Laboratorio 1

Introducción - Errores - Álgebra Lineal
Métodos de Computación Científica

Manuel Lagos

September 14, 2023

0 Introduction

Los ejercicios del laboratorio fueron resueltos utilizando el software sugerido Octave y fueron incluidas capturas de pantallas del código escrito y el resultado obtenido con el programa. El trabajo fue realizado con un ϵ_{mach} de 2^{-5} en una computadora de 64 bits.

1 Ejercicio 1

Se evaluó la función $f(x) = x^3 - x^2 + 3x - 1$ en el intervalo $[0, 2]$ con un paso $h = 0.001$, y se obtuvo un resultado esperable. Luego, fue evaluada alrededor de $x = 1 \pm 0.000025$ en 640 puntos distintos, logrando el efecto de nube de puntos con el objetivo de reproducir el fenómeno visto en la teoría como *"Ruido en la evaluación de funciones"*. La banda difusa de puntos de este segundo resultado presenta una dificultad para determinar la raíz del polinomio en cuestión, no está claro dónde interseca el eje de las abscisas.

De manera similar se procedió con la función $f(x) = x^3 + 2x^2 - x - 2$, sin embargo se obtuvieron distintos resultados. En este segundo experimento, la banda de puntos sí logra evidenciar la raíz del polinomio, aún con una cantidad escasa de puntos.

La diferencia entre ambos resultados se puede explicar por la inclinación que presentan los polinomios alrededor de la raíz. En el primer caso, al haber un punto de inflexión sobre la raíz, la función es prácticamente indistinguible del eje en un entorno del punto, y es esta falta de diferencia lo que dificulta determinar el punto de intersección. En cambio, en el segundo caso la función pasa por el eje con una inclinación bien marcada y la raíz puede ser determinada sin dificultad.

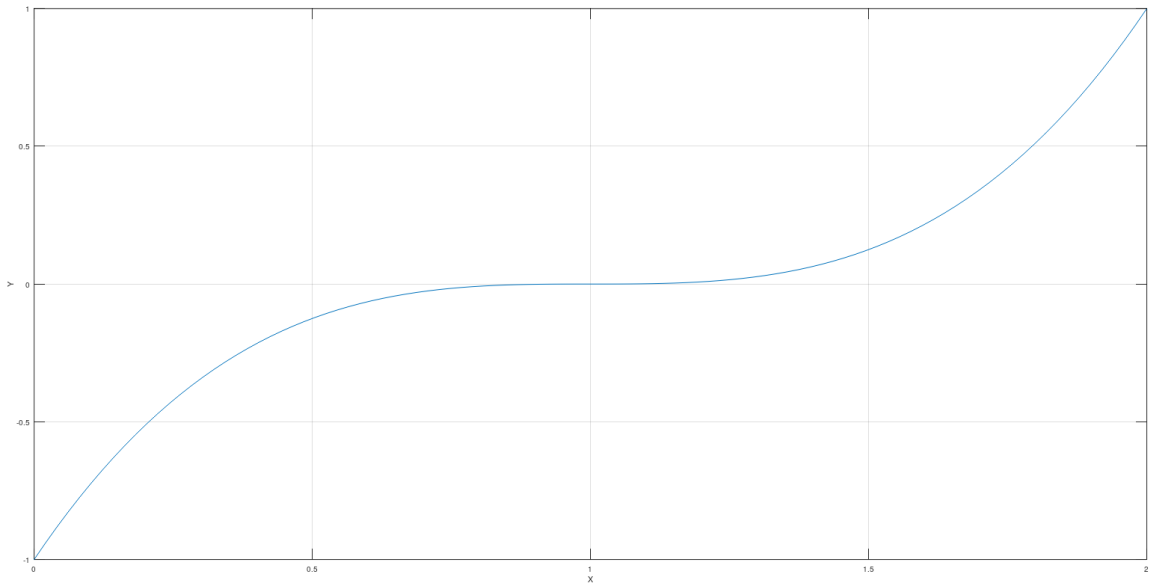

Figure 1: $f(x) = x^3 - x^2 + 3x - 1$ en $[0, 2]$ con un paso $h = 0.001$

```
clf; % limpiar la gráfica del plano

delta = 1;

x = linspace(1-delta, 1+delta);
y = (x.^3)-(3*(x.^2))+(3*x)-1;

plot(x, y);
xlabel('X');
ylabel('Y');
grid;
```

Figure 2: Código de la figura anterior

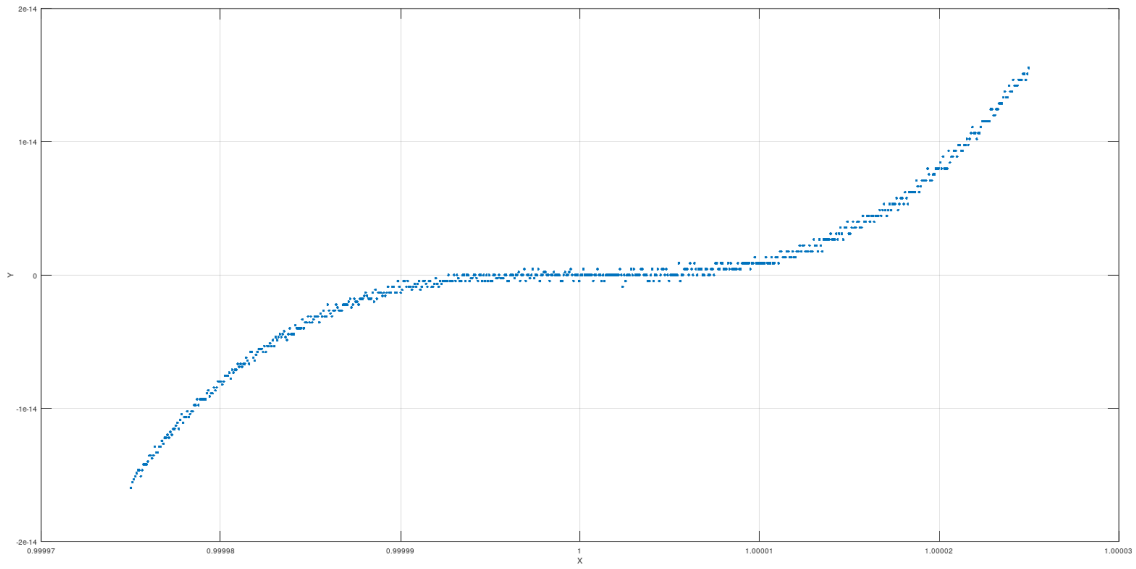

Figure 3: $f(x) = x^3 - x^2 + 3x - 1$ evaluada alrededor de $x = 1 \pm 0.000025$ en 640 puntos

```
clf; % limpiar la gráfica del plano

delta = 0.000025;

x = linspace(1-delta, 1+delta, 640); % tomar 640 puntos linealmente espaciados cerca de 1
y = (x.^3)-(3*(x.^2))+(3*x)-1;

plot(x, y, '.'); % graficar los puntos
xlabel('X');
ylabel('Y');
grid;
```

Figure 4: Código de la figura anterior

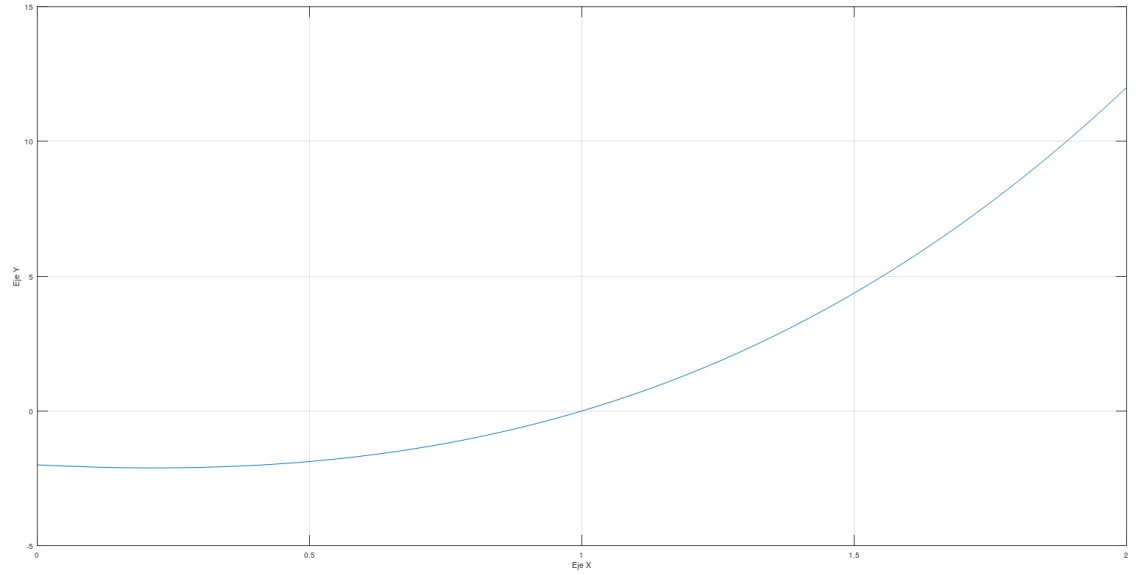

Figure 5: $f(x) = x^3 + 2x^2 - x - 2$ en $[0, 2]$ con un paso $h = 0.001$

```
clf; % limpiar la gráfica del plano

delta = 1;

x = linspace(1-delta, 1+delta);
y = (x.^3)+(2*(x.^2))-x-2;

plot(x, y);

xlabel('Eje X');
ylabel('Eje Y');
grid;
```

Figure 6: Código de la figura anterior

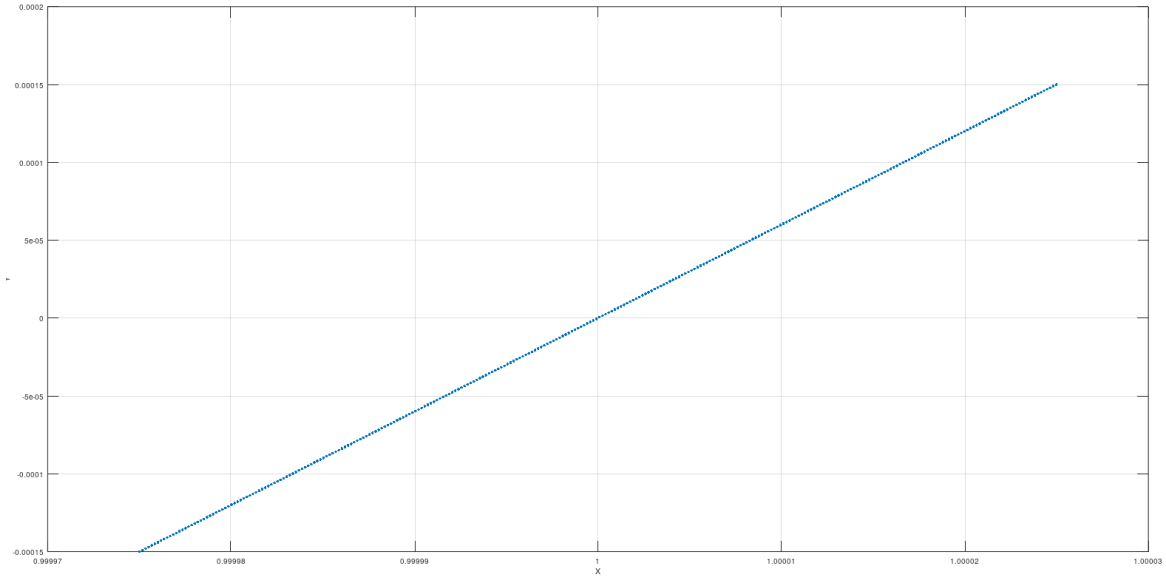

Figure 7: $f(x) = x^3 + 2x^2 - x - 2$ evaluada alrededor de $x = 1 \pm 0.000025$ en 300 puntos

```
clf; % limpiar la gráfica del plano

delta = 0.000025;

x = linspace(1-delta, 1+delta, 300); % tomar 300 puntos linealmente espacios cerca de 1
y = (x.^3)+(2*(x.^2))-x-2;

plot(x, y, '.'); % graficar los puntos

xlabel('X');
ylabel('Y');
grid;
```

Figure 8: Código de la figura anterior

2 Ejercicio 2

Se realizó un programa para construir la matriz de Hilbert con n=20, y se calcularon sus autovectores y sus autovalores asociados empleando las funciones provistas por el software.

$$H_{ij} = \frac{1}{i+h-1} \tag{1}$$

La matriz de Hilbert es un conocido ejemplo de una matriz mal condicionada, lo que significa que es especialmente sensible a pequeñas perturbaciones numéricas en la entrada. Esto la hace una matriz problemática para trabajar en aplicaciones numéricas.

aclaración: Los autovectores se observan como los vectores columna que conforman la matriz autovectores. El n-ésimo autovalor está asociado al elemento de la matriz columna de autovalores de la n-ésima fila.

```
autovectores =

 Columns 1 through 8:

  -9.0222e-09   4.0896e-09   2.0139e-09   6.0881e-10   3.2668e-10  -1.2472e-08  -9.8297e-08  -6.4020e-07
   1.1244e-06  -6.5954e-07  -3.6184e-07  -1.5168e-07  -1.1478e-07   2.0581e-06   1.3282e-05   6.9945e-05
  -3.3440e-05   2.6336e-05   1.5986e-05   7.8609e-06   7.2037e-06  -8.3416e-05  -4.3859e-04  -1.8547e-03
   4.0692e-04  -4.5381e-04  -3.0476e-04  -1.6359e-04  -1.7571e-04   1.4470e-03   6.1219e-03   2.0460e-02
  -2.4519e-03   4.1673e-03   3.1245e-03   1.7586e-03   2.1995e-03  -1.3273e-02  -4.4162e-02  -1.1319e-01
   7.9289e-03  -2.2412e-02  -1.9242e-02  -1.1049e-02  -1.6147e-02   7.0928e-02   1.7814e-01   3.2995e-01
  -1.5386e-02   7.2170e-02   7.5509e-02   4.3232e-02   7.4408e-02  -2.2815e-01  -3.9841e-01  -4.5951e-01
   2.9911e-02  -1.2791e-01  -1.9417e-01  -1.0853e-01  -2.2060e-01   4.2637e-01   4.1217e-01   1.1288e-01
  -8.1553e-02   5.8597e-02   3.2934e-01   1.7766e-01   4.1078e-01  -3.7253e-01   3.5057e-02   3.3917e-01
   1.4327e-01   2.4195e-01  -3.6957e-01  -2.0326e-01  -4.1495e-01  -7.3429e-02  -3.6846e-01  -4.7093e-02
  -4.7333e-02  -5.2628e-01   3.0130e-01   2.2015e-01   3.4960e-02   3.5997e-01  -4.8515e-02  -3.1123e-01
  -1.7853e-01   3.1987e-01  -2.7691e-01  -2.9040e-01   4.5678e-01  -3.0880e-02   3.3665e-01  -1.4271e-01
   1.2841e-01   2.8619e-01   3.6709e-01   2.6594e-01  -5.0175e-01  -1.8208e-01   1.6398e-01   1.8110e-01
   1.2368e-01  -5.3145e-01  -4.3412e-01   6.6523e-02   1.0298e-01  -2.5247e-01  -2.1360e-01   2.8427e-01
   2.9820e-02   1.3409e-01   3.8025e-01  -4.9962e-01   1.2576e-01   3.6951e-01  -2.9656e-01   7.5094e-02
  -2.4371e-01   2.8148e-01  -2.5236e-01   5.8714e-01   5.4889e-02   2.1189e-01   4.2232e-02  -2.2160e-01
  -2.1466e-01  -2.6458e-01   1.0822e-01  -3.2455e-01  -2.5247e-01  -4.3074e-01   3.4566e-01  -2.7557e-01
   7.1495e-01   7.0395e-02  -4.2976e-03   7.5475e-02   2.0917e-01   6.7829e-02   3.0823e-02   3.6212e-02
  -5.2162e-01   1.0235e-02  -2.0872e-02   2.7527e-03  -7.7044e-02   1.2417e-01  -2.9770e-01   3.7534e-01
   1.2690e-01  -6.0783e-03   6.9784e-03  -3.0760e-03   1.1218e-02  -4.8475e-02   1.1699e-01  -1.8179e-01
```

Figure 9: Autovectores de H_{20} de 1 a 8

```
Columns 9 through 16:

 -3.7024e-06  -1.9503e-05  -9.3877e-05   4.1352e-04   1.6664e-03  -6.1281e-03   2.0460e-02   6.1434e-02
  3.2303e-04   1.3336e-03   4.9181e-03  -1.6139e-02  -4.6763e-02   1.1804e-01  -2.5372e-01  -4.4573e-01
 -6.7745e-03  -2.1630e-02  -5.9948e-02   1.4238e-01   2.8326e-01  -4.5287e-01   5.3303e-01   3.5229e-01
  5.7794e-02   1.3812e-01   2.7354e-01  -4.3198e-01  -5.0065e-01   3.2905e-01   7.9079e-02   3.8294e-01
 -2.3621e-01  -3.9217e-01  -4.8279e-01   3.6029e-01  -6.6759e-03   3.4519e-01  -2.4943e-01   1.9669e-01
  4.5729e-01   4.1666e-01   1.1649e-01   2.6726e-01   3.2528e-01   5.9563e-02  -3.2353e-01   6.3646e-03
 -2.7322e-01   1.1700e-01   3.5640e-01  -1.2381e-01   2.7641e-01  -1.8126e-01  -2.4601e-01  -1.3137e-01
 -2.7179e-01  -2.9907e-01   9.9499e-02  -3.0740e-01   5.4163e-02  -2.8043e-01  -1.1167e-01  -2.1180e-01
  1.7299e-01  -2.4839e-01  -2.0402e-01  -2.2753e-01  -1.5252e-01  -2.5577e-01   2.2963e-02  -2.4478e-01
  3.0837e-01   5.8518e-02  -2.8956e-01  -2.1689e-02  -2.5693e-01  -1.5665e-01   1.3038e-01  -2.4239e-01
  6.4865e-02   2.6921e-01  -1.5936e-01   1.6702e-01  -2.4909e-01  -2.9998e-02   2.0045e-01  -2.1518e-01
 -2.1873e-01   2.3744e-01   5.5858e-02   2.5678e-01  -1.5818e-01   8.9538e-02   2.3227e-01  -1.7140e-01
 -2.7109e-01   3.1735e-02   2.2107e-01   2.2958e-01  -2.6305e-02   1.8001e-01   2.2939e-01  -1.1722e-01
 -7.8194e-02  -1.8476e-01   2.5821e-01   1.1180e-01   1.0583e-01   2.2960e-01   1.9708e-01  -5.7101e-02
  1.7830e-01  -2.6990e-01   1.5926e-01  -4.6834e-02   2.0510e-01   2.3369e-01   1.4097e-01   5.7775e-03
  2.8122e-01  -1.6948e-01  -2.5682e-02  -1.8942e-01   2.4770e-01   1.9234e-01   6.6361e-02   6.9206e-02
  1.2620e-01   6.2847e-02  -2.0872e-01  -2.6213e-01   2.1851e-01   1.0851e-01  -2.2095e-02   1.3167e-01
 -1.8839e-01   2.7665e-01  -2.8400e-01  -2.1961e-01   1.0968e-01  -1.3342e-02  -1.2044e-01   1.9217e-01
 -3.3336e-01   2.5104e-01  -1.4344e-01  -2.7245e-02  -8.1024e-02  -1.6803e-01  -2.2539e-01   2.5005e-01
  2.3042e-01  -2.7518e-01   3.1246e-01   3.3879e-01  -3.5177e-01  -3.5025e-01  -3.3431e-01   3.0493e-01
```

Figure 10: Autovectores de H_{20} de 9 a 16

```
Columns 17 through 20:

 -1.6305e-01   3.6932e-01   6.4934e-01   6.4120e-01
  5.8802e-01  -4.5542e-01   5.8229e-02   4.0494e-01
  9.6414e-02  -4.0814e-01  -1.0387e-01   3.0887e-01
 -1.7490e-01  -2.8491e-01  -1.6647e-01   2.5377e-01
 -2.7866e-01  -1.7412e-01  -1.9314e-01   2.1714e-01
 -2.9322e-01  -8.5712e-02  -2.0395e-01   1.9065e-01
 -2.6385e-01  -1.7024e-02  -2.0697e-01   1.7043e-01
 -2.1457e-01   3.6108e-02  -2.0591e-01   1.5440e-01
 -1.5793e-01   7.7295e-02  -2.0266e-01   1.4133e-01
 -1.0044e-01   1.0934e-01  -1.9823e-01   1.3044e-01
 -4.5377e-02   1.3434e-01  -1.9320e-01   1.2120e-01
  5.7322e-03   1.5390e-01  -1.8792e-01   1.1325e-01
  5.2292e-02   1.6918e-01  -1.8258e-01   1.0633e-01
  9.4213e-02   1.8111e-01  -1.7729e-01   1.0024e-01
  1.3167e-01   1.9037e-01  -1.7213e-01   9.4845e-02
  1.6497e-01   1.9749e-01  -1.6714e-01   9.0021e-02
  1.9445e-01   2.0290e-01  -1.6234e-01   8.5682e-02
  2.2049e-01   2.0692e-01  -1.5774e-01   8.1757e-02
  2.4344e-01   2.0981e-01  -1.5334e-01   7.8187e-02
  2.6363e-01   2.1178e-01  -1.4914e-01   7.4925e-02
```

Figure 11: Autovectores de H_{20} de 17 a 20

```
>> diag(autovalores)
ans =

  -1.2625e-16
  -1.0662e-17
  -4.3010e-18
   2.8428e-18
   9.8716e-18
   2.2343e-17
   3.3033e-16
   1.7344e-14
   6.7404e-13
   2.1929e-11
   6.0361e-10
   1.4140e-08
   2.8277e-07
   4.8305e-06
   7.0334e-05
   8.6767e-04
   8.9611e-03
   7.5596e-02
   4.8704e-01
   1.9071e+00
```

Figure 12: Autovalores de H_{20}.

```
clc; % limpiar la consola

n = 20; % definir la dimensión
A = zeros(n); % crear la matriz A

% generar la matriz de Hilbert para n en A
for i = 1:n
  for j = 1:n
    A(i, j) = 1/((i+j)-1);
  endfor
endfor

[autovectores, autovalores] = eig(A) % obtener los autovectores y autovalores de A
```

Figure 13: Código del programa

3 Ejercicio 3

Se escribió un programa para aproximar la derivada de la función $f(x) = tan(x)$ en $x = 1$ empleando primero la fórmula de diferencias finitas hacia adelante, y luego la fórmula de diferencias finitas centrales, en ambas ocasiones computando 100 puntos. También, se graficó para cada una la magnitud del error absoluto con respecto al valor real de la derivada en función del paso h utilizado, en una escala logarítmica de ambos ejes para una correcta apreciación. Además, se calculó la mejor aproximación obtenida, su correspondiente error, y el paso h utilizado, y se lo comparó con el sugerido por la regla del pulgar $h \approx \sqrt{\epsilon_{mach}}$.

Los resultados del experimento muestran que la regla del pulgar obtiene mejores resultados con la fórmula de aproximación por diferencias finitas hacia adelante que con la de diferencias centrales, esto es así posiblemente porque fue hecha a medida para la primera fórmula y no así para la segunda.

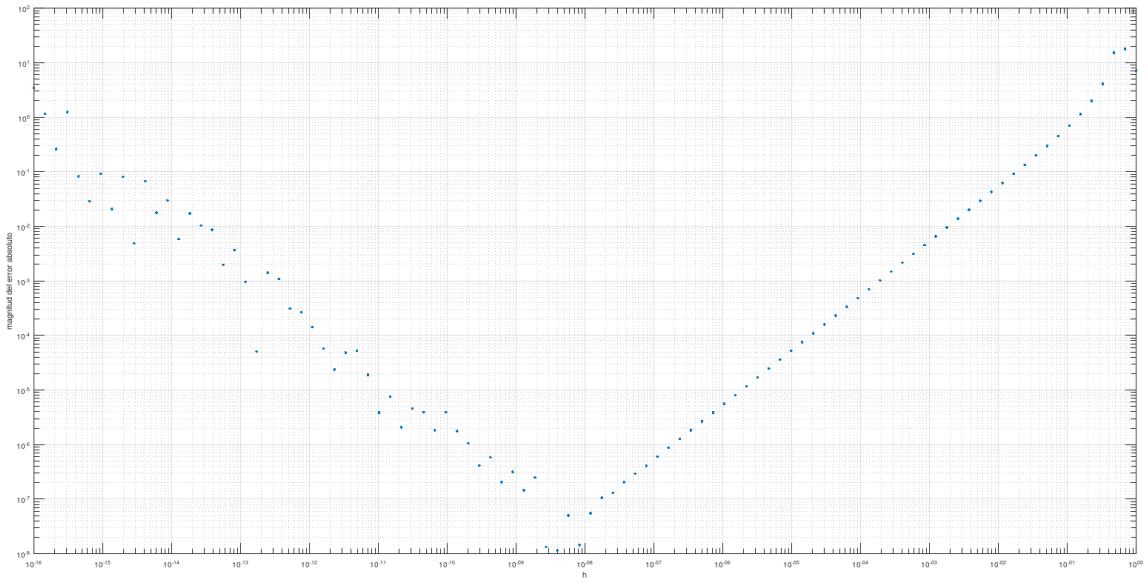

Figure 14: **Diferencias finitas hacia adelante**: magnitud del error absoluto en función del paso h

```
>Aproximación por diferencias finitas hacia adelante

función a proximar: y = @tan
para el valor: x=1
valor real: 3.425519e+00
mejor aproximación: 3.425519e+00
menor error absoluto: 1.143195e-08
mejor paso h encontrado: 3.944206e-09
paso h sugerido sqrt(eps): 1.490116e-08
diferencia entre los dos pasos h: 1.095696e-08
>>
```

Figure 15: Salida por consola

```
clc; %limpiar la consola

h = logspace(-16, 0, 100); % distribución logarítmica de valores a probar
                           % entre 10^-16 y 10^0

x = 1; % punto a evaluar
y = @tan; % función a derivar
dydx = sec(x).^2; % derivada real de la función

aprox = (y(x+h)-y(x))./h; % aproximación de la derivada por diferencias
                          % finitas hacia adelante

error = abs(aprox - dydx); % error absoluto de la aproximación

[min_e, I] = min(error); % valor mínimo e índice entre los errores calculados
min_h = h(I); % valor de h para el cual se obtuvo el mínimo error
min_v = aprox(I); % mejor aproximación

loglog(h, error, '.'); % graficar los puntos calculados en escala logarítmica
xlabel("h");
ylabel("magnitud del error absoluto");
grid;

printf(">Aproximación por diferencias finitas hacia adelante\n\n");
printf("función a proximar: "); display(y);
printf("para el valor: x=%d\n", x);
printf("valor real: %Le\n", dydx);
printf("mejor aproximación: %Le\n", min_v);
printf("menor error absoluto: %Le\n", min_e);
printf("mejor paso h encontrado: %Le\n", min_h);
printf("paso h sugerido sqrt(eps): %Le\n", sqrt(eps));
printf("diferencia entre los dos pasos h: %Le\n", abs(min_h-sqrt(eps)));

```

Figure 16: Código de la figura anterior

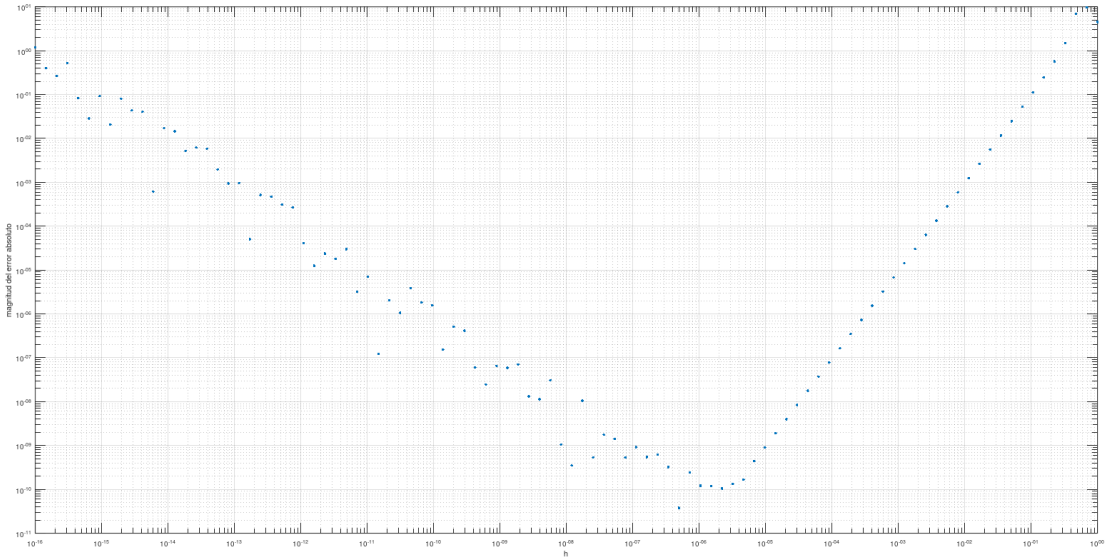

Figure 17: **Diferencias finitas centrales**: magnitud del error absoluto en función del paso h

```
>Aproximación por diferencias finitas centrales

función a proximar: y = @tan
para el valor: x=1
valor real: 3.42552
mejor aproximación: 3.42552
menor error absoluto: 3.78466e-11
mejor paso h encontrado: 4.97702e-07
paso h sugerido sqrt(eps): 1.49012e-08
diferencia entre los dos pasos h: 4.828012e-07
>>
```

Figure 18: Salida por consola

```
clc; %limpiar la consola

h = logspace(-16, 0, 100); % distribución logarítmica de valores a probar
                           % entre 10^-16 y 10^0

x = 1; % punto a evaluar
y = @tan; % función a derivar
dydx = sec(x).^2; % derivada real de la función

aprox = (y(x+h)-y(x-h))./(2*h); % aproximación de la derivada por diferencias
                                % finitas centrales

error = abs(aprox - dydx); % error absoluto de la aproximación

[min_e, I] = min(error); % valor mínimo e índice entre los errores calculados
min_h = h(I); % valor de h para el cual se obtuvo el mínimo error
min_v = aprox(I); % mejor aproximación

loglog(h, error, '.'); % graficar los puntos calculados en escala logarítmica
xlabel("h");
ylabel("magnitud del error absoluto");
grid;

printf(">Aproximación por diferencias finitas centrales\n\n");
printf("función a proximar: "); display(y);
printf("para el valor: x=%d\n", x);
printf("valor real: %d\n", dydx);
printf("mejor aproximación: %d\n", min_v);
printf("menor error absoluto: %d\n", min_e);
printf("mejor paso h encontrado: %d\n", min_h);
printf("paso h sugerido sqrt(eps): %d\n", sqrt(eps));
printf("diferencia entre los dos pasos h: %Le\n", abs(min_h-sqrt(eps)));

```

Figure 19: Código de la figura anterior

4 Ejercicio 4

Se realizó un programa para computar la serie infinita conocida como serie armónica $\sum_{n=1}^{\infty} 1/n$, en simple y en doble precisión. La serie en cuestión es divergente. Sin embargo, puesto que se realizó la

sumatoria con una precisión finita, se obtuvo un resultado finito. Podría pensarse que eventualmente el número $1/n$ será tan pequeño que producirá un undeflow, o que el contador n será tan grande que producirá un overflow, estos superan las posibilidades de representación propias de la máquina. Aunque, de hecho, la sumatoria terminará antes de producirse cualquiera de estas dos excepciones, precisamente cuando 1/n resulte despreciable con respecto a la suma parcial. Esto es así debido a la operación de suma en punto flotante que debe igualar los exponentes perdiendo bits de la mantisa, y, eventualmente cuando la diferencia de magintud es muy grande, igualar el número más pequeño a cero. Esto ocurrirá puntualmente cuando $1/n < \epsilon_{mach} \sum_{k=1}^{n-1}(1/k)$. Luego la serie divergente resulta en una suma finita. Se realizó un gráfico de cómo crece la suma parcial en función de la n-ésima iteración para una mejor apreciación.

Para la sumatoria en doble precisión se procedió de manera similar pero estableciendo un criterio de parada al tratarse de un problema de mayor complejidad computacional, al menos para las computadoras actuales. El experimento fue interrumpido abrutamente a las siete horas de haber comenzado por cuestiones de limitaciones técnincas, el mismo no solo insumió mucho tiempo sino que también ocupó gran cantidad de la memoria de la máquina, y fue trabajoso luego reconstrir los datos obtenidos para generar el gráfico.

Como conclusión, con la aritmética de doble precisión se obtiene una considerable mejora en la precisión de los cálculos pero implica un mayor consumo de los recursos de la máquina, afectando por tanto al tiempo de cómputo. Este *tradeoff* debe ser tenido en cuenta en cada caso particular de aplicación.

```
>> >Serie Armónica en simple precisión
El resultado de la sumatoria es: 1.540368e+01
se realizaron 2097153 iteraciones y demoró 24.6309 segundos
```

Figure 20: Sumatoria Serie Armónica $\sum_{n=1}^{\infty} 1/n$ en precisión simple.

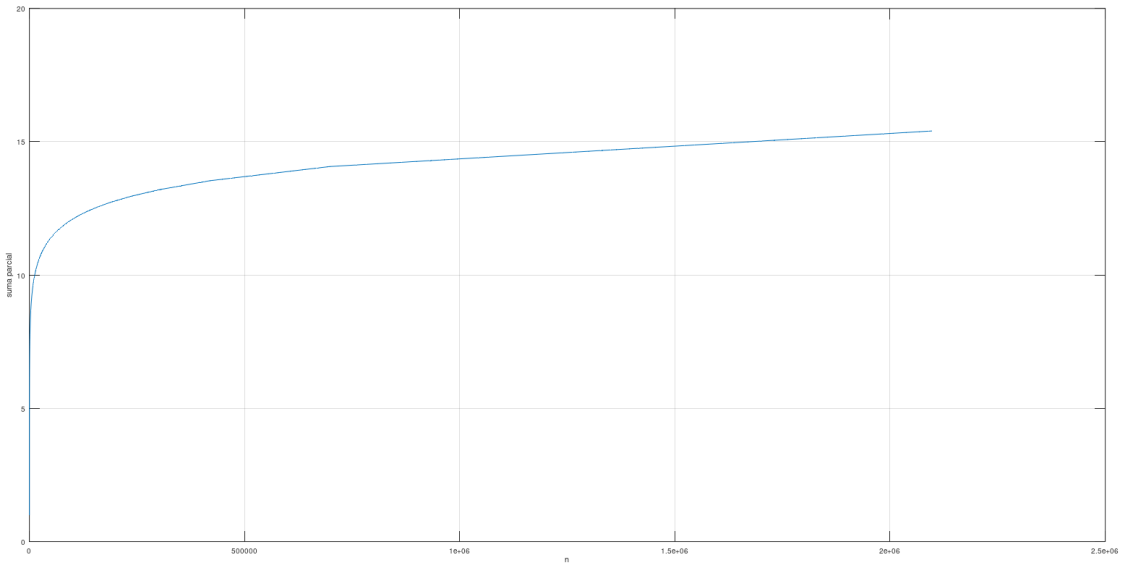

Figure 21: Suma parcial en función de la n-ésima iteración

```
n = single(1); % comienzar en la iteración 1

anterior = single(1);
siguiente = single(0);
parciales = [];

tic(); % empezar el contador

while(anterior != siguiente)
  anterior = siguiente;
  siguiente = anterior + (1/n);
  parciales(n++) = siguiente; % guardar sumas parciales
endwhile

tiempo_total = toc(); % obtener el tiempo demorado
plot([1:n-1], parciales) % graficar suma parcial en función de la iteración n
xlabel('n');
ylabel('suma parcial');
grid;

printf(">Serie Armónica en simple precisión\n");
printf("El resultado de la sumatoria es: %Le\n", siguiente);
printf("se realizaron %ld iteraciones y demoró %ld segundos", n, tiempo_total);
```

Figure 22: Código de la figura anterior

```
>Serie Armónica en doble precisión
El resultado de la sumatoria es: 1.923011e+01
se realizaron 126138549 iteraciones y demoró 7.472504e+00 horas
>>
```

Figure 23: Sumatoria Serie Armónica $\sum_{n=1}^{\infty} 1/n$ en precisión doble acotada.

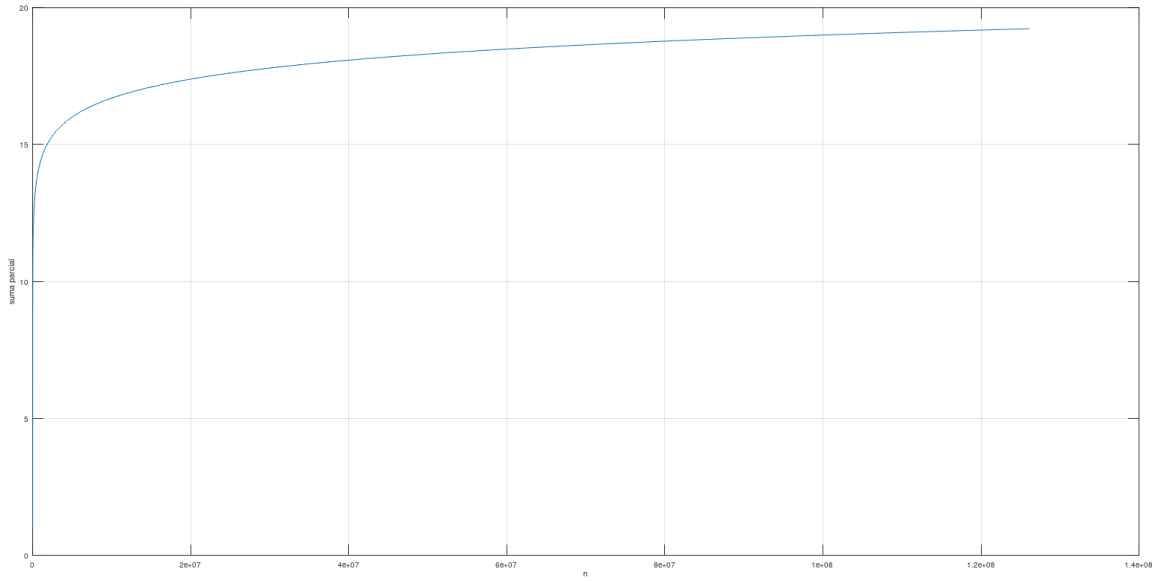

Figure 24: Suma parcial en función de la n-ésima iteración

```
n = 1; % comienzar en la iteración 1

anterior = 1;
siguiente = 0;
parciales = [];

tic(); % empezar el contador

while(anterior != siguiente)
  anterior = siguiente;
  siguiente = anterior + (1/n);
  parciales(n++) = siguiente; % guardar sumas parciales
endwhile

tiempo_total = toc(); % obtener el tiempo demorado
plot([1:n-1], parciales) % graficar suma parcial en función de la iteración n
xlabel('n');
ylabel('suma parcial');
grid;

printf(">Serie Armónica en doble precisión\n");
printf("El resultado de la sumatoria es: %Le\n", siguiente);
printf("se realizaron %ld iteraciones y demoró %Le horas", n, tiempo_total/3600);
```

Figure 25: Código de la figura anterior

5 Ejercicio 5

Se realizó un programa para construir la matriz A propuesta con $n = 50$ y calcular su inversa. Además, se calculó el error sumando todos los elementos de la matriz $C = A \cdot A^{-1}$ y se lo comparó con I_n.

Si A^{-1} fue correctamente calculada, entonces $C = A \cdot A^{-1} = I$. Luego, la suma de los elementos de C debe ser igual a la suma de los elementos de I_{50}, esto es $\sum_{i=1}^{50} \sum_{j=1}^{50} Cij = 50$. Calculando C y la suma de sus elementos, y restándole 50 al resultado, se obtiene una medida del error.

$$A = \begin{bmatrix} \frac{n+2}{2n+2} & -\frac{1}{2} & 0 & 0 & \dots & 0 & 0 & \frac{1}{2n+2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \dots & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & \dots & 0 & 0 & 0 \\ . & . & . & . & \dots & . & . & . \\ . & . & . & . & \dots & . & . & . \\ . & . & . & . & \dots & . & . & . \\ . & . & . & . & \dots & . & . & . \\ 0 & 0 & 0 & 0 & \dots & -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{2n+2} & 0 & 0 & 0 & \dots & 0 & -\frac{1}{2} & \frac{n+2}{2n+2} \end{bmatrix}$$

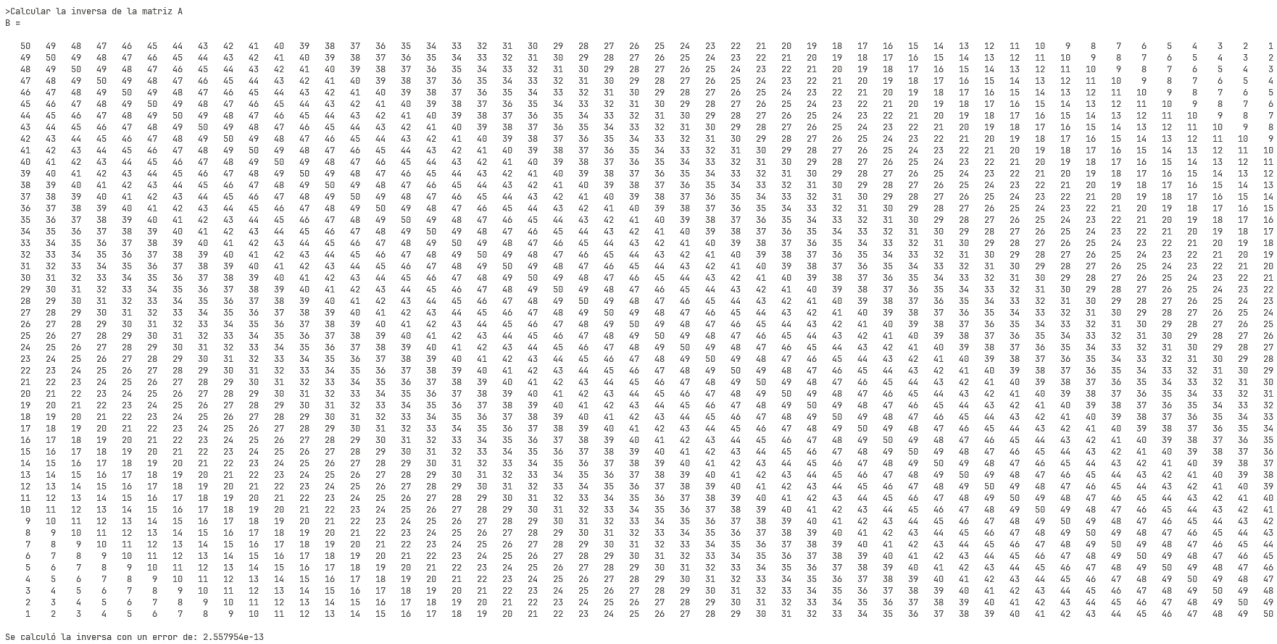

Figure 26: Cálculo de A^{-1} con 2.557954e−13 de error.

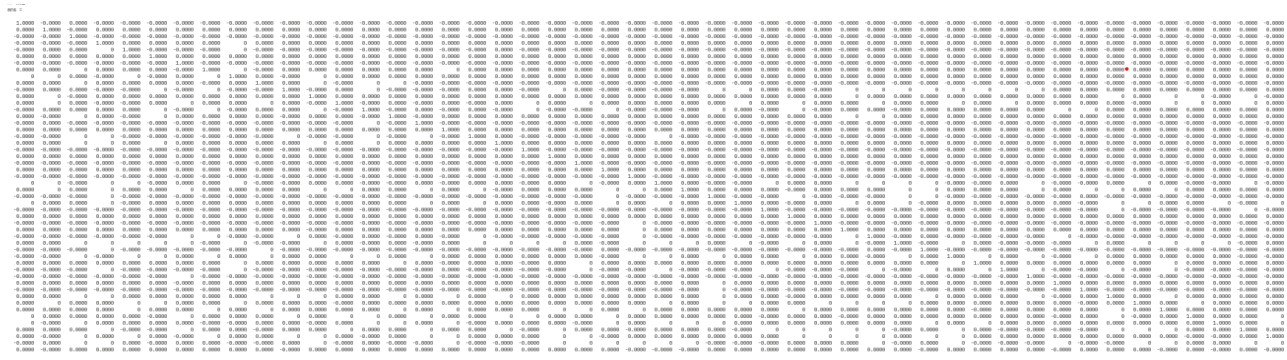

Figure 27: Matric C = $A \cdot A^{-1}$ obtenida

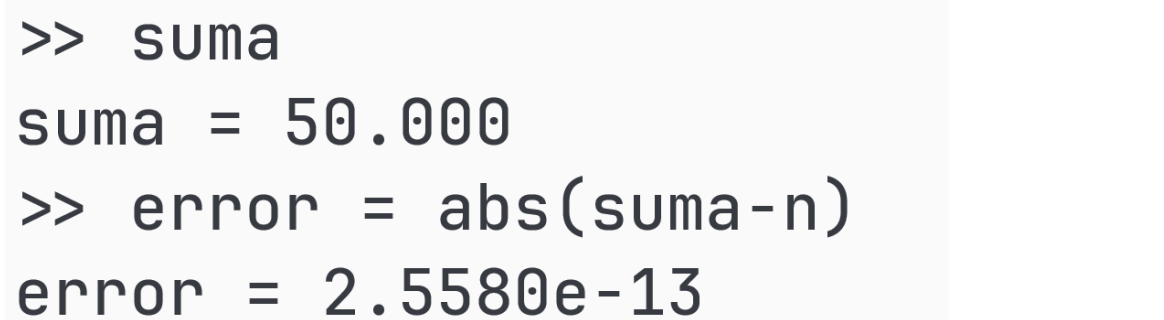

Figure 28: Diferencia absoluta de la suma de los elementos de C entre n

```
clc; % limpiar la consola

n = 50;

function resultado = obtener_matriz(n)
  x1 = (n+2)/(2*n+2);
  x2 = 1/(2*n+2);
  x3 = [-1/2, 1, -1/2];

  primera_fila = [x1, x3(3), zeros(1, n-3), x2];

  medio = [];
  for i=1:n-2
    nueva_fila = [zeros(1, i-1), x3, zeros(1, n-(i-1)-3)];
    medio = [medio; nueva_fila]; % agregar la nueva fila
  endfor

  ultima_fila = [x2, zeros(1, n-3), x3(1), x1];

  resultado = [primera_fila; medio; ultima_fila];
endfunction

A = obtener_matriz(n);
B = inv(A); % obtener la inversa de A

C = A*B;
suma = 0;

for i=1:n
  for j=1:n
    suma += C(i, j);
  endfor
endfor

printf(">Calcular la inversa de la matriz A\n");
display(B); % imrpimir la matriz B
printf("Se calculó la inversa con un error de: %Le\n", abs(n-suma));

```

Figure 29: Código de la figura anterior

6 Conclusión

Como conclusión, la teoría de errores es fundamental para el científico en computación, ya que le permite manejar los métodos numéricos con precisión y exáctictud, para resolver problemas complejos con soluciones óptimas y confiables. Esto es fundamental en problemas interdisiplinarios del mundo real, que trabajan con variables contínuas y que serán ampliamente utilizados.