**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 en la resolución del grupo para estimar la raíz de mediante el método de refinamiento de Newton-Raphson (igualando dicha función a 0).

Una vez verificadas las condiciones iniciales del enunciado, se pasó a un caso práctico en donde se cumplen las mismas, cuya tarea fue encontrar las raíces . En primer lugar, como en ninguna de las ecuaciones presentadas existe una combinación lineal respecto a la variable a estimar (en este caso ), se probó con todas en cuál podría haber un método en el cual converja a una solución. Para , ocurrió que la más apta resultó ser la ecuación (4) de donde, igualando a 0 la misma (esto se realiza restando a ambos lados de la igualdad) se logró hallar un . Graficando la función reescrita en el párrafo anterior, se observó que tenía una raíz doble cercana a 0 (referencia en anexo 1), por lo que con una semilla lo suficientemente cercana a una de las raíces (en este caso consideramos el lado derecho), resulta posible aplicar el método de Newton-Raphson, ya que esto asegura su convergencia. Inicialmente y como no hay observaciones realizadas sobre los datos de entrada del enunciado, se consideraron los valores como exactos (se desprecian errores inherentes), y el error resultante fue de truncamiento, que por método de Newton-Raphson, la cota resultante se expresa como . A priori esto también dependería de la tolerancia máxima que se le impone al método de refinamiento. Analizando el número de iteraciones, se encontró que con 7 iteraciones se llegaba a una cota de , y con una más a , con lo cual se ganaban 10 decimales más de precisión y se decidió ir por esa tolerancia (no era posible establecer una más baja porque era el rango de la mantisa del ordenador).

Ya obtenido el y expresado bien redondeado, se procedió a hallar de la ecuación (3), que despejando queda como (con las condiciones iniciales ). Para encontrar su cota, se analizan los errores que acarrean los distintos parámetros de la igualdad. En este caso, el único es que ingresa a esta ecuación con un error inherente. Para propagar dicho error se deriva respecto de y se multiplica por su cota correspondiente. Una vez realizados estos cálculos, se pudo expresar bien redondeado acotando por , donde indica la cantidad de decimales significativos.

Por último en esta parte, se pidió hallar . Para ello, se utilizó la ecuación original (1) de la catenaria, y reemplazando los datos se llegó a la solución de manera inmediata, y para expresarlo bien redondeado se halló su cota con la misma idea conceptual que para , pero en este caso las variables de entrada que poseían error inherente eran dos, en vez de una: y . Por ende,

Siendo y las cotas halladas previamente de y respectivamente.

Posteriormente, se busca analizar qué ocurre en el mismo caso sabiendo que los valores de entrada están bien redondeados a un decimal, lo que significa que

Es importante aclarar que los cálculos y los métodos empleados para llevarlos a cabo no se modifican, por consiguiente solo hay que analizar la modificación de los errores en cada caso.

Primero se calculó la propagación de errores inherentes para . Por definición, se haría como fue hecho previamente para , pero en diferente contexto ya que fue hallado por el método de Newton-Raphson. Es por ello que debe propagarse el error tomando el y derivándolo (además de multiplicar por la cota correspondiente) en cada parámetro declarado con error inherente. Para este caso particular se expresa como

Para la ecuación (4) despejada, no dependía de . La cota de error resultante es entonces

Donde el error de truncamiento es el mismo que el anterior, dado que se estableció en el uso del método.

Una observación relevante fue que en este punto no resultó tan sencillo saber cómo propagar el error para dado que no se sabía si tomar la ecuación (4) original en vez de la operada, en donde también tomaba protagonismo el y eso hacía que aparezca como término en la cota de error inherente. El grupo logró decantarse por esta última opción de la ecuación (4) dado que la raíz se aproximaba de la misma manera, pero comparando los errores esta opción obtuvo una disminución del error original.

La manera recientemente descrita en la cual se obtuvo la cota de fue de manera analítica, pero también se analizó sobre la cota de forma experimental. Para ello, se realizaron pequeñas perturbaciones al medido a ambos lados del mismo y se graficó para tener una noción (ver en sección de resultados). Efectivamente, se encontró una perturbación con la cual la cota era aún inferior a la descubierta analíticamente (se comparó realizando la diferencia).

Con la cota ya hallada de , se efectuó la propagación de errores inherentes de manera similar para . Como este valor se encontró con un despeje, no resultó propagarlo en base a un , y por definición en la ecuación de se propagó como

Despreciando los errores de otro índole.

De manera similar se procedió para y la ecuación de dónde fue obtenida (catenaria):

En realidad no se conoce el valor de , ya que los errores inherentes que se conocen acerca de es solo para los extremos según lo establecido en el enunciado, pero de igual manera en este caso se evalúa , por lo que se anula la derivada de la catenaria en esa posición, ergo ese término de la cota, por lo que finalmente quedaría

A criterio del grupo, se decidió expresarlo bien redondeado y como intervalo a estos dos últimos casos () debido a que las cotas no eran lo suficientemente aptas (aún siendo las menores que se han experimentado para estos casos) para despejar un de .

Tras todo este análisis abstracto se logró la idea de entender la manera en que una catenaria funciona, qué parámetros tiene y como analizar sus errores. Ahora se desarrollará lo mismo pero inversamente, es decir, a partir de un caso real se simulará el ejercicio (ver figuras de casos en anexo 2).

En primera medida, hubo un debate sobre qué materiales elegir para modelar la catenaria: con qué soportarlo, sobre qué plataforma ubicar los soportes, cómo ubicar los ejes, entre otras. La idea es construir un modelo real que se aproxime lo mejor posible al teórico, por lo que mientras más se minimicen las diferencias, mejor sería.

La cadena utilizada mide 50cm y cada eslabón tenía un espesor de 0,3cm. Como soportes se colocaron agujas, y la cadena enganchada en esas hizo que se desprecie 0,5cm de cada extremo, por lo que la longitud final considerada en el proyecto fue de 49cm.

Las agujas se pusieron sobre 4 hojas cuadriculadas A4, las cuales ofrecían como beneficio la escala; cada 2 unidades de cuadrado se forma 1cm.

Antes de colocar los ejes, para que sea lo más parecido al sistema de coordenadas que se aprecia en el enunciado, se priorizó colocar la cadena sobre arriba del eje x, por lo que todas las ordenadas queden positivas. El eje y se intentó ubicar de manera que interseccione perpendicularmente con el vértice formado por la cadena.

Luego, para cada caso se colgó la cadena de los soportes a una distancia entre ellos de 0,8, 0,7 y 0.3 veces la longitud de la cadena. Tomando 20 puntos entre el intervalo definido por las posiciones de los soportes sobre el eje x (en los gráficos anotados como y en el código como ), y siendo la distancia entre los puntos consecutivas definida por el grupo, se pudieron tomar los mismos puntos en los casos de 0,8 y 0,7 veces la longitud (cada 1,5cm) y cada 0,5cm en el caso 0,3 veces la longitud. Otra aclaración importante fue que la altura e se tomó en todos los casos por igual (24cm), dado que el eje x se mantuvo en la misma altura.

Con los puntos ya marcados, se tomaron fotos de cada maqueta y con el software recomendado por la cátedra ImageJ, se midieron la proyección de los puntos sobre la cadena (como convención, se tomaron por encima de ella) y con estos datos fue posible pasar a la implementación en código. Como cota de error inherente sobre las ordenadas, se tomó 0,3cm que corresponde al espesor, esto ya que si bien se decidió tomar la proyección por encima de la cadena, también podría haberse tomado por debajo, y el caso hipotético planteó esta cota del espesor (distancia entre esa hipotética ordenada y la medida).

En Python, se comenzó inicializando los valores conocidos y la cantidad de puntos por caso . También en cada maqueta se conocen la posición de los soportes sobre el eje x en las fotos, en el código.

A esta altura para modelar la ecuación de la catenaria se necesitaba hallar para cada uno de estos casos. En base a como se construyó la maqueta en cada situación, se cumplen las condiciones iniciales del enunciado (2), (3) y (4), por lo que resultó posible encontrar los valores de la misma manera que en la parte anterior.

Ya con obtenidos, se planteó la catenaria para cada caso (evaluando en sus respectivos puntos) y se guardaron los datos en un arreglo. Posteriormente, se realizó un ajuste por cuadrados mínimos de manera tal que la catenaria se aproxime por una cuadrática, o sea:

Donde y se hallaron los coeficientes por dicho método.

Una vez obtenida , se evaluó para cada caso y en los respectivos puntos de cada uno, y también se almacenaron estos datos en un arreglo.

Finalmente para analizar la comparación de las distintas estimaciones, se calculó el error cuadrático medio entre las mediciones de las fotos y la catenaria, y entre las mediciones de las fotos y el ajuste cuadrático. Además, para tener una idea más intuitiva, se graficaron las funciones (imágenes en sección de resultados).

Como conclusiones sobre las comparaciones, en todos los casos de manera analítica, el error cuadrático medio dio menor en los ajustes cuadráticos, y es también visible en los gráficos el hecho de que se aproxima más a los puntos de las fotos por lo que resultó más precisa.

**Resultados**

Parte 1)

b) Valores bien redondeados:

-

-

-

c)

-

-

-

-

-

-

-

-

Parte 2)

Caso 0.8L: Valores bien redondeados

Caso 0.7L: Valores bien redondeados

Caso 0.3L: Valores bien redondeados

Gráficos:

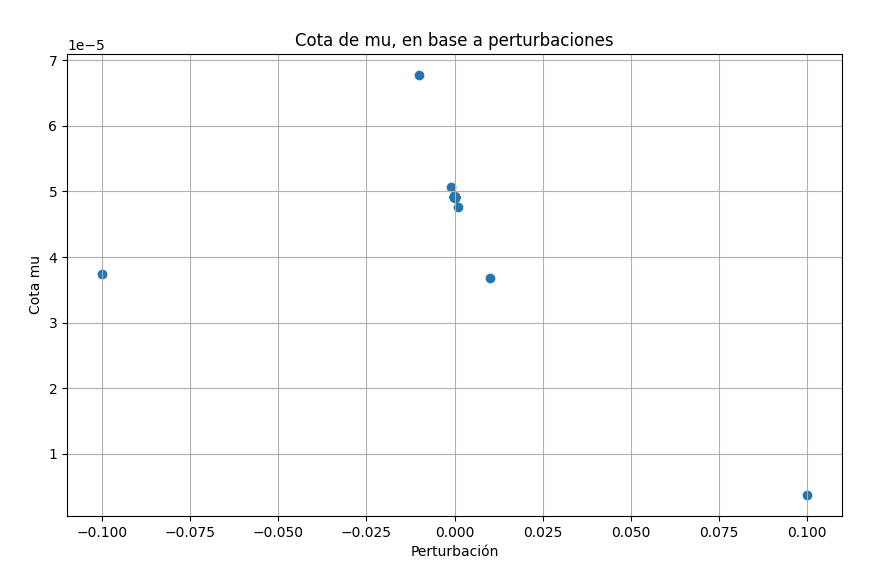


Figura 1: Cota de en función de las distintas perturbaciones

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figura 2: Comparación de estimaciones para 0.8L

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figura 3: Comparación de estimaciones para 0.7L

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figura 4: Comparación de estimaciones para 0.3L

**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.
* Otro conflicto fue el hecho de encontrar . Si tenía sentido físico, de qué ecuación corresponde mejor, entre otros pensamientos.
* Se aprendió que algunas veces resulta más preciso transformar el problema original a uno más equivalente, como ocurrió con .
* Se lograron profundizar los temas de las unidades 1, 2 y 3
* 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**

**primera-parte.py**

import math

import numpy as np

from matplotlib import pyplot as plt

*#LA PARTE A) DE ESTA PRIMERA PARTE FUE REALIZADA EN HOJA. LEER INFORME PARA MÁS DETALLES.*

*#--------------------------- ITEM (B) ---------------------------*

*#Mediciones EXACTAS*

P1 = 106716

P2 = 106031

P = (P1 + P2) / 2

x0 = -P/3500

x1 = P/3500

y0 = y1 = P/2100

L = P/1300

*#Por Newton-Raphson, hallamos mu igualando la ecuación 4) a 0 (ver anexo), dado que en este caso se cumplen las condiciones*

def f(*mu*):

    return ((2\*np.sinh(*mu*\*x1)) / *mu*) - L

*# Derivada de f(mu)*

def df(*mu*):

    return ((2\**mu*\*x1\*np.cosh(*mu*\*x1)) - (2\*np.sinh(*mu*\*x1))) / *mu*\*\*2

tolerancia = 0.5e-16 *#el menor error de truncamiento que este ordenador permite debido a su mantisa (16 decimales)*

def newton\_raphson(*semilla*):

    x0 = *semilla*

    x1 = x0 - f(x0) / df(x0)

    while abs(x1-x0) > tolerancia:

        x0 = x1

        x1 = x0 - f(x0) / df(x0)

    return x1

mu = newton\_raphson(0.1)

mu = round(mu, 16) *#para redondear a 16 decimales significativos*

*#Cálculo del C2*

C2 = y1\*mu - (np.cosh(mu\*x1)) *#despejando de la ecuación (3) del enunciado*

*#propagación de errores c2 -> mu tiene error inherente para su cálculo*

dc2\_mu = y1 - (x1\*np.sinh(mu\*x1)) *#derivada respecto a mu de C2*

ec2 = float(f"{(abs(dc2\_mu) \* abs(tolerancia)):.1g}" ) *#cota de C2 ya redondeada*

t = math.ceil(-math.log10(2 \* ec2)) *#decimales significativos*

C2 = round(C2, t)

*#para la ecuacion original, mu y C2 como datos de entrada tienen error inherente*

def catenaria(*x*):

    return ((np.cosh(mu\**x*))/mu) + C2

def df\_catenaria\_mu(*x*):

    return ((*x*\*mu\*np.sinh(mu\**x*)) - np.cosh(mu\**x*))/mu\*\*2

*#y la derivada respecto de C2 es 1 -> se propaga su error como es de entrada*

y\_x\_0 = catenaria(0)

e\_y = (abs(df\_catenaria\_mu(0)) \* tolerancia) + 1\*ec2

t = math.ceil(-math.log10(2 \* e\_y))

y\_x\_0 = round(y\_x\_0, t)

print("------------- ITEM B) -------------")

print("Valor de mu: ", mu)

print("Valor de C2: ", C2)

print("Valor de y(x=0): ", y\_x\_0)

*#--------------------------- ITEM (C) ---------------------------*

*#A priori, los cálculos serían los mismos. Solo se modifican sus errores*

print("------------- ITEM C) -------------")

COTA\_X0 = COTA\_X1 = COTA\_Y0 = COTA\_Y1 = COTA\_L = 0.05

def cota\_error\_propagado\_mu(*mu\_medido*, *x1\_medido*):

    mu\_anterior = *mu\_medido* - tolerancia

    dphi\_x1 = (((mu\_anterior\*\*2)\*(np.sinh(2\*mu\_anterior\**x1\_medido*)))-(mu\_anterior\**x1\_medido*\*(L\*mu\_anterior\*np.sinh(mu\_anterior\**x1\_medido*)+2))) / (2\*np.sinh(mu\_anterior\**x1\_medido*)-(mu\_anterior\**x1\_medido*\*np.cosh(mu\_anterior\**x1\_medido*\*\*2)))\*\*2

    dphi\_l = (mu\_anterior\*\*2)/((2\**x1\_medido*\*mu\_anterior\*np.cosh(mu\_anterior\**x1\_medido*))-(2\*np.sinh(*x1\_medido*\*mu\_anterior)))

    cota = COTA\_X1

    return (cota \* abs(dphi\_x1) + cota \* abs(dphi\_l))

cota\_error\_mu = cota\_error\_propagado\_mu(mu, x1) + tolerancia *#error total = error inh + error trunc (se desprecian err redondeo)*

t = math.ceil(-math.log10(2 \* cota\_error\_mu)) *#decimales significativos*

cota\_error\_mu\_red = round(cota\_error\_mu, t)

print("Cota de error propagado de mu (analítica):", cota\_error\_mu)

perturbaciones = []

cotas\_exp = []

for i in range(1,16):

    delta\_mu = 0.1\*\*i

    perturbaciones.append(delta\_mu)

    perturbaciones.append(-delta\_mu)

    cotas\_exp.append(cota\_error\_propagado\_mu(mu+delta\_mu, x1))

    cotas\_exp.append(cota\_error\_propagado\_mu(mu-delta\_mu, x1))

cota\_min\_exp = np.min(cotas\_exp)

perturbaciones = np.ravel(perturbaciones)

cotas\_exp = np.ravel(cotas\_exp)

plt.figure(*figsize*=(10, 6))

plt.scatter(perturbaciones, cotas\_exp)

plt.xlabel('Perturbación')

plt.ylabel('Cota mu')

plt.title('Cota de mu, en base a perturbaciones')

plt.grid(True)

plt.show()

print("Cota de error propagado de mu (experimental)", cota\_min\_exp)

dif\_cota\_mu = abs(cota\_error\_mu - cota\_min\_exp)

print("Diferencia entre cota analítica y experimental:", dif\_cota\_mu)

mu\_redondeado = round(mu, t-1)

print("Valor de mu bien redeondeado:", mu\_redondeado)

*#c2 = f(mu, x1, y1) = y1 - cosh(mu \* x1)/mu*

def cota\_error\_total\_c2(*mu\_medido*, *x1\_medido*):

    df\_mu = (np.cosh(*x1\_medido*\**mu\_medido*) - (*mu\_medido*\**x1\_medido*\*np.sinh(*mu\_medido*\**x1\_medido*))) / *mu\_medido*\*\*2

    df\_x1 = - *mu\_medido*\*np.sinh(*mu\_medido* \* *x1\_medido*)

    df\_y1 = 1

    return abs(cota\_error\_mu\*df\_mu) + abs(COTA\_X1\*df\_x1) + abs(COTA\_Y1 \* df\_y1)

cota\_error\_c2 = cota\_error\_total\_c2(mu\_redondeado, x1)

print("Cota de error de C2:",cota\_error\_c2)

*#erc2 = (cota\_error\_c2/C2)\*100 -> Por si se desea obtener error relativo de C2. Da alrededor de 35%*

print("C2 bien redondeado:", round(C2)) *#t=0 y es un decimal, por lo que redondea a 0*

print("C2 expresado como intervalo: ", C2, "±", math.ceil(cota\_error\_c2 \* 100) / 100)

*#el error inherente de mu (dato de entrada para esta ecuación) es el obtenido anteriormente, por lo que se reutiliza esa variable*

def cota\_total\_error\_y(*mu\_medido*, *x\_medido*):

    df\_mu = df\_catenaria\_mu(*x\_medido*)

    df\_x = *mu\_medido*\*np.sinh(*mu\_medido* \* *x\_medido*)

    df\_c2 = 1

    return abs(cota\_error\_mu\*df\_mu) + abs(cota\_error\_c2 \* df\_c2) + abs(COTA\_X1\*df\_x)

cota\_error\_y = cota\_total\_error\_y(mu\_redondeado, 0)

print("Cota de error de y(x):", cota\_error\_y)

print("Valor de y(0) bien redeondeado:", round(y\_x\_0))

print("El valor expresado como intervalo", y\_x\_0, "±", round(cota\_error\_y, 1))

**segunda-parte.py**

import numpy as np

from matplotlib import pyplot as plt

*#----------------- GENERALIDADES -----------------*

*#Las siguientes medidas y funciones valen para todos los casos del TPM (eg. todos los casos medimos la altura a 24cm)*

L = 49   *# Longitud de la cadena. Realmente mide 50cm, pero se consideró el espacio perdido en el soporte del primer eslabón de cada lado*

n = 22  *#Puntos considerando los extremos*

y1 = y0 = 24.0 *#altura del soporte*

*# Definimos la función f(mu) basada en la condición de longitud*

def f(*mu*, *x1*):

    return ((2\*np.sinh(*mu*\**x1*)) / *mu*) - L

*# Derivada de f(mu)*

def df(*mu*, *x1*):

    return ((2\**mu*\**x1*\*np.cosh(*mu*\**x1*)) - (2\*np.sinh(*mu*\**x1*))) / *mu*\*\*2

*# Newton-Raphson loop*

tolerancia = 0.5e-16

def newton\_raphson(*semilla*, *soporte\_derecha*):

    mu0 = *semilla*

    mu1 = mu0 - f(mu0, *soporte\_derecha*) / df(mu0, *soporte\_derecha*)

    while abs(mu1-mu0) > tolerancia:

        mu0 = mu1

        mu1 = mu0 - f(mu0, *soporte\_derecha*) / df(mu0, *soporte\_derecha*)

    return mu1

*#ecuación original*

def catenaria(*x*):

    return ((np.cosh((mu\_raiz\**x*)))/mu\_raiz + C2)

*#auxiliar para obtener valores de la ecuación original en cada caso*

def obtener\_soluciones\_catenaria(*puntos*):

    y = []

    for punto in *puntos*:

        y.append(catenaria(punto))

    return y

*#función que muestra ajuste y comparaciones según el caso*

def resultados\_soluciones(*puntos*, *soluciones\_caso*, *titulo*):

    soluciones\_catenaria = np.matrix(obtener\_soluciones\_catenaria(*puntos*))

*puntos* = np.matrix(*puntos*)

    fi0=np.matrix(np.ones(n))

    fi1=*puntos*

    fi2=np.power(*puntos*, 2)

    M=(

        (np.inner(fi0,fi0)), (np.inner(fi0,fi1)), (np.inner(fi0,fi2)),

        (np.inner(fi1,fi0)), (np.inner(fi1,fi1)), (np.inner(fi1,fi2)),

        (np.inner(fi2,fi0)), (np.inner(fi2,fi1)), (np.inner(fi2,fi2))

    )

    M=np.array(M).reshape((3,3))

    M=np.matrix(M).reshape((3,3))

    b=(np.inner(fi0, *soluciones\_caso*), np.inner(fi1, *soluciones\_caso*), np.inner(fi2, *soluciones\_caso*))

    b=np.array(b).reshape(3,1)

    b=np.matrix(b).reshape(3,1)

    c=np.linalg.inv(M)\*b

    soluciones\_cuadratica = c[0,0]\*fi0+c[1,0]\*fi1+c[2,0]\*fi2

    dif=*soluciones\_caso* - soluciones\_catenaria

    ecm=np.sqrt(np.inner(dif, dif) / n)

    print("Coeficientes del ajuste:", c) *#Coeficientes de ajuste cuadrático c0\*fi0 + c1\*fi1 + c2\*fi2 respectivamente*

    print("ECM Catenaria:", ecm[0,0])

    dif=*soluciones\_caso* - soluciones\_cuadratica

    ecm=np.sqrt(np.inner(dif, dif) / n)

    print("ECM Cuadrática:", ecm[0,0])

*puntos* = np.ravel(*puntos*)

    soluciones\_catenaria = np.ravel(soluciones\_catenaria)

    soluciones\_cuadratica = np.ravel(soluciones\_cuadratica)

*# Gráficos para comparar ajuste, catenaria y datos medidos*

    plt.figure(*figsize*=(10, 6))

    plt.scatter(*puntos*, *soluciones\_caso*, *color*='blue', *label*='Datos Medidos')

    plt.plot(*puntos*, soluciones\_catenaria, *color*='green', *linestyle*='-', *label*='Catenaria')

    plt.plot(*puntos*, soluciones\_cuadratica, *color*='red', *linestyle*='--', *label*='Ajuste Cuadrático')

    plt.xlabel('X (cm)')

    plt.ylabel('Y (cm)')

    plt.title(*titulo*)

    plt.legend()

    plt.grid(True)

    plt.show()

*#----------------- CASO 0.8L -----------------*

print("----------------- CASO 0.8L -----------------")

x0 = -19.6

x1 = 19.6

puntos = np.array([x0, -15.0, -13.5, -12.0, -10.5, -9.0, -7.5, -6.0, -4.5, -3.0, -1.5, 1.5, 3.0, 4.5, 6.0, 7.5, 9.0, 10.5, 12.0, 13.5, 15.0, x1])

soluciones\_foto = np.array([

    y0,

    18.864, 17.129, 15.749, 14.508, 13.355, 12.468, 11.722, 11.203, 10.836, 10.659,

    10.635, 10.826, 11.256, 11.775, 12.498, 13.387, 14.313, 15.467, 16.826, 18.397,

    y1

])

*#calculo de mu y c2*

mu\_raiz = newton\_raphson(0.1, x1)

C2 = y1\*mu\_raiz - (np.cosh(mu\_raiz\*x1)) *#despejando de la ecuación (3)*

*# Resultados*

print("Valor de mu:", mu\_raiz)

print("Valor de C2", C2)

resultados\_soluciones(puntos, soluciones\_foto, 'Caso: Comparación de funciones para 0.8L')

*#CASO 0.7L*

print("----------------- CASO 0.7L -----------------")

x0 = -17.15

x1 = 17.15

puntos = np.array([x0, -15.0, -13.5, -12.0, -10.5, -9.0, -7.5, -6.0, -4.5, -3.0, -1.5, 1.5, 3.0, 4.5, 6.0, 7.5, 9.0, 10.5, 12.0, 13.5, 15.0, x1])

soluciones\_foto = np.array([

    y0,

    20.951, 17.932, 15.504, 13.615, 11.892, 10.566, 9.522, 8.775, 8.254, 7.944,

    7.972, 8.311, 8.846, 9.636, 10.623, 11.952, 13.559, 15.560, 17.846, 20.413,

    y1

])

*#calculo de mu y c2*

mu\_raiz = newton\_raphson(0.1, x1)

C2 = y1\*mu\_raiz - (np.cosh(mu\_raiz\*x1)) *#despejando de la ecuación (3)*

*# Resultados*

print("Valor de mu:", mu\_raiz)

print("Valor de C2", C2)

resultados\_soluciones(puntos, soluciones\_foto, 'Caso: Comparación de funciones para 0.7L')

*#----------------- CASO 0.3L -----------------*

*# Definimos las constantes según nuestro caso del modelo*

print("----------------- CASO 0.3L -----------------")

x0 = -7.35

x1 = 7.35

puntos = np.array([x0, -5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, x1])

soluciones\_foto = np.array([

    y0,

    11.464, 8.760, 6.519, 4.898, 3.809, 2.975, 2.309, 1.815, 1.469, 1.241,

    1.119, 1.278, 1.525, 1.846, 2.343, 2.911, 3.761, 4.593, 5.880, 7.287,

    y1

])

*#calculo de mu y c2*

mu\_raiz = newton\_raphson(0.1, x1)

C2 = y1\*mu\_raiz - (np.cosh(mu\_raiz\*x1)) *#despejando de la ecuación (3)*

*# Resultados*

print("Valor de mu:", mu\_raiz)

print("Valor de C2", C2)

resultados\_soluciones(puntos, soluciones\_foto, 'Caso: Comparación de funciones para 0.3L')

**Anexo II**

**primera-parte.py**

------------- ITEM B) -------------

Valor de mu: 0.0452347114575708

Valor de C2: 0.1877466921834858

Valor de y(x=0): 22.29466343327906

------------- ITEM C) -------------

Cota de error propagado de mu (analítica): 4.912575795925581e-05

Cota de error propagado de mu (experimental) 3.7389579801427036e-06

Diferencia entre cota analítica y experimental: 4.5386799979113107e-05

Valor de mu bien redeondeado: 0.0452

Cota de error de C2: 0.0647017779296913

C2 bien redondeado: 0

C2 expresado como intervalo: 0.1877466921834858 ± 0.07

Cota de error de y(x): 0.08871031044939455

Valor de y(0) bien redeondeado: 22

El valor expresado como intervalo 22.29466343327906 ± 0.1

**segunda-parte.py**

----------------- CASO 0.8L -----------------

Valor de mu: 0.06034313978532292

Valor de C2 -0.3366139483485733

Coeficientes del ajuste: [[ 1.05275216e+01]

[-6.20736398e-03]

[ 3.52954329e-02]]

ECM Catenaria: 5.369972573010284

ECM Cuadrática: 0.09547278207066552

----------------- CASO 0.7L -----------------

Valor de mu: 0.08830641188186822

Valor de C2 -0.26408122686673563

Coeficientes del ajuste: [[ 7.60906662e+00]

[-3.00032964e-03]

[ 5.61343595e-02]]

ECM Catenaria: 2.813739754120727

ECM Cuadrática: 0.21835433294394332

----------------- CASO 0.3L -----------------

Valor de mu: 0.4078014209306157

Valor de C2 -0.2538203665716541

Coeficientes del ajuste: [[ 0.01480042]

[-0.17107422]

[ 0.41749618]]

ECM Catenaria: 1.180772193276597

ECM Cuadrática: 1.0866821557099977

**Anexo III**

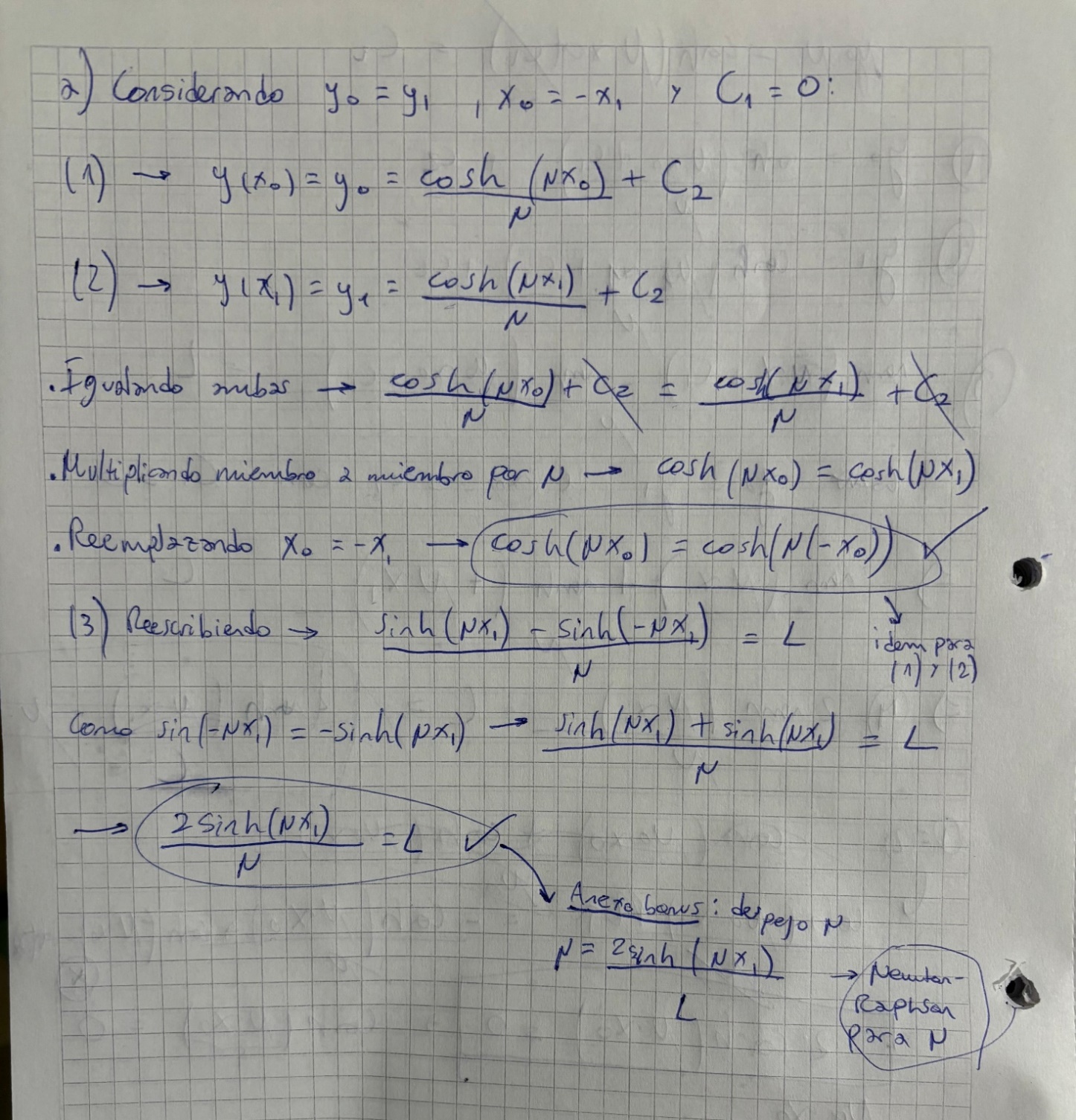
****

Figura 5: Análisis de a) primera parte

**Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja**

Figura 6: Gráfica de

**Bibliografía**

* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 1: Errores en el Cálculo Numérico*
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 2: Representación de números, errores de redondeo*
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 1, Módulo 3: Gráfica de proceso, condición del problema y término de estabilidad*
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 2, Módulo 2: Métodos de punto fijo y de Newton-Raphson*
* Facultad de Ingeniería, Universidad de Buenos Aires. *Unidad 3, Módulo 1: Ajuste por cuadrados mínimos (lineal y no lineal)*
* González, H. *Análisis Numérico, Primer curso.* Nueva Librería