FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Practico - Análisis Numérico I

[75.12] Análisis Numérico I
Curso Tarela
Primer cuatrimestre de 2021

Alumno 1:	Argel, Ignacio
Número de padrón:	104351
Email:	iargel@fi.uba.ar
Alumno 2:	Bareiro, Facundo
Número de padrón:	103807
Email:	fbareiro@fi.uba.ar

Índice

1. Introducción

Los objetivos de este informe son: analizar el efecto de los errores de redondeo al resolver sistemas de ecuaciones lineales por métodos directos, poder desarrollar un algoritmo que obtenga la descomposición LU de una matriz, su inversa, el número de condición y el error de redondeo con respecto a la solución exacta matemática. Los cálculos fueron hechos en la precisión de python que son 28 dígitos significativos. A su vez poder obtener alguna relación entre los errores de redondeo y el número de condición. Por último se desea perturbar el término independiente con el fin de analizar el impacto en el error de redondeo.
Para este se utilizo Python como lenguaje de programación y el siguiente conjunto de librerías:

- NumPy : Para trabajar con matrices.
- Matplotlib: Para generar los gráficos utilizados en este informe.

2. Desarrollo

2.1. Obtener Matriz de Hilbert

Para este trabajo se utilizó la matriz de Hilbert con el objetivo de lograr una solución afectada por errores de redondeo apreciables sin tener la necesidad de trabajar con sistemas de ecuaciones muy grandes, ya que esta matriz está mal condicionada. A su vez esta matriz es simétrica y definida positiva y esto facilita los cálculos ya que al aplicar la triangulación no es necesario para este tipo de matrices realizar pivoteo alguno. Se procede a mostrar las distintas matrices de Hilbert obtenidas con la formula dada en el enunciado al variar N:

$$H_{i,j} = (\frac{1}{i+j-1}) \qquad j,k = 0,...,N$$

2.1.1. Matriz hilbert 4x4

$$\begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 \\ 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 \\ 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 \\ 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 \end{bmatrix}$$

2.1.2. Matriz hilbert 5x5

$$\begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 \\ 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 \\ 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 \\ 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 \\ 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 \end{bmatrix}$$

2.1.3. Matriz hilbert 6x6

$$\begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 \\ 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 \\ 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 \\ 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 \\ 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 \\ 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 \end{bmatrix}$$

2.1.4. Matriz hilbert 7x7

$$\begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 \\ 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 \\ 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 \\ 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 \\ 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 \\ 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 \\ 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 & 0{,}07692 \end{bmatrix}$$

2.1.5. Matriz hilbert 8x8

$$\begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 \\ 0{,}50000 & 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 \\ 0{,}33333 & 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 \\ 0{,}25000 & 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 \\ 0{,}20000 & 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 \\ 0{,}16667 & 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 & 0{,}07692 \\ 0{,}14286 & 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 & 0{,}07692 & 0{,}07143 \\ 0{,}12500 & 0{,}11111 & 0{,}10000 & 0{,}09091 & 0{,}08333 & 0{,}07692 & 0{,}07143 & 0{,}06667 \end{bmatrix}$$

2.2. Descomposición LU

Una forma práctica de resolver sistemas de ecuaciones lineales cuando varía el término independiente es aplicar la descomposición LU. La misma requiere nombrar a la matriz A del sistema $Ax = b$, como:

$$A = LU \tag{1}$$

Siendo L la matriz triangular inferior que se construye con los multiplicadores obtenidos en la eliminación de Gauss y U la matriz triangulada que surge de la misma eliminación. Desarrollando a partir de la ecuación 1, llegamos a lo siguiente:

$$Ly = b \tag{2}$$

$$Ux = y \tag{3}$$

Por lo que hallar x implicaría resolver 2 para hallar Y y luego resolver 3 y hallar X. Esto facilita mucho el cálculo ya que resolver 2 es resolver un sistema de una matriz triangular inferior mediante sustitución directa con la siguiente fórmula:

$$y_i = \frac{b_i}{L_{i,i}} - \frac{1}{L_{i,i}} \sum_{j=0}^{i-1} (L_{i,j} y_j) \qquad i \in 0, \ldots, n \tag{4}$$

Y luego en 3 se tiene una sustitución inversa ya que U es triangular superior y por ende X se obtiene con:

$$x_i = \frac{y_i}{U_{i,i}} - \frac{1}{U_{i,i}} \sum_{j=i-1}^{n} (U_{i,j} x_j) \qquad i \in n, \ldots, 0 \tag{5}$$

A continuación se muestra las matrices L y U obtenidas mediante el programa para $N = 4$:

2.2.1. Matriz L y U hilbert 4x4

$$L = \begin{bmatrix} 1{,}00000 & 0{,}00000 & 0{,}00000 & 0{,}00000 \\ 0{,}50000 & 1{,}00000 & 0{,}00000 & 0{,}00000 \\ 0{,}33333 & 1{,}00000 & 1{,}00000 & 0{,}00000 \\ 0{,}25000 & 0{,}90000 & 1{,}50000 & 1{,}00000 \end{bmatrix} U = \begin{bmatrix} 1{,}00000 & 0{,}50000 & 0{,}33333 & 0{,}25000 \\ 0{,}00000 & 0{,}08333 & 0{,}08333 & 0{,}07500 \\ 0{,}00000 & 0{,}00000 & 0{,}00556 & 0{,}00833 \\ 0{,}00000 & 0{,}00000 & 0{,}00000 & 0{,}00036 \end{bmatrix}$$

3. Cálculo de la inversa

Para calcular la inversa de una matriz, como:

$$AA^{-1} = I$$

lo que se hace es plantear N sistemas de ecuaciones lineales a resolver (siendo N la dimensión de las matrices)

$$Ax^{(j)} = I^{(j)} \qquad j = 1, \ldots, n$$

donde en cada SEL la incógnita es una columna de la inversa y el término independiente es su respectivo canónico. Una vez que se obtienen las N soluciones entonces, se tiene la matriz inversa. Este es un caso donde la descomposición LU facilita los cálculos ya que el término independiente varía y no es conveniente aplicar N veces la descomposición de Gauss. El programa desarrollado arrojó las siguientes inversas para N perteneciente entre 4 y 8:

3.1. Inversa 4x4

$$\begin{bmatrix} 16{,}00000 & -120{,}00000 & 240{,}00000 & -140{,}00000 \\ -120{,}00000 & 1200{,}00000 & -2700{,}00000 & 1680{,}00000 \\ 240{,}00000 & -2700{,}00000 & 6480{,}00000 & -4200{,}00000 \\ -140{,}00000 & 1680{,}00000 & -4200{,}00000 & 2800{,}00000 \end{bmatrix}$$

3.2. Inversa 5x5

$$\begin{bmatrix} 25{,}00000 & -300{,}00000 & 1050{,}00000 & -1400{,}00000 & 630{,}00000 \\ -300{,}00000 & 4800{,}00000 & -18900{,}00000 & 26880{,}00000 & -12600{,}00000 \\ 1050{,}00000 & -18900{,}00000 & 79380{,}00000 & -117600{,}00000 & 56700{,}00000 \\ -1400{,}00000 & 26880{,}00000 & -117600{,}00000 & 179200{,}00000 & -88200{,}00000 \\ 630{,}00000 & -12600{,}00000 & 56700{,}00000 & -88200{,}00000 & 44100{,}00000 \end{bmatrix}$$

3.3. Inversa 6x6

$$\begin{bmatrix} -630{,}00000 & 14700{,}00000 & -88200{,}00001 & 211680{,}00002 & -220500{,}00002 & 83160{,}00001 \\ 3360{,}00000 & -88200{,}00001 & 564480{,}00004 & -1411200{,}00011 & 1512000{,}00013 & -582120{,}00005 \\ -7560{,}00000 & 211680{,}00002 & -1411200{,}00011 & 3628800{,}00030 & -3969000{,}00033 & 1552320{,}00013 \\ 7560{,}00000 & -220500{,}00002 & 1512000{,}00013 & -3969000{,}00033 & 4410000{,}00037 & -1746360{,}00015 \\ -2772{,}00000 & 83160{,}00001 & -582120{,}00005 & 1552320{,}00013 & -1746360{,}00015 & 698544{,}00006 \end{bmatrix}$$

3.4. Inversa 7x7

$$\begin{bmatrix} 49{,}00000 & -1176{,}00000 & 8820{,}00003 & -29400{,}00010 & 48510{,}00019 & -38808{,}00016 & 12012{,}00005 \\ -1176{,}00000 & 37632{,}00011 & -317520{,}00106 & 1128960{,}00412 & -1940400{,}00758 & 1596672{,}00656 & -504504{,}00216 \\ 8820{,}00003 & -317520{,}00106 & 2857680{,}01023 & -10584000{,}03991 & 18711000{,}07335 & -15717240{,}06350 & 5045040{,}02088 \\ -29400{,}00010 & 1128960{,}00413 & -10584000{,}03992 & 40320000{,}15569 & -72765000{,}28610 & 62092800{,}24767 & -20180160{,}08144 \\ 48510{,}00019 & -1940400{,}00758 & 18711000{,}07337 & -72765000{,}28612 & 133402500{,}52573 & -115259760{,}45508 & 37837800{,}14963 \\ -38808{,}00016 & 1596672{,}00657 & -15717240{,}06352 & 62092800{,}24768 & -115259760{,}45506 & 100590336{,}39389 & -33297264{,}12951 \\ 12012{,}00005 & -504504{,}00216 & 5045040{,}02089 & -20180160{,}08143 & 37837800{,}14961 & -33297264{,}12949 & 11099088{,}04257 \end{bmatrix}$$

3.5. Inversa 8x8

$$\begin{bmatrix} 64{,}0 & -2016{,}0 & 20160{,}0 & -92400{,}0 & 221760{,}0 & -288288{,}0 & 192192{,}0 & -51480{,}0 \\ -2016{,}0 & 84672{,}0 & -952560{,}0 & 4656960{,}1 & -11642400{,}2 & 15567552{,}3 & -10594584{,}2 & 2882880{,}1 \\ 20160{,}0 & -952560{,}0 & 11430720{,}2 & -58212001{,}0 & 149688002{,}5 & -204324123{,}3 & 141261122{,}2 & -38918880{,}6 \\ -92400{,}0 & 4656960{,}1 & -58212001{,}0 & 304920004{,}8 & -800415011{,}9 & 1109908815{,}5 & -776936170{,}3 & 216216002{,}7 \\ 221760{,}0 & -11642400{,}2 & 149688002{,}4 & -800415011{,}8 & 2134440028{,}8 & -2996753797{,}4 & 2118916824{,}6 & -594594006{,}5 \\ -288288{,}0 & 15567552{,}3 & -204324123{,}2 & 1109908815{,}4 & -2996753797{,}2 & 4249941743{,}7 & -3030051055{,}1 & 856215368{,}1 \\ 192192{,}0 & -10594584{,}2 & 141261122{,}1 & -776936170{,}1 & 2118916824{,}3 & -3030051054{,}9 & 2175421267{,}9 & -618377765{,}1 \\ -51480{,}0 & 2882880{,}1 & -38918880{,}6 & 216216002{,}7 & -594594006{,}3 & 856215368{,}0 & -618377765{,}1 & 176679361{,}3 \end{bmatrix}$$

A simple vista parecería que las matrices obtenidas no tienen error con respecto a las proporcionadas en el anexo por lo menos hasta N=6. Sin embargo, en la conclusión se retomarán estos resultados y se analizarán en profundidad.

4. Cálculo del error de redondeo y del número de condición

Para poder verificar si las matrices de la sección anterior arrastran algún tipo de error de redondeo, lo que se debe hacer es calcular el mismo mediante la siguiente fórmula:

$$\frac{\| H^{-1} - H^{-1}_{anexo} \|}{\| H^{-1}_{anexo} \|} \tag{6}$$

En este caso se pide la norma infinito, donde para una matriz es el mayor valor de un vector que posee la suma de sus filas en módulo. A su vez para comprobar si la matriz de Hilbert está bien o mal condicionada, el parámetro a utilizar es el número de condición (K), que se obtiene como

$$K(H) \equiv \| H \| \| H^{-1} \| \tag{7}$$

donde las matrices a utilizar son las de Hilbert calculadas y las respectivas inversas proporcionadas en el anexo del enunciado. Para poder analizar este número se tiene en cuenta que si:

- $K(H)$ chico / orden 1 $\rightarrow$ H bien condicionada.
- $K(H)$ grande / $>>> 1 \rightarrow$ H mal condicionada.

Con estas fórmulas se procedió a graficar el logaritmo en base 10 tanto del error como de K en función de N:
En rojo es el logaritmo en base 10 del número de condición y en azul el logaritmo del error de redondeo.

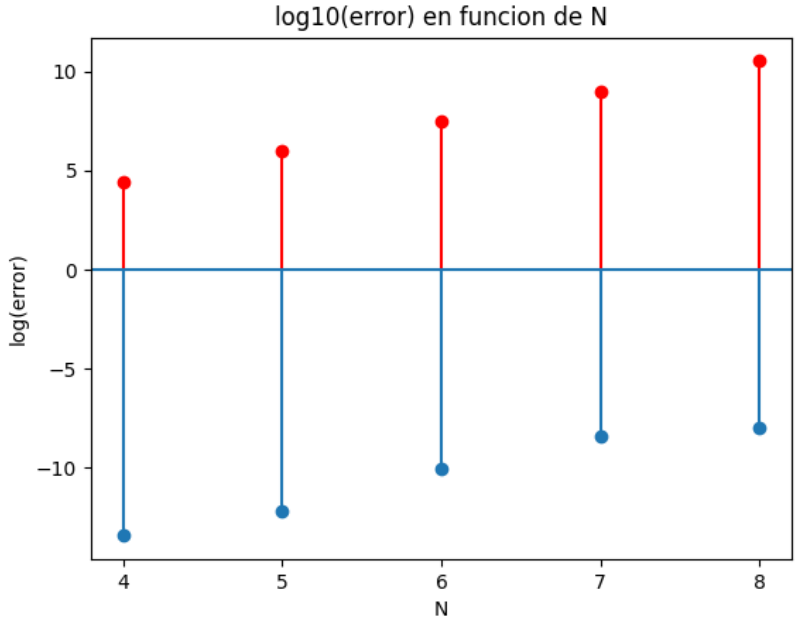

Figura 1: Grafica comparativa errores de redondeo y número de condición en función de N.

Los siguientes son los número de condición de cada matriz:
Número de condición de matriz de 4x4: 28374.999999999996
Número de condición de matriz de 5x5: 943656.0
Número de condición de matriz de 6x6: 29070278.999999996
Número de condición de matriz de 7x7: 985194886.4999999
Número de condición de matriz de 8x8: 33872791094.999996

5. Perturbaciones

El enunciado pedía una vez calculadas las inversas y sus errores, perturbar el término independiente con

$$\Delta b_i = (-1)^i 10^{-p} \qquad p \in 1, 3, 5, 7$$

Para esto los cálculos se mantienen pero los resultados varían y lo que hay que analizar es cómo es que varían. Por eso se procede a calcularles el error de redondeo con la fórmula 6 y a compararlos en un mismo gráfico con los errores obtenidos sin perturbación alguna. El error con perturbación se representa en azul y el obtenido sin perturbación en rojo.

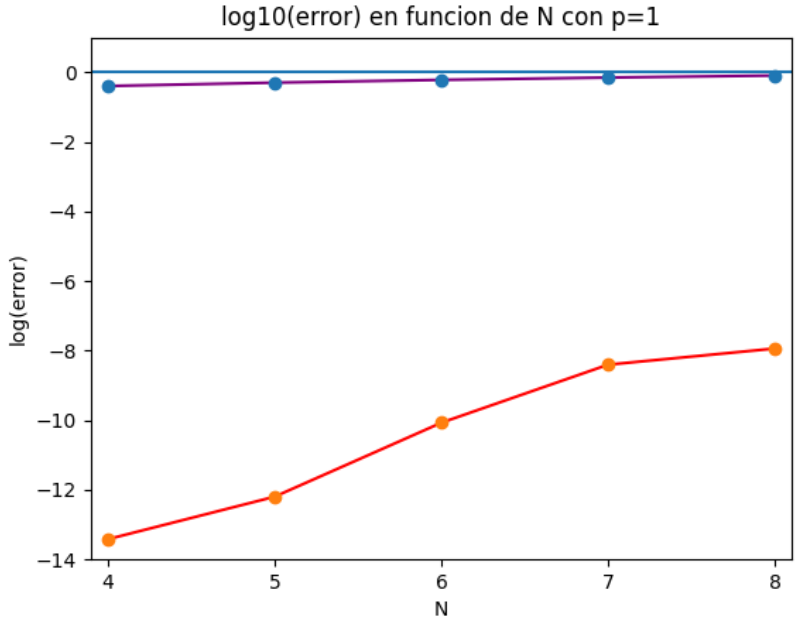

Figura 2: Grafica comparativa errores de redondeo sin perturbación y con perturbación $p = 1$.

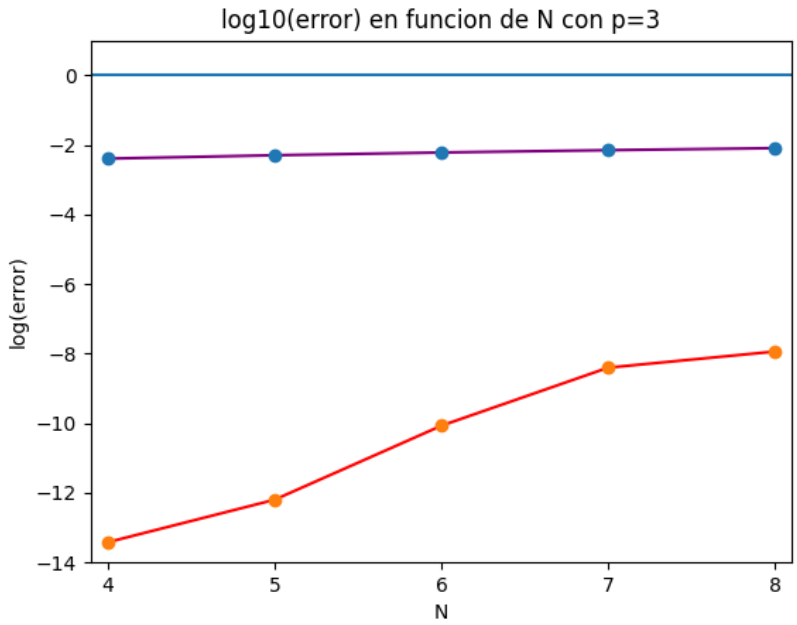

Figura 3: Grafica comparativa errores de redondeo sin perturbación y con perturbación $p = 3$.

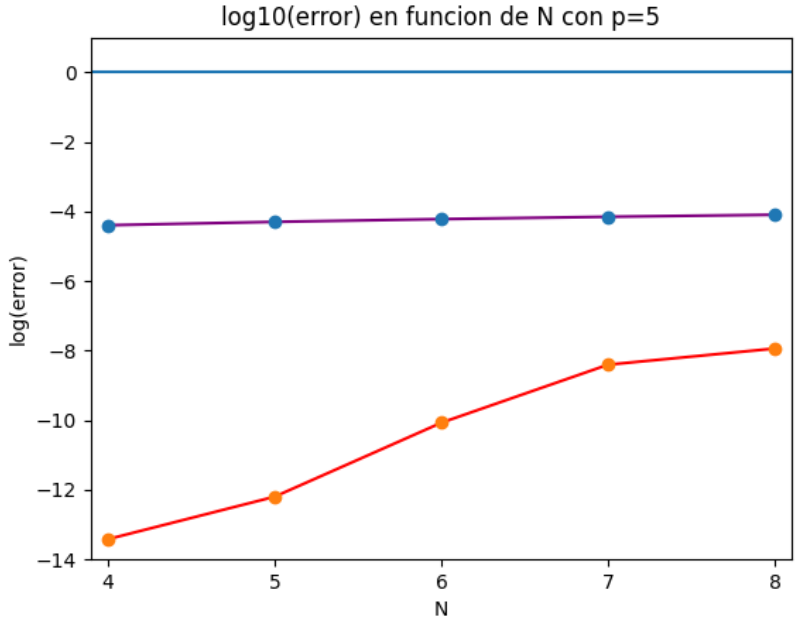

Figura 4: Grafica comparativa errores de redondeo sin perturbación y con perturbación $p = 5$.

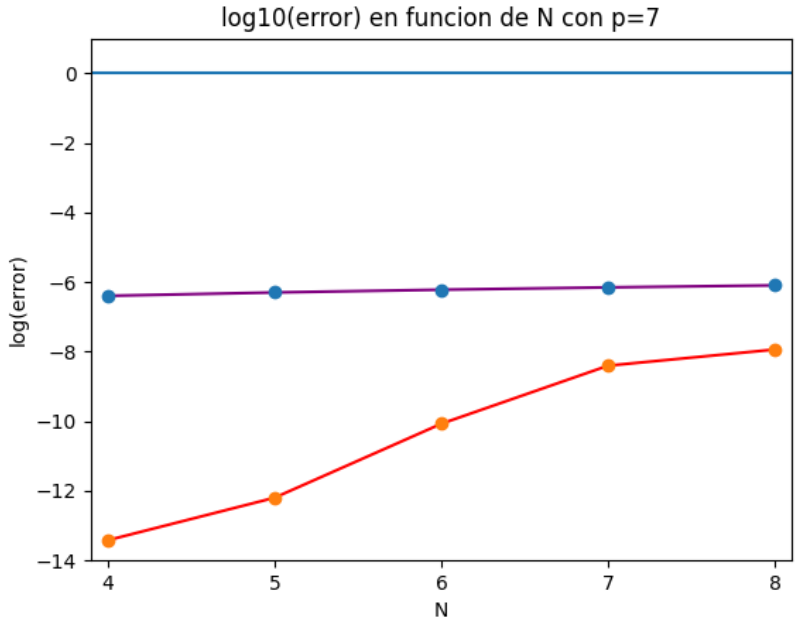

Figura 5: Grafica comparativa errores de redondeo sin perturbación y con perturbación $p = 7$.

A modo de obtener conclusiones más certeras, se propuso la misma perturbacion pero sin cambiar el signo de la misma:

$$\Delta b_i = 10^{-p} \qquad p \in 1, 3, 5, 7$$

En verde se muestra con $p = 1$, rojo es $p = 3$, cyan es $p = 5$ y morado $p = 7$. El azul es sin perturbación.

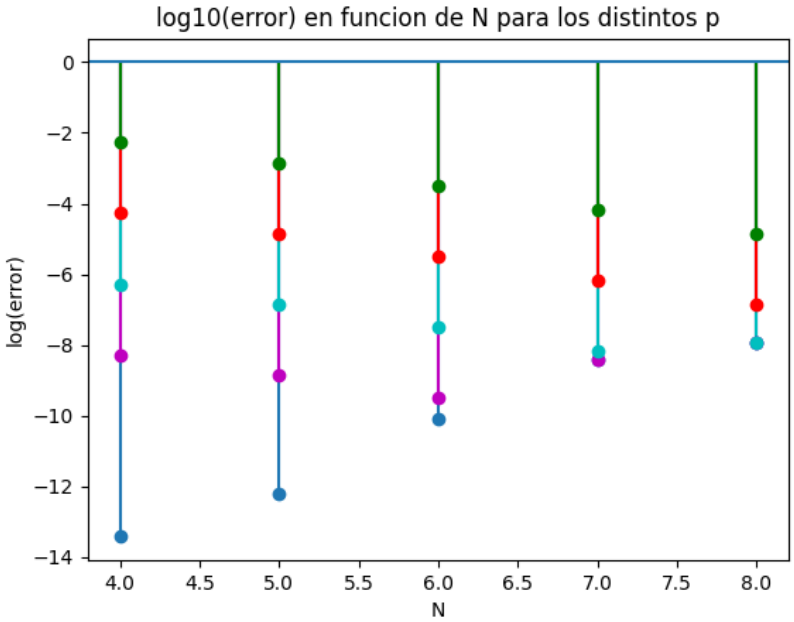

Figura 6: Grafica comparativa errores de redondeo sin perturbación y con perturbación con los distintos p.

6. Conclusión

6.1. Error de redondeo

Con respecto al cálculo de las matrices inversas, a simple vista como fue mencionado el error aparece en N=6. Sin embargo, si nos remontamos a la Figura 1, podemos observar que en realidad sí aparece un error para los N más bajos. Este gráfico presenta un logaritmo creciente y todos de signo negativo lo que quiere decir que a mayor N, mayor es el error; y este resultado es respaldado por la teoría ya que para N mayores, la cantidad de operaciones aumenta significativamente y por lo tanto se arrastra un error mayor.

6.2. Número de condición

A su vez la Figura 1 muestra el logaritmo del número de condición, el cual es positivo y a medida que N crece, este también lo hace. También podemos ver los resultados que se tienen para el número de condición, los cuales son mucho mayores a 1 para todos los N, lo que indica que se está en presencia de una matriz mal condicionada. Esto último si lo analizamos en base a la fórmula 7 y a las matrices de la sección 2.1 y 3, vemos que inevitablemente el valor de K siempre será alto ya que la norma infinito de H siempre tomará la suma en módulo de la primer fila (porque es la fila que tiene los números mayores) y será siempre un número mayor a 1. Y para la norma infinito de la inversa a pesar de no saber con certeza que fila proveerá el mayor valor, si se puede intuir que la misma será de un número de orden elevado mucho mayor a 1.
Como conclusión es correcto decir que todas estas matrices de Hilbert están mal condicionadas y que a N mayores peor condicionada estará la matriz.

6.3. Relación entre error de redondeo y número de condición

A pesar de estas magnitudes indicar resultados que no parecerían tener relación alguna, en el gráfico de la Figura 1 podemos notar cierta tendencia que se repite entre estas.
Ya fue mencionado que a mayor N, mayor es el número de condición y mayor el error de redondeo. Esto último está relacionado de la manera siguiente: a mayor N, peor condicionada estará la matriz, mayor será el K y por lo tanto mayor error de redondeo se arrastra en los cálculos.

6.4. Análisis del efecto de las perturbaciones en el error de redondeo

Para la perturbación propuesta en el enunciado se tiene que a mayor p, menor será la perturbación, por lo que se esperaría que a mayor p, menor sea la diferencia con la solución original. Mirando los gráficos de la sección 5 vemos que esto último se comprueba, ya que a mayor p, menor es la diferencia entre el error de redondeo con perturbación y el error de redondeo sin perturbación. Por eso se concluye que mientras mayor es la perturbación, por un lado mayor error se tiene con respecto a la solución exacta matemática y por otro que más lejos se estará de la solución numérica sin perturbación.
Vale destacar que para una misma perturbación, si el N aumenta o disminuye no dice nada en cuanto a si el resultado es más preciso o no. Esto lo podemos ver para las dos perturbaciones utilizadas donde para la dada por el enunciado, para un mismo p aumenta el error a medida que N crece y para el otro disminuye el error a medida que N crece, por eso no se puede asegurar que haya una dependencia entre el tamaño de la matriz y el error de redondeo para estas perturbaciones. Lo que sí se puede decir es que va a depender de la perturbación que se utilice.

6.5. Anexo

En el código presentado se muestran las 2 funciones que calcular el logaritmo del error de la inversa de una matriz pasandole la exacta por parámetro. El primero lo calcula sin perturbaciones y el segundo con las perturbaciones propuestas en el enunciado.La última función calcula la norma a infinito de una matriz. Decidimos no adjuntar las partes en las que se llama a las funciones, ni cuando se crean los gráficos o se exportan las matrices csv para preparar el informe ya que nos pareció irrelevante.

```
1
2   def calculoerror(N,Hinversa):
3   'Defino la matriz de hilbert'
4   arreglos =[]
5
6   for i in range(N):
7       subarreglo = [0 for i in range(N)]
8
9       for j in range(N):
10          constanteHilbert = (i+1)+(j+1)-1
11          subarreglo[j] = (1/(constanteHilbert))
12      arreglos.append(subarreglo)
13
14  hilbertMatrix = np.array(arreglos)
15
16  matrizU = hilbertMatrix.copy()
17
18  #defino las matrices L
19  matrizL= np.identity(N)
20  'defino la matriz L'
21  columna=0
22  while columna < N:
23      for fila in range (N):
24          if fila>columna:
25              m_ia=(matrizU[fila][columna])/(matrizU[columna][columna])
26              for j in range(1,N-columna):
```

```
                    matrizU[fila][columna+j] = matrizU[fila][columna+j]-m_ia*matrizU[
                        columna][columna+j]
                matrizL[fila][columna]=m_ia
        columna+=1
    'defino la matriz U'
    for fil in range(N):
            for col in range(N):
                if (col < fil) :
                    matrizU[fil,col] = 0

    x= np.zeros((N,N))
    def sumatoria1(i,y):
        acum=0
        for j in range(i):
            acum+=matrizL[i][j]*y[j]
        return(acum)
    def sumatoria2(i,xk):
        acum=0
        for j in range(i+1,N):
            acum+=matrizU[i][j]*xk[j]
        return(acum)
    canonico=0
    while canonico < N:
        #creo el termino independiente
        b=np.zeros(N) #obs deberia quedar transpuesto
        for i in range(N):
            if i==canonico:
                b[i]=1
        #calculo Ly=b con este b
        y=np.zeros(N) #transpuesto deberia
        for i in range(N):
            for j in range(N):
                if i>=j:
                    y[i]=(b[i]/(matrizL[i][i]))-(1/(matrizL[i][i])*sumatoria1(i,y))
        #calculo Ux=y
        xk=np.zeros(N)

        for i in range(N-1,-1,-1):
            for j in range(N-1,-1,-1):
                if i<=j:
                    xk[i]=(y[i]/(matrizU[i][i]))-(1/(matrizU[i][i])*sumatoria2(i,xk))

        x[:,canonico]=xk
        canonico+=1
        #calculo norma infinito

    difmatrices = x-Hinversa
    normainf1 = normaInfinito(difmatrices, N)
    normainf2 = normaInfinito(Hinversa, N)
    Error = normainf1/normainf2
    logerror = math.log10(Error)
    numeroCondicion = normaInfinito(hilbertMatrix,N)*normaInfinito(Hinversa,N)
    logNumeroCondicion = math.log10(numeroCondicion)
    return(logerror,numeroCondicion)


def calculoerrorConPerturbaciones(N,Hinversa):
'Defino la matriz de hilbert'
arreglos =[]

for i in range(N):
    subarreglo = [0 for i in range(N)]

    for j in range(N):
        constanteHilbert = (i+1)+(j+1)-1
        subarreglo[j] = (1/(constanteHilbert))
    arreglos.append(subarreglo)
```

```
93
94 hilbertMatrix = np.array(arreglos)
95 matrizU = hilbertMatrix.copy()
96
97 #defino las matrices L
98 matrizL= np.identity(N)
99 'defino la matriz L'
100 columna=0
101 while columna < N:
102     for fila in range (N):
103         if fila>columna:
104             m_ia=(matrizU[fila][columna])/(matrizU[columna][columna])
105             for j in range(1,N-columna):
106                 matrizU[fila][columna+j] = matrizU[fila][columna+j]-m_ia*matrizU[
                        columna][columna+j]
107             matrizL[fila][columna]=m_ia
108     columna+=1
109 'defino la matriz U'
110 for fil in range(N):
111         for col in range(N):
112             if (col < fil) :
113                 matrizU[fil,col] = 0
114 x= np.zeros((N,N))
115 def sumatoria1(i,y):
116     acum=0
117     for j in range(i):
118         acum+=matrizL[i][j]*y[j]
119     return(acum)
120 def sumatoria2(i,xk):
121     acum=0
122     for j in range(i+1,N):
123         acum+=matrizU[i][j]*xk[j]
124     return(acum)
125 inversas=[]
126 logaritmoError=[]
127 for p in range(1,8,2):
128     canonico=0
129     while canonico < N:
130         #creo el termino independiente
131         b=np.zeros(N) #obs deberia quedar transpuesto
132         for i in range(N):
133             if i==canonico:
134                 b[i]=1+((-1)**(i+1))*10**(-p)
135             else:
136                 b[i]=((-1)**(i+1))*10**(-p)
137         #calculo Ly=b con este b
138         y=np.zeros(N) #transpuesto deberia
139         for i in range(N):
140             for j in range(N):
141                 if i>=j:
142                     y[i]=(b[i]/(matrizL[i][i]))-(1/(matrizL[i][i])*sumatoria1(i,y))
143         #calculo Ux=y
144         xk=np.zeros(N)
145
146         for i in range(N-1,-1,-1):
147             for j in range(N-1,-1,-1):
148                 if i<=j:
149                     xk[i]=(y[i]/(matrizU[i][i]))-(1/(matrizU[i][i])*sumatoria2(i,xk
                            ))
150
151         x[:,canonico]=xk
152         canonico+=1
153         #calculo norma infinito
154
155     difmatrices = x-Hinversa
156     normainf1 = normaInfinito(difmatrices, N)
157     normainf2 = normaInfinito(Hinversa, N)
```

```
158     Error = normainf1/normainf2
159     logerror = math.log10(Error)
160     numeroCondicion = normaInfinito(hilbertMatrix,N)*normaInfinito(Hinversa,N)
161     logNumeroCondicion = math.log10(numeroCondicion)
162     logaritmoError.append(logerror)
163 return(logaritmoError)
164
165 'calculo norma infinito'
166 def normaInfinito(matriz,N):
167 max=0
168 for fila in range (N):
169     sumafila= 0
170     for columna in range (N):
171         sumafila += abs(matriz[fila][columna])
172     if max < sumafila:
173         max = sumafila
174 return(max)
```