Ejercicio 1

a)

```
function[Q,R] = qr1(A)
    % Tamaño de la matriz A
    [m, n] = size(A); %m filas y n columnas

    % Inicializar Q con ceros y tamaño como A
    Q = zeros(m, n);

    % Inicializar R de n x n
    R = zeros(n, n);

    %Ciclo que recorre las columnas
    for j = 1:n
        % Tomar la columna j de A y la asigna a v_j
        vj = A(:, j);

        % Bucle interno para calcular las proyecciones
        % Note que cuando j=1 el ciclo va de 1:0, es decir no se ejecuta.
        for i = 1:j-1

            % Calcular r_ij, es producto interno
            R(i, j) = Q(:, i)' * A(:, j);

            % Modifica v_j restando la proyección de a_j en q_i
            vj = vj - R(i, j) * Q(:, i);
        end

        % Paso 3: calcula r_jj como la norma de v_j y normaliza para obtener q_j
        R(j, j) = norm(vj);
        Q(:, j) = vj / R(j, j);
    end
end
```

b)

```
function[Q,R] = qr2(A)
    % Tamaño de la matriz A
    [m, n] = size(A); %m filas y n columnas

    % Inicializar Q con ceros y tamaño como A
    Q = zeros(m, n);

    % Inicializar R de n x n
    R = zeros(n, n);
```

```
    % Definir la matriz A igual a una V
    V = A;

    % Ciclo for que recorre columnas de v
    for i = 1:n

        % Llenar el valor r_ii
        R(i, i) = norm(V(:, i));

        % Llenar la columna q_i
        Q(:,i) = V(:, i)/R(i, i);

        % Ciclo for interno de las proyecciones
        for j = i+1:n

            % Llenar la fila de la triangular
            R(i, j) = Q(:, i)' * V(:, j);

            % Modifica v_j restando la proyección de a_j en q_i
            V(:, j) = V(:,j) - R(i, j) * Q(:, i);
        end
    end
end
```

c)

```
function [Q, R] = qr3(A)
    [m, n] = size(A);        % Tamaño de la matriz A
    R = A;                   % Inicializamos R como una copia de A
    M = eye(m);              % Matriz identidad de m x m para construir Q

    for j = 1:n
        % Selecciona el vector x desde la fila j hasta m de la columna j de R
        x = R(j:m, j);

        % Construye el vector de Householder vj
        e1 = zeros(length(x), 1);
        e1(1) = 1;
        vj = sign(x(1)) * norm(x) * e1 + x;
        vj = vj / norm(vj);  % Normaliza vj

        % Actualiza R aplicando la transformación de Householder
        R(j:m, j:n) = R(j:m, j:n) - 2 * vj * (vj' * R(j:m, j:n));

        % Actualiza M aplicando la misma transformación para construir Q
        M(j:m, :) = M(j:m, :) - 2 * vj * (vj' * M(j:m, :));
    end

    % La matriz Q es la transpuesta de M
```

```
    Q = M';
end
```

d)

Cálculo de la factorización implementando los 3 algoritmos

```
format long

% Definición de dimensiones
m = 20; % m = n

% Generar matriz aleatoria
A = rand(m);

% Generar matriz identidad mxm
I = eye(m);

% Algoritmo 1
[Q1,R1] = qr1(A)
```

```
Q1 = 20×20
   0.253706085111658   0.116419547947744  -0.211415383325299   0.015355156192298 ···
   0.282064864590495  -0.204263719273224   0.024870624014221  -0.028043526227370
   0.039543870591570   0.377637722140463   0.295623542495380   0.123253731198064
   0.284426507497794   0.226222975638028  -0.026573427342612  -0.106059646590977
   0.196917545694853   0.172071920465345  -0.415399352365584   0.080101369387014
   0.030374217303243   0.340851500850871   0.068028028883216   0.459981408352025
   0.086724731340670   0.289579570203147  -0.041129228969524   0.017716265227491
   0.170299663032571   0.055074157060259   0.251170228132287  -0.102114764168137
   0.298168955613811   0.081384858195125   0.015483824596259  -0.434651490381349
   0.300467627151315  -0.153508085362657   0.318981686990089   0.025608723789521
     ⋮

R1 = 20×20
   3.211289496799465   1.631233712087509   2.038904546911871   1.914408311036381 ···
                   0   2.077716194786124   1.064101540049071   1.169928269853722
                   0                   0   0.957453274857126  -0.526927976771277
                   0                   0                   0   1.170083891092091
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
     ⋮
```

```
% Algoritmo 2
[Q2,R2] = qr2(A)
```

```
Q2 = 20×20
   0.253706085111658   0.116419547947744  -0.211415383325300   0.015355156192298 ···
   0.282064864590495  -0.204263719273224   0.024870624014221  -0.028043526227370
   0.039543870591570   0.377637722140463   0.295623542495380   0.123253731198064
   0.284426507497794   0.226222975638028  -0.026573427342612  -0.106059646590977
   0.196917545694853   0.172071920465345  -0.415399352365584   0.080101369387013
```

```
   0.030374217303243   0.340851500850871   0.068028028883216   0.459981408352025
   0.086724731340670   0.289579570203147  -0.041129228969524   0.017716265227491
   0.170299663032571   0.055074157060259   0.251170228132287  -0.102114764168137
   0.298168955613811   0.081384858195125   0.015483824596259  -0.434651490381349
   0.300467627151315  -0.153508085362657   0.318981686990089   0.025608723789521
      ⋮
      ⋮
R2 = 20×20
   3.211289496799465   1.631233712087509   2.038904546911871   1.914408311036381 ···
                   0   2.077716194786124   1.064101540049071   1.169928269853722
                   0                   0   0.957453274857126  -0.526927976771278
                   0                   0                   0   1.170083891092091
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
      ⋮
```

```
% Algoritmo 3
[Q3,R3] = qr3(A)
```

```
Q3 = 20×20
  -0.253706085111658   0.116419547947744   0.211415383325299   0.015355156192298 ···
  -0.282064864590495  -0.204263719273224  -0.024870624014221  -0.028043526227370
  -0.039543870591570   0.377637722140463  -0.295623542495380   0.123253731198064
  -0.284426507497794   0.226222975638029   0.026573427342612  -0.106059646590977
  -0.196917545694853   0.172071920465345   0.415399352365584   0.080101369387013
  -0.030374217303243   0.340851500850871  -0.068028028883216   0.459981408352025
  -0.086724731340670   0.289579570203147   0.041129228969524   0.017716265227491
  -0.170299663032571   0.055074157060259  -0.251170228132287  -0.102114764168137
  -0.298168955613811   0.081384858195125  -0.015483824596259  -0.434651490381349
  -0.300467627151315  -0.153508085362657  -0.318981686990089   0.025608723789521
      ⋮
      ⋮
R3 = 20×20
  -3.211289496799465  -1.631233712087509  -2.038904546911871  -1.914408311036381 ···
                   0   2.077716194786125   1.064101540049071   1.169928269853721
                   0  -0.000000000000000  -0.957453274857126   0.526927976771278
  -0.000000000000000  -0.000000000000000                   0   1.170083891092090
                   0  -0.000000000000000                   0   0.000000000000000
                   0  -0.000000000000000                   0   0.000000000000000
                   0  -0.000000000000000                   0   0.000000000000000
                   0  -0.000000000000000                   0  -0.000000000000000
   0.000000000000000  -0.000000000000000                   0  -0.000000000000000
                   0   0.000000000000000                   0  -0.000000000000000
      ⋮
      ⋮
```

Cálculo de las normas con los Q y R de los algoritmos

Para el algoritmo 1:

Cálculo de $\|A - Q_1 R_1\|_2$

```
disp(norm(A - Q1 * R1))
```

```
    7.520006994808369e-16
```

Cálculo de $\|Q_1Q_1^T - I\|_2$

```
disp(norm(Q1 * Q1' - I))
```

```
3.010556258530773e-14
```

Para el algoritmo 3:

Cálculo de $\|A - Q_2R_2\|_2$

```
disp(norm(A - Q2 * R2))
```

```
7.612339524839162e-16
```

Cálculo de $\|Q_2Q_2^T - I\|_2$

```
disp(norm(Q2 * Q2' - I))
```

```
2.687249457186055e-15
```

Para el algoritmo 2:

Cálculo de $\|A - Q_3R_3\|_2$

```
disp(norm(A - Q3 * R3))
```

```
6.098700635199346e-15
```

Cálculo de $\|Q_3Q_3^T - I\|_2$

```
disp(norm(Q3 * Q3' - I))
```

```
1.201349042010106e-15
```

¿Se cumple que A = QR?

En los 3 algoritmos se obtuvo que la difencia $\|A - QR\|_2$ es extremadamente pequeña, siendo casi nivel de precisión de máquina, por lo que se puede concluir que si satisfacen la propiedad.

¿Se cumple que Q es ortogonal?

De manera similar se obtuvo que la diferencia $\|QQ^T - I\|_2$ en los 3 algoritmos fue sumamente baja a nivel de precisión de máquina, por lo que se puede concluir que las matrices Q generadas son ortogonales.

Determinar si algún algoritmo es mejor

```
%Función para ver el error pormedio de los 3 algoritmos y así hacer una
%comparación
function error_promedio = media(num_alg, A)

    %num_alg es 1 usa la funcion qr1, si es 2 usa qr2 y si es 3 usa qr3
```

```
    % Inicializar el vector para almacenar los resultados
    resultados = zeros(1, 100);

    for i = 1:100
        % Seleccionar la función correspondiente según el valor de num_alg
        if num_alg == 1
            [Q, R] = qr1(A);
        elseif num_alg == 2
            [Q, R] = qr2(A);
        elseif num_alg == 3
            [Q, R] = qr3(A);
        elseif num_alg == 4
            [Q, R] = qr(A);
        else
            error('El valor de num_alg debe ser 1, 2 o 3');
        end

        % Calcular el producto Q * R y almacenar el resultado en el vector
        resultados(i) = norm(A - Q * R, 'fro'); % Se guarda la norma de la
diferencia
    end
    error_promedio = mean(resultados);
end

% Mostrar errores promedio

fprintf('El promedio del la diferencia Q*R - A en el algoritmo 1: %.20f\n',
media(1, A));
```

```
El promedio del la diferencia Q*R - A en el algoritmo 1: 0.00000000000000167513
```

```
fprintf('El promedio del la diferencia Q*R - A en el algoritmo 2: %.20f\n',
media(2, A));
```

```
El promedio del la diferencia Q*R - A en el algoritmo 2: 0.00000000000000155780
```

```
fprintf('El promedio del la diferencia Q*R - A en el algoritmo 3: %.20f\n',
media(3, A));
```

```
El promedio del la diferencia Q*R - A en el algoritmo 3: 0.00000000000000701755
```

Del algoritmo 1 y 2 no fue posible determinar cual es mejor, pues a pesar de hacer 100 iteraciones y obtener el promedio siguen sin mostrar una superioridad, no obstante con estos promedios si es posible notar que el algoritmo 3 es menos preciso que el 1 y el 2. Por lo tanto estos son mejores.

e)

Cálculo de la factorización implementando los 3 algoritmos con A como la matriz de Hilbert

```
% Definición de dimensiones
```

```
m = 20; % m = n

% Generar matriz aleatoria
A = hilb(m);

% Generar matriz identidad mxm
I = eye(m);

% Algoritmo 1
[Q1,R1] = qr1(A)
```

```
Q1 = 20×20
   0.791519005081713  -0.556499824361258   0.236121645758677  -0.084946081139037 ···
   0.395759502540857   0.201480277642668  -0.613359598328218   0.539945879756187
   0.263839668360571   0.294230248369544  -0.335343759038868  -0.173296594888645
   0.197879751270428   0.292632214750653  -0.123193797273439  -0.358717084265336
   0.158303801016343   0.272484186986386   0.006149901184197  -0.339140494012994
   0.131919834180286   0.249914450575479   0.083641604827828  -0.259732346984270
   0.113074143583102   0.228898004263368   0.130263976681730  -0.171843039373418
   0.098939875635214   0.210280132685099   0.158249825451403  -0.092105270084689
   0.087946556120190   0.194022787421174   0.174711181267072  -0.024690899394759
   0.079151900508171   0.179853928931674   0.183883052754369   0.030571435821536
     ⋮
     ⋮
R1 = 20×20
   1.263393542770036   0.753827623887346   0.556804494916790   0.447678262802663 ···
                   0   0.173708753589238   0.200397760350369   0.199920969280484
                   0                   0   0.017505010673033   0.029825115652517
                   0                   0                   0   0.001556567066944
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
     ⋮
     ⋮
```

```
% Algoritmo 2
[Q2,R2] = qr2(A)
```

```
Q2 = 20×20
   0.791519005081713  -0.556499824361258   0.236121645758666  -0.084946081140209 ···
   0.395759502540857   0.201480277642668  -0.613359598328214   0.539945879759436
   0.263839668360571   0.294230248369544  -0.335343759038862  -0.173296594886906
   0.197879751270428   0.292632214750653  -0.123193797273434  -0.358717084264728
   0.158303801016343   0.272484186986386   0.006149901184202  -0.339140494013070
   0.131919834180286   0.249914450575479   0.083641604827832  -0.259732346984758
   0.113074143583102   0.228898004263368   0.130263976681735  -0.171843039374152
   0.098939875635214   0.210280132685099   0.158249825451407  -0.092105270085565
   0.087946556120190   0.194022787421174   0.174711181267076  -0.024690899395729
   0.079151900508171   0.179853928931674   0.183883052754372   0.030571435820516
     ⋮
     ⋮
R2 = 20×20
   1.263393542770036   0.753827623887346   0.556804494916790   0.447678262802663 ···
                   0   0.173708753589238   0.200397760350368   0.199920969280484
                   0                   0   0.017505010673033   0.029825115652525
                   0                   0                   0   0.001556567066944
                   0                   0                   0                   0
                   0                   0                   0                   0
```

```
                0                   0                   0                   0
                0                   0                   0                   0
                0                   0                   0                   0
                0                   0                   0                   0
    ⋮
```

```
% Algoritmo 3
[Q3,R3] = qr3(A)
```

```
Q3 = 20×20
  -0.791519005081713   0.556499824361259   0.236121645758672  -0.084946081140241 ···
  -0.395759502540857  -0.201480277642667  -0.613359598328209   0.539945879759388
  -0.263839668360571  -0.294230248369544  -0.335343759038861  -0.173296594886909
  -0.197879751270428  -0.292632214750653  -0.123193797273431  -0.358717084264757
  -0.158303801016343  -0.272484186986386   0.006149901184204  -0.339140494013110
  -0.131919834180286  -0.249914450575478   0.083641604827833  -0.259732346984771
  -0.113074143583102  -0.228898004263368   0.130263976681737  -0.171843039374181
  -0.098939875635214  -0.210280132685099   0.158249825451409  -0.092105270085592
  -0.087946556120190  -0.194022787421174   0.174711181267077  -0.024690899395738
  -0.079151900508171  -0.179853928931674   0.183883052754374   0.030571435820492
     ⋮
R3 = 20×20
  -1.263393542770036  -0.753827623887346  -0.556804494916790  -0.447678262802663 ···
                   0  -0.173708753589238  -0.200397760350368  -0.199920969280484
                   0  -0.000000000000000   0.017505010673033   0.029825115652525
                   0  -0.000000000000000                   0   0.001556567066944
                   0  -0.000000000000000                   0  -0.000000000000000
                   0  -0.000000000000000                   0  -0.000000000000000
                   0  -0.000000000000000                   0  -0.000000000000000
                   0  -0.000000000000000                   0  -0.000000000000000
                   0  -0.000000000000000                   0  -0.000000000000000
     ⋮
```

Factorización con la funcion qr de MATLAB

```
[Q_m, R_m] = qr(A)
```

```
Q_m = 20×20
  -0.791519005081713   0.556499824361259   0.236121645758672  -0.084946081140241 ···
  -0.395759502540857  -0.201480277642667  -0.613359598328210   0.539945879759392
  -0.263839668360571  -0.294230248369544  -0.335343759038859  -0.173296594886920
  -0.197879751270428  -0.292632214750652  -0.123193797273431  -0.358717084264759
  -0.158303801016343  -0.272484186986386   0.006149901184205  -0.339140494013109
  -0.131919834180286  -0.249914450575478   0.083641604827834  -0.259732346984767
  -0.113074143583102  -0.228898004263368   0.130263976681737  -0.171843039374179
  -0.098939875635214  -0.210280132685099   0.158249825451409  -0.092105270085590
  -0.087946556120190  -0.194022787421174   0.174711181267077  -0.024690899395736
  -0.079151900508171  -0.179853928931674   0.183883052754374   0.030571435820500
     ⋮
R_m = 20×20
  -1.263393542770036  -0.753827623887346  -0.556804494916789  -0.447678262802663 ···
                   0  -0.173708753589238  -0.200397760350368  -0.199920969280484
                   0                   0   0.017505010673033   0.029825115652525
                   0                   0                   0   0.001556567066944
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
```

```
         0         0         0         0
         0         0         0         0
    ⋮
```

Cálculo de las normas con los Q y R de los algoritmos

Para el algoritmo 1:

Cálculo de $\|A - Q_1 R_1\|_2$

```
fprintf('La norma es: %.20f\n', norm(A - Q1 * R1))
```

```
La norma es: 0.00000000000000008464
```

Cálculo de $\|Q_1 Q_1^T - I\|_2$

```
fprintf('La norma es: %.20f\n', norm(Q1 * Q1' - I))
```

```
La norma es: 12.97161351231601145173
```

Para el algoritmo 2:

Cálculo de $\|A - Q_2 R_2\|_2$

```
fprintf('La norma es: %.20f\n', norm(A - Q2 * R2))
```

```
La norma es: 0.00000000000000008098
```

Cálculo de $\|Q_2 Q_2^T - I\|_2$

```
fprintf('La norma es: %.20f\n', norm(Q2 * Q2' - I))
```

```
La norma es: 0.99895633841753628257
```

Para el algoritmo 3:

Cálculo de $\|A - Q_3 R_3\|_2$

```
fprintf('La norma es: %.20f\n', norm(A - Q3 * R3))
```

```
La norma es: 0.00000000000000041743
```

Cálculo de $\|Q_3 Q_3^T - I\|_2$

```
fprintf('La norma es: %.20f\n', norm(Q3 * Q3' - I))
```

```
La norma es: 0.00000000000000113713
```

Para el algoritmo MATLAB:

Cálculo de $\|A - Q_m R_m\|_2$

```
fprintf('La norma es: %.20f\n', norm(A - Q_m * R_m))
```

```
La norma es: 0.00000000000000042275
```

Cálculo de $\|Q_m Q_m^T - I\|_2$

```
fprintf('La norma es: %.20f\n', norm(Q_m * Q_m' - I))
```

```
La norma es: 0.00000000000000135288
```

Determinar si algún algoritmo es mejor

Al analizar la precisión de los algoritmos para calcular el producto QRQR y aproximarlo a la matriz original AA, se observa que los algoritmos 2 y 1 son los más exactos, logrando hasta 17 cifras decimales de precisión. Sin embargo, ambos algoritmos presentan un desempeño deficiente en la construcción de matrices QQ ortogonales, lo que compromete su calidad general, ya que estas matrices están lejos de ser realmente ortogonales.

Por otro lado, el algoritmo 3 y la función nativa de MATLAB muestran un rendimiento superior en términos globales. El algoritmo 3, en particular, ofrece una buena precisión en ambas métricas y se destaca como la mejor opción para esta matriz específica, seguido de cerca por la función de MATLAB. En comparación, los algoritmos 2 y 1 quedan por detrás en el orden de preferencia, esto por la poca precisión en la matriz ortogonal.

Ejercicio 2

Definiciones

```
m = 12;
x = ones(m, 1);
A = hilb(m);
b = A * x;
```

a) Cálculo del número de condición de A

```
cond_A = cond(A);
disp(cond_A)
```

```
    1.622382674081028e+16
```

El número de condición de A proporciona una medida de sensibilidad de la solución del sistema a pequeños cambios en los datos de entrada. Entre más elevado el numero de condición que indica la matriz A más mal

condicionada está, es decir, que pequeños errores durante los cálculos o los datos pueden provocar mayores errores en la solución aproximada del sistema.

En este caso, la matriz A está dando un número de condición extremadamente alto, entonces incluso errores numéricos pequeños debido a la precisión limitada de las operaciones, a pesar de tener bastante exactitud en decimales, los errores pueden amplificarse significativamente en la solución.

b)

Cálculo de la factorización PA = LU

```
[L, U, P] = lu(A);
```

El sistema $Ax = b$ se convierte en $PAx = Pb$. Sustuituyento $PA = LU$ se tiene que $LUx = Pb$, despejamos Ux y se tiene:

```
Ux = L \ (P * b);
```

Tomando $Ux = y$ y usando la barra "\", MATLAB busca la solución de $Ux = y$ usando una operación optimizada para matrices triangulares superiores como U, entonces:

```
y = Ux;
x_hat = U \ y;

% Cálculo de la norma L2 de la diferencia entre x y x_hat
error_norm = norm(x - x_hat, 2);

% Mostrar resultados
disp(x_hat);
```

```
   0.999999965056470
   1.000004409407701
   0.999861743769333
   1.001879710648280
   0.986241982741676
   1.060375974230446
   0.831939173215308
   1.303973363208347
   0.643862006187990
   1.260684018118645
   0.891666923800637
   1.019510752834270
```

```
disp(error_norm);
```

```
   0.575664393204197
```

Calcule la factorización A = QR

```
% Factorización QR
[Q, R] = qr(A);
```

```
% Resolución del sistema
x_hat = R \ (Q' * b);

% Cálculo de la norma 2 de la diferencia entre x y x_hat
error_norm = norm(x - x_hat, 2);
disp(x_hat);
```

```
   0.999999959388693
   1.000005283186343
   0.999830403310794
   1.002349309075760
   0.982541671473555
   1.077584665633242
   0.781738775313878
   1.398351690179006
   0.529638931676855
   1.346635837865845
   0.855086526655507
   1.026236980262808
```

```
disp(error_norm);
```

```
   0.758772683659291
```

Calcule la factorización $A = LL^T$

```
% Factorización de Cholesky
L = chol(A, 'lower');

% Resolución del sistema usando LL^T
y = L \ b;               % Resuelve Ly = b para obtener y
x_hat = L' \ y;          % Resuelve L^T x_hat = y para obtener x_hat

% Cálculo de la norma L2 de la diferencia entre x y x_hat
error_norm = norm(x - x_hat, 2);

disp(x_hat);
```

```
   0.999999959226938
   1.000005157294675
   0.999838010235511
   1.002205336511813
   0.983841672808578
   1.070969584044129
   0.802313840856519
   1.357764261676641
   0.580634574173165
   1.307093753268461
   0.872333978007935
   1.022999899390717
```

```
disp(error_norm);
```

```
   0.677775250243075
```

Calcule la factorizaci´on $A = USV^T$

```
% Factorización SVD
[U, S, V] = svd(A);

% Resolución
x_hat = V * (S \ (U' * b));
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  6.163774e-17.
```

```
% Cálculo de la norma L2 de la diferencia entre x y x_hat
error_norm = norm(x - x_hat, 2);
disp(x_hat);
```

```
   0.999999916894989
   1.000010516060504
   0.999669678757335
   1.004496252243775
   0.967065745818753
   1.144601092073345
   0.597366584601017
   1.728382796746141
   0.146537052529202
   1.624734728068396
   0.740380238209129
   1.046755453281029
```

```
disp(error_norm);
```

```
   1.379474758199625
```

f) ¿Algún método brinda alguna solución aceptable en térrminos del error?

Considero que ningún método brinda alguna solución aceptable, se entiende que el número de condición es alto lo que sube la dificultad, no obstante ningun método brinda resultados aceptables

g) Factorización de valores singulares

Utilizando la factorización SVD con rango 9 y despejando se tiene:

```
A = hilb(m); % Definiendo de nuevo la matriz A como la de Hilbert
[U, S, V] = svd(A);
nu = 9;

% Creación de la sumatoria
A_nu = zeros(size(A));
for i = 1:nu
    A_nu = A_nu + S(i, i) * (U(:, i) * V(:, i)'); % S(i,i) sería el escalar de la
matriz diagonal
end


x_hat = A_nu \ b; % Solución del sistema
```

```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  4.857866e-19.
```

```
disp(x_hat)
```

```
   1.000066517126225
   0.996354037779178
   1.030367790041151
   1.201901178557731
  -2.602878583222456
  19.579875661058018
 -46.805934171007962
  66.981452741489136
 -42.373475968967931
   3.434593940238302
  13.255680904913740
  -3.698134105981338
```

```
% Cálculo de la norma
norm_dif = norm(x - x_hat, 2);
disp(norm_dif)
```

```
  95.166451572480653
```

Nótese que que el $9 = \nu < Rango(A) = 12$, por lo tanto la aproximación desde el inicio va a ser menos precisa que si tomara el Rango de la matriz A. Además al realizar la factorización se indica que un warning y esto se puede deber a que la matriz de Hilbert es extremadamente mal condicionada, por lo tanto un error tan grande como el obtenido de la norma se debe a la acumulación de errores de redondeo y a la falta de precisión en los cálculos de punto flotante.

Ejercicio 3:

a)

Definir los vectores:

```
x = linspace(0,1,100);
y = linspace(0,2,200);
```

Generar malla de puntos:

```
[xx,yy]=meshgrid(x,y);
```

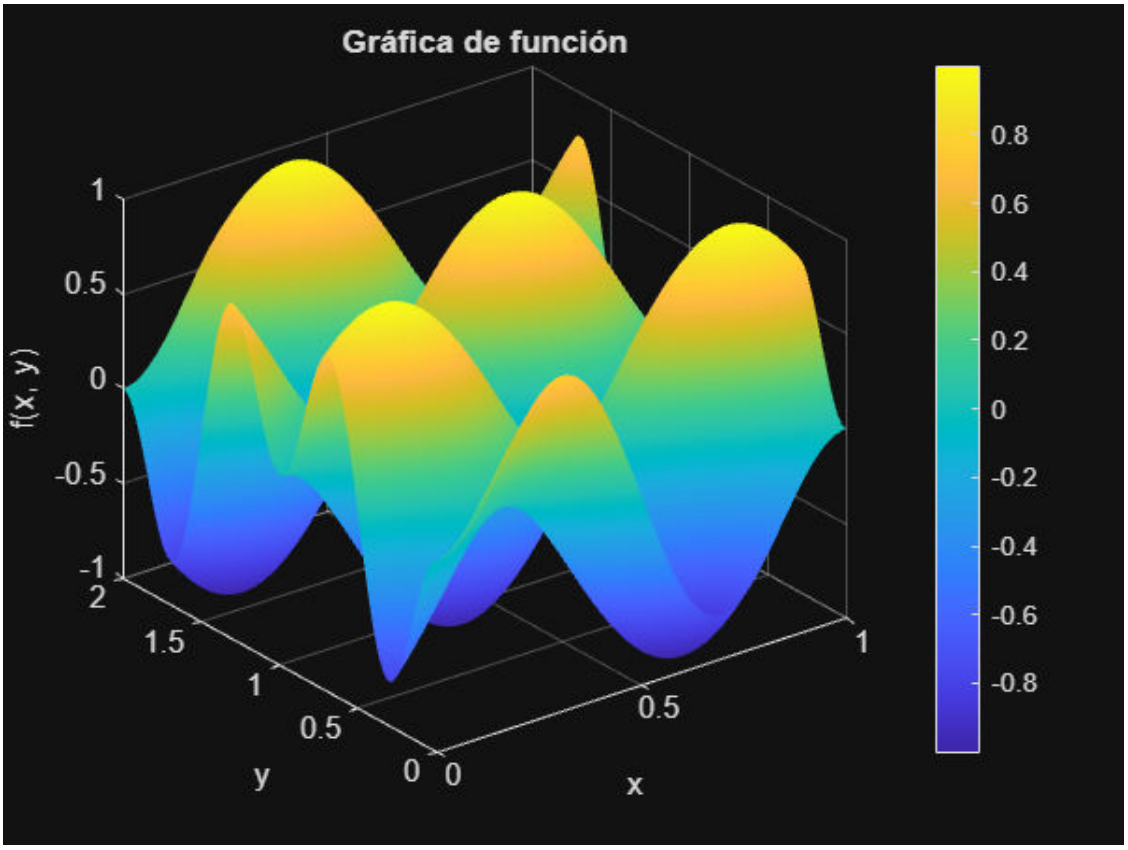
Crear función:

```
f = sin(2 * pi * (xx + yy)) .* sin(pi * (xx - yy));
```

Graficar la función usando surf:

```
surf(xx, yy, f);
colorbar;        % Para agregar colores de niveles
shading interp;  % Para que no se vea la maya
view(3);         % Ajustar la vista en 3D
xlabel('x');
ylabel('y');
```

```
zlabel('f(x, y)');
title('Gráfica de función');
```



b)

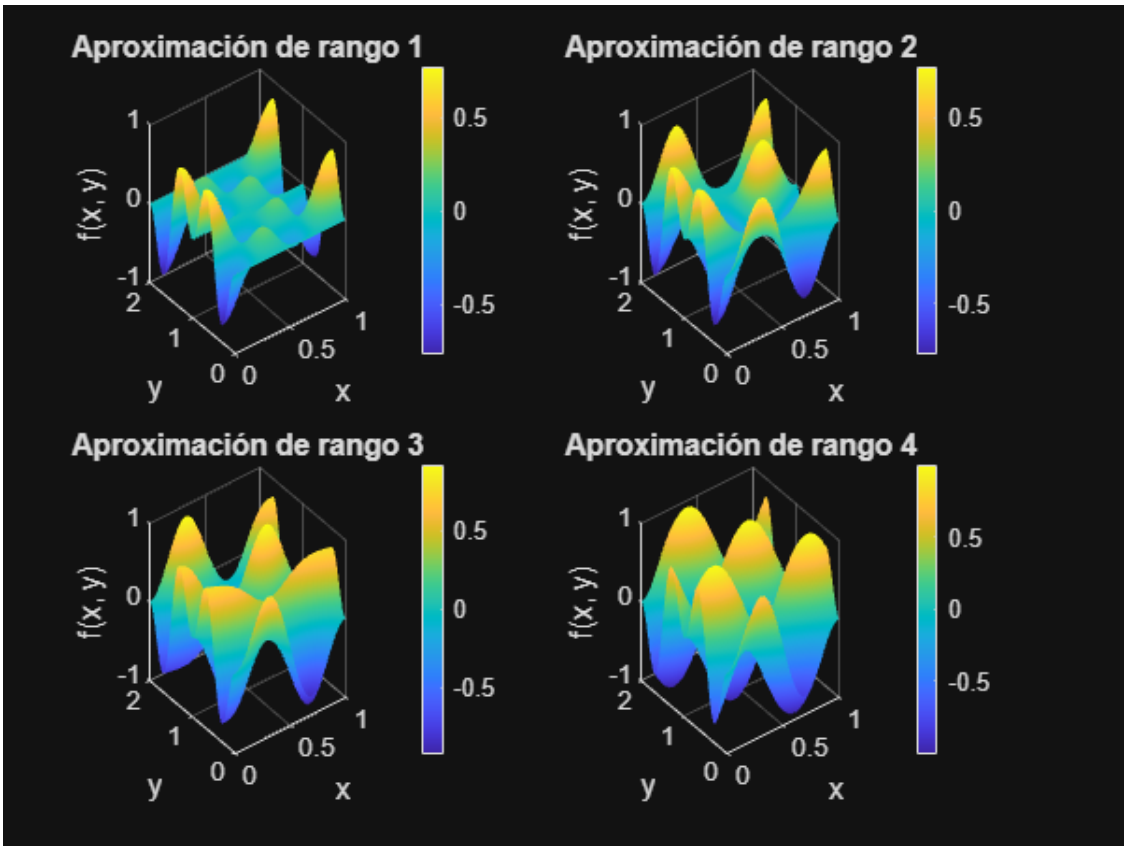
Calcular la función f(x, y) en la malla y asignarla a zz de esta manera, ya que zz=f(xx,yy) me da problemas

```
zz = sin(2 * pi * (xx + yy)) .* sin(pi * (xx - yy));
```

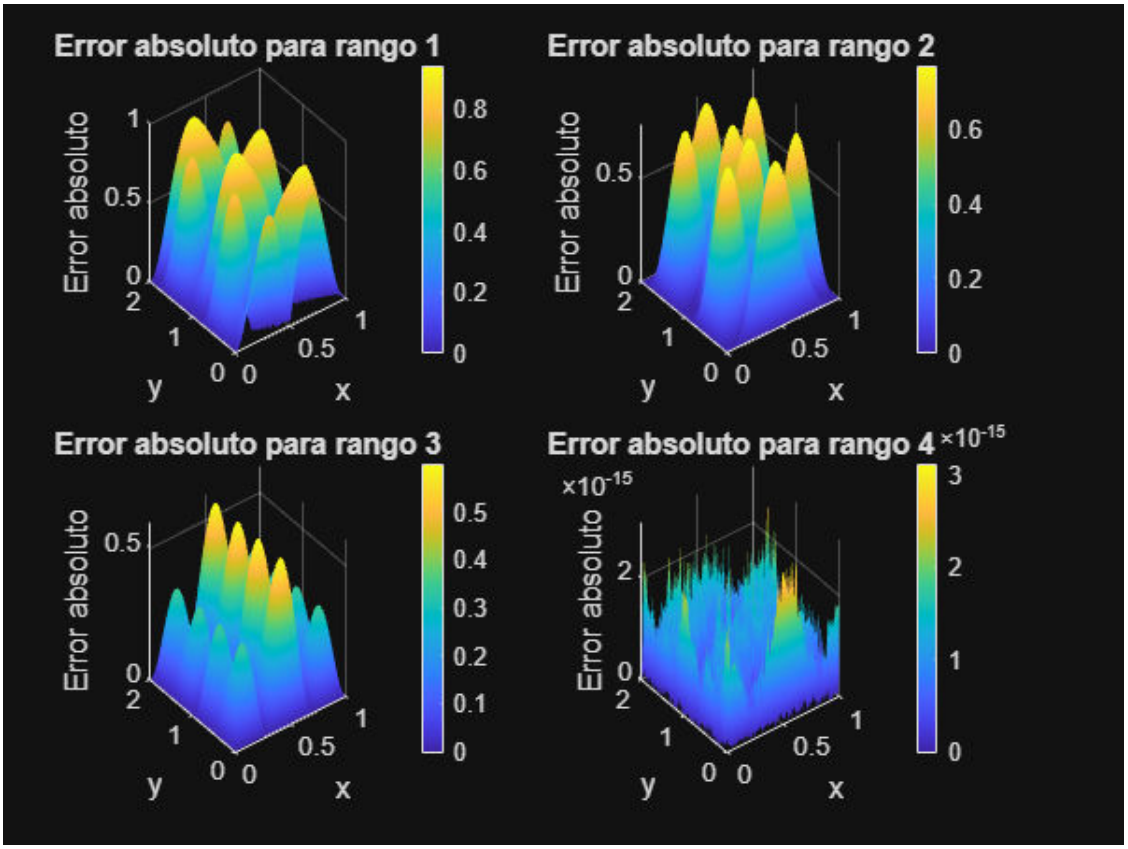
Obtener la descomposición en valores singulares de zz

```
[U, S, V] = svd(zz);

% Crear subgráficas de las mejores aproximaciones 4
figure;
for k = 1:4
    % Aproximación de rango k
    zz_k = U(:, 1:k) * S(1:k, 1:k) * V(:, 1:k)';

    subplot(2, 2, k);
    surf(xx, yy, zz_k);
    colorbar;
    shading interp;
    view(3);
    xlabel('x');
    ylabel('y');
```

```
    zlabel('f(x, y)');
    title(['Aproximación de rango ', num2str(k)]);
end
```




```
% Graficar el error absoluto para cada aproximación
figure;
for k = 1:4
    % Error absoluto
    zz_k = U(:, 1:k) * S(1:k, 1:k) * V(:, 1:k)';
    error_abs = abs(zz - zz_k);

    subplot(2, 2, k);
    surf(xx, yy, error_abs);
    colorbar;
    shading interp;
    view(3);
    xlabel('x');
    ylabel('y');
    zlabel('Error absoluto');
    title(['Error absoluto para rango ', num2str(k)]);
end
```



c)

Mostrar los valores singulares

```
% Se usa un for para que muestre mas decimales
for i = 1:min(size(S))
    fprintf('%20.16f\n', S(i, i));
end
```

```
 35.7919334487535039
 35.4409720521319684
 35.0900626958687667
 35.0900626958687667
  0.0000000000000318
  0.0000000000000190
  0.0000000000000178
  0.0000000000000167
  0.0000000000000155
  0.0000000000000146
  0.0000000000000127
  0.0000000000000121
  0.0000000000000119
  0.0000000000000109
  0.0000000000000103
  0.0000000000000097
  0.0000000000000095
  0.0000000000000088
  0.0000000000000085
  0.0000000000000080
  0.0000000000000077
  0.0000000000000076
```

0.0000000000000075
0.0000000000000073
0.0000000000000065
0.0000000000000061
0.0000000000000059
0.0000000000000057
0.0000000000000053
0.0000000000000047
0.0000000000000046
0.0000000000000045
0.0000000000000037
0.0000000000000037
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035

```
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000035
0.0000000000000029
0.0000000000000029
0.0000000000000021
0.0000000000000014
0.0000000000000014
0.0000000000000011
0.0000000000000004
```

Para ver el rango de zz

```
disp(rank(zz))
```

```
     4
```