**Informe de TPM2**

*Facultad de Ingeniería, Universidad de Buenos Aires*

Materia: Modelación Numérica

Código: EB051

Curso: 06

Carrera: Ingeniería en Informática (ambos integrantes)

Cuatrimestre y año: 2C2024

Integrantes

|  |  |  |
| --- | --- | --- |
| **Apellido y nombre** | **E-mail** | **Padrón** |
| Riat Sapulia, Mateo | mriat@fi.uba.ar | 106031 |
| Reimundo, Martín | mreimundo@fi.uba.ar | 106716 |

Nombre del grupo: **Python01**

Docentes

|  |  |
| --- | --- |
| **Apellido y nombre** | **E-mail** |
| Rodriguez, Daniel | drodriguez@fi.uba.ar |
| Machiunas, Valeria | vmachiunas@fi.uba.ar |

**Enunciado**

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media**

**Texto

Descripción generada automáticamente**

**Introducción**

En el presente trabajo se resolverá numéricamente el problema de la catenaria explicado en el enunciado. Inicialmente, se estudiará el modelo teórico de manera analítica con el fin de comprender el funcionamiento de la ecuación original de la catenaria y qué significado tienen cada uno de sus parámetros. Luego, se pasará a un modelo práctico que consiste en un experimento que simula este fenómeno, se tomarán mediciones sobre distintos casos y se resolverá numéricamente con el objetivo de comparar los errores que existen entre como ajusta el modelo teórico, el de los distintos casos en la realidad, y el de un ajuste cuadrático por mínimos cuadrados.

Una vez realizado el trabajo, se espera haber puesto en práctica los conocimientos adquiridos en las unidades 1 (errores), 2 (ecuaciones no lineales) y 3 (ajuste de funciones), además de aprender a hacer los cálculos, operaciones y experimentos pertinentes en un lenguaje de programación donde se pueda realizar este análisis.

**Desarrollo**

Se arrancó verificando la congruencia de las ecuaciones (2), (3) y (4) del enunciado, para las condiciones iniciales del mismo. Para ello, partiendo de la igualdad se igualan las ecuaciones (2) y (3), donde mediante operaciones básicas (ver anexo 1) se llega a la igualdad

Verificando la primera condición, es decir que .

Luego, si se reescribe la ecuación (4) con esta noción, se obtiene tras operaciones básicas:

Ecuación utilizada con mucha frecuencia para estimar la raíz de mediante el método de refinamiento de Newton-Raphson (igualando dicha función a 0).

//borrar esto después, indicador de que terminó el desarrollo de parte 1 a)

Con los intervalos obtenidos mediante el método de bisección, y eligiendo una semilla es posible aplicar el método de refinamiento de Newton-Raphson para encontrar la raíz solicitada con una convergencia más rápida. Para implementarlo en código, previamente se debe definir la derivada de la función, que se llamó df(x) y resulta

Posteriormente, se definió la función que implementa Newton-Raphson (en el código aparece como “NR”), en la cual se evalúa que la tolerancia entre la solución y la semilla sea mayor que la tolerancia máxima permitida (que es el número de máquina en doble () o simple precisión (), según el caso). Es decir, la tolerancia se interpreta como

Además, previo a la evaluación de la condición y mientras que la condición se cumpla, se define a la solución como

donde también la semilla se redefine al principio del ciclo “while” (con el fin de simular al de la teoría) como

de manera tal que el en el marco teórico se interpretaría como

Llamando a la función NR y pasándole por parámetro una semilla que pertenezca a cada uno de los intervalos calculados, se obtienen las 4 raíces de cada inciso.

Por otro lado, en el código se analiza el término de estabilidad de forma experimental, es decir, con la fórmula

siendo el resultado en **doble precisión** de aplicar Newton-Raphson sobre una semilla e el mismo pero calculado en **simple precisión**.

De esta manera, resultó conveniente almacenar (en ambos formatos) el resultado de aplicar Newton-Raphson en una variable, para luego pasar ambos valores como parámetros de la función término de estabilidad (en el código aparece como “te”). Dicha función retorna el número de estabilidad producto de la cuenta descrita.

Además, el término de estabilidad (como el algoritmo en sí) se analiza mediante la gráfica de procesos que se puede observar en el anexo III.

Para continuar el análisis de errores, se hizo una función para simular la condición del problema, la cuál se utiliza con aquellos valores que poseen error inherente, en el inciso a) se trata del coeficiente “e”, en el b) de “b”, y en el c) de ambos, mientras que en d) ambos valores se consideran exactos, y al no estar bien redondeados, a priori no se asume ningún coeficiente para evaluar la condición del problema.

La función condición de problema (“cp” dentro del código), realiza el siguiente cálculo:

donde en este contexto el que se le pasa es el valor con error inherente, y el se corresponde a la perturbación óptima para la condición del problema. En este caso , o sea puede pensarse a porcentualmente, siendo el mínimo, un valor como 1%, y el 1 como el máximo (100%). Por otro lado equivale a la función de Newton-Raphson.

Para encontrar el óptimo para el problema, se cargó un vector de exponentes desde a y después se evaluó la función cp en cada uno de estos exponentes (siendo cada uno el a probar). Cada valor resultante de aplicar la función fue también almacenado en un vector para finalmente graficar la función cp. Para hacer el gráfico en cuestión, fue necesario utilizar la librería “pyplot”, que se importó como “plt”. Siguiendo con el procedimiento, mediante plt.plot y plt.show se observa el gráfico correspondiente a cada raíz de cada inciso al ejecutar el código.

**Resultados**

* a)
* **Raíz 1**: 1.0580715529730582
* Término de estabilidad: 0.033938721666938974
* Condición del problema (óptima): 3.6785754273544473
* **Raíz 2**: 1.887489573513979
* Término de estabilidad: 0.023541407601073715
* Condición del problema (óptima): 3.6863999808756125
* **Raíz 3**: 3.0547849633482493
* Término de estabilidad: 0.07478383649736262
* Condición del problema (óptima): 1.0109117249622304
* **Raíz 4**: 19.999653910164714
* Término de estabilidad: 5.480617679930992e-06
* Condición del problema (óptima): 0.0012434713052191357
* b)
* **Raíz 1**: 1.0748681099273683
* Término de estabilidad: 0.0837144153626382
* Condición del problema (óptima): 1.2303345409057382
* **Raíz 2**: 1.5042040770569831
* Término de estabilidad: 0.04193262415627334
* Condición del problema (óptima): 2.8518060957560314
* **Raíz 3**: 4.357933213569889
* Término de estabilidad: 0.026261616289443106
* Condición del problema (óptima): 3.8336442903672805
* **Raíz 4**: 17.03094459944576
* Término de estabilidad: 0.015167833256955728
* Condición del problema (óptima): 2.213977935360985
* c)
* **Raíz 1**: 1.2146291341449174
* Término de estabilidad: 0.11132982241501785
* Condición del problema (óptima): 20.516324666538043
* **Raíz 2**: 1.3483005402656294
* Término de estabilidad: 0.03003398882799861
* Condición del problema (óptima): 21.301793714171446
* **Raíz 3**: 4.374716655075904
* Término de estabilidad: 0.0271644762730282
* Condición del problema (óptima): 4.020195080136319
* **Raíz 4**: 17.030303670513554
* Término de estabilidad: 0.015182617237065283
* Condición del problema (óptima): 2.2166782392599096
* d)
* **Raíz 1**: 1.2146291341449174
* Término de estabilidad: 0.11132982241501785
* **Raíz 2**: 1.3483005402656294
* Término de estabilidad: 0.03003398882799861
* **Raíz 3**: 4.374716655075904
* Término de estabilidad: 0.0271644762730282
* **Raíz 4**: 17.030303670513554
* Término de estabilidad: 0.015182617237065283

Gráficos del cp (sólo se introduce uno del inciso a, b y c por cuestiones de espacio en el informe, para ver todos los gráficos se debe ejecutar el código):

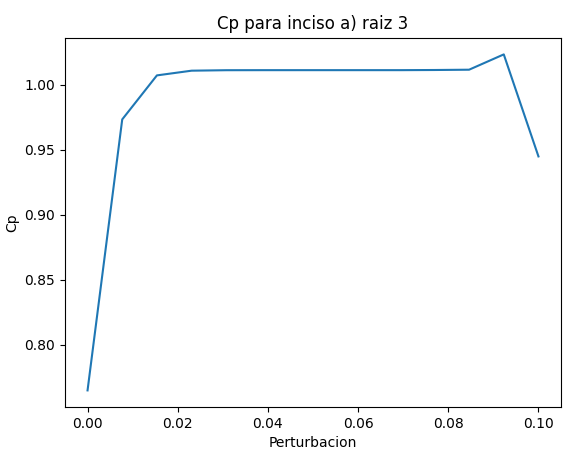


Figura 1: Valores del cp en función de las perturbaciones para la raíz 3 del inciso a)

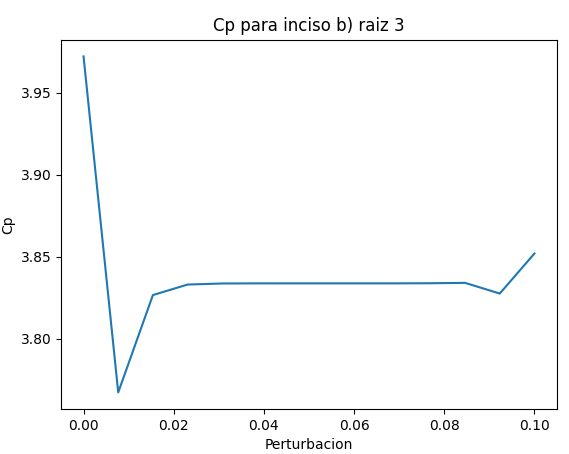


Figura 2: Valores del cp en función de las perturbaciones para la raíz 3 del inciso b)

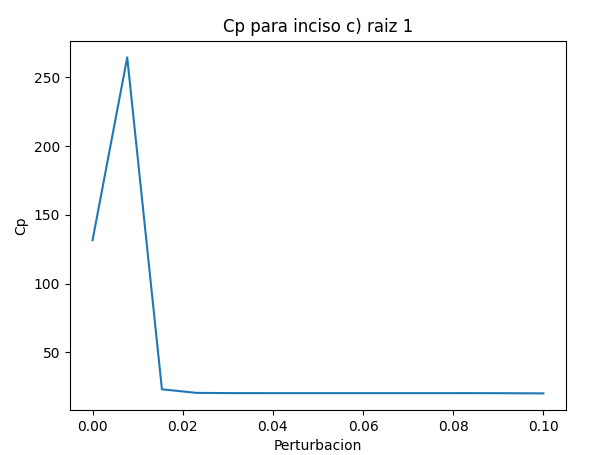


Figura 3: Valores del cp en función de las perturbaciones para la raíz 1 del inciso c)

**Conclusiones**

* A la hora de realizar el trabajo, surgieron complicaciones en el grupo sobre cómo relacionar lo visto en clase con el problema, y aún más, sobre cómo plasmarlo en el código.
* Las raíces halladas con Newton-Raphson corresponden con las reales, por lo que el método funciona siempre y cuando el intervalo esté reducido preferentemente por un método de arranque.
* Cuando la semilla que se introduce en Newton-Raphson está más lejos de la raíz, tiene que realizar más iteraciones para encontrarla adecuadamente.
* Los errores se analizan en tiempo de ejecución. Con los resultados en doble y simple precisión se pudo hallar el término de estabilidad correctamente, mientras que la condición del problema existe únicamente para las variables con error inherente y probando distintos resultados se pudo hallar la perturbación "m" óptima en cada caso.
* Dado que Python tiene un manejo de tipo de datos dinámico, hubo que especificar en todo valor si se trabaja en simple precisión o doble precisión, lo cual dificulta la legibilidad del código.
* Se lograron profundizar los temas de las unidades 1 y 2
* Hacer el mismo análisis a mano resulta más engorroso, y gracias a las herramientas de la computadora se pueden realizar gráficos para visualizar mejor los conceptos.

**Anexo I**

**TPM1-Compilacion.py (La conclusión son comentarios, que no son parte del código)**

#Librerias

import numpy as np

from math import \*

from matplotlib import pyplot as plt

#1) Determinar el número de máquina (lo llamaremos n)

#Lo calculamos a través de n = 1+a, siendo a un número iterable que empieza en 0.5 y se expone a la cantidad de iteraciones, de manera tal que en un momento n=1+a=1 y obtenemos la mantisa

n=0

i=np.float64(1)

while (n!=np.float64(1)):

n=np.float64(1)+((np.float32(0.5))\*\*i)

i += np.float64(1)

#print(n)En la ultima iteración, se pueden observar los decimales que corresponden a la mantisa "t", estos son 14 dígitos.

#Al ser 14 dígitos, podemoss acotar el número de máquina como 0.5e(-t+1), por ende el número de máquina final es n=0.5e-13 (ó 5e-14)

n=np.float64(5\*(10\*\*-14))

print('Numero de máquina en doble precisión:', n)

#Análogamente calculamos en formato de simple precisión el número de máquina, al cuál denominaremos "ns":

ns=0

i=np.float32(1)

while (ns!=np.float32(1)):

ns=np.float32(1)+((np.float32(0.5))\*\*i)

i += np.float32(1)

#print(ns)

#Como puede observarse en la ultima impresión del while, se observan 6 dígitos antes de que el valor se considere como 1, por ende la mantisa es t y el número de máquina (en simple precisión)

#final es

ns=np.float32(5\*(10\*\*-6))

print('Número de máquina en simple precisión:', ns)

#Finalmente, queda respondido con cuantos dígitos trabaja el ordenador en cuestión para ambos formatos.

#---------------------------------------------------------------------------

#2) Errores

#Definimos la ecuación del enunciado (fmt es un parámetro booleano que indica si se desea que trabaje en doble precisión o no)

def f(x,a,b,c,d,e,fmt):

return a\*(x\*\*np.float64(4))+b\*(x\*\*np.float64(3))+c\*(x\*\*np.float64(2))+d\*(x)+e if fmt else a\*(x\*\*np.float32(4))+b\*(x\*\*np.float32(3))+c\*(x\*\*np.float32(2))+d\*(x)+e

#Las constantes (doble precisión):

a=np.float64(1)

b=np.float64(-26)

c=np.float64(131)

d=np.float64(-226)

e=np.float64(120)

#Constantes (simple precisión):

asi=np.float32(1)

bsi=np.float32(-26)

csi=np.float32(131)

dsi=np.float32(-226)

esi=np.float32(120)

#Grafico

x=np.linspace(0,25, 100)

y=f(x,a,b,c,d,e,True)

plt.plot(x,y)

plt.ylim([0, 20])

plt.title('Ecuación original')

plt.show() #Como puede observarse, las raices de la ecuación original son 1, 2, 3 y 20

#a) Obtenemos las raíces mediante Newton-Raphson, para lo cual primero definimos nuestra nueva constante "e"

P = (np.float64(106716) + np.float64(99489))/np.float64((10\*\*5))

e = P + np.float64(119.95) #número bien redondeado

eE = np.float32(0.000005)

eEr = eE/e

CotaeEr = 5\*10\*\*(-8)

Ps = (np.float32(106716) + np.float32(99489))/(np.float32((10\*\*5)))

esi = Ps + np.float32(119.95) #numero bien redondeado

#Para aplicar el método, hallamos la derivada de la ecuación (ya puede omitirse a "e" de los parámetros porque su derivada es 0)

def df(x,a,b,c,d,fmt):

return a\*np.float64(4)\*(x\*\*np.float64(3))+b\*np.float64(3)\*(x\*\*np.float64(2))+(c\*np.float64(2))\*(x)+d if fmt else a\*np.float32(4)\*(x\*\*np.float32(3))+b\*np.float32(3)\*(x\*\*np.float32(2))+(c\*np.float32(2))\*(x)+d

#Método de Newton-Raphson (NR). Necesitamos la semilla (x0), y las constantes a,b,c,d y e para aplicarlo. Además de esto, solo a fines prácticos se añade "fmt" para identificar al formato

#de doble precisión si es verdadero (por defecto se calculan todos los valores en este formato), o falso para realizar los cálculos en simple precisión

def NR(x0,a,b,c,d,e,fmt):

x1=x0-f(x0,a,b,c,d,e,fmt)/df(x0,a,b,c,d,fmt) #Fórmula de obtención de raíz por Newton-Raphson

#Usamos como tolerancia máxima al número de máquina obtenido anteriormente

if(fmt):

while abs(x1-x0)>n:

x0=x1

x1=x0 - (f(x0,a,b,c,d,e, fmt)/df(x0,a,b,c,d,fmt))

#print('Tolerancia', abs(x1-x0))

else:

while abs(x1-x0)>ns:

x0=x1

x1=x0 - (f(x0,a,b,c,d,e, fmt)/df(x0,a,b,c,d,fmt))

#print('Tolerancia', abs(x1-x0))

return x1

#Método de Bisección para realizar el arranque y obtener una semilla cercana para garantizar la convergencia.

def biseccionFn(a0,b0,ErrorDeseado,a,b,c,d,e,fmt):

#Defino el metodo de biseccion para doble precision

if fmt:

#Defino el error de la primera etapa y la primera solucion

ErrorActual = (b0-a0)/np.float64(2)

c1 = (a0+b0)/np.float64(2)

i = 0

# print('Raiz = ', c1,', Error = ', ErrorActual, ', Iteracion = ', i)

#Defino el bucle para seleccionar los nuevos margenes del intervalo que contiene el 0

while(ErrorActual>ErrorDeseado):

#Observo si la nueva raiz esta en el primer intervalo

if f(a0,a,b,c,d,e,fmt)\*f(c1,a,b,c,d,e,fmt)<0 :

b0 = np.float64(c1)

c1 = (a0+b0)/np.float64(2)

ErrorActual = (b0-a0)/np.float64(2)

#Observo si la nueva raiz esta en el segundo intervalo

elif f(c1,a,b,c,d,e,fmt)\*f(b0,a,b,c,d,e,fmt)<0 :

a0 = c1

c1 = (a0+b0)/np.float64(2)

ErrorActual = (b0-a0)/np.float64(2)

i = i+np.float64(1)

# print('Raiz = ', c1,', Error = ', ErrorActual, ', Iteracion = ', i)

#Defino el metodo de biseccion para simple presicion

else:

#Defino el error de la primera etapa y la primera solucion

ErrorActual = (b0-a0)/np.float32(2)

c1 = (a0+b0)/np.float32(2)

i = 0

# print('Raiz = ', c1,', Error = ', ErrorActual, ', Iteracion = ', i)

#Defino el bucle para seleccionar los nuevos margenes del intervalo que contiene el 0

while(ErrorActual>ErrorDeseado):

#Observo si la nueva raiz esta en el primer intervalo

if f(a0,a,b,c,d,e,fmt)\*f(c1,a,b,c,d,e,fmt)<0 :

b0 = c1

c1 = (a0+b0)/np.float32(2)

ErrorActual = (b0-a0)/np.float32(2)

#Observo si la nueva raiz esta en el segundo intervalo

elif f(c1,a,b,c,d,e,fmt)\*f(b0,a,b,c,d,e,fmt)<0 :

a0 = c1

c1 = (a0+b0)/np.float32(2)

ErrorActual = (b0-a0)/np.float32(2)

i = i+np.float64(1)

# print('Raiz = ', c1,', Error = ', ErrorActual, ', Iteracion = ', i)

#Defino la salida se la funcion como un vector que contiene la solucion y el error

return c1

#Condicion del problema en forma experimental (cp):

#Para hallarla, se necesita de un valor x junto a su error m y las constantes a,b,c,d y e

def cpa(x,m,a,b,c,d,e,fmt):

return abs((NR(x,a,b,c,d,e\*(1+m),fmt)-NR(x,a,b,c,d,e,fmt))/(NR(x,a,b,c,d,e,fmt)\*m))

def cpb(x,m,a,b,c,d,e,fmt):

return abs((NR(x,a,b\*(1+m),c,d,e,fmt)-NR(x,a,b,c,d,e,fmt))/(NR(x,a,b,c,d,e,fmt)\*m))

def cpc(x,m,a,b,c,d,e,fmt):

termino1 = cpa(x,m,a,b,c,d,e,fmt)

termino2 = cpb(x,m,a,b,c,d,e,fmt)

return abs(termino1+termino2)

#Término experimental de estabilidad (te):

#Para aplicar el "te", necesitamos la solución en doble precisión (yd), y en simple (ys).

def te(yd,ys):

return (abs(yd-ys))/(ns\*abs(ys)) #ns es el número de máquina en simple precisión

#Ahora debemos tener una semilla en el intervalo [0, 20], que es donde se concentran todas las raices. Este valor es arbitrario pero como Newton-Raphosn aumenta la rapidez de la

#convergencia en un intervalo mas cercano a la raiz, es conveniente asignar semillas en un intervalo.

#sabemos que entre [0, 1.5], por teorema de Bolzano, existe una raiz (1). Asignemos la semilla en 0

print('--------------------Inciso a)----------------------')

#Selecciono los limites del intervalo donde hallo la raiz

ErrorSemilla = np.float32(0.01)

A1 = np.float32(0.5)

B1 = np.float32(1.2)

Semilla1D = biseccionFn(A1, B1, ErrorSemilla, a, b, c, d, e, True)

Semilla1S = biseccionFn(A1, B1, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r1 = NR(Semilla1D,a,b,c,d,e,True)

r1s = NR(Semilla1S,asi,bsi,csi,dsi,esi,False)

print('Raiz 1 mediante Newton-Raphson:', r1)

print('Término de estabilidad:', te(r1, r1s))

#Busco obtener el Cp para la raiz

exps = []

for i in range(14):

exps.append(np.float64((np.float64(10)\*\*-(np.float64(i+np.float64(1))))))

conds1 = []

for j in range(len(exps)):

conds1.append(cpa(Semilla1D,exps[j],a,b,c,d,e,True))

ejeX=np.linspace(10\*\*-14, 10\*\*-1, 14)

plt.plot(ejeX, conds1, color='b', label='Raiz 1')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.title('Cp para inciso a) raiz 1')

#plt.show()

print('Condicion del problema:', conds1[3])

print('Error relativo:', conds1[3]\*CotaeEr)

print('Error absoluto:', r1\*conds1[3]\*CotaeEr)

print('Cota del Error absoluto:', 0.5\*10\*\*(-6))

print('Raiz 1 mediante Newton-Raphson bien redondeada', 1.058072)

#Para la segunda raiz

A2 = np.float32(1.6)

B2 = np.float32(2.3)

Semilla2D = biseccionFn(A2, B2, ErrorSemilla, a, b, c, d, e, True)

Semilla2S = biseccionFn(A2, B2, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r2 = NR(Semilla2D,a,b,c,d,e,True)

r2s = NR(Semilla2S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 2 mediante Newton-Raphson:', r2)

print('Término de estabilidad:', te(r2, r2s))

#Busco obtener el Cp para la segunda raiz

conds2 = []

for j in range(len(exps)):

conds2.append(cpa(Semilla2D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds2, color='r', label='Raiz 2')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.title('Cp para inciso a) raiz 2')

#plt.show()

print('Condicion del problema:', conds2[4])

print('Error relativo:', conds2[4]\*CotaeEr)

print('Error absoluto:', r2\*conds2[4]\*CotaeEr)

print('Cota del Error absoluto:', 0.5\*10\*\*(-6))

print('Raiz 2 mediante Newton-Raphson bien redondeada', 1.887490)

#Para la tercera raiz

A3 = np.float32(2.8)

B3 = np.float32(3.2)

Semilla3D = biseccionFn(A3, B3, ErrorSemilla, a, b, c, d, e, True)

Semilla3S = biseccionFn(A3, B3, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r3 = NR(Semilla3D,a,b,c,d,e,True)

r3s = NR(Semilla3S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 3 mediante Newton-Raphson:', r3)

print('Término de estabilidad:', te(r3, r3s))

#Obtengo el Cp del problema para la tercera raiz

conds3 = []

for j in range(len(exps)):

conds3.append(cpa(Semilla3D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds3, color='g', label='Raiz 3')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.title('Cp para inciso a) raiz 3')

#plt.show()

print('Condicion del problema:', conds3[3])

print('Error relativo:', conds3[3]\*CotaeEr)

print('Error absoluto:', r3\*conds3[3]\*CotaeEr)

print('Cota del Error absoluto:', 0.5\*10\*\*(-6))

print('Raiz 3 mediante Newton-Raphson bien redondeada', 3.054785)

#Para la última raiz

A4 = np.float64(18)

B4 = np.float64(22)

Semilla4D = biseccionFn(A4, B4, ErrorSemilla, a, b, c, d, e, True)

Semilla4S = biseccionFn(A4, B4, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r4 = NR(Semilla4D,a,b,c,d,e,True)

r4s = NR(Semilla4S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 4 mediante Newton-Raphson:', r4)

print('Término de estabilidad:', te(r4, r4s))

#Cp para cuarta raiz

conds4 = []

for j in range(len(exps)):

conds4.append(cpa(Semilla4D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds4, color='y', label='Raiz 4')

plt.xlabel('Perturbacion')

plt.ylabel('Cp')

plt.title('Cp para inciso a)')

plt.legend()

plt.show()

print('Condicion del problema:', conds4[11])

print('Error relativo:', conds4[11]\*CotaeEr)

print('Error absoluto:', r4\*conds4[11]\*CotaeEr)

print('Cota del Error absoluto:', 0.5\*10\*\*(-8))

print('Raiz 4 mediante Newton-Raphson bien redondeada', 19.99965391)

print('--------------------Inciso b)----------------------')

e=np.float64(120)

esi=np.float32(120)

b = P - np.float64(26.03) #bien redondeado

bsi = Ps - np.float32(26.03)

bE = np.float32(0.000005)

bEr = bE/abs(b)

CotabEr = np.float32(0.5\*10\*\*(-6))

A1 = np.float32(0.5)

B1 = np.float32(1.2)

Semilla1D = biseccionFn(A1, B1, ErrorSemilla, a, b, c, d, e, True)

Semilla1S = biseccionFn(A1, B1, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r1 = NR(Semilla1D,a,b,c,d,e,True)

r1s = NR(Semilla1S,asi,bsi,csi,dsi,esi,False)

print('Raiz 1 mediante Newton-Raphson:', r1)

print('Término de estabilidad:', te(r1, r1s))

#Cp para raiz 1

conds1 = []

for j in range(len(exps)):

conds1.append(cpb(Semilla1D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds1, color='b', label='Raiz 1')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.title('Cp para inciso b) raiz 1')

#plt.show()

print('Condicion del problema:', conds1[3])

print('Condicion del problema:', conds1[3])

print('Error relativo:', conds1[3]\*CotabEr)

print('Error absoluto:', r1\*conds1[3]\*CotaeEr)

print('Cota del Error absoluto:', 0.5\*10\*\*(-6))

print('Raiz 1 mediante Newton-Raphson bien redondeada', 1.074868)

#Segunda raiz

A2 = np.float32(1.4)

B2 = np.float32(2.3)

Semilla2D = biseccionFn(A2, B2, ErrorSemilla, a, b, c, d, e, True)

Semilla2S = biseccionFn(A2, B2, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r2 = NR(Semilla2D,a,b,c,d,e,True)

r2s = NR(Semilla2S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 2 mediante Newton-Raphson:', r2)

print('Término de estabilidad:', te(r2, r2s))

#Cp para la segunda raiz

conds2 = []

for j in range(len(exps)):

conds2.append(cpb(Semilla2D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds2, color='r', label='Raiz 2')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.title('Cp para inciso b) raiz 2')

#plt.show()

print('Condicion del problema:', conds2[2])

print('Condicion del problema:', conds2[2])

print('Error relativo:', conds2[2]\*CotabEr)

print('Error absoluto:', r2\*conds2[2]\*CotaeEr)

print('Cota del Error absoluto:', np.float32(0.5\*10\*\*(-6)))

print('Raiz 2 mediante Newton-Raphson bien redondeada', 1.504204)

#Para la tercera raiz

A3 = np.float64(4)

B3 = np.float32(4.5)

Semilla3D = biseccionFn(A3, B3, ErrorSemilla, a, b, c, d, e, True)

Semilla3S = biseccionFn(A3, B3, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r3 = NR(Semilla3D,a,b,c,d,e,True)

r3s = NR(Semilla3S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 3 mediante Newton-Raphson:', r3)

print('Término de estabilidad:', te(r3, r3s))

#Cp para la tercera

conds3 = []

for j in range(len(exps)):

conds3.append(cpb(Semilla3D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds3, color='g', label='Raiz 3')

#plt.title('Cp para inciso b) raiz 3')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.show()

print('Condicion del problema:', conds3[4])

print('Condicion del problema:', conds3[4])

print('Error relativo:', conds3[4]\*CotabEr)

print('Error absoluto:', r3\*conds3[4]\*CotaeEr)

print('Cota del Error absoluto:', np.float32(0.5\*10\*\*(-5)))

print('Raiz 3 mediante Newton-Raphson bien redondeada', 4.35793)

#Cuarta raiz

A4 = np.float64(16)

B4 = np.float64(18)

Semilla4D = biseccionFn(A4, B4, ErrorSemilla, a, b, c, d, e, True)

Semilla4S = biseccionFn(A4, B4, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r4 = NR(Semilla4D,a,b,c,d,e,True)

r4s = NR(Semilla4S,asi,bsi,csi,dsi,esi,False)

print('\nRaiz 4 mediante Newton-Raphson:', r4)

print('Término de estabilidad:', te(r4, r4s))

#Cp para la cuarta raiz

conds4 = []

for j in range(len(exps)):

conds4.append(cpb(Semilla4D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds4, color='y', label='Raiz 4')

plt.xlabel('Perturbacion')

plt.ylabel('Cp')

plt.title('Cp para inciso b)')

plt.legend()

plt.show()

print('Condicion del problema:', conds4[3])

print('Condicion del problema:', conds4[3])

print('Error relativo:', conds4[3]\*CotabEr)

print('Error absoluto:', r4\*conds4[4]\*CotaeEr)

print('Cota del Error absoluto:', np.float32(0.5\*10\*\*(-5)))

print('Raiz 4 mediante Newton-Raphson bien redondeada', 17.03094)

#c) Ahora necesitamos variar las constantes e y d juntas..:

print('--------------------Inciso c)----------------------')

e=P + np.float64(119.95)

esi=Ps + np.float32(119.95)

A1 = np.float32(0.3)

B1 = np.float32(1.3)

Semilla1D = biseccionFn(A1, B1, ErrorSemilla, a, b, c, d, e, True)

Semilla1S = biseccionFn(A1, B1, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r1 = NR(Semilla1D,a,b,c,d,e,True)

r1s = NR(Semilla1S,asi,bsi,csi,dsi,esi,False)

print('Raiz 1 mediante Newton-Raphson:', r1)

print('Término de estabilidad:', te(r1, r1s))

#Busco el Cp para la primera raiz

conds1 = []

for j in range(len(exps)):

conds1.append(cpc(Semilla1D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds1, color='b', label='Raiz 1')

#plt.title('Cp para inciso c) raiz 1')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.show()

print('Condicion del problema:', conds1[3])

#Segunda raiz

A2 = np.float32(1.3)

B2 = np.float32(2.3)

Semilla2D = biseccionFn(A2, B2, ErrorSemilla, a, b, c, d, e, True)

Semilla2S = biseccionFn(A2, B2, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r2 = NR(Semilla2D,a,b,c,d,e,True)

r2s = NR(Semilla2S,asi,bsi,csi,dsi,esi,False)

print('Raiz 2 mediante Newton-Raphson:', r2)

print('Término de estabilidad:', te(r2, r2s))

#Cp para la segunda raiz

conds2 = []

for j in range(len(exps)):

conds2.append(cpc(Semilla2D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds2, color='r', label='Raiz 2')

#plt.title('Cp para inciso c) raiz 2')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.show()

print('Condicion del problema:', conds2[3])

#Raíz 3

A3 = np.float64(4)

B3 = np.float32(4.5)

Semilla3D = biseccionFn(A3, B3, ErrorSemilla, a, b, c, d, e, True)

Semilla3S = biseccionFn(A3, B3, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r3 = NR(Semilla3D,a,b,c,d,e,True)

r3s = NR(Semilla3S,asi,bsi,csi,dsi,esi,False)

print('Raiz 3 mediante Newton-Raphson:', r3)

print('Término de estabilidad:', te(r3, r3s))

#Obtengo el Cp del problema para la tercera raiz

conds3 = []

for j in range(len(exps)):

conds3.append(cpc(Semilla3D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds3, color='g', label='Raiz 3')

#plt.title('Cp para inciso c) raiz 3')

#plt.xlabel('Perturbacion')

#plt.ylabel('Cp')

#plt.show()

print('Condicion del problema:', conds3[4])

#Para la cuarta raiz

A4 = np.float64(16)

B4 = np.float64(18)

Semilla4D = biseccionFn(A4, B4, ErrorSemilla, a, b, c, d, e, True)

Semilla4S = biseccionFn(A4, B4, ErrorSemilla, asi, bsi, csi, dsi, esi, False)

r4 = NR(Semilla4D,a,b,c,d,e,True)

r4s = NR(Semilla4S,asi,bsi,csi,dsi,esi,False)

print('Raiz 4 mediante Newton-Raphson:', r4)

print('Término de estabilidad:', te(r4, r4s))

#Cp para cuarta raiz

conds4 = []

for j in range(len(exps)):

conds4.append(cpc(Semilla4D,exps[j],a,b,c,d,e,True))

plt.plot(ejeX, conds4, color='y', label='Raiz 4')

plt.xlabel('Perturbacion')

plt.ylabel('Cp')

plt.title('Cp para inciso b)')

plt.legend()

plt.show()

print('Condicion del problema:', conds4[3])

#d) Con las mismas constantes de c) pero ahora exactas (su error inherente es nulo):

print('--------------------Inciso d)----------------------')

#Utilizo las semillas ya calculadas del inciso anterior

r1=NR(Semilla1D,a,b,c,d,e,True)

r1s=NR(Semilla1S,asi,bsi,csi,dsi,esi,False)

print('Raiz 1 mediante Newton-Raphson:', r1)

print('Término de estabilidad:', te(r1, r1s))

r2=NR(Semilla2D,a,b,c,d,e,True)

r2s=NR(Semilla2S,asi,bsi,csi,dsi,esi,False)

print('Raiz 2 mediante Newton-Raphson:', r2)

print('Término de estabilidad:', te(r2, r2s))

r3=NR(Semilla3D,a,b,c,d,e,True)

r3s=NR(Semilla3S,asi,bsi,csi,dsi,esi,False)

print('Raiz 3 mediante Newton-Raphson:', r3)

print('Término de estabilidad:', te(r3, r3s))

r4=NR(Semilla4D,a,b,c,d,e,True)

r4s=NR(Semilla4S,asi,bsi,csi,dsi,esi,False)

print('Raiz 4 mediante Newton-Raphson:', r4)

print('Término de estabilidad:', te(r4, r4s))

**Anexo II**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Perturbac.** | **Cp de “e” en raíz 1 de a)** | **Cp de “e” en raíz 2 de a)** | **Cp de “e” en raíz 3 de a)** | **Cp de “e” en raíz 4 de a)** |
|  | 21.07964326352779 | 7.422340698003286 | 0.7649103973740456 | 0.0010498027982278318 |
|  | 3.963675173422167 | 3.7862384917442444 | 0.9734311357974111 | 0.001049471495968602 |
|  | 3.701521687550057 | 3.694080355369846 | 1.0072891531130592 | 0.0010494383848502144 |
|  | 3.6785754273544473 | 3.68708029501239 | 1.0109117249622304 | 0.0010494350741966836 |
|  | 3.676308496126414 | 3.6863999808756125 | 1.0112765568059896 | 0.0010494347491177567 |
|  | 3.6760820746326797 | 3.68633214315072 | 1.0113130655875506 | 0.001049434891228763 |
|  | 3.6760594368483486 | 3.686325367429524 | 1.0113167316530278 | 0.0010494347135900051 |
|  | 3.676057149397952 | 3.6863245580651154 | 1.0113169875129389 | 0.0010494542538533728 |
|  | 3.6760576530567546 | 3.686322664058288 | 1.0113176998501914 | 0.0010494897816049505 |
|  | 3.6760587023459275 | 3.686318429011965 | 1.011315373850999 | 0.0010498450591207274 |
|  | 3.6759747592121004 | 3.686184319211749 | 1.0114316738106175 | 0.001048068671541843 |
|  | 3.6760796881293847 | 3.685666702438984 | 1.0116642737298542 | 0.0012434713052191357 |
|  | 3.6767092616330888 | 3.671549881363571 | 1.023439644641209 | 0.0017763875788844796 |
|  | 3.5885689711144875 | 3.4233291107875656 | 0.9449371718988434 | 0 |

Figura 4: Tabla de valores de la condición del problema según la perturbación, para el inciso a)

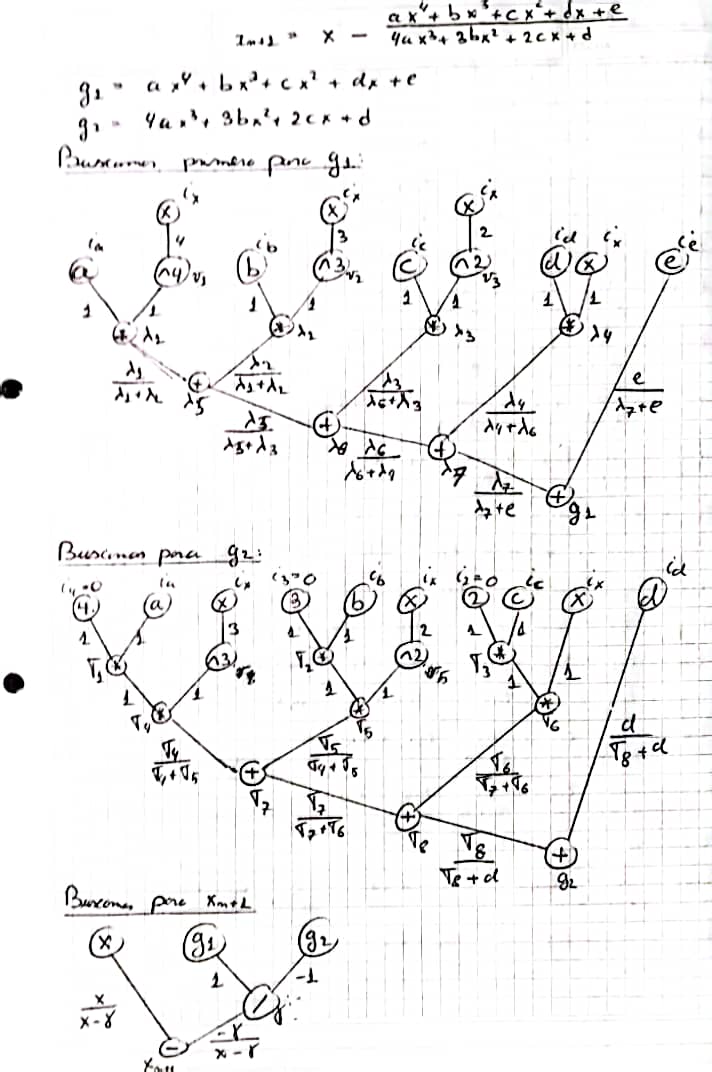
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Perturbac.** | **Cp de “b” en raíz 1 de b)** | **Cp de “b” en raíz 2 de b)** | **Cp de “b” en raíz 3 de b)** | **Cp de “b” en raíz 4 de b)** |
|  | 0.7821618253001377 | 7.463186533924787 | 3.972326996443781 | 2.035599339352093 |
|  | 1.1506824647698555 | 2.871049442397982 | 3.767129395544171 | 2.190309275224709 |
|  | 1.222385494809023 | 2.8518060957560314 | 3.8265779410984697 | 2.211746137395376 |
|  | 1.2303345409057382 | 2.850815907665587 | 3.8329974994143567 | 2.213977935360985 |
|  | 1.2311382722405382 | 2.850726732702229 | 3.8336442903672805 | 2.214202040056857 |
|  | 1.2312187340777785 | 2.8507179122175117 | 3.833709017595938 | 2.214224459357254 |
|  | 1.2312267927037235 | 2.8507170293261526 | 3.8337154929601454 | 2.2142267035130927 |
|  | 1.2312275900966008 | 2.8507168197114177 | 3.833716043239742 | 2.2142269183746386 |
|  | 1.2312287262781618 | 2.850718915858767 | 3.833717653317082 | 2.2142277110677218 |
|  | 1.2312345104752 | 2.850717587314672 | 3.8337184685461145 | 2.214229588498707 |
|  | 1.2312902866609257 | 2.850848965564072 | 3.8337979533768154 | 2.214304685738141 |
|  | 1.231827390671616 | 2.850612779947174 | 3.834022141360841 | 2.214951356411043 |
|  | 1.2312076552746656 | 2.8209419618243743 | 3.8275003090982667 | 2.2153685632967863 |
|  | 1.342760026725726 | 2.952320211223835 | 3.8519571800829198 | 2.2320568387265176 |

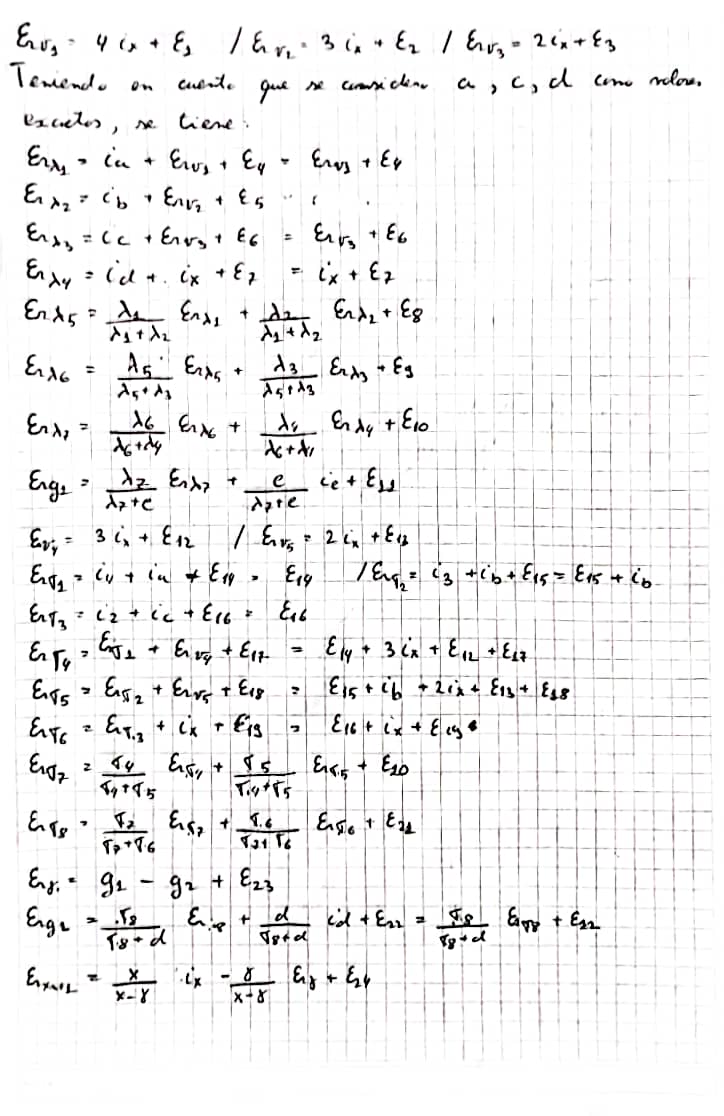
Figura 5: Tabla de valores de la condición del problema según la perturbación, para el inciso a)

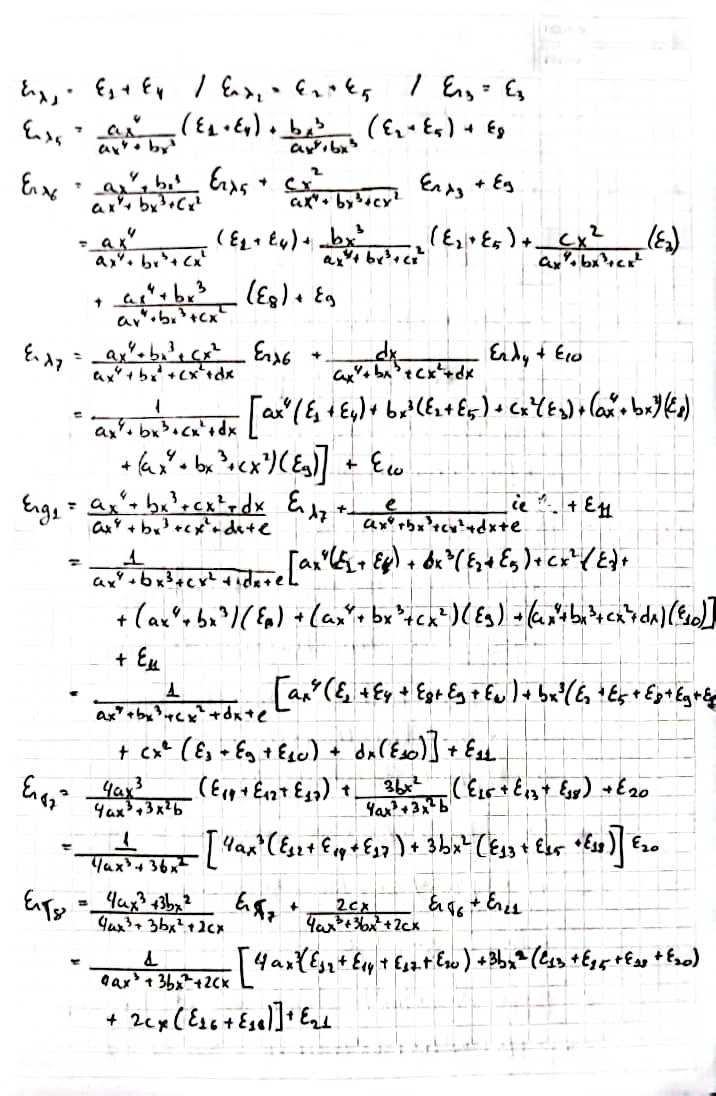
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Perturbac.** | **Cp de “b” y “e” en raíz 1 de c)** | **Cp de “b” y “e” en raíz 2 de c)** | **Cp de “b” y “e” en raíz 3 de c)** | **Cp de “b” y “e” en raíz 4 de c)** |
|  | 131.573632114168 | 25.403789976889005 | 3.9227501611421136 | 2.038150886516324 |
|  | 264.5315489545277 | 230.57022148587285 | 3.9499985562982642 | 2.1929859518383057 |
|  | 23.13651427840833 | 23.668584046704503 | 4.012782579726023 | 2.214444155664162 |
|  | 20.516324666538043 | 21.301793714171446 | 4.019516976464354 | 2.2166782392599096 |
|  | 20.34707370311149 | 21.14868902812998 | 4.020195080136319 | 2.2169025741103034 |
|  | 20.330745794614945 | 21.1339165805885 | 4.020262936489979 | 2.216925016659295 |
|  | 20.329118807374073 | 21.13244458436857 | 4.0202697295154675 | 2.216927266114874 |
|  | 20.328955758589867 | 21.132297537310578 | 4.02027043807397 | 2.216927445520586 |
|  | 20.32894749564288 | 21.132283226201448 | 4.020272630747848 | 2.2169280713544666 |
|  | 20.328964679647676 | 21.132264616819043 | 4.020279736635416 | 2.2169289057996413 |
|  | 20.328897040479873 | 21.13220533029101 | 4.020328462721591 | 2.216891355766797 |
|  | 20.332516650000116 | 21.131200753010507 | 4.020917236262877 | 2.2169122168961546 |
|  | 20.28260991267555 | 21.115885066602814 | 4.019902109467557 | 2.21336582490526 |
|  | 20.14550349145422 | 21.112591370601166 | 4.040204645373958 | 2.1486963238948333 |

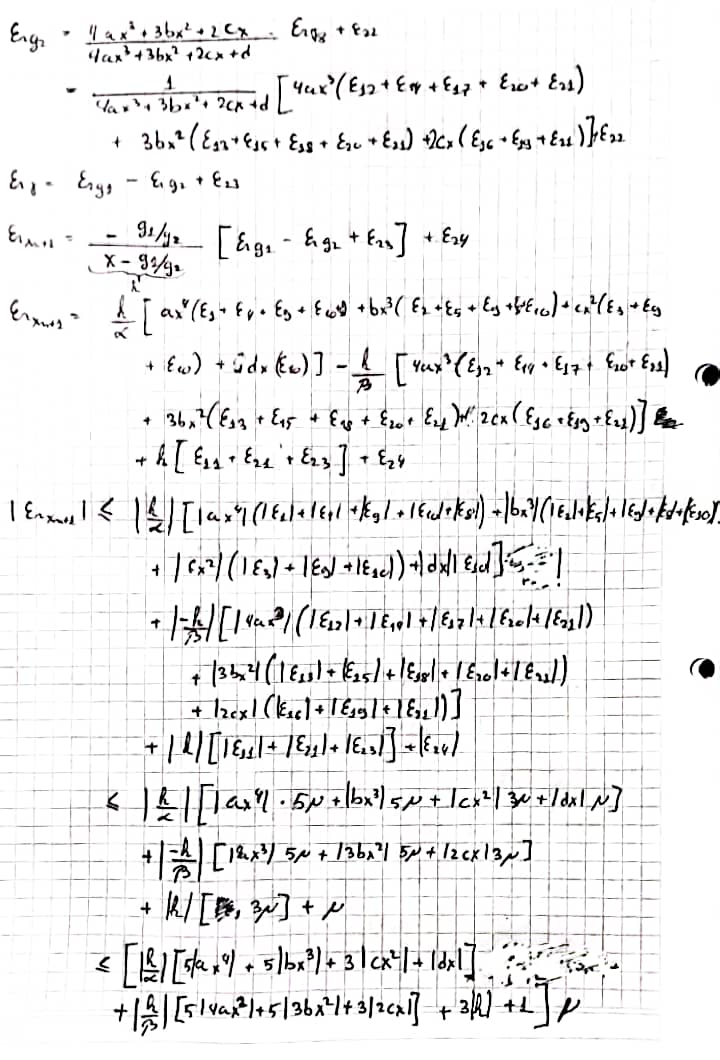
Figura 6: Tabla de valores de la condición del problema según la perturbación, para el inciso a)

**Anexo III**

****

****

****

****

**Bibliografía**

* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 1: Errores en el Cálculo Numérico.* https://campusgrado.fi.uba.ar/mod/lesson/view.php?id=90369
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 2: Representación de números, errores de redondeo.* https://campusgrado.fi.uba.ar/mod/lesson/view.php?id=90370
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 3: Gráfica de proceso, condición del problema y término de estabilidad.*

https://campusgrado.fi.uba.ar/mod/lesson/view.php?id=90899

* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 2, Módulo 1: Métodos de bisección y de Regula-Falsi.* https://campusgrado.fi.uba.ar/mod/lesson/view.php?id=90900
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 2, Módulo 2: Métodos de punto fijo y de Newton-Raphson.* https://campusgrado.fi.uba.ar/mod/lesson/view.php?id=28174
* González, H. *Análisis Numérico, Primer curso.* Nueva Librería