```
format long e
```

Q1

```
n = 5;  % You can choose other values for n
a = 1;
b = 10;
r = a + (b-a).*rand(n,1);
D = diag(r);
B = randn(n);
[Q, ~] = qr(B);
A = Q' * D * Q;

G = mychol(A)
```

```
G = 5×5
     2.704004053488558e+00     4.988998203369643e-02     9.590509139876384e-02 ···
                         0     2.226224383436023e+00     5.391436683726226e-01
                         0                         0     2.557896465648252e+00
                         0                         0                         0
                         0                         0                         0
```

```
G_builtin = chol(A)
```

```
G_builtin = 5×5
     2.704004053488558e+00     4.988998203369643e-02     9.590509139876384e-02 ···
                         0     2.226224383436023e+00     5.391436683726226e-01
                         0                         0     2.557896465648252e+00
                         0                         0                         0
                         0                         0                         0
```

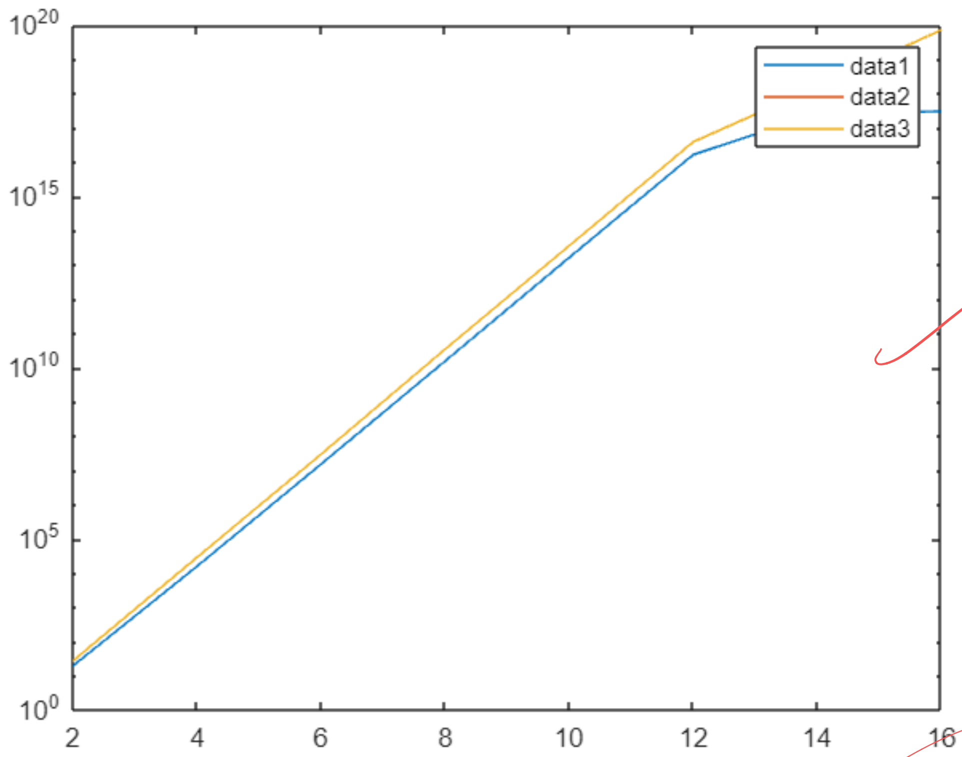
```
fprintf('Norm difference between custom and builtin Cholesky: %e\n', norm(G -
G_builtin));
```

```
Norm difference between custom and builtin Cholesky: 3.925231e-17
```

Q2

```
C = [];
C1 = [];
Cinf = [];
N = 2:2:16;
for n = N
    H = hilb(n);
    C = [C; cond(H)];
    C1 = [C1; cond(H, 1)];
    Cinf = [Cinf; cond(H, inf)];
end
figure;
semilogy(N,C);
hold on;
semilogy(N,C1);
```

```
semilogy(N,Cinf);
legend();
hold off;
```



Conjecture: Cond(H) varies exponentially with n till n = 12. For n > 12, Cond(H) takes almost a constant value with little variation.

1-norm and inf-norm also show very similar dependence on n.

Q3

```
n = 8;
H = hilb(n);
HI = invhilb(n);
x = rand(n,1);
b = H*x;
x1 = H\b;
x2 = HI*b;
x3 = gepp(H, b);
[x x1 x2 x3]
```

For n = 10, 12 ?

print the number of correct digits

```
ans = 8×4
    6.160446761466392e-01    6.160446761635532e-01    6.160446761678031e-01 ···
    4.732888489027293e-01    4.732888480144311e-01    4.732888480648398e-01
    3.516595070629968e-01    3.516595184753939e-01    3.516595233231783e-01
    8.308286278962909e-01    8.308285669285420e-01    8.308285176753998e-01
    5.852640911527243e-01    5.852642535895640e-01    5.852643251419067e-01
    5.497236082911395e-01    5.497233804087566e-01    5.497236847877502e-01
```

```
9.171936638298100e-01    9.171938248489369e-01    9.171939194202423e-01
2.858390188203735e-01    2.858389736630688e-01    2.858389914035797e-01
```

```
[cond(H) norm(x-x1)/norm(x) norm(x-x2)/norm(x) norm(x-x3)/norm(x)]
```

```
ans = 1×4
    1.525757553064797e+10    1.920755608287811e-07    2.158605517579707e-07 ···
```

As the cond(H) ~ 10^10, we can take t = 10 and s = 16.

Hence, s - t = 6.

Using Cholesky (H\b), invhilb and GEPP methods, the norm(x-x1)/norm(x) <= 10^ -(s-t). Also, looking at the arrays element-wise, we can see that the elements agree for atleast 6 places after the decimal.

Q4

```
n = 10;
H = hilb(n);
x = randn(n, 1);
b = H*x;
x1 = H\b;
r = H*x1 - b;
disp([norm(r)/norm(b) norm(x-x1)/norm(x)])
```

```
    1.482818332939167e-16    1.246074181472704e-05
```

Here, the norm(x-x1)/norm(x) > 10^ -6. This shows that having a small norm(r)/norm(b) does not imply norm(x − x1)/norm(x) to be small.

Q5

This is not wilkinson matrix.
See the Lab 1:

```
n_values = [32, 64];
results = [];

for n = n_values
    W = wilkinson(n);
    x = randn(n, 1);
    b = W * x;

    % GEPP solution
    x_gepp = gepp(W, b);
    error_gepp = norm(x - x_gepp, inf) / norm(x, inf);
    cond_W = cond(W, inf);
    r_gepp = norm(W * x_gepp - b, inf) / norm(b, inf);

    % QR decomposition solution
    [Q, R] = qr(W);
    x_qr = R \ (Q' * b);
    error_qr = norm(x - x_qr, inf) / norm(x, inf);
```

observation.

```
    r_qr = norm(W * x_qr - b, inf) / norm(b, inf);

    % Store results
    results = [results; n, error_gepp, r_gepp, error_qr, r_qr];
end

disp('n | GEPP Error | GEPP Residual | QR Error | QR Residual');
```

```
n | GEPP Error | GEPP Residual | QR Error | QR Residual
```

```
disp(results);
```

```
     3.200000000000000e+01     1.527125115052313e-16     1.304061606663360e-16     6.108500460209251e-16     5.21624
     6.400000000000000e+01     1.758073095716209e-16     1.180678356531078e-16     3.516146191432417e-16     2.36135
```

```
function [L, U, p, detP] = geppsolve(A)
    detP = 1;
    n = size(A, 1);
    p = (1:n)';

    for i = 1:n-1
        [~, max_row] = max(abs(A(i:n, i)));
        max_row = max_row + i - 1;

        if max_row ~= i
            detP = -detP;
            A([i, max_row], :) = A([max_row, i], :);
            p([i, max_row]) = p([max_row, i]);
        end

        if A(i, i) == 0
            warning('Matrix is singular or nearly singular. Pivoting failed.');
            break; % If no pivot is found, exit the loop gracefully
        end
```

continue

should not exit,

```
        A(i+1:n, i) = A(i+1:n, i) / A(i, i);
        A(i+1:n, i+1:n) = A(i+1:n, i+1:n) - A(i+1:n, i) * A(i, i+1:n);
    end

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

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

    % PA = LU, Ax = b => LUX = Pb => Ly = Pb, Ux = y
    b = b(p);
    y = rowforward(L, b);
    x = colbackward(U, y);
```

```
end

function x = colbackward(U,b)
    n = size(b, 1);
    x = zeros(n, 1);
    for i = n:-1:1
        if U(i, i) == 0
            error('Provided U is singular');
        end
        x(i) = b(i) / U(i, i);
        b(1 : i - 1) = b(1 : i - 1) - U(1 : i-1, i)*x(i);
    end
end

function x = rowforward(L,b)
    n = size(b, 1);
    x = zeros(n, 1);
    for i = 1:n
        if L(i, i) == 0
            error('Provided L is singular');
        end
        x(i) = (b(i) - L(i, 1:i-1)*x(1:i-1)) / L(i, i);
    end
end

function G = mychol(A)
    [n, m] = size(A);
    if n ~= m
        error('Matrix must be square');
    end

    G = zeros(n, n);

    for j = 1:n
        G(j, j) = sqrt(A(j, j) - sum(G(1:j-1, j).^2));
        for i = j+1:n
            G(j, i) = (A(j, i) - sum(G(1:j-1, j) .* G(1:j-1, i))) / G(j, j);
        end
    end
end
```

It is not recursive & not outer product