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
   0.183242217753921  -0.025636422065451  -0.352651470586100   0.331938685304752 ···
   0.186936712599269   0.170718576649507  -0.126336909281979  -0.119337435101800
   0.351455863958444  -0.345029547841881   0.139000384476336   0.236081624135451
   0.146174042328889  -0.125390940897712   0.184405722334473  -0.022778430899064
   0.043987639723182   0.416720141116407   0.171009863410556  -0.235774747158227
   0.090278285129134   0.343957992582164   0.117258269046658   0.344344955034490
   0.257180976423353   0.200041634326475  -0.330236401481578   0.171706692304328
   0.315137081639972   0.057117338632047   0.115556410384379  -0.260275646161804
   0.364286245413978  -0.435277064287946   0.188008570206007   0.084467495340759
   0.080795509986395   0.182984990057412   0.198275182997108   0.375904713439168
     ⋮
R1 = 20×20
   2.663138246670700   2.394600405738687   2.498543055851519   1.424680701228067 ···
                   0   1.919334918682500   0.367524729256336   0.517527290179470
                   0                   0   1.232834742595248   0.126573617405416
                   0                   0                   0   1.050808871544815
                   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
   0.183242217753921  -0.025636422065451  -0.352651470586100   0.331938685304752 ···
   0.186936712599269   0.170718576649507  -0.126336909281979  -0.119337435101800
   0.351455863958444  -0.345029547841881   0.139000384476336   0.236081624135450
   0.146174042328889  -0.125390940897712   0.184405722334473  -0.022778430899064
   0.043987639723182   0.416720141116407   0.171009863410557  -0.235774747158227
```

```
   0.090278285129134   0.343957992582164   0.117258269046658   0.344344955034490
   0.257180976423353   0.200041634326475  -0.330236401481578   0.171706692304329
   0.315137081639972   0.057117338632047   0.115556410384379  -0.260275646161804
   0.364286245413978  -0.435277064287946   0.188008570206006   0.084467495340759
   0.080795509986395   0.182984990057412   0.198275182997108   0.375904713439168
     ⋮
R2 = 20×20
   2.663138246670700   2.394600405738687   2.498543055851519   1.424680701228067 ···
                   0   1.919334918682500   0.367524729256336   0.517527290179470
                   0                   0   1.232834742595248   0.126573617405416
                   0                   0                   0   1.050808871544815
                   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
  -0.183242217753921   0.025636422065452   0.352651470586100   0.331938685304752 ···
  -0.186936712599269  -0.170718576649508   0.126336909281979  -0.119337435101800
  -0.351455863958444   0.345029547841881  -0.139000384476335   0.236081624135450
  -0.146174042328889   0.125390940897712  -0.184405722334473  -0.022778430899064
  -0.043987639723182  -0.416720141116407  -0.171009863410556  -0.235774747158227
  -0.090278285129134  -0.343957992582164  -0.117258269046658   0.344344955034490
  -0.257180976423353  -0.200041634326475   0.330236401481577   0.171706692304329
  -0.315137081639972  -0.057117338632047  -0.115556410384379  -0.260275646161804
  -0.364286245413978   0.435277064287947  -0.188008570206006   0.084467495340759
  -0.080795509986395  -0.182984990057412  -0.198275182997108   0.375904713439168
     ⋮
R3 = 20×20
  -2.663138246670700  -2.394600405738687  -2.498543055851519  -1.424680701228067 ···
   0.000000000000000  -1.919334918682500  -0.367524729256336  -0.517527290179470
   0.000000000000000   0.000000000000000  -1.232834742595248  -0.126573617405416
   0.000000000000000   0.000000000000000   0.000000000000000   1.050808871544815
   0.000000000000000  -0.000000000000000   0.000000000000000  -0.000000000000000
   0.000000000000000  -0.000000000000000                   0   0.000000000000000
   0.000000000000000  -0.000000000000000                   0   0.000000000000000
   0.000000000000000  -0.000000000000000   0.000000000000000  -0.000000000000000
   0.000000000000000   0.000000000000000                   0  -0.000000000000000
   0.000000000000000  -0.000000000000000   0.000000000000000   0.000000000000000
     ⋮
```

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

Para el algoritmo 1:

Cálculo de $\|A - Q_1R_1\|_2$

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

```
    7.593833434274956e-16
```

Cálculo de $\|Q_1Q_1^T - I\|_2$

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

```
1.205141340744771e-13
```

Para el algoritmo 3:

Cálculo de $\|A - Q_2R_2\|_2$

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

```
7.402143115756298e-16
```

Cálculo de $\|Q_2Q_2^T - I\|_2$

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

```
1.512252993219856e-14
```

Para el algoritmo 2:

Cálculo de $\|A - Q_3R_3\|_2$

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

```
3.065714060388978e-15
```

Cálculo de $\|Q_3Q_3^T - I\|_2$

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

```
9.936569346476342e-16
```

¿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: 0.00000000000000159591
```

```
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: 0.00000000000000165087
```

```
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: 0.00000000000000443449
```

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
```

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$
$$\vdots$$

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

Para el algoritmo 1:

Cálculo de $\|A - Q_1R_1\|_2$

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

```
La norma es: 0.00000000000000008464
```

Cálculo de $\|Q_1Q_1^T - I\|_2$

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

```
La norma es: 12.97161351231601145173
```

Para el algoritmo 2:

Cálculo de $\|A - Q_2R_2\|_2$

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

```
La norma es: 0.00000000000000008098
```

Cálculo de $\|Q_2Q_2^T - I\|_2$

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

```
La norma es: 0.99895633841753628257
```

Para el algoritmo 3:

Cálculo de $\|A - Q_3R_3\|_2$

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

```
La norma es: 0.00000000000000041743
```

Cálculo de $\|Q_3Q_3^T - I\|_2$

```
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 QR y aproximarlo a la matriz original A, 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 Q 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);
```

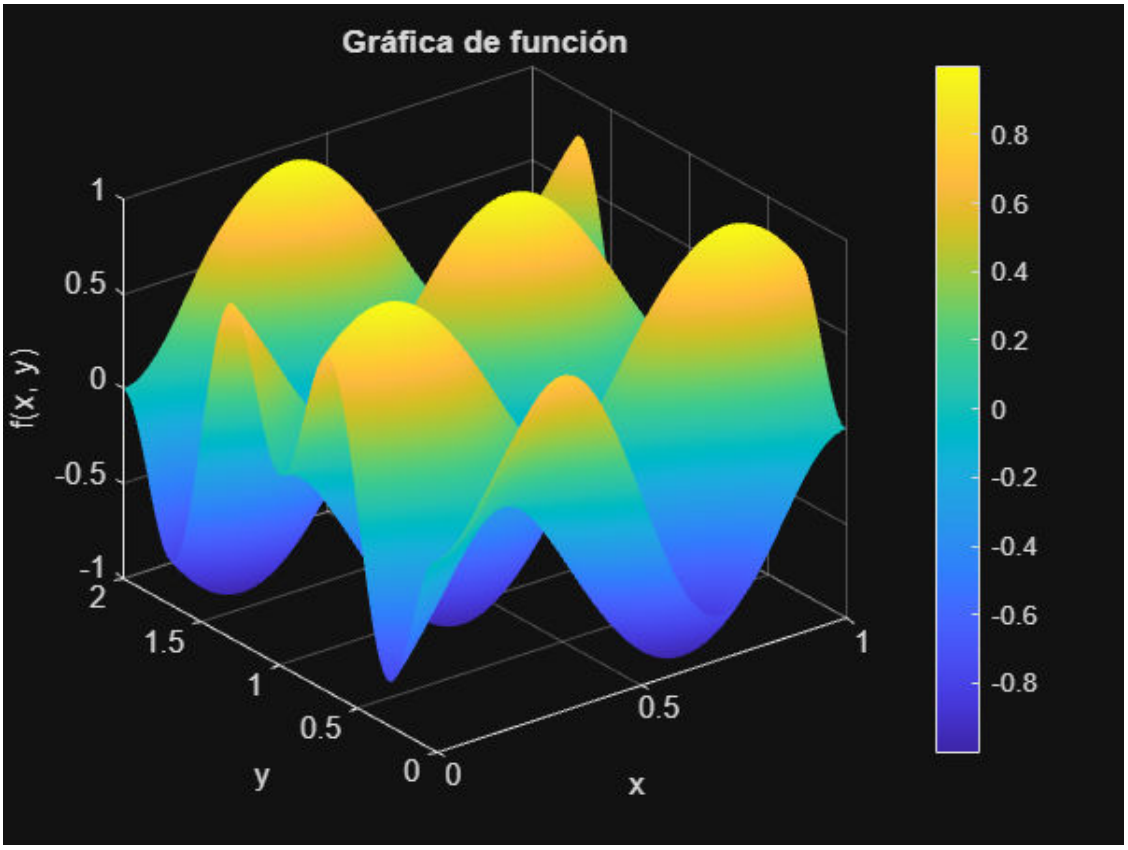
Crear función:

```
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)

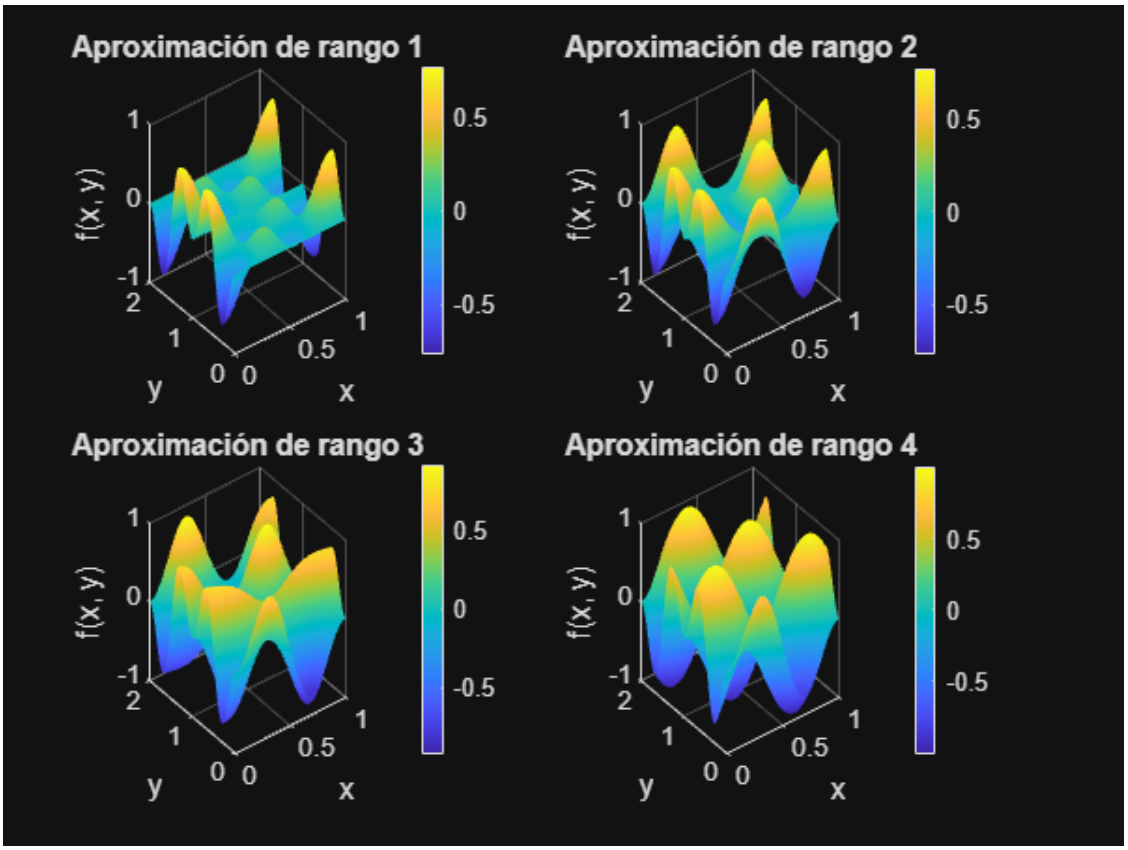
Calcular la función f(x, y) en la malla y asignarla a zz de esta manera, ya que zz=f(xx,yy) me da problemas

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

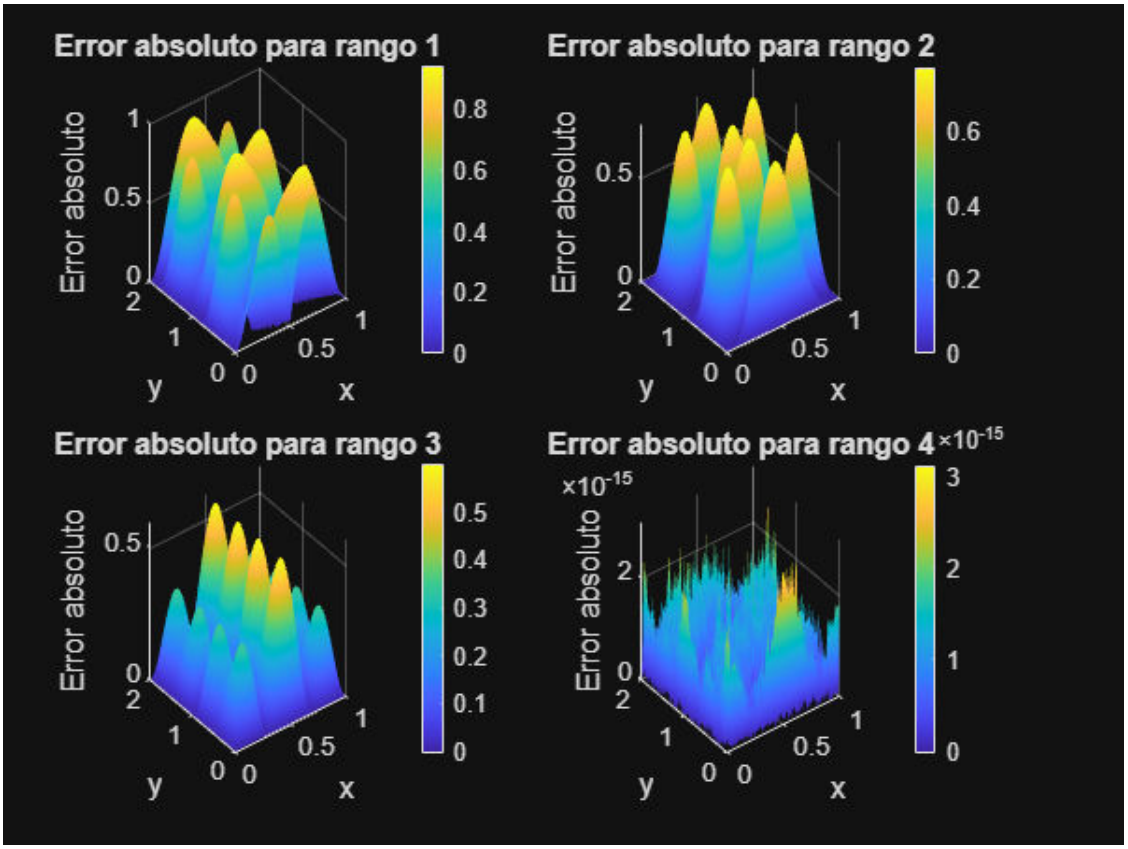
Obtener la descomposición en valores singulares de zz

```
[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 = 1:6
    fprintf('%20.16f\n', S(i, i));
end
```

```
 35.7919334487535039
 35.4409720521319684
 35.0900626958687667
 35.0900626958687667
  0.0000000000000318
  0.0000000000000190
```

Dado que a partir del cuarto valor singular son número muy cercanos a cero, solo se consideran los primeros 4 valores.

Para ver el rango de zz utilizamos la función rank:

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

```
     4
```

d)

A partir de la función dada y aplicando porpiedades de senos:

$$f(x,y) = \sin(2\pi(x+y))\sin(\pi(x-y)) = (\sin(2\pi x)\cos(2\pi y) + \sin(2\pi y)\cos(2\pi x)) \cdot (\sin(\pi x)\cos(\pi y) - \sin(\pi y)\cos(\pi x))$$

Al evaluar en la función los vectores xx y yy se tiene la expresión que representa la matriz zz:

$$zz = \sin(2\pi x) \cdot \sin(\pi x)(\cos(2\pi y)\cos(\pi y))^T - \sin(2\pi x)\cos(\pi x) \cdot (\cos(2\pi y)\sin(\pi y))^T$$

$$+ \sin(\pi x) \cdot \cos(2\pi x)(\sin(2\pi y)\cos(\pi y))^T - \cos(2\pi x)\cos(\pi x)(\sin(2\pi y)\sin(\pi y))^T$$

Al normalizar los vectores, se obtienen los vectores unitarios:

$$zz = ||\sin(2\pi x)\sin(\pi x)|| \cdot ||\cos(2\pi y)\cos(\pi y)|| \cdot \left(\frac{\sin(2\pi x)\sin(\pi x)}{||\sin(2\pi x)\sin(\pi x)||}\right)\left(\frac{\cos(2\pi y)\cos(\pi y)}{||\cos(2\pi y)\cos(\pi y)||}\right)^T$$

$$-||\sin(2\pi x)\cos(\pi x)|| \cdot ||\cos(2\pi y)\sin(\pi y)|| \cdot \left(\frac{\sin(2\pi x)\cos(\pi x)}{||\sin(2\pi x)\cos(\pi x)||}\right)\left(\frac{\cos(2\pi y)\sin(\pi y)}{||\cos(2\pi y)\sin(\pi y)||}\right)^T$$

$$+||\sin(2\pi y)\cos(\pi y)|| \cdot ||\sin(\pi x)\cos(2\pi x)|| \cdot \left(\frac{\sin(\pi x)\cos(2\pi x)}{||\sin(\pi x)\cos(2\pi x)||}\right)\left(\frac{\sin(2\pi y)\cos(\pi y)}{||\sin(2\pi y)\cos(\pi y)||}\right)^T$$

$$-||\sin(2\pi y)\sin(\pi y)|| \cdot ||\cos(2\pi x)\cos(\pi x)|| \cdot \left(\frac{\cos(2\pi x)\cos(\pi x)}{||\cos(2\pi x)\cos(\pi x)||}\right)\left(\frac{\sin(2\pi y)\sin(\pi y)}{||\sin(2\pi y)\sin(\pi y)||}\right)^T$$

Según el lema 5.19 se tiene $A = \sum_{j=1}^{r} \sigma_j u_j v_j^*$ donde $\{u_j\}_{j=1}^{r}$ y $\{v_j\}_{j=1}^{r}$ son los vectores singulares izquierdos y derechos,

respectivamente, por ende pertenecen al espacio de filas y columnas respectivamente y $\{\sigma_j\}_{j=1}^{r}$ son escalares. Note que podemos expresar la matriz zz como el lema, si tomamos los σ_j como las normas al inicio de cada sumando que representan la magnitud o peso a cada componente de rango 1, los vectores dependientes de x como los u_j, ya que estos son del espacio filas y por último los vectoes transpuestos (para que se obtenga la matriz) dependientes de y se toman como los v_j. Dado que con este lema la cantidad de sumandos indican el rango de la matriz, entonces se concluye que es rango 4.

Ejercicio 4

a)

Note que $\frac{\partial E(c_0,...,c_{n-1})}{\partial c_k}$ donde $E(c_0,...,c_{n-1}) := ||f-p||_2^2 = \int_0^1 |f(x)-p(x)|^2 dx \to$ mín está dada por

$\frac{\partial E}{\partial c_k} = -2\int_0^1 (f(x) - c_0 - ... - c_{n-1}x^{n-1}) \cdot x^k dx$, donde $k = 0,...,n-1$. Igualando a 0 obtenemos que:

$$
\begin{aligned}
&-2\int_0^1 (f(x) - c_0 - ... - c_{n-1}x^{n-1}) \cdot x^k dx = 0 \\
&\iff \int_0^1 (f(x) - c_0 - ... - c_{n-1}x^{n-1}) \cdot x^k dx = 0 \\
&\iff \int_0^1 f(x)\cdot x^k dx - c_0\int_0^1 x^k dx - c_1\int_0^1 x\cdot x^k dx - ... - c_{n-1}\int_0^1 c^{n-1}\cdot x^k dx = 0 \\
&\iff \int -^1 f(x)\cdot x^k dx = c_0\int_0^1 x^k dx - c_1\int_0^1 x\cdot x^k dx - ... - c_{n-1}\int_0^1 c^{n-1}\cdot x^k dx \\
&\iff \int_0^1 f(x)\cdot x^k = \sum_{k=0}^{n-1} c_k \int_0^1 c^{k+j} dx
\end{aligned}
$$

Note que $\int_0^1 x^{k+j} dx = \frac{x^{k+j+1}}{k+j+1}\Big|_0^1 = \frac{1}{k+j+1}$. Por lo tanto, el sistema de n ecuaciones con n incógnitas $M\mathbf{c} = \mathbf{b}$ está dado por:

$$
\begin{cases}
\sum_{k=0}^{n-1} \frac{1}{k+1}\cdot c_k = \int_0^1 f(x)dx \\
\sum_{k=0}^{n-1} \frac{1}{k+2}\cdot c_k = \int_0^1 f(x)\cdot x dx \\
\quad\vdots \\
\sum_{k=0}^{n-1} \frac{1}{k+n}\cdot c_k = \int_0^1 f(x)\cdot x^{n-1} dx
\end{cases}
$$

Expresado en términos matriciales, obtenemos:

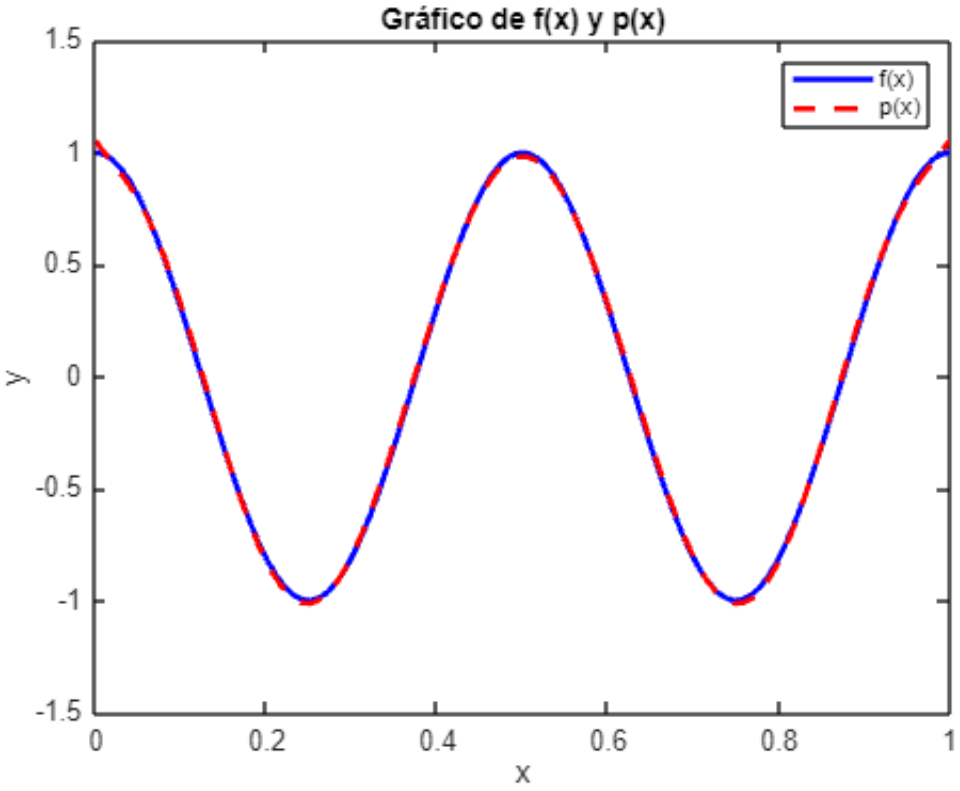
$$
\begin{pmatrix}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\
\frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\
\vdots & \vdots & \vdots & & \vdots \\
\frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1}
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}
=
\begin{pmatrix}
\int_0^1 f(x)dx \\
\int_0^1 f(x)xdx \\
\vdots \\
\int_0^1 f(x)x^{n-1}dx
\end{pmatrix}
$$

Observe que la matriz Mes una matriz de Hilbert y para resolver cualquier sistema $H\mathbf{x} = \mathbf{b}$ se utiliza el comando $\mathbf{x} = H\backslash b$. Este sistema de ecuaciones presenta varios problemas, uno de ellos es que al utilizar polinomios de un grado alto, es posible obtener interpoladores con grandes oscilaciones que no aproximan de manera correcta la función que modelo el fenómeno correspondiente al conjunto de datos en casos particulares (fenómeno de Runge). Además, se sabe que la matriz de Hilbert está mal condicionada, lo cual implica que cualquier error en los datos de entrada o en el cálculo numérico se amplifica significativamente en la solución. Esto hace que los resultados sean inestables y poco confiables.

b)

```
n = 10;
M = hilb(n);
b = zeros(n,1);

% Calcular el vector b
for i = 1:n
    f_b = @(x) cos(4*pi*x) .* x.^(i-1); % Definir f para cada i
    b(i) = integral(f_b, 0, 1);
end

% Resolver el sistema para obtener los coeficientes c
c = M\b;

% Definir la función original f(x)
f  = @(x) cos(4*pi*x);

% Definir la función aproximada p(x) usando los coeficientes c
p = @(x) 0;
for i = 1:n
    p = @(x) p(x) + c(i) * x.^(i-1);
end

% Evaluar f(x) y p(x) en cada punto de xx
xx = linspace(0, 1, 100);
yy = f(xx);
p_evaluado = p(xx);

% Graficar f(x) y p(x) en el intervalo [0, 1]
figure;
plot(xx, yy, 'b-', 'LineWidth', 2, 'DisplayName', 'f(x)');
hold on;
plot(xx, p_evaluado, 'r--', 'LineWidth', 2, 'DisplayName', 'p(x)');
xlabel('x');
ylabel('y');
legend;
title('Gráfico de f(x) y p(x)');
```

```
hold off;
```



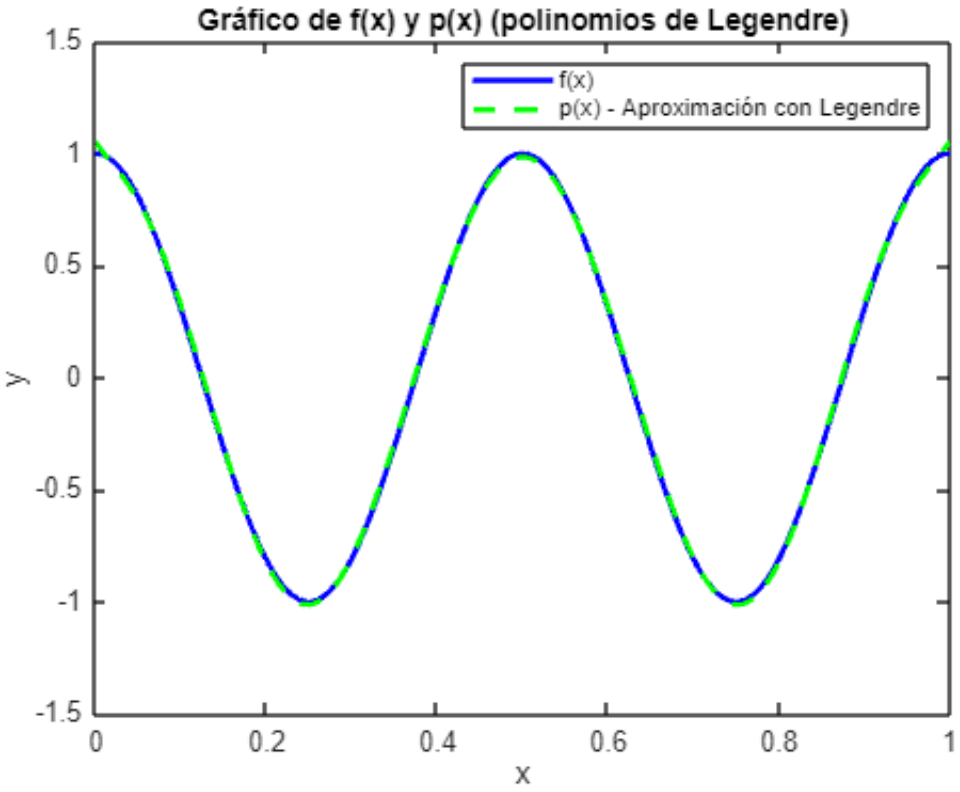

```
% Calcular ||f - p||_2^2
f_p = @(x) (f(x) - p(x)).^2;
int_f_p = integral(f_p, 0, 1);

fprintf('El valor de ||f - p||_2^2 es: %.6f\n', int_f_p);
```

```
El valor de ||f - p||_2^2 es: 0.000173
```

c)

```
% Definir la función recursiva para calcular los polinomios de Legendre
function y = polin_legendre(k, x)
    if k == 0
        y = ones(size(x)); % ϕ_0(x) = 1
    elseif k == 1
        y = x; % ϕ_1(x) = x
    else
        y = ((2 * k - 1) * (x) .* polin_legendre(k - 1, x) - (k - 1) *
polin_legendre(k - 2, x)) / k;
    end
end
```

```
% Código principal
n = 10;
f = @(x) cos(4 * pi * x); % Definir la función f(x)

% Inicializar los coeficientes beta
beta = zeros(n, 1);

% Calcular los coeficientes beta_k
for k = 0:(n-1)
    % Definir el polinomio de Legendre phi_k con la traslación de x del intervalo
[-1,1] a
    % [0,1]
    phi_k = @(x) polin_legendre(k, 2*x-1);

    % Calcular el producto interno <f, phi_k>
    f_phi_k = @(x) f(x) .* phi_k(x);
    numerador = integral(f_phi_k, 0, 1);

    % Calcular el producto interno <phi_k, phi_k>
    phi_k_2 = @(x) phi_k(x).^2;
    denominador = integral(phi_k_2, 0, 1);

    % Calcular beta_k
    beta(k + 1) = numerador / denominador;
end

% Construir la función aproximada p(x) usando los coeficientes beta
p = @(x) 0;
for k = 0:(n-1)
    phi_k = @(x) polin_legendre(k, 2*x-1);
    p = @(x) p(x) + beta(k + 1) * phi_k(x);
end

% Graficar f(x) y p(x)
xx = linspace(0, 1, 1000);
yy = f(xx);
p_evaluado = p(xx); % Evaluar p en cada punto de xx

figure;
plot(xx, yy, 'b-', 'LineWidth', 2, 'DisplayName', 'f(x)');
hold on;
plot(xx, p_evaluado, 'g--', 'LineWidth', 2, 'DisplayName', 'p(x) - Aproximación con
Legendre');
xlabel('x');
ylabel('y');
legend;
title('Gráfico de f(x) y p(x) (polinomios de Legendre)');
hold off;
```




```
% Calcular la norma ||f - p||_2
f_p = @(x) (f(x) - p(x)).^2;
int_f_p = sqrt(integral(f_p, 0, 1));

fprintf('El valor de ||f - p||_2 es: %.6f\n', int_f_p);
```

```
El valor de ||f - p||_2 es: 0.013167
```

Se observa que el valor de $||f - p||_2^2$ es mayor cuando se realiza la aproximación con los polinomios ortogonales de Legendre que resolviendo el sistema de ecuaciones propuesto en el **inciso b)**, esto puede darse ya que n es muy pequeño.

Ejercicio 5

a)

Por hipótesis, se asume que la matriz $A = I + \mathbf{u}\mathbf{v}^T$ es invertible. Sea $A^{-1} = I + \alpha\mathbf{u}\mathbf{v}^T$ su inversa, observe lo siguiente:

$$A \cdot A^{-1} = (I + \mathbf{u}\mathbf{v}^T)(I + \alpha\mathbf{u}\mathbf{v}^T)$$
$$\iff A \cdot A^{-1} = I \cdot I + I \cdot \alpha \cdot \mathbf{u}\mathbf{v}^T + \mathbf{u}\mathbf{v}^T \cdot I + \mathbf{u}\mathbf{v}^T \cdot \alpha\mathbf{u}\mathbf{v}^T$$
$$\iff A \cdot A^{-1} = I + \alpha\mathbf{u}\mathbf{v}^T + \mathbf{u}\mathbf{v}^T + \alpha\mathbf{u}\mathbf{v}^T\mathbf{u}\mathbf{v}^T$$

Note que $\mathbf{v}^T\mathbf{u}$ es el producto interno de $\mathbf{u} \cdot \mathbf{v}$, es decir, son escalares de forma que:

$$\mathbf{v}^T\mathbf{u} = \begin{pmatrix} v_1 & \dots & v_m \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = v_1u_1 + v_2u_2 + \dots + v_mu_m$$

Por lo tanto

$$\iff A \cdot A^{-1} = I + \alpha\mathbf{u}\mathbf{v}^T + \alpha\mathbf{u}(v^Tu)\mathbf{v}^T$$
$$\iff A \cdot A^{-1} = I + \mathbf{u}\mathbf{v}^T(\alpha + 1 + \alpha(v^Tu))$$

Por propiedades de la matriz inversa, sabemos que $A \cdot A^{-1} = I$, por lo que necesitamos que

$$\mathbf{u}\mathbf{v}^T(\alpha + 1 + \alpha(v^Tu)) = 0$$
$$\iff \alpha + 1 + \alpha(v^Tu) = 0$$
$$\iff \alpha(1 + v^Tu) = -1$$
$$\iff \alpha = \frac{-1}{1 + v^Tu}.$$

Conluya que si A es invertible, entonces $A^{-1} = I + \alpha\mathbf{u}\mathbf{v}^T$ es su inversa donde $\alpha = \frac{-1}{1 + v^Tu}$.

b)

Por definición, sabemos que si una matriz es singular, entonces su determinante es igual a cero. Por lo tanto, debemos de encontrar los valores de $\mathbf{u}$ y $\mathbf{v}$ que hacen que el determinante de la matriz A se anule:

$det(A) = det(I + \mathbf{u}\mathbf{v}^T) = 1 + v^Tu$ (sacado de https://math.stackexchange.com/questions/3008399/determinant-of-iuv)

Para que $1 + v^Tu = 0 \iff v^Tu = -1$. El núcelo de la matriz A es dado por:

$$Ax = 0$$
$$\iff (I + \mathbf{u}\mathbf{v}^T)x = 0$$
$$\iff x + x\mathbf{u}\mathbf{v}^T = 0$$
$$\iff x = -x\mathbf{u}\mathbf{v}^T$$
$$\iff x = -(v^Tx)u$$

Es decir, x es un múltiplo de u tal que $Au = 0$ de forma que $ker(A) =< u >$.

Ejercicio 7

a)

```
load('data.mat'); % Cargar los datos x e y
```

b)

```
function [ck, iteracion, norma_delta_ck, condiciones_J] = minimo_cuadrados(c0, x, y)

% Definición de la función f y sus derivadas con respecto a cada parámetro
f = @(x, c1, c2, c3, c4) c1 .* exp(-c2 .* x) .* sin(c3 .* x + c4);

% Derivadas parciales de f respecto a c1, c2, c3, y c4
der_c1 = @(x, c2, c3, c4) exp(-c2 .* x) .* sin(c3 .* x + c4);
der_c2 = @(x, c1, c2, c3, c4) -x .* c1 .* exp(-c2 .* x) .* sin(c3 .* x + c4);
der_c3 = @(x, c1, c2, c3, c4) c1 .* exp(-c2 .* x) .* x .* cos(c3 .* x + c4);
der_c4 = @(x, c1, c2, c3, c4) c1 .* exp(-c2 .* x) .* cos(c3 .* x + c4);

% Parámetros de iteración y tolerancia
tolerancia = 10^(-8);
max_iteracion = 100000;
ck = c0;
iteracion = 1;
norma_delta_ck = []; % Para almacenar la norma de delta_c en cada iteración
condiciones_J = []; % Para almacenar el número de condición de J en cada iteración

while iteracion < max_iteracion

    % Inicialización de g(c^(k)) y el Jacobiano
    g = zeros(100, 1);
    derivada_c1 = zeros(100, 1);
    derivada_c2 = zeros(100, 1);
    derivada_c3 = zeros(100, 1);
    derivada_c4 = zeros(100, 1);

    for i = 1:100
        g(i) = f(x(i), ck(1), ck(2), ck(3), ck(4));
        derivada_c1(i) = der_c1(x(i), ck(2), ck(3), ck(4));
        derivada_c2(i) = der_c2(x(i), ck(1), ck(2), ck(3), ck(4));
        derivada_c3(i) = der_c3(x(i), ck(1), ck(2), ck(3), ck(4));
        derivada_c4(i) = der_c4(x(i), ck(1), ck(2), ck(3), ck(4));
    end

    % Cálculo del vector de error
    b = y - g;
```

```
    % Definición del Jacobiano como matriz 100x4
    jacobiano = [derivada_c1, derivada_c2, derivada_c3, derivada_c4];

    % Calcular el número de condición de J y almacenarlo
    condiciones_J = [condiciones_J; cond(jacobiano)];

    % Descomposición en valores singulares del Jacobiano
    [U, S, V] = svd(jacobiano, 'econ');

    % Invertir los valores singulares positivos de S
    S_inv = diag(1 ./ diag(S));

    % Cálculo de delta_ck
    delta_ck = V * S_inv * U' * b;
    norma_delta_ck = [norma_delta_ck; norm(delta_ck, Inf)]; % Almacenar la norma de
delta_ck
    disp(norma_delta_ck(end));
    disp('--------');

    % Verifica si el criterio de parada se cumple
    if norma_delta_ck(end) < tolerancia
        break;
    end

    % Actualiza los parámetros c_k
    ck = ck + delta_ck;
    disp(ck);

    iteracion = iteracion + 1;
end

if iteracion == max_iteracion
    disp('Se alcanzó el número máximo de iteraciones sin convergencia.');
end

end
```

c)

```
c0 = [1.1; 0.4; 2.1; 0.2];
[ck_final, num_iter, norma_delta_ck, ~]  = minimo_cuadrados(c0, x, y);
fprintf('Número de iteraciones: %d\n', num_iter);
disp('c resultante:');
disp(ck_final);

f = @(x, c1, c2, c3, c4) c1 .* exp(-c2 .* x) .* sin(c3 .* x + c4);
```

```
c_exacto = transpose([1, 1/2, 2, 0]);

% Graficar los nodos {xi, yi}
figure;
scatter(x, y, 'filled'); % Graficar los nodos {xi, yi}
hold on;

% Graficar la función exacta f(x, c_exacto)
xx = linspace(min(x), max(x), 1000);
y_exact = f(xx, c_exacto(1), c_exacto(2), c_exacto(3), c_exacto(4));
plot(xx, y_exact, 'r-', 'LineWidth', 2, 'DisplayName', 'Función Exacta f(x,
c_{exacto})');

% Graficar la función aproximada f(x, c(k))
y_aprox = f(xx, ck_final(1), ck_final(2), ck_final(3), ck_final(4));
plot(xx, y_aprox, 'b--', 'LineWidth', 2, 'DisplayName', 'Función Aproximada f(x,
c^{(k)})');

xlabel('x');
ylabel('f(x, c^{(k)})');
legend('Nodos \{(x_i, y_i)\}^m_{i=1}', 'Función Exacta', 'Función Aproximada');
title('Problema Mínimos Cuadrados');

% Graficar ||Δc^(k)||_∞ en función de k
figure;
semilogy(norma_delta_ck, '-o', 'LineWidth', 2);
xlabel('Iteración k');
ylabel('||Δc^{(k)}||_∞');
title('||Δc^{(k)}||_∞ en función de k');
grid on;
```

Analizando la el gráfico de $||\Delta c^{(k)}||_\infty$ en escala logarítmica, se puede deducir que la velocidad de convergencia es superlineal, podría ser incluso cuadrática.

d)

```
c0 = [1.1; 0.4; 2.1; 0.2];
[~, ~, ~, condiciones_J] = minimo_cuadrados(c0, x, y);

% Graficar el número de condición en función de las iteraciones
figure;
plot(condiciones_J, '-o', 'LineWidth', 2);
xlabel('Iteración k');
ylabel('\kappa(J(c^{(k)}))');
title('Número de Condición de _{\kappa}(J(c^{(k)})) en función de k');
grid on;
```

En el grafico se puede observar que el número de condición no es muy alto, además, este disminuye considerablemente de la primera a la segunda iteración de forma que se acerca más a 1 y entre más cerca, mejor condicionado. Esto contribuye de manera positiva a la estabilidad y precisión del algoritmo.

e)

```
c0 = transpose([1; 1; 1; 1]);
[ck_final, num_iter, norma_delta_ck, condiciones_J] = minimo_cuadrados(c0, x, y);
fprintf('Número de iteraciones: %d\n', num_iter);
disp('c resultante:');
disp(ck_final);

f = @(x, c1, c2, c3, c4) c1 .* exp(-c2 .* x) .* sin(c3 .* x + c4);
c_exacto = transpose([1, 1/2, 2, 0]);

% Graficar los nodos {xi, yi}
figure;
scatter(x, y, 'filled'); % Graficar los nodos {xi, yi}
hold on;

% Graficar la función exacta f(x, c_exacto)
xx = linspace(min(x), max(x), 1000);
y_exact = f(xx, c_exacto(1), c_exacto(2), c_exacto(3), c_exacto(4));
plot(xx, y_exact, 'r-', 'LineWidth', 2, 'DisplayName', 'Función Exacta f(x,
c_{exacto})');

% Graficar la función aproximada f(x, c(k))
y_aprox = f(xx, ck_final(1), ck_final(2), ck_final(3), ck_final(4));
plot(xx, y_aprox, 'b--', 'LineWidth', 2, 'DisplayName', 'Función Aproximada f(x,
c^{(k)})');

xlabel('x');
ylabel('f(x, c^{(k)})');
legend('Nodos \{(x_i, y_i)\}^m_{i=1}', 'Función Exacta', 'Función Aproximada');
title('Problema Mínimos Cuadrados');

% Graficar ||Δc^(k)||_∞ en función de k
figure;
semilogy(norma_delta_ck, '-o', 'LineWidth', 2);
xlabel('Iteración k');
ylabel('||Δc^{(k)}||_∞');
title('||Δc^{(k)}||_∞ en función de k');
grid on;

% Graficar el número de condición en función de las iteraciones
figure;
```

```
plot(condiciones_J, '-o', 'LineWidth', 2);
xlabel('Iteración k');
ylabel('\kappa(J(c^{(k)}))');
title('Número de Condición de J(c^{(k)}) en función de k');
grid on;
```

Se puede observar que para $c_0 = [1, 1, 1, 1]^T$ el algoritmo no converge, por lo que en el gráfico de los nodos, la función exacta y la función aproximada no se puede observar la función aproximada. Después de la sétima iteración, sólo se obtienen valores nulos para $\Delta c^{(k)}$.

Ejercicio 8

a)

Se tiene un conjunto de $m+1$ elementos, por lo tanto se tiene la partición $0 = x_0 < x_1 < ... < x_{m-1} < x_m = 1$, Dado que la función es periódica, entonces por partición:

1. $s_2/_{[x_i-1, x_i]} \in P_3 \forall i \in \{1, ..., n\}$
2. $s(x_i) = y_i$
3. $s \in c^2[a, b[$

Se tienen las condiciones:

1. $(m+1)$ ecuaciones de interpolación.
2. s_2 continua $\implies (m+1)$ ecuaciones de nodos internos.
3. $s_2^{'}$ continua $\implies (m-1)$ ecuaciones de $s'(x_i^-) = s'(x_i^+)$.
4. Condición de periodicidad de $s_2 \implies s_2^{'}(x_0) = s_2^{'}(x_m)$

Por ende, en total se tienen $3m$ ecuaciones, que son las necesarias.

Construcción del sistema:

Se define $\sigma_i := s_2^{'}(x_i)\ \ (i = 0, 1, ..., m)$. Al s_2 ser cuadrática en cada intervalo y continua, entonces se puede expresar como el polinomio:

$$s_2(x) = \alpha_i + \sigma_i(x - x_i) + \beta_i(x - x_i)^2 \ \ x \in I_i$$

Para que se cumplan las condiciones de interpolación:

$$s_2(x_i) = y_i \text{ y } s_2(x_{i+1}) = y_{i+1}$$

Por ende, si $x = x_i$:

$$s_2(x_i) = y_i = \alpha_I$$

Si $x = x_{i+1}$:

$$s_2(x_i) = \alpha_i + \sigma_i(x_{i+1} - x_i) + \beta_i(x_{i+1} - x_i)^2 = \alpha + \sigma_i h_{i+1} + \beta h_{i+1}^2$$

Note que la derivada está dada por:

$$s_2^{'}(x) = \sigma_i + 2\beta_i(x - x_i)$$

Para que satisfaha la condición de continuidad debe pasar que $s_2^{'}(x_{i+1}^{-}) = s_2^{'}(x_{i+1}^{+})$, por ende:

$$\sigma_i + 2\beta_i(x_{i+1} - x_i) = \sigma_{i+1} \Longrightarrow \sigma_i + 2\beta_i h_{i+1} = \sigma_{i+1} \ (*)$$

También se debe cumplir la condición de periodicidad:

$$s_2^{'}(x_0) = s_2^{'}(x_m) \Longrightarrow \sigma_0 = \sigma_m$$

Ahora, para encontrar el valor de β_i, observe que:

$$\begin{aligned} & s_2(x_{i+1}) = \alpha_i + \sigma_i(x_{i+1} - x_i) + \beta_i(x_{I+1} - x_i)^2 = y_{i+1} \\ & \Longrightarrow y_i + \sigma_i h_{i+1} + \beta_{i+1} h_{i+1}^2 = y_{i+1} \\ & \Longrightarrow \beta_i = \frac{y_{i+1} - y_i - \sigma_i h_{i+1}}{h_{i+1}^2} \end{aligned}$$

Para encontrar el sistema de ecuaciones, primer sustituimos β_i en la ecuación obenida de la continuidad de la derivada:

$$\begin{aligned} & \Longrightarrow \sigma)i + 2\left(\frac{y_{i+1} - y_i - \sigma_i h_{i+1}}{h_{i+1}^2}\right) h_{i+1} = \sigma_{i+1} \\ & \Longrightarrow \sigma_i + 2\left(\frac{y_{i+1} - y_i}{h_{i+1}}\right) - 2\sigma_i = \sigma_{i+1} \\ & \Longrightarrow \sigma_{i+1} + \sigma_i = \frac{2(y_i - y_{i+1})}{h_{i+1}} \end{aligned}$$

Tomando $2\left(\frac{y_i - y_{i+1}}{h_{i+1}}\right) = b_{i+1}$, entonces:

$$\sigma_{i+1} + \sigma_i = b_{i+1}$$

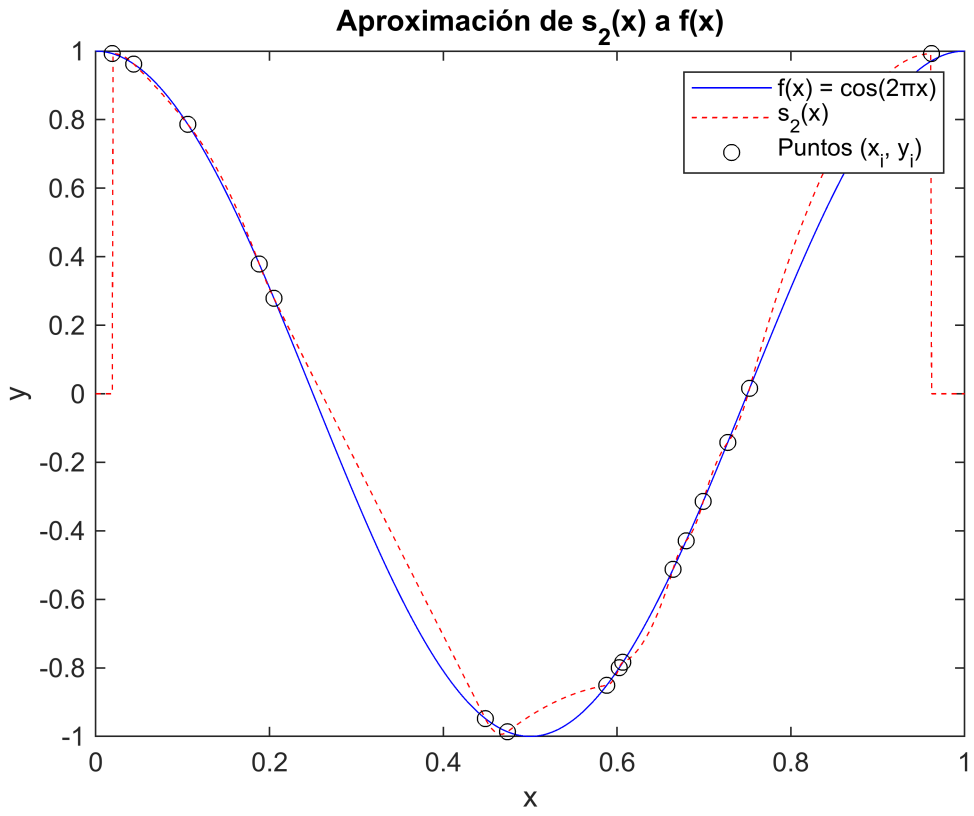
De esta manera, se obtiene el sistema de ecuaciones $A\sigma = b$. Al considerar la condiciónn de periodicidad $\sigma_0 = \sigma_m$, entonces instroducimos la última fila impartiendo la igualdad:

$$A\sigma = b \iff \begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 1 & 0 & \dots & \dots & 1 \end{pmatrix} \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_m \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m_1} \end{pmatrix}.$$

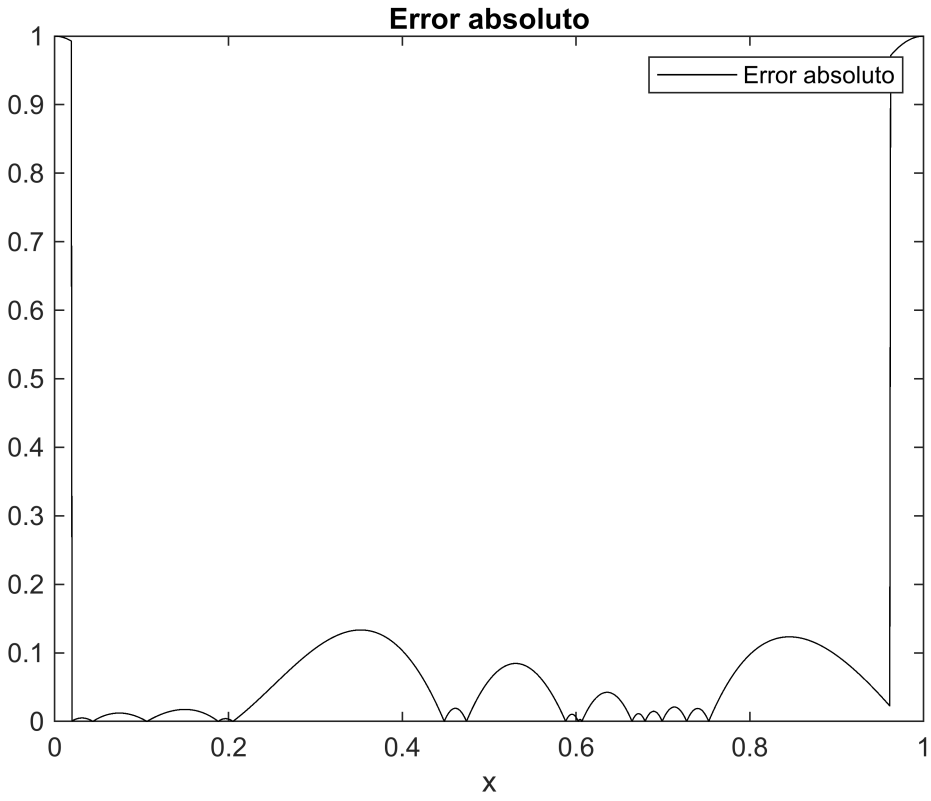
b)

```
% Inicializar variables dadas en el enunciado
m = 15;
f = @(x) cos(2 * pi * x);
rng(2024);
x = sort(rand(1, m + 1));
y = f(x);
y(m+1) = y(1);

h = diff(x);

% Construir matriz A del sistema de ecuaciones Ax = b
A = zeros(m, m);
A(m, 1) = 1;
for i = 1:m
    A(i, i) = 1;
    if i == m
        break;
    end
    A(i, i+1) = 1;
end

% Construir b del sistema de ecuaciones
b = zeros(m, 1);
for i = 2:m+1
    b(i-1) = 2 * (y(i) - y(i-1)) / h(i-1);
end

% Encontrar sigma al resolver el sistema de ecuaciones
sigma = A \ b;

% Encontrar alpha y beta
alpha = y(1:m); % Ya que alpha_i = y_i
beta = zeros(m, 1);
for i = 2:m+1
    beta(i-1) = (y(i) - y(i-1) - sigma(i-1) * h(i-1)) / (h(i-1)^2);
```

```
end

% Valores para graficar
xx = linspace(0, 1, 1000); % Puntos donde evaluaremos s_2
s2 = zeros(size(xx)); % Inicializamos el vector de s_2(x) con ceros

% Construir s2(x) para cada intervalo
for i = 2:m+1
    % Encontrar los índices correspondientes en x para el intervalo actual
    indices = (xx >= x(i-1)) & (xx <= x(i));

    % Calcular s_2(x) en x para el intervalo actual
    s2(indices) = alpha(i-1) + sigma(i-1) * (xx(indices) - x(i-1)) + ...
                      beta(i-1) * (xx(indices) - x(i-1)).^2;
end

% Calcular el error absoluto entre f y la aproximación
f_exacta = f(xx);
error = abs(f_exacta - s2);

% Graficar
figure;
plot(xx, f(xx), 'b', 'DisplayName', 'f(x) = cos(2πx)'); % Función original
hold on;
plot(xx, s2, 'r--', 'DisplayName', 's_2(x)'); % Aproximación s_2
plot(x, y, 'ko', 'DisplayName', 'Puntos (x_i, y_i)'); % Nodos
legend;
title('Aproximación de s_2(x) a f(x)');
xlabel('x');
ylabel('y');
hold off;
```




```
figure;
plot(xx, error, 'k', 'DisplayName', 'Error absoluto');
title('Error absoluto');
xlabel('x');
legend;
```



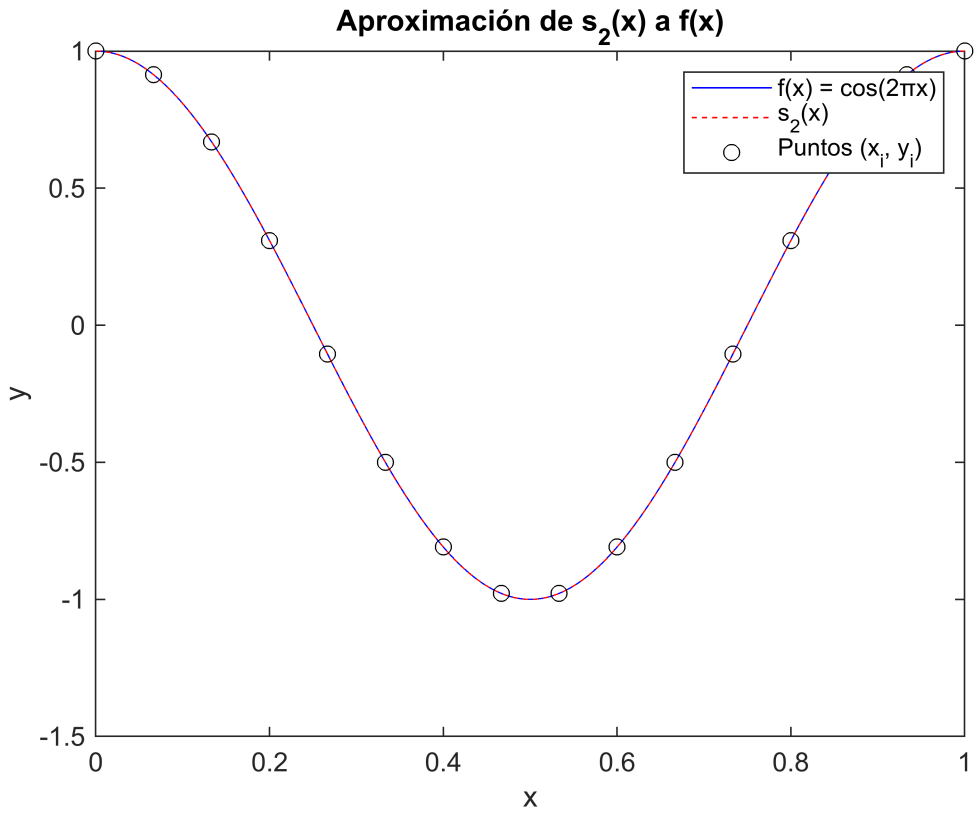

```
% Estimar norma infinito del error
error_norma = max(error);
disp(['Norma infinito del error entre f'' y s''_2: ', num2str(error_norma)]);
```

```
Norma infinito del error entre f' y s'_2: 1
```

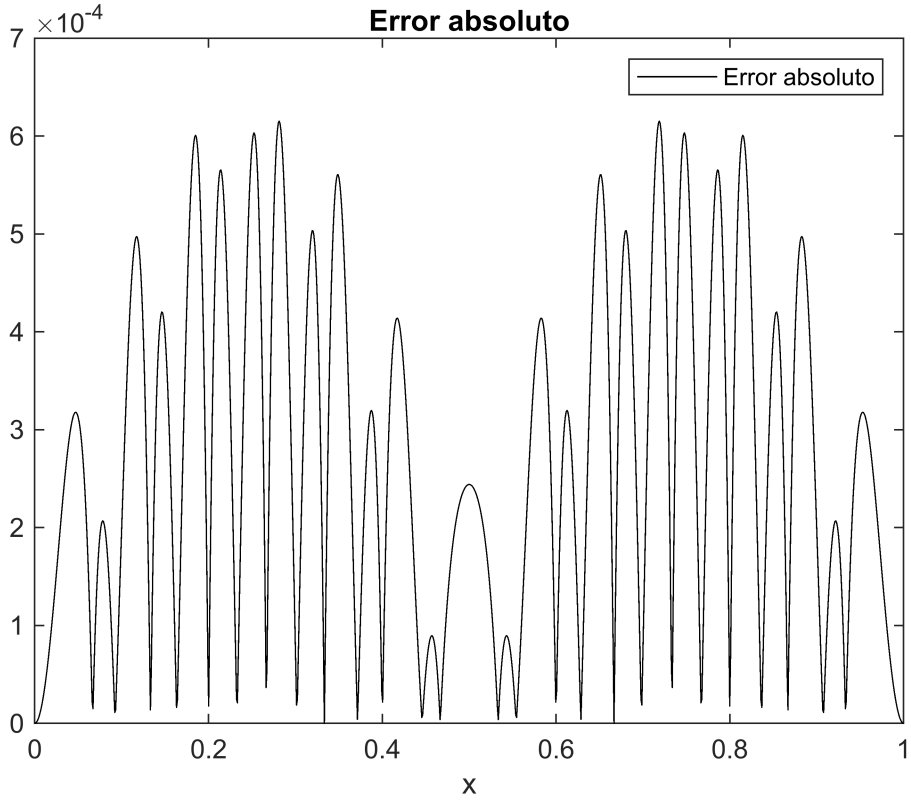
c)

```
% Inicializar variables dadas en el enunciado
m = 15;
f = @(x) cos(2 * pi * x);
rng(2024);
x = linspace(0, 1, m + 1);
y = f(x);
y(m+1) = y(1);

h = diff(x);

% Construir matriz A del sistema de ecuaciones Ax = b
A = zeros(m, m);
A(m, 1) = 1;
for i = 1:m
    A(i, i) = 1;
    if i == m
```

```
        break;
    end
    A(i, i+1) = 1;
end

% Construir b del sistema de ecuaciones
b = zeros(m, 1);
for i = 2:m+1
    b(i-1) = 2 * (y(i) - y(i-1)) / h(i-1);
end

% Encontrar sigma al resolver el sistema de ecuaciones
sigma = A \ b;

% Encontrar alpha y beta
alpha = y(1:m); % Ya que alpha_i = y_i
beta = zeros(m, 1);
for i = 2:m+1
    beta(i-1) = (y(i) - y(i-1) - sigma(i-1) * h(i-1)) / (h(i-1)^2);
end

% Valores para graficar
xx = linspace(0, 1, 1000); % Puntos donde evaluaremos s_2
s2 = zeros(size(xx)); % Inicializamos el vector de s_2(x) con ceros

% Construir s2(x) para cada intervalo
for i = 2:m+1
    % Encontrar los índices correspondientes en x para el intervalo actual
    indices = (xx >= x(i-1)) & (xx <= x(i));

    % Calcular s_2(x) en x para el intervalo actual
    s2(indices) = alpha(i-1) + sigma(i-1) * (xx(indices) - x(i-1)) + ...
                      beta(i-1) * (xx(indices) - x(i-1)).^2;
end

% Calcular el error absoluto entre f y la aproximación
f_exacta = f(xx);
error = abs(f_exacta - s2);

% Graficar
figure;
plot(xx, f(xx), 'b', 'DisplayName', 'f(x) = cos(2πx)'); % Función original
hold on;
plot(xx, s2, 'r--', 'DisplayName', 's_2(x)'); % Aproximación s_2
plot(x, y, 'ko', 'DisplayName', 'Puntos (x_i, y_i)'); % Nodos
legend;
title('Aproximación de s_2(x) a f(x)');
xlabel('x');
ylabel('y');
hold off;
```




```
figure;
plot(xx, error, 'k', 'DisplayName', 'Error absoluto');
title('Error absoluto');
xlabel('x');
legend;
```




```
% Estimar norma infinito del error
error_norma = max(error);
disp(['Norma infinito del error entre f'' y s''_2: ', num2str(error_norma)]);
```

```
Norma infinito del error entre f' y s'_2: 0.00061525
```

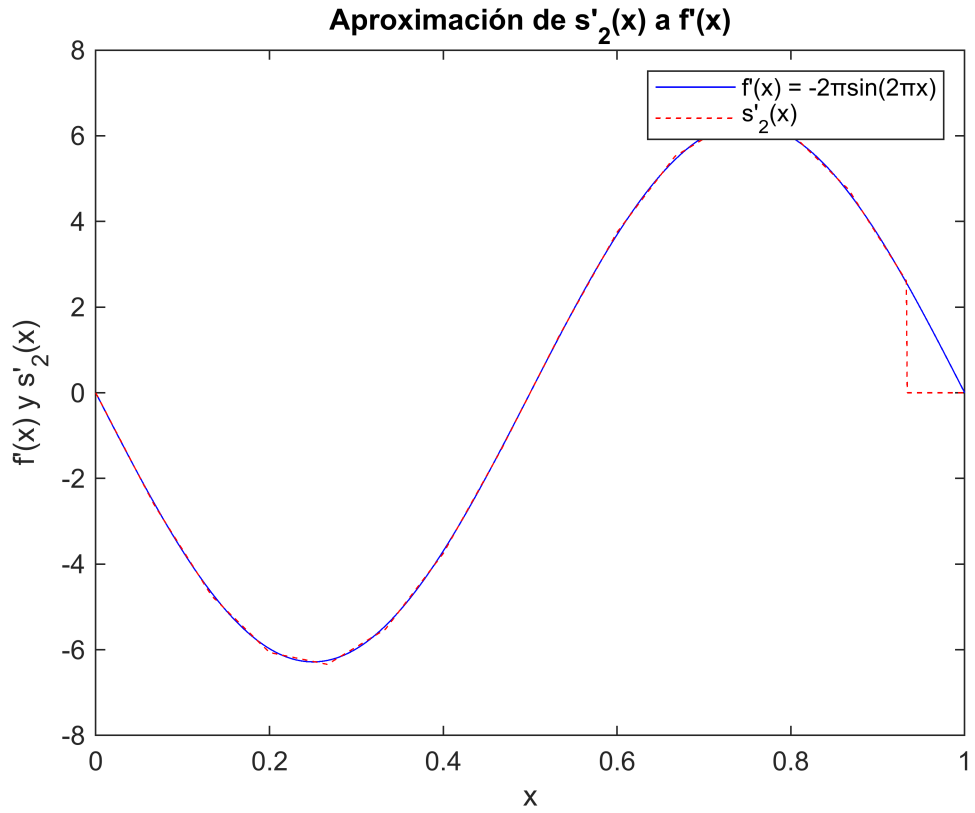
Al comparar el gráfico que utiliza números aleatorios para x, y con el gráfico que utiliza **linespace(0, 1, m+1)** se puede observar que el segundo aproxima exponencialmente mejor que el primero. A pesar de que el primer gráfico interseca los mismo puntos que la función, el segundo se aproxima de manera casi perfecta. Esto se puede deber a que los nodos del segundo son equidistantes y fijos, mientras que en el primer caso son aleatorios en distancia y número.

d)

```
% Inicializar variables dadas en el enunciado
m = 15;
f = @(x) cos(2 * pi * x);
f_derivada = @(x) -2 * pi * sin(2 * pi * x); % Derivada de la función original
x = linspace(0, 1, m+1);
y = f(x);
h = diff(x);

% Construir matriz A del sistema de ecuaciones Ax = b
A = zeros(m, m);
```

```
A(m, 1) = 1;
for i = 1:m
    A(i, i) = 1;
    if i == m

        break;
    end
    A(i, i+1) = 1;
end

% Construir b del sistema de ecuaciones
b = zeros(m, 1);
for i = 2:m+1
    b(i-1) = 2 * (y(i) - y(i-1)) / h(i-1);
end

% Encontrar sigma al resolver el sistema de ecuaciones
sigma = A \ b;

% Encontrar alpha y beta
alpha = y(1:m); % Ya que alpha_i = y_i
beta = zeros(m, 1);
for i = 2:m+1
    beta(i-1) = (y(i) - y(i-1) - sigma(i-1) * h(i-1)) / (h(i-1)^2);
end

% Valores para graficar
xx = linspace(0, 1, 1000); % Puntos donde evaluaremos s_2 y s'_2
s2 = zeros(size(xx)); % Inicializamos el vector de s_2(x) con ceros
s2_derivada = zeros(size(xx)); % Inicializamos el vector de s'_2(x) con ceros

% Construcción de s2(x) y s2_prime(x) en cada intervalo
for i = 2:m
    indices = (xx >= x(i-1)) & (xx <= x(i));

    % Calcular s_2(x) en x para el intervalo actual
    s2(indices) = alpha(i-1) + sigma(i-1) * (xx(indices) - x(i-1)) + ...
                  beta(i-1) * (xx(indices) - x(i-1)).^2;

    % Calcular s'_2(x) en x para el intervalo actual
    s2_derivada(indices) = sigma(i-1) + 2 * beta(i-1) * (xx(indices) - x(i-1));
end

% Calcular el error absoluto entre f' y s'_2
f_derivada_exacta = f_derivada(xx);
error_derivada = abs(f_derivada_exacta - s2_derivada);

% Graficar f' y s'_2 en el mismo gráfico
figure;
```

```
plot(xx, f_derivada(xx), 'b', 'DisplayName', "f'(x) = -2πsin(2πx)"); % Derivada de
la función original
hold on;
plot(xx, s2_derivada, 'r--', 'DisplayName', "s'_2(x)"); % Aproximación s'_2
legend;
title("Aproximación de s'_2(x) a f'(x)");
xlabel('x');
ylabel("f'(x) y s'_2(x)");
hold off;
```




```
% Graficar el error absoluto
figure;
plot(xx, error_derivada, 'k', 'DisplayName', 'Error absoluto');
title("Error absoluto");
xlabel('x');
legend;
```

```
% Estimar norma infinito del error
error_norma_infinito = max(error_derivada);
disp(['Norma infinito del error entre f'' y s''_2: ',
num2str(error_norma_infinito)]);
```

```
Norma infinito del error entre f' y s'_2: 2.5339
```

SI se quiere aproximar $f'(x)$ por una función s de manera tal que s'' sea continuo, se pueden utilizar *splines cúbicos* ya que una de sus restricciones es tener $(n-1)$ condiciones de continuidad para la segunda derivada de s.