



Instituto Tecnológico  
de Buenos Aires

# TRABAJO PRÁCTICO N°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	2
2. Selección del método a utilizar	2
3. Función <i>solver</i> ( <i>L,l,n</i> )	4
4. Función <i>test</i> ()	5
5. Función <i>graphRvL</i> ()	5
6. Anexo	6
6.1. Código completo en Python . . . . .	6

## 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:

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

Para ello se comenzó realizando una gráfica de la función a analizar, con los valores adecuados. En ésta gráfica se pudo observar una gran pendiente, lo cual indicaba que el método de Newton-Raphson funcionaría de manera óptima. 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 gran pendiente en todos los rangos razonables de valores. Para ello se utilizó el graficador de [Desmos](#). También se graficó que la derivada de la función a analizar, para asegurar que su pendiente siempre sea positiva para los valores razonables de las variables.

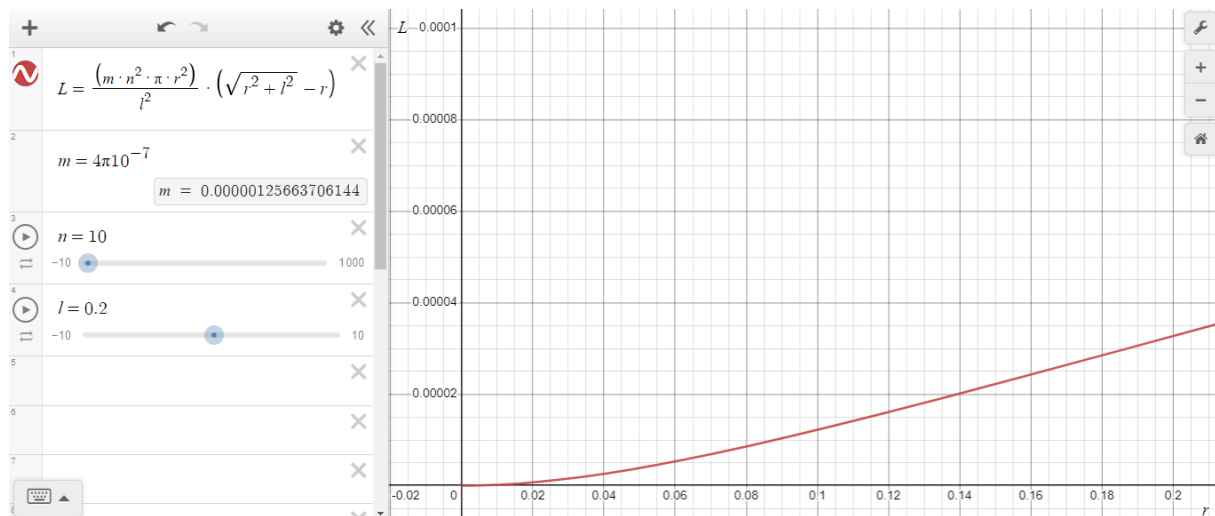


Figura 1: Estimación de  $L$  en el orden de las centenas de  $\mu H$  con  $n=10$

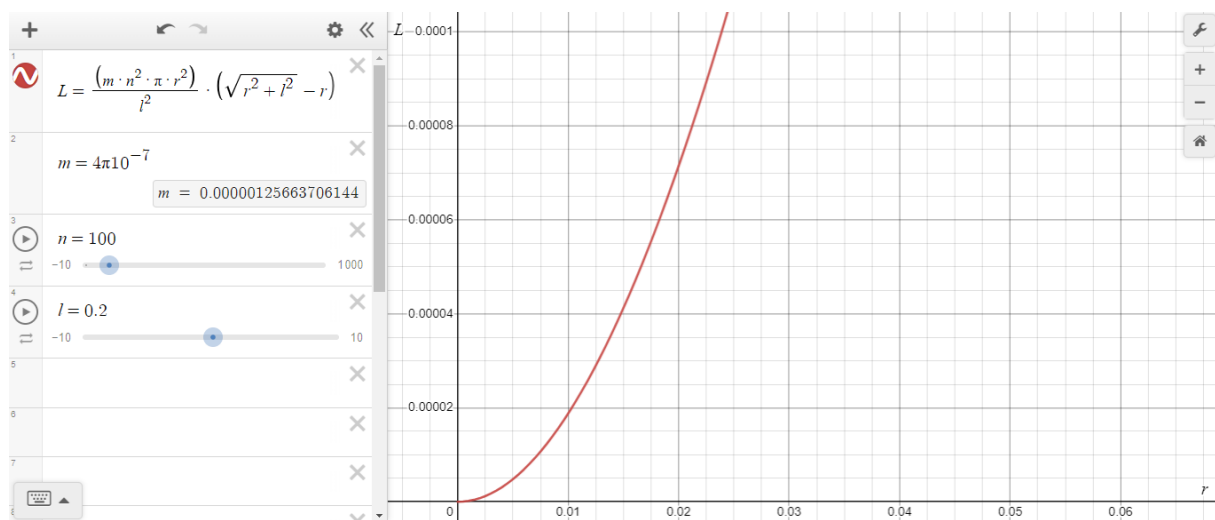


Figura 2: Estimación de L en el orden de las centenas de  $\mu H$  con  $n=100$

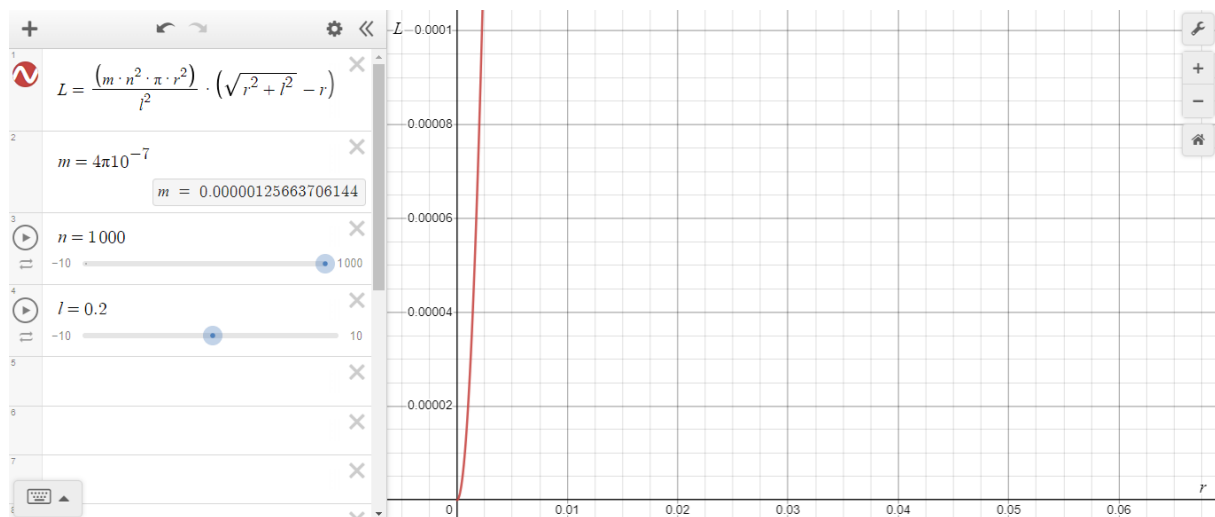


Figura 3: Estimación de L en el orden de las centenas de  $\mu H$  con  $n=1000$

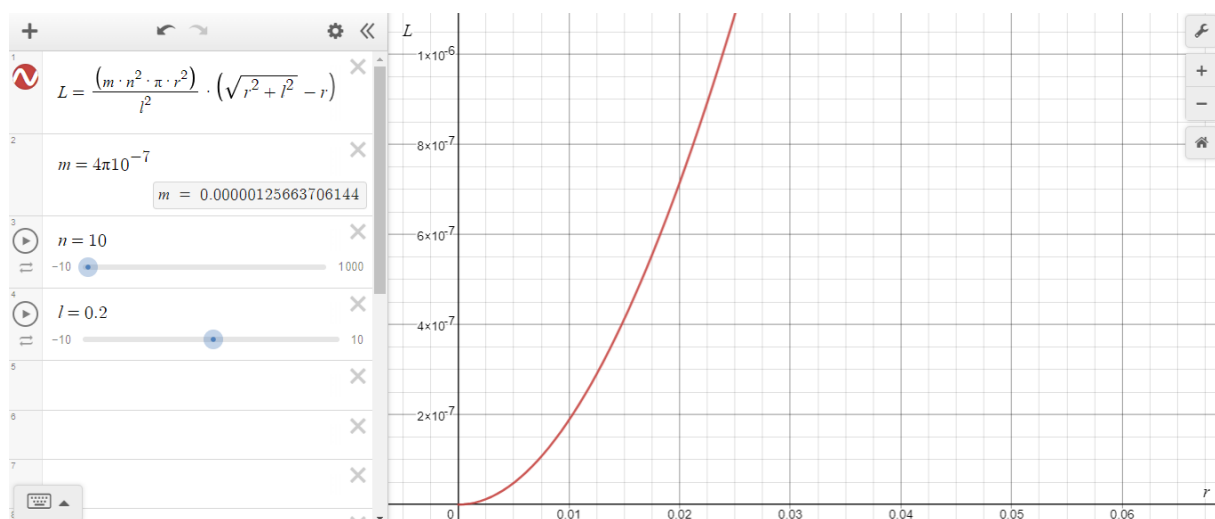


Figura 4: Estimación de L en el orden de las centenas de  $nH$  con  $n=10$

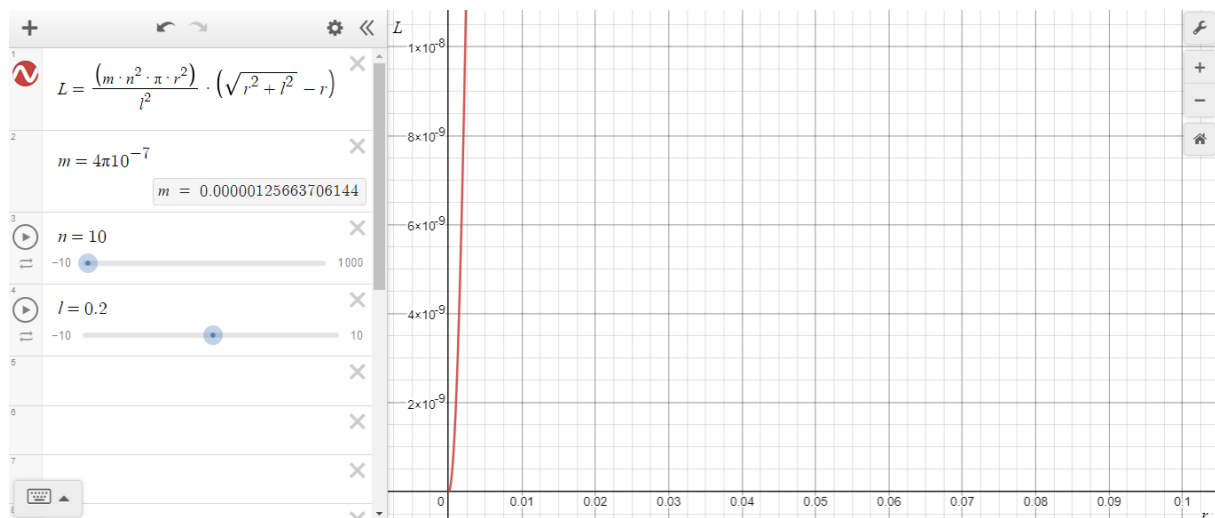


Figura 5: Estimación de L en el orden de las unidades de  $nH$  con  $n=10$

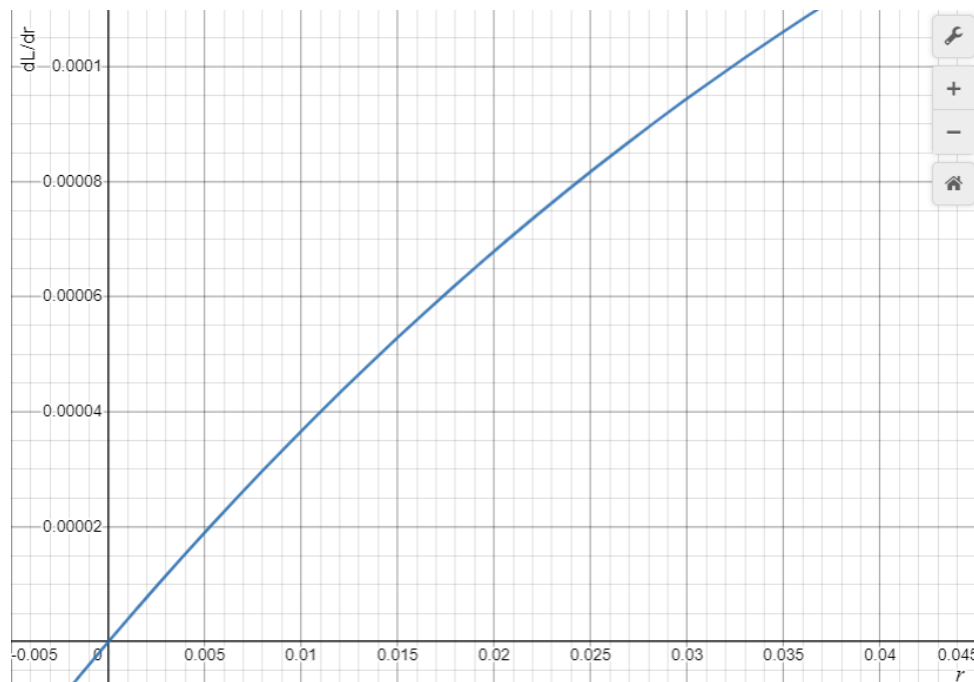


Figura 6: Derivada de la función con pendiente positiva  $\forall r \in$  los valores lógicos

A pesar de que esta prueba era necesaria, no era suficiente para asegurar que la convergencia de éste método fuera la mas 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. Fue el algoritmo de Newton-Raphson el que mejor resultados arrojó.

### 3. Función $\text{solver}(L, l, n)$

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  $1 \cdot 10^{-9}$  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, 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 las vueltas  $r$ .

#### 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:

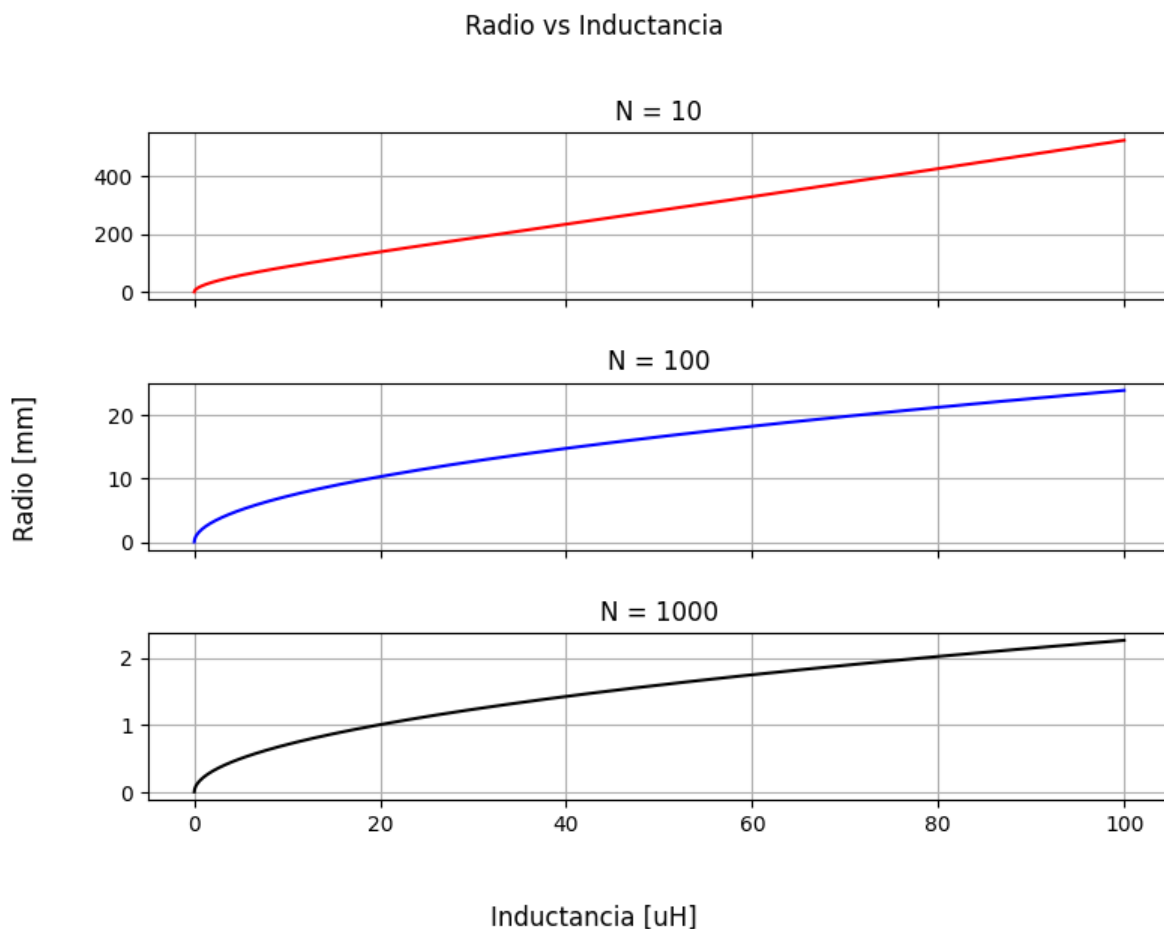


Figura 7: Gráfica de  $r$  en función de  $L$  para distintos  $N$

## 6. Anexo

### 6.1. Código completo en Python

```

1 # -----
2 # @file      +mri.py+
3 # @brief     +Ecuaciones no-lineales+
4 # @author    +Grupo 4+
5 # -----
6
7 # -----
8 # LIBRARIES
9 # -----
10 import numpy as np
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 import math as mt
14
15 # -----
16 # FUNCTION DEF
17 # -----
18 # Resolución de ecuaciones no-lineales
19 def solver(L,l,n):
20     tol = 1e-9
21     maxiter = 100
22
23     mu = 4 * np.pi * 10**(-7) # Permeabilidad del vacío
24     ro = 0.1                   # Radio del solenoide en metros
25
26     # Función a resolver
27     f = lambda r: (((mu * n**2 * np.pi * r**2) / (l**2)) * (np.sqrt(r**2 + l**2) - r)) - L
28     # Derivada de la función a resolver
29     df = lambda r: (n**2*np.pi*r*(-r + np.sqrt(l**2 + r**2))*(-r + 2*np.sqrt(l**2 + r**2))
30     )*mu)/(l**2 * np.sqrt(l**2 + r**2))
31
32     #----- NEWTON-RAPHSON -----
33     for iters in range(maxiter): # Iteramos como máximo maxiter veces
34         dr = f(ro)/df(ro)
35         ro = ro-dr
36         if abs(dr) < tol: # Si se llega a un error menor al deseado, terminamos
37             break
38
39     return ro, iters
40
41 def graphRvL():
42     C = 10000 # Cantidad de puntos a tomar
43     l = 0.2   # Longitud del solenoide en metros
44     L = np.linspace(1e-9,100e-6,C) # Inductancia del solenoide H
45     N = [10, 100, 1000] # Cant de vueltas
46
47     #Inicialización de arreglos
48     r1 = np.zeros(C); r2 = np.zeros(C); r3 = np.zeros(C)
49
50     for i in range(C): # Calculo de radios para cada numero de espiras
51         r1[i],_ = solver(L[i],l,N[0])
52         r2[i],_ = solver(L[i],l,N[1])
53         r3[i],_ = solver(L[i],l,N[2])
54
55     # Graficación
56     fig, axs = plt.subplots(3, sharex=True)
57     fig.suptitle('Radio vs Inductancia')
58     axs[0].plot(L*1e6, r1*1000, color='r')
59     axs[1].plot(L*1e6, r2*1000, color='b')
60     axs[2].plot(L*1e6, r3*1000, color='k')
61
62     # Configuraciones de los ejes
63     axs[0].set_title('N = 10'); axs[0].grid('both')
64     axs[1].set_title('N = 100'); axs[1].grid('both')
65     axs[2].set_title('N = 1000'); axs[2].grid('both')
66     fig.supxlabel('Inductancia [uH]')
67     fig.supylabel('Radio [mm]')

```

```
68 plt.tight_layout()
69 figManager = plt.get_current_fig_manager()
70 figManager.window.showMaximized()
71 plt.show()
72
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) -
84     r)
85
86     result=0 ; tot=0
87
88     for l_ in l:
89         for L_ in L:
90             for n_ in n:
91                 tot+=1
92
93                 # Calculo del radio para cada valor de cada parámetro
94                 r,_ = solver(L_,l_,n_)
95
96                 # Verificación del radio obtenido
97                 result = result + 1 if abs(L_ - Lcalc(r,l_,n_)) < tol else result
98
99     print('Se pasaron con éxito', result, '/', tot, 'casos')
```