Trabajo Práctico N^{o}3

Ecuaciones no-lineales

93.54 Métodos Numéricos

Grupo N°4

Legajo N°	Nombre
61428	Kevin Amiel Wahle
61430	Francisco Basili
61431	Nicolás Bustelo

05/05/2022

Índice

1. Introducción

El objetivo de este informe es presentar y explicar el funcionamiento de un programa realizado en *Python*, el cual consiste en encontrar la solución a una ecuación no-lineal. A su vez también se realizaron pruebas gráficas para comprobar el correcto funcionamiento. El código utilizado se encuentra en el Anexo (sección 6).

2. Selección del método a utilizar

Se debía elegir un método a utilizar, dentro de los 4 métodos vistos en clase, para encontrar la solución a la ecuación no-lineal que se observa a continuación:

$$L = \frac{\mu n^2 \pi r^2}{l^2} \left[\sqrt{r^2 + l^2} - r \right] \tag{1}$$

Para ello se comenzó realizando una gráfica de la función a analizar, con los valores adecuados. En ésta gráfica se pudo observar una pendiente no nula (y por ende tampoco se modifica la concavidad), lo cual indicaba que el método de Newton-Raphson funcionaría correctamente. A continuación se pueden observar una serie de imágenes, en las cuales se graficó la función para distintos valores de n, con diferentes escalas de L (dentro de los posibles valores a obtener), pudiéndose observar que se cuenta con una notable pendiente en todos los rangos razonables de valores. Para ello se utilizó el graficador de Desmos. También se gráfico la derivada de la función a analizar (véase 6), para asegurar que su pendiente siempre sea positiva para los valores razonables de las variables.

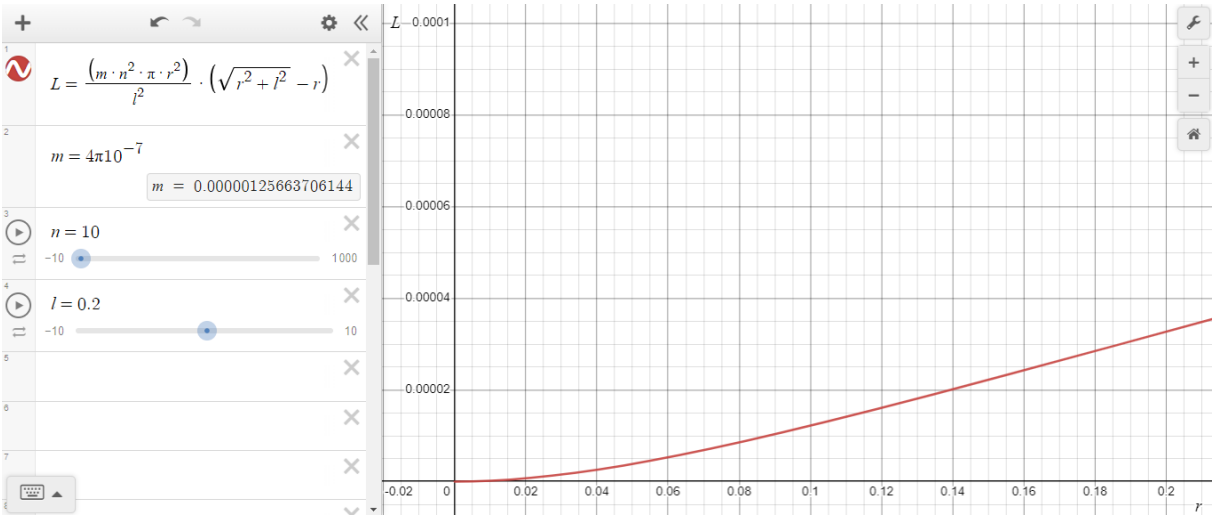

Figura 1: Estimación de L en el orden de las centenas de μH con n=10

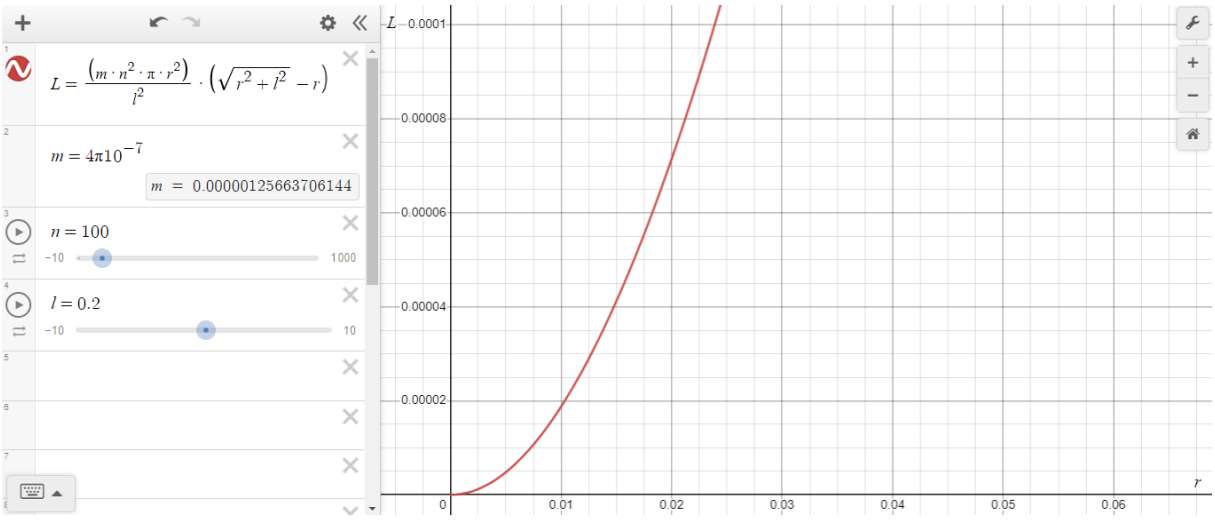

Figura 2: Estimación de L en el orden de las centenas de μH con n=100

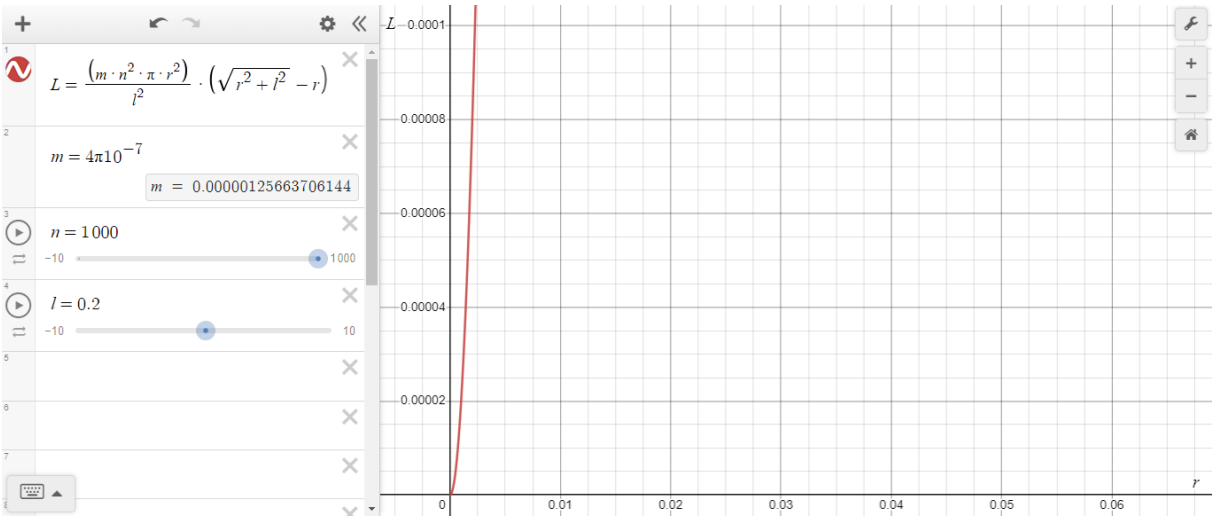

Figura 3: Estimación de L en el orden de las centenas de μH con n=1000

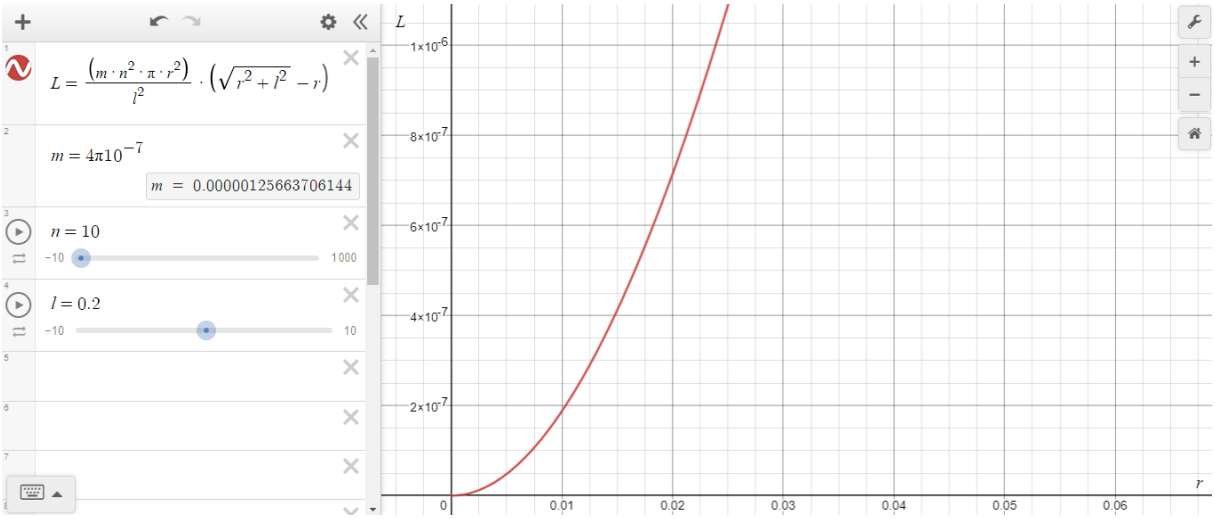

Figura 4: Estimación de L en el orden de las centenas de nH con n=10

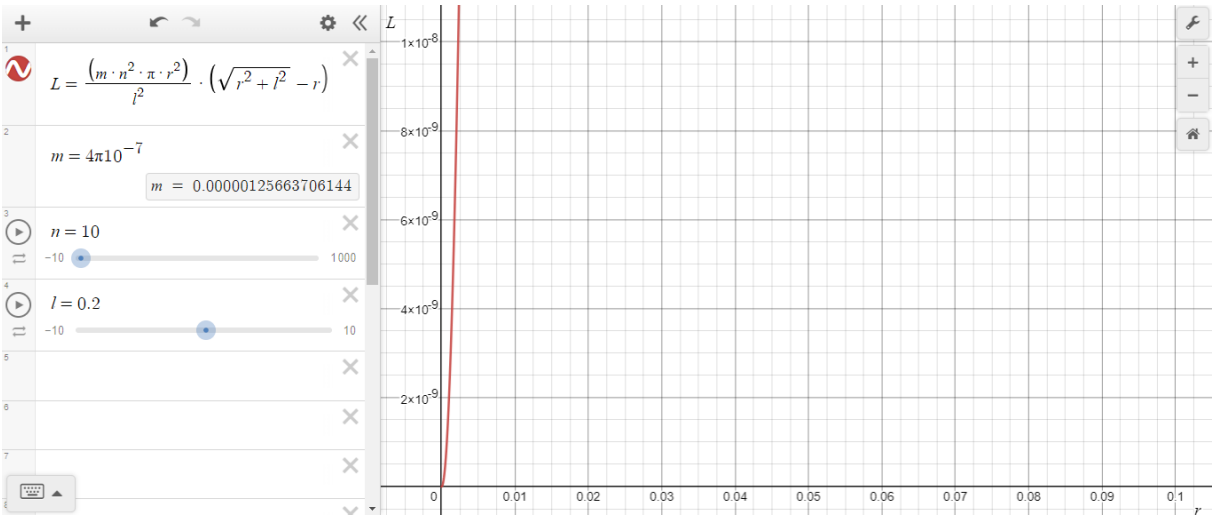

Figura 5: Estimación de L en el orden de las unidades de *nH* con n=10

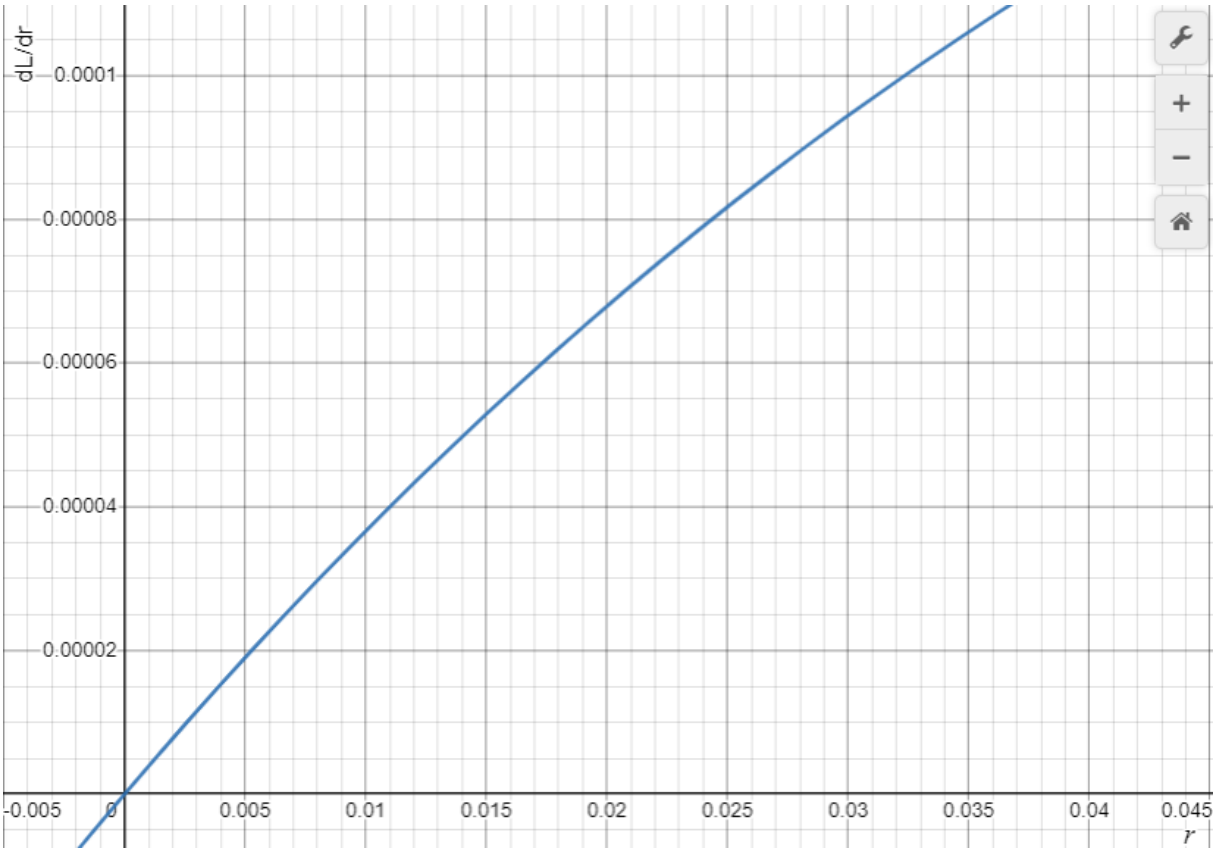

Figura 6: Derivada de la función con valores positivos y misma concavidad $\forall r \in$ los valores lógicos

A su vez, se calculó analíticamente:

$$\frac{\partial L}{\partial r} = \frac{2\pi\mu N^2 r(\sqrt{l^2 + r^2} - l)}{l^2} + \frac{\pi\mu N^2 r^3}{l^2\sqrt{l^2 + r^2}}$$

de donde se puede notar que la única manera que $\frac{\partial L}{\partial r} = 0$ es que $N = 0$ ó que $r = 0$, los cuales no representarían situaciones lógicas.

A pesar de que esta prueba era necesaria, no era suficiente para asegurar que la convergencia de éste método fuera la más rápida. Es por este motivo que se realizaron diversas pruebas ceteris paribus en python con el objetivo de ver en cual de los métodos se requería un menor número de iteraciones del algoritmo. Finalmente el algoritmo de Newton-Raphson el que mejor resultados arrojó.

3. Función *solver(L,l,n)*

Como se mencionó anteriormente, se utilizó el método de Newton-Raphson para hallar la solución a la ecuación no lineal planteada. Para ello se definió las funciones:

$$f(r) = \frac{\mu n^2 \pi r^2}{l^2}\left[\sqrt{r^2 + l^2} - r\right] - L$$

$$\frac{\partial f}{\partial r} = \frac{\pi \mu n^2 r\left(\frac{r^2}{\sqrt{l^2+r^2}} + 2\sqrt{l^2 + r^2} - 2L - 3r\right)}{l^2}$$

Se seleccionó una tolerancia *tol* de $1 \cdot 10^{-9} m$ debido a que los valores razonables de radio de los inductores no podían ser menores a $1nm$.

Finalmente se realizaron las iteraciones necesarias, teniendo en cuenta como límite la tolerancia mencionadas, considerando que:

$$X_{K+1} = X_K - \frac{f(X_K)}{f'(X_K)}$$

4. Función *test()*

Se realizó una función *test()* para probar el correcto funcionamiento del código. Para ello se reemplazó en la ecuación (1) el valor de radio r obtenido en la función *solver(L,l,n)*, y se comparó el resultado con el valor inicial de L planteado. Para las diferentes pruebas, se varió la inductancia L, la longitud l y el número de vueltas n.

5. Función *graphRvL()*

Se graficó r en función de L, para $L \in [100nH, 100\mu H]$ y $N = 10, 100, 1000$ vueltas. El gráfico que se llevó a cabo se puede observar a continuación:

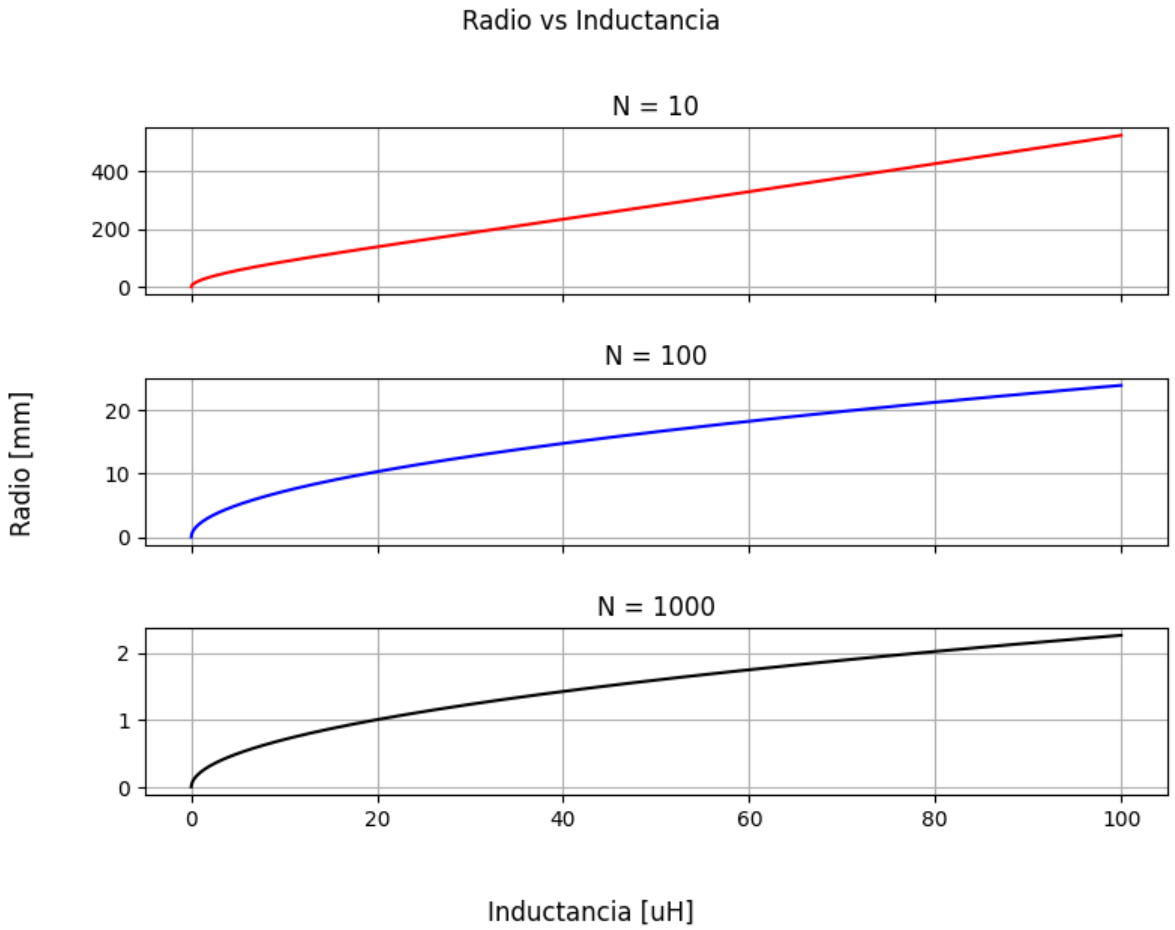

Figura 7: Gráfica de r en función de L para distintos N

6. Anexo

6.1. Código completo en Python

```python
# ------------------------------------------------------------------------------
#  @file     +mri.py+
#  @brief    +Ecuaciones no-lineales+
#  @author   +Grupo 4+
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# LIBRARIES
# ------------------------------------------------------------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as mt

# ------------------------------------------------------------------------------
# FUNCTION DEF
# ------------------------------------------------------------------------------
# Resolución de ecuaciones no-lineales
def solver(L,l,n):
    tol = 1e-9
    maxiter = 100

    mu = 4 * np.pi * 10**(-7)    # Permeabilidad del vacío
    ro = 0.1                     # Radio del solenoide en metros

    # Función a resolver
    f = lambda r:(((mu * n**2 * np.pi * r**2) / (l**2)) * (np.sqrt(r**2 + l**2) - r))-L
    # Derivada de la función a resolver
    df = lambda r: (n**2*np.pi*r*(-r + np.sqrt(l**2 + r**2))*(-r + 2*np.sqrt(l**2 + r**2)
    )*mu)/(l**2 * np.sqrt(l**2 + r**2))

    #--------------- NEWTON-RAPHSON -----------------------------------------
    for iters in range(maxiter): # Iteramos como máximo maxiter veces
        dr = f(ro)/df(ro)
        ro = ro-dr
        if abs(dr) < tol:   # Si se llega a un error menor al deseado, terminamos
            break

    return ro, iters


def graphRvL():
    C = 10000                       # Cantidad de puntos a tomar
    l = 0.2                         # Longitud del solenoide en metros
    L = np.linspace(1e-9,100e-6,C)  # Inductancia del solenoide H
    N = [10, 100, 1000]             # Cant de vueltas

    #Inicialización de arreglos
    r1 = np.zeros(C); r2 = np.zeros(C); r3 = np.zeros(C)

    for i in range(C):  # Calculo de radios para cada numero de espiras
        r1[i],_ = solver(L[i],l,N[0])
        r2[i],_ = solver(L[i],l,N[1])
        r3[i],_ = solver(L[i],l,N[2])

    # Graficación
    fig, axs = plt.subplots(3, sharex=True)
    fig.suptitle('Radio vs Inductancia')
    axs[0].plot(L*1e6, r1*1000, color="r")
    axs[1].plot(L*1e6, r2*1000, color="b")
    axs[2].plot(L*1e6, r3*1000, color="k")

    # Configuraciones de los ejes
    axs[0].set_title('N = 10'); axs[0].grid('both')
    axs[1].set_title('N = 100') ; axs[1].grid('both')
    axs[2].set_title('N = 1000') ; axs[2].grid('both')
    fig.supxlabel('Inductancia [uH]')
    fig.supylabel('Radio [mm]')
```

```
68
69      plt.tight_layout()
70      figManager = plt.get_current_fig_manager()
71      figManager.window.showMaximized()
72      plt.show()
73
74
75  def test():
76      tol = 1e-9              # Tolerancia aceptada
77      mu = 4 * np.pi * 10**(-7)   # Permeabilidad del vacío
78      l = np.linspace(0.02,1,100) # Longitudes del solenoide en metros
79      L = np.linspace(1e-9,100e-6,100) # Inductancias del solenoide H
80      n = np.linspace(1,100,100)  # Cantidad de espiras
81
82      # Calculo de la inductancia
83      Lcalc = lambda r,l,n:((mu * n**2 * np.pi * r**2) / (l**2)) * (np.sqrt(r**2 + l**2) -
        r)
84
85      result=0 ; tot=0
86
87      for l_ in l:
88          for L_ in L:
89              for n_ in n:
90                  tot+=1
91
92                  # Calculo del radio para cada valor de cada parámetro
93                  r,_ = solver(L_,l_,n_)
94
95                  # Verificación del radio obtenido
96                  result = result + 1 if abs(L_ - Lcalc(r,l_,n_)) < tol else result
97
98      print('Se pasaron con éxito', result, '/', tot, 'casos')
```