. In any given matrix, to perform gaussian elimination, we may have to swap a maximum of n rows. Thus, the maximum cost of gaussian elimination without pointers is n^2 times the cost of using pointers. For n = 5,10,20, this is not a lot, but for larger matrices, this could be a problem. On the other hand, the actual process of using gaussian elimination is another factor of n more expensive than this row swapping, so the vast majority of the computational cost is still going to come from the process of gaussian elimination.