Clase_01_Parte01

October 12, 2019

1 1er Curso de Fluidodinámica con Python

1.1 Introducción

Bienvenido a este primer curso esperamos sea de tu agrado el contenido que presentemos, para participar tendremos algunas consideraciones:

1. ¿Recuerdas las EDO's y las EDP's.?
2. ¿Sabes los fundamentos de la mecánica de fluidos.?
3. ¿De programación conoces lo básico en algún lenguaje?

Dicho esto es claro que tienes una motivación ya que nos gastaremos el tiempo en aprender algo nuevo, vamos a apoyarnos en el lenguaje de programación Python para desarrollar este curso, imagino que has escuchado de este lenguaje de programación ¿si?, bueno al final es un lenguaje de programación más aunque recomendable si es que deseas programar en muy poco tiempo ya que es bastante fácil, total, todos los presentes somos de ingenierías y ciencias por ello no debemos presentar complicaciones en cuanto a aprender algo como esto.

Python es un lenguaje de programación: * Orientado a objetos * Multipropósito * Interpretado * Fácil de aprender * Multiplataforma * Open-source

1.2 Mira a Python como un lenguaje "pegamento" con otros

Con el tiempo notarás que has aprendido un lenguaje moderno y con un campo enorme de aplicación, aunque debo señalar desde un principio que como tal a este lenguaje lo sostiene su enorme comunidad en todo el mundo, siempre que tengas dudas búsca, un buen lugar al que puedes entrar a consultar es a StackOverFlow, pero ojo, preguntar es un privilegio, sugiero la verdad puedas primero leerte esto "Cómo hacer preguntas de manera inteligente"; regresando al titulo de este parrafo lo digo porque Python es fabuloso, pero ojo, "hay programas que se sobre salen en ciertas áreas y pues Python también podría hacer la tarea, pero tienes que evaluar bien", conocer Python para mi es como andar sobre un todo-terreno, puedes hacer de todo, incluso hay artificios que podrías llevar, pero al final la decisión vendrá del desarrollador y esto dependerá de su expertís; alguna vez conversando con alguien con mayor experiencia en desarrollo me hizo entender esto; para nuestro curso va bien, así que sigamos adelante.

1.3 Conocimientos previos

1.3.1 Python es un lenguaje interpretado

Significa que a medida que vayas escribiendo tu código este podrá interactuar directamente con el ordenador, las ventajas pues que vas desarrollando y puedes ir ejecutando; esto es algo que no hacen los lenguajes compilados ya que este último tendría que convertir tu código fuente a un archivo binario que solamente es entendido por la CPU; entonces si sabemos que Python es un lenguaje interpretado significa que es mejor que otros lenguajes ¿verdad?, sinceramente esa es una pregunta difícil de responder ya que depende de tus indicadores que pongas, si te refieres a que es mejor en cuanto a legibilidad de lectura, pues si; te resultará fácil de comprender el código, además que como lenguaje interpretado te tomará menos lineas de código para escribir un programa; los lenguajes compilados suelen tener muchas filas; - oye pero si me dices todo esto sigo pensando que Python es un mejor lenguaje - , no , no he terminado de contarte, Python es cierto que tiene un montón de ventajas, pero en cuanto a velocidad de calculo pues queda corto ante los lenguajes compilados y te guste o no siempre será así y es que es lógico, la tarea de compilar ya dije que trata de que tienes que crear un nuevo archivo en binario y por ser algo que el CPU lo entiende más "natural" hace todos sus calculos más rápidos, por ese motivo es que Python queda chico en velocidad, si vas a trabajar resolviendo grandes ecuaciones, meterte a computación de alto rendimiento (HPC) que tal vez puedas pensar que no va tan bien; pero sabes con Python se pueden hacer artificios, podrías por ejemplo escribir cierto código en Fortran y este podría ser ejecutado desde un script principal , llamaríamos a ese archivo Python con F2PY por ejemplo, o sino podríamos paralizar las tareas en el CPU y la GPU para el lenguaje Python con Numba.

1.3.2 Entornos virtuales

Un entorno virtual no es otra cosa que un ambiente de trabajo aislado , te permite contar con una versión del interprete de Python aislado teniendo la opción de poder interactuar con otras librerías que podrás instalar.

¿Por qué necesito un crear un entorno virtual?

Python nació en 1991, desde esa fecha ha venido evolucionando y teniendo muchas mejoras; pero como tal, es cierto que hay librerías que solamente pueden interactuar con ciertas versiones de otras dependencias, bueno se pueden ocasionar conflictos de versiones y para evitar ello creamos los famosos "entornos virtuales", así de simple.

Facilitamos enlace para la gestión de los entornos: * Documentación para entornos virtuales con PIP * Documentación para entornos virtuales con Conda

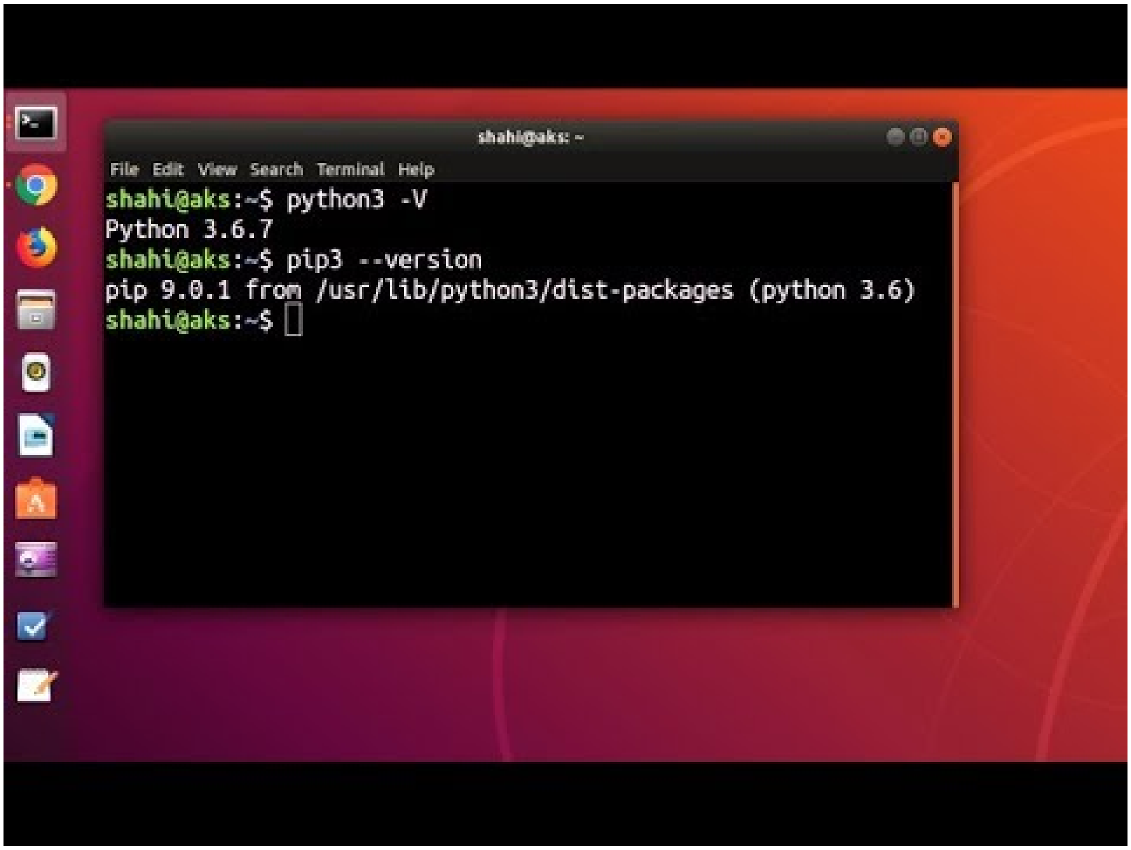
Nota: Los gestores de paquetes para Python más populares son PIP y CONDA, si usas GNU/Linux tu sistema operativo ya tiene todo listo, si eres de Windows necesitarás instalar Python, posiblemente les interese comenzar con Anaconda ya que es relativamente fácil para los "novatos".

[1]:
```
from IPython.display import YouTubeVideo
# Vídeo en inglés para instalar Anaconda en Ubuntu OS y derivados
YouTubeVideo('DYODB_NwEu0')
```

[1]:

[2]:
```
# Vídeo en inglés para instalar Python3-pip en Ubuntu OS y derivados
YouTubeVideo('FfkPLekXuL4')
```

[2]:



[3]:
```python
# Vídeo en inglés para instalar Python3-pip en Ubuntu OS y derivados
YouTubeVideo('5mDYijMfSzs')
```

[3]:

[4]:
```
# Vídeo en inglés para instalar Python3-pip en Ubuntu OS y derivados
YouTubeVideo('gFNApsyhpKk')
```

[4]:

2 Programación aplicada

2.1 Variables

[5]:
```python
x1 = 96
x2 = 17
x3 = 35
x4 = 96.
x5 = 17.0
x6 = 17.00
x7 = 'avión'
x8 = 'casaca'
x9 = 'cuaderno'
```

2.2 Tipos de objetos

[6]:
```python
type(x1)
```

[6]: int

[7]:
```python
print(type(x1))
print(type(x2))
print(type(x3))
print(type(x4))
print(type(x5))
print(type(x6))
print(type(x7))
print(type(x8))
print(type(x9))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'str'>
<class 'str'>
<class 'str'>
```

2.3 Operadores matemáticos

[8]:
```python
12 + 12.
```

[8]: 24.0

[9]:
```python
12 + 12
```

[9]: 24

[10]:
```python
14/7
```

[10]: 2.0

[11]:
```python
14/7.
```

[11]: 2.0

[12]:
```python
15/7
```

[12]: 2.142857142857143

Para la versión 2 del interprete de Python se tenía que agregar la siguiente línea de comando en caso era de interés ejecutar divisiones entre enteros y estos de manera automática se quisiera presente el resultado como un decimal.

[13]:
```
from __future__ import division
```

[14]:
```
14/7
```

[14]: 2.0

2.4 La identación

A diferencia de otros lenguajes de programación en Python no se utilizan signos de colección para ejecutar bucles o estructuras repetitivas, esto se logra gracias a que se respeta la identación, la idea es que el programador use o las barras espaciadoras o las tabulaciones pero no en simultaneo ambas ya que el hacer esto traería inconsistencia en el código.

2.5 Esquema For

[15]:
```
for i in range(10):
    print("Bienvenido a este primer curso")
```

```
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
Bienvenido a este primer curso
```

2.6 Bucles anidados caso For

[16]:
```
for i in range(2):
    for j in range(2):
        for k in range(2):
            print(i, j, k)
            print("-----------")
            print("x: " + str(i))
            print("y: " + str(j))
```

```python
            print("z: " + str(k))
            print('***********')
            print("Estamos en la sentencia que incluye a la i, j, k")
        print("Estamos en la sentencia que incluye a la i, j")
    print("Estamos en la sentencia que incluye a la i")
```

```
0 0 0
-----------
x: 0
y: 0
z: 0
***********
Estamos en la sentencia que incluye a la i, j, k
0 0 1
-----------
x: 0
y: 0
z: 1
***********
Estamos en la sentencia que incluye a la i, j, k
Estamos en la sentencia que incluye a la i, j
0 1 0
-----------
x: 0
y: 1
z: 0
***********
Estamos en la sentencia que incluye a la i, j, k
0 1 1
-----------
x: 0
y: 1
z: 1
***********
Estamos en la sentencia que incluye a la i, j, k
Estamos en la sentencia que incluye a la i, j
Estamos en la sentencia que incluye a la i
1 0 0
-----------
x: 1
y: 0
z: 0
***********
Estamos en la sentencia que incluye a la i, j, k
1 0 1
-----------
x: 1
```

```
y: 0
z: 1
***********
Estamos en la sentencia que incluye a la i, j, k
Estamos en la sentencia que incluye a la i, j
1 1 0
-----------
x: 1
y: 1
z: 0
***********
Estamos en la sentencia que incluye a la i, j, k
1 1 1
-----------
x: 1
y: 1
z: 1
***********
Estamos en la sentencia que incluye a la i, j, k
Estamos en la sentencia que incluye a la i, j
Estamos en la sentencia que incluye a la i
```

2.7 Guía de estilo al momento de programar en Python PEP - 8

El PEP-8 es una sugerencia que se hace a los nuevos, es decir si vas a comenzar algo es mejor que lo hagas de buena manera y aprendas los hábitos que se recomiendan, pasa que en el mundo del desarrollo puede que te toque tocar código ajeno y para ello deberas entender que es lo que hizo la otra persona, aunque si no se pone un guía de como programar gastarás más tiempo en tratar de entender que empezar a programar, son como 88 normas, sugiero que los leas en:

Guía de estilo PEP-8 en ingles

2.8 Matrices

Hemos de usar la librería Numpy

[17]:
```python
import numpy as np
```

[18]:
```python
var01 = np.array([1, 2, 3, 4])
```

[19]:
```python
print(var01)
```

```
[1 2 3 4]
```

[20]:
```python
var01[0]
```

[20]: 1

[21]:
```
var01[0:4]
```

[21]: array([1, 2, 3, 4])

[22]:
```
var01[0], var01[3]
```

[22]: (1, 4)

[23]:
```
var02 = np.array([1, -3, 0, 1, -5])
```

[24]:
```
var02[-1]
```

[24]: -5

[25]:
```
var02[0:-1]
```

[25]: array([1, -3, 0, 1])

[26]:
```
var02[4:5]
```

[26]: array([-5])

[27]:
```
var02[4]
```

[27]: -5

[28]:
```
var02[4:-1]
```

[28]: array([], dtype=int64)

[29]:
```
var02[-2:-1]
```

[29]: array([1])

2.8.1 Array de una dimensión

[30]:
```
var03 = np.linspace(-10, 10, 5)
var03
```

[30]: array([-10., -5.,   0.,   5., 10.])

[31]:
```
print(var03)
```

```
[-10.  -5.   0.   5.  10.]
```

2.8.2 Creamos una copia de var02

1° Con la sintaxis foo02 = foo01 lo que se está creando es una declaración que referencia los valores de foo01 a foo01, se está haciendo un puntero.

[32]:
```
var04 = var03
```

[33]:
```
var04
```

[33]: array([-10., -5.,   0.,   5., 10.])

[34]:
```
var04[0]
```

[34]: -10.0

[35]:
```
var04[0] = 0
```

[36]:
```
var04
```

[36]: array([0., -5., 0., 5., 10.])

[37]:
```
var03
```

[37]: array([0., -5., 0., 5., 10.])

[38]:
```
var03[0]
```

[38]: 0.0

[39]:
```
var03[0] = 30
```

[40]:
```
var03
```

[40]: array([30., -5., 0., 5., 10.])

[41]:
```
var04
```

[41]: array([30., -5., 0., 5., 10.])

Observamos que si cambiamos el valor del primer elemento a var03 en consecuencia simulteanea ocurré lo mismo para var04, por ello se identifica la referenciación que existe entre ambos.

[42]:
```
print(var03)
```

[30. -5. 0. 5. 10.]

[43]:
```
type(var03)
```

```
[43]: numpy.ndarray
```

Creamos una copia de var03 como var05

var05[:] = var03[:] Este es el esquema correcto para crear una copia con todos sus elementos para una matriz, aunque antes deberíamos de contar con una matriz de las mismas dimensiones vacía.

Notamos que nos imprime error, ello ocurre porque hacer una copia de esta manera exige que primero se genere una matriz vacía con la misma cantidad de elmeentos de var03, para ello hacemos:

```
[44]: #print(var05)
```

```
[45]: print(np.empty_like(var03))
```

```
[30. -5.  0.  5. 10.]
```

```
[46]: var05 = np.empty_like(var03)
      print(var05)
```

```
[30. -5.  0.  5. 10.]
```

```
[47]: len(var05)
```

```
[47]: 5
```

2.9 Entendiendo np.empty_like(varX)

Debemos contar con la variable 'a' , esta guarda en memoria a una matriz de dimensiones nxm , lo que se desea es crear una matriz equivalente en las dimensiones, por lo tanto hacemos:

```
[48]: a = ([1, 2, 3], [4, 5, 6])
      print(a)
```

```
([1, 2, 3], [4, 5, 6])
```

```
[49]: np.empty_like(a)
```

```
[49]: array([[48690896,        0,        0],
             [       0,        0,        0]])
```

La nueva matriz generada a partir de 'a' tiene la misma cantidad de elementos y en las mismas posiciones, aunque los elementos son random

```
[50]: np.empty_like(var03)
```

[50]: array([30., -5., 0., 5., 10.])

```
[51]: var003 = [30, 5, 0, 5, 10]
```

```
[52]: np.empty_like(var003)
```

```
[52]: array([          54561008,     139663746531328,                   0,
                        0, 2314885530818453536])
```

Acabamos de descubrir que la función 'np.empty_like(var)' crea una nueva matriz equivalente siempre y cuando sus terminos hayan sido guardados de la forma de números enternos, en el caso sean decimales la nueva matriz será identica a la anterior.

```
[53]: a = np.array([[1., 2., 3.],[4., 5., 6.]])
```

```
[54]: np.empty_like(a)
```

```
[54]: array([[2.32040184e-316, 0.00000000e+000, 0.00000000e+000],
             [0.00000000e+000, 0.00000000e+000, 0.00000000e+000]])
```

```
[55]: type(a)
```

[55]: numpy.ndarray

```
[56]: np.empty_like(var03)
```

[56]: array([1.5e-322, 2.5e-323, 0.0e+000, 2.5e-323, 4.9e-323])

```
[57]: var05 = np.empty_like(var03)
```

```
[58]: print(var05)
```

```
[1.5e-322 2.5e-323 0.0e+000 2.5e-323 4.9e-323]
```

Para mejor entendimiento de np.empty_like(a) revisar el siguiente enlace: Documentación desde Scipy

```
[59]: var05[:] = var03[:]
```

```
[60]: print(var05)
```

```
[30. -5.  0.  5. 10.]
```

```
[61]: type(var05)
```

[61]: numpy.ndarray

2.10 Matemática básica con una matriz de 1D en Numpy

[62]:
```python
a = [1, 2, 3,]
print(a)
print(a+a)
```

```
[1, 2, 3]
[1, 2, 3, 1, 2, 3]
```

[63]:
```python
b = a
type(b)
```

[63]:
```
list
```

[64]:
```python
import numpy as np
```

[65]:
```python
b = np.array(a)
print(type(b))
print(b)
```

```
<class 'numpy.ndarray'>
[1 2 3]
```

[66]:
```python
print(b+b)
print(b + 1)
print(b**2)
print(np.sin(b))
```

```
[2 4 6]
[2 3 4]
[1 4 9]
[0.84147098 0.90929743 0.14112001]
```

[67]:
```python
A = 5
print("Imprimir los " + str(A) + " primeros números usando la función 'range'")
for i in range(A):
    print( i, end=', ')
```

```
Imprimir los 5 primeros números usando la función 'range'
0, 1, 2, 3, 4,
```

2.11 Framework Qt, el complemento para el desarrollo de software GUI

Qt es un framework multiplataforma para el desarrollo de software, también es posible hacer software embebido para vehículos y equipos industriales.

La metodología que usaremos para crear aplicaciones será que usaremos el Qt Designer donde se usará para ir creando la interfaz de usuario, es posible luego convertir este archivo a uno con extensión .py, eso se logra bajo la sintaxis:

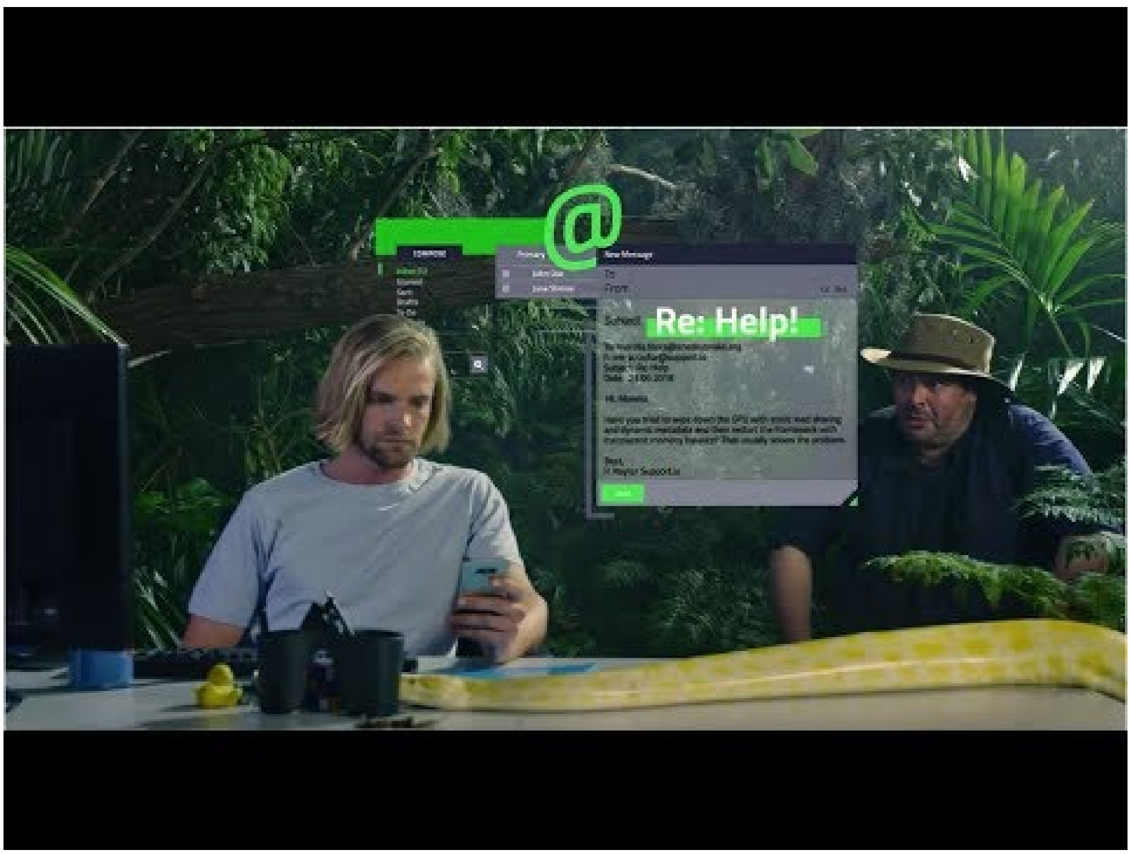
$ pyuic5 file_GUI.ui -o file_GUI.py

En este curso además tendremos la posibilidad de desarrollar nuestras aplicaciones propias aplicaciones, si hablamos del lenguaje Python existen las librerías PyQt, Tkinter, PySide, WxPython, WxPython; en nuestro caso usaremos PyQt en su versión 5 con el cual daremos la funcionalidad a los archivos en formato .ui que nos brinde el Qt Designer.

Para la instalación sugerimos hacerlo desde la página web oficial, prueben la versión open-source.

[68]:
```python
# En este vídeo se muestra un spot que hace Qt a su librería PySide para trabajar con Python
YouTubeVideo('ic0PnF04N-Q')
```

[68]:



2.12 Ecuación de convección lineal 1D

Esta es la ecuación más sencilla con la que podremos aprender sobre dinámica de fluidos computacional, resulta interesante ya que es una ecuación pequeña y nos muestra mucho:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

La ecuación anterior manifiestaa la onda inicial con una velocidad c para su propagación. Contando con la condición inicial $u(x,0) = u_0(x)$. El resultado exacto de la ecuación es $u(x,t) = u_0(x - ct)$

La ecuación para resolverse en el ordenador debe discretizarse, usaremos el método de diferencias finitas donde para el tiempo será diferencias finitas adelantadas y para el caso del espacio serán diferencias finitas retrazadas. Para el espacio en el eje x para los puntos de $i = 0$ a N, el tiempo tiene un tamaño de paso Δt .

La ecuación para la coordenada x resulta:

$$\frac{\partial u}{\partial x} \approx \frac{u(x + \Delta x) - u(x)}{\Delta x}$$

Si la discretizamos es:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

Siendo n y $n + 1$ puntos consecutivos en el tiempo; $i - 1$ y i son aquellos puntos vecinos de la coordenada x discretizada. Si se brindan las condiciones iniciales, entonces la única incógnita en esta discretización queda ser u_i^{n+1}. Se logra resolver la incógnita y tener una ecuación que nos permita continuar en el tiempo, de la siguiente manera:

$$u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

Implementamos esto para el lenguaje Python y así mismo haremos nuestra GUI.

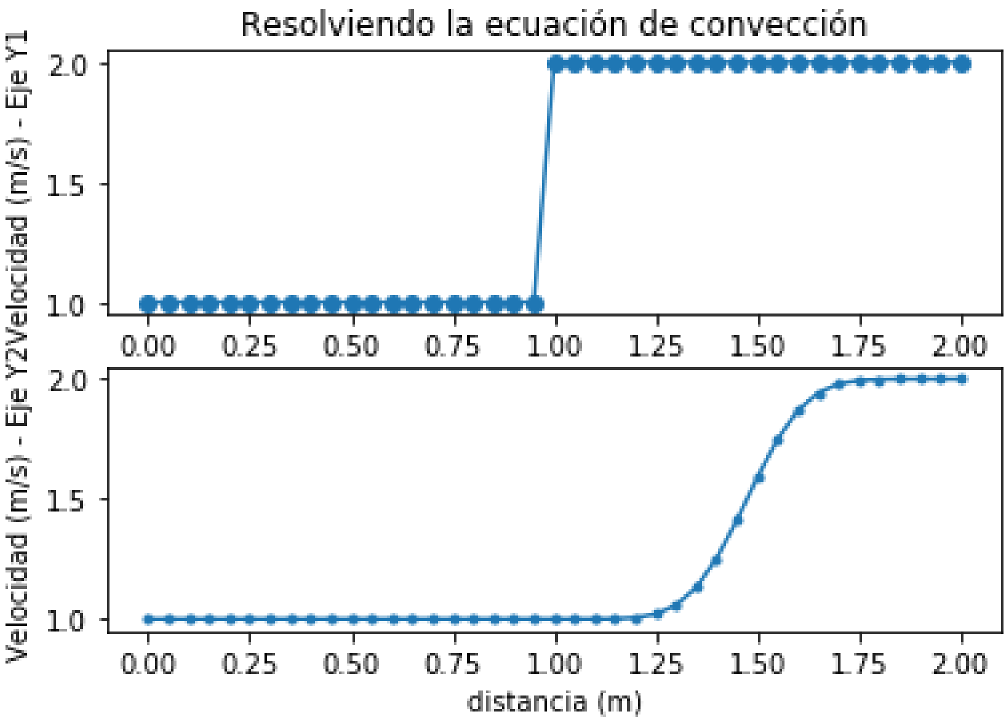
[69]:
```
# Importando librerías
import sys # Función de Python
import numpy as np # Numpy contraída como np
import matplotlib.pyplot as plt # Matplotlib contraída como plt
from IPython.core.display import clear_output
from PyQt5.QtWidgets import QMainWindow, QApplication, QAction, QFileDialog #␣
↪PyQt5 el binding que hace posible la GUI
from UI_N01_V01 import * # GUI convertida de Qt Designer
```

[70]:
```python
# Clase principal
class MyForm(QMainWindow): # Clase de la ventana principal
    def __init__(self): # Inicializamos el programa
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton_Paso01_LC_solver.clicked.connect(self.
 ↪dispsolver_Paso01_LC_01) # El botón es conectado a nuestro método que␣
 ↪resolverá la ecuación lineal de convección 1-D
        self.show() # Muestra resultados de la tarea ejecutada en la línea␣
 ↪anterior

    def dispsolver_Paso01_LC_01(self): # Método que usamos para resolver la␣
 ↪ecuación de convección lineal 1-D
        Paso01_LC_1D_nx = int(self.ui.lineEdit_Paso01_LC_01_nx.text()) # Input␣
 ↪para el número de puntos en el que se divide la distancia en el eje x
        Paso01_LC_1D_x = float(self.ui.lineEdit_Paso01_LC_02_x.text()) # Input␣
 ↪de la distancia en el eje x
        Paso01_LC_1D_dx = Paso01_LC_1D_x/(Paso01_LC_1D_nx-1)
        Paso01_LC_1D_nt = int(self.ui.lineEdit_Paso01_LC_03_nt.text()) # Número␣
 ↪de veces que el dt se repite
        Paso01_LC_1D_dt = float(self.ui.lineEdit_Paso01_LC_04_dt.text()) # dt␣
 ↪es el tamaño del paso del tiempo
        Paso01_LC_1D_c = float(self.ui.lineEdit_Paso01_LC_05_c.text()) #␣
 ↪velocidad de la onda , c = 1
        Paso01_LC_1D_ue = float(self.ui.lineEdit_Paso01_LC_06_u.text()) # Es la␣
 ↪velocidad inicial u_0 , u=2
        Paso01_LC_1D_x1 = float(self.ui.lineEdit_Paso01_LC_07_x1.text()) # es␣
 ↪la u inicial u_0 para el tramo izquierdo
        Paso01_LC_1D_x2 = float(self.ui.lineEdit_Paso01_LC_08_x2.text()) # Es␣
 ↪la u inicial u_0 para el tramo derecho

        # Barra de progreso
        x = 0
        while x < 100:
            x += 0.0001
            self.ui.progressBar_Paso01_LC_01.setValue(x)

        # Ecuación de Convección - Función sombrero
        print("Ecuación de convección - Función sombrero")
        u = np.ones(Paso01_LC_1D_nx) # Creamos una ecuación de puros unos con␣
 ↪la cantidad de elementos de nx
        u[int(Paso01_LC_1D_x1/Paso01_LC_1D_dx) : int(Paso01_LC_1D_x2/
 ↪Paso01_LC_1D_dx)] = Paso01_LC_1D_ue # Remplezamos el valor de u_0 en el␣
 ↪tramo de las cotas en el eje x para x1 y x2
```

```python
        # Dominio01:
        Paso01_LC_1D_x1 = np.linspace(0,Paso01_LC_1D_x, Paso01_LC_1D_nx)
        print("Dominio01 x1:")
        print(Paso01_LC_1D_x1)

        # Rango01:
        print("Rango01 u:")
        print(u)

        y1 = u
        print("Rango01 y1:")
        print(y1)

        # Gráfico
        plt.subplot(2,1,1)
        plt.plot(Paso01_LC_1D_x1, y1, 'o-')
        plt.title('Resolviendo la ecuación de convección')
        plt.ylabel('Velocidad (m/s) - Eje Y1')

        self.ui.label_Paso01_LC_response_01.setText("uini: " + str(y1))

        #Ecuación de Convección - Discretizada
        print("Ecuación de convección - Discretizada")
        un = np.ones(Paso01_LC_1D_nx)

        for n in range(Paso01_LC_1D_nt):
          un[:] = u[:]
          for i in range(1,Paso01_LC_1D_nx):
              u[i] = un[i]-Paso01_LC_1D_c*Paso01_LC_1D_dt/
↪Paso01_LC_1D_dx*(un[i]-un[i-1])

        # Dominio02
        print("Dominio02 x2:")
        Paso01_LC_1D_x2 = np.linspace(0,Paso01_LC_1D_x, Paso01_LC_1D_nx)
        print(Paso01_LC_1D_x2)

        # Rango02
        print("Rango02 u:")
        print(u)
        y2 = u
        print("Rango02 y2:")
        print(y2)

        plt.subplot(2,1,2)
        plt.plot(Paso01_LC_1D_x2, y2, '.-')
        plt.xlabel('distancia (m)')
        plt.ylabel('Velocidad (m/s) - Eje Y2')
```

```python
        plt.show()

        self.ui.label_Paso01_LC_response_02.setText("Ufin : " + str(y2))

if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    w.show()
    sys.exit(app.exec_())
```

```
Ecuación de convección - Función sombrero
Dominio01 x1:
[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.   1.05 1.1  1.15 1.2  1.25 1.3  1.35
 1.4  1.45 1.5  1.55 1.6  1.65 1.7  1.75 1.8  1.85 1.9  1.95 2.  ]
Rango01 u:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
Rango01 y1:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
Ecuación de convección - Discretizada
Dominio02 x2:
[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.   1.05 1.1  1.15 1.2  1.25 1.3  1.35
 1.4  1.45 1.5  1.55 1.6  1.65 1.7  1.75 1.8  1.85 1.9  1.95 2.  ]
Rango02 u:
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.00000095 1.00002003 1.00020123 1.00128841
 1.00590897 1.02069473 1.05765915 1.13158798 1.25172234 1.41190147
 1.58809853 1.74827766 1.86841202 1.94234085 1.97930527 1.99409103
 1.99871159 1.99979877 1.99997997 1.99999905 2.        ]
Rango02 y2:
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.00000095 1.00002003 1.00020123 1.00128841
 1.00590897 1.02069473 1.05765915 1.13158798 1.25172234 1.41190147
 1.58809853 1.74827766 1.86841202 1.94234085 1.97930527 1.99409103
 1.99871159 1.99979877 1.99997997 1.99999905 2.        ]
```




```
An exception has occurred, use %tb to see the full traceback.


SystemExit: 0


/home/jhongesell/Documentos/Informatica/Entornos_virtuales/Entornos_pip_venv/ent
orno01N01/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3334:
UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

2.12.1 Entendiendo lo ocurrido:

Teníamos que de nuestra ecuación linea de convección 1-D en su forma discretizada habíamos hecho la descomposición para la derivada parcial de u_i^{n+1} con respecto al tiempo ya que de esta manera hacemos diferencias finitas adelantadas, quedando:

$$u_i^{n+1} = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

Por ello necesitamos como variables de entrada a:

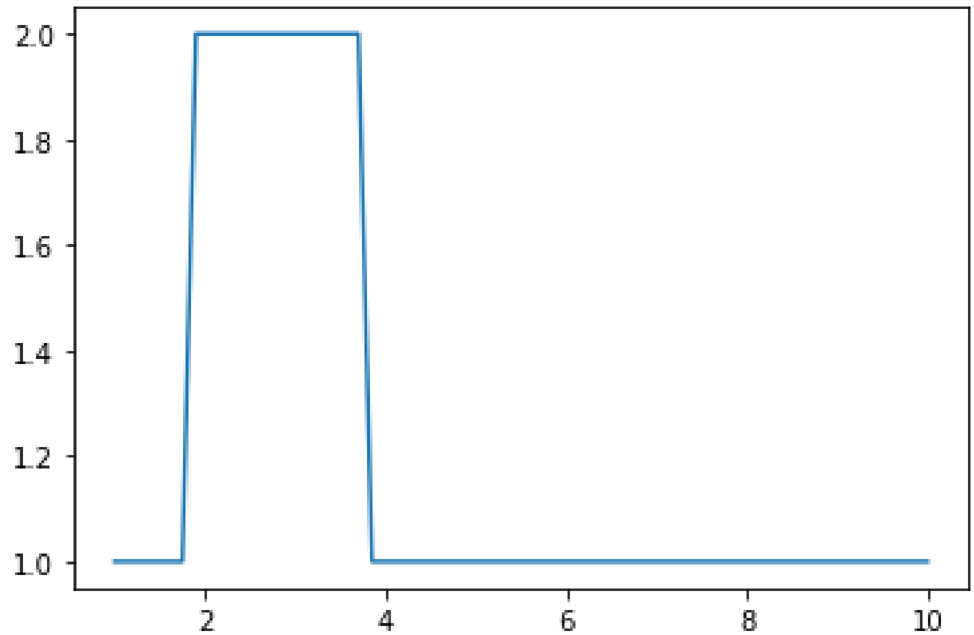
```python
# Librerías
import numpy as np # Numpy contraída como np
import matplotlib.pyplot as plt # Matplotlib contraída como plt
```

Creamos una red de puntos uniforme con lo cual se llega a definir el dominio para 4 unidades de longitud de ancho x = [0,4], nx es el número de puntos para la malla, dx es el espacio entre cada punto de la malla.

[71]:
```python
x = 10. # Longitud en el eje X
nx = 61 # Número de divisiones que se le hace a x
dx = x/(nx-1)
nt = 300 # Tiempo t total
dt = 0.010 # Paso del tiempo t
c = 1. # Velocidad de la onda
ue = 2
x1 = 1
x2 = 3

# Condiciones iniciales
u = np.ones(nx)      # numpy función ones() creando un array de 1 con nx␣
 ↪elmentos
u[int(x1/dx) : int(x2/dx+1)]=ue
print("Esto es u:")
print(u)
print("Esto es nuestra función sombrero:")
plt.plot(np.linspace(1,x,nx), u) # Es esquema es para hacer una gráfica simple␣
 ↪plot(X,Y)
```

```
Esto es u:
[1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Esto es nuestra función sombrero:
```

[71]:
```
[<matplotlib.lines.Line2D at 0x7f05ebfe8be0>]
```

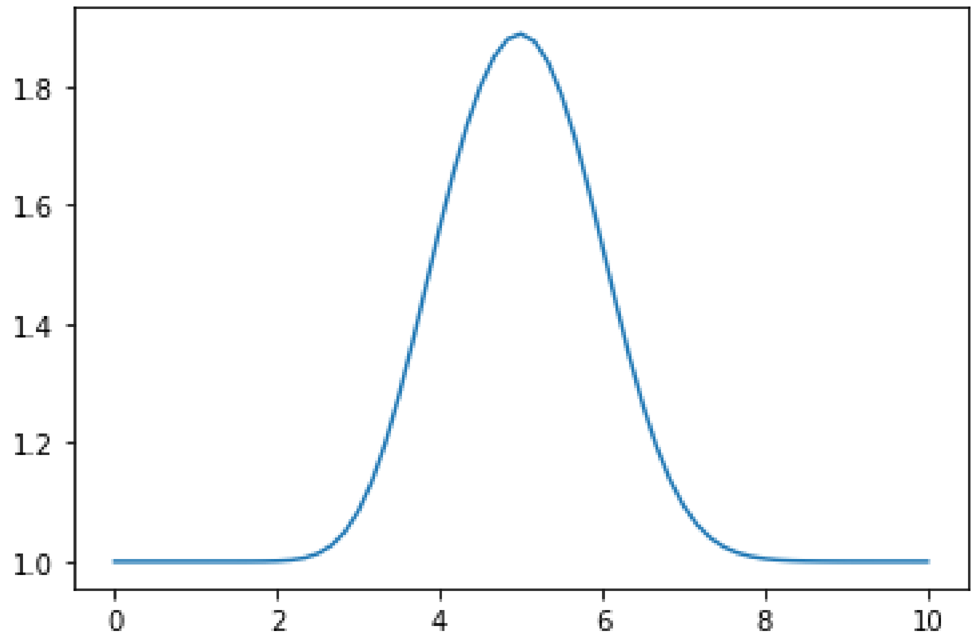


La forma de la función sombrero es definida por lineas rectas ya que esta asumiendo valores exactos que hemos asignado al vector u, en un principio habíamos dicho que u sería un arreglo conformado por *unos* con la misma cantidad de de elementos que *nx*, luego según las condiciones iniciales definimos a ue al cual le dimos un valor mayor que sería reemplezado en *u* para las cotas x1 y x2, derecha e izquierda respectivamente, lo que estamos visualizando ocurre para el instante $t = 0$ s.

Ahora vamos a implementar la ecuación de convección ya discretizada, esto lo logramos haciendo un *for* que irá desde el *1* hasta el valor de *nx*.

[72]:
```
un = np.ones(nx)

for n in range(nt):
    un[:] = u[:]
    for i in range(1,nx):
        u[i] = un[i]-c*(dt/dx)*(un[i]-un[i-1])
plt.plot(np.linspace(0,x,nx),u)
```

[72]: [<matplotlib.lines.Line2D at 0x7f05ebf47898>]



2.13 Ecuación de convección no-lineal 1D

Es la siguiente ecuación de convección representada en su forma no-lineal:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

El valor de la velocidad de propagación de la onda c ya no se encuentra disponible, en su lugar se presenta a u quien reemplaza a la constante anterior hablada, ya hemos visto antes como se hace la discretización por lo tanto para este caso nos queda:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_i^n \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0$$

Recordemos que nuestro interés es obtener el valor de u_i^{n+1} , por ello si despejamos queda:

$$u_i^{n+1} = u_i^n - u_i^n \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n)$$

Para nuestro programa GUI sería lo siguiente:

```
# Importando librerías
import sys
```

```python
import numpy as np
import matplotlib.pyplot as plt
from IPython.core.display import clear_output
from PyQt5.QtWidgets import QMainWindow, QApplication, QAction, QFileDialog
from UI_N01_V02 import *

# Clase principal
class MyForm(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton_Paso01_LC_solver.clicked.connect(self.dispsolver_Paso01_LC)
        self.ui.pushButton_Paso01_NLC_solver.clicked.connect(self.dispsolver_Paso01_NCL)
        self.show()

    def dispsolver_Paso01_LC(self):
        Paso01_LC_1D_nx = int(self.ui.lineEdit_Paso01_LC_01_nx.text())
        Paso01_LC_1D_x = float(self.ui.lineEdit_Paso01_LC_02_x.text())
        Paso01_LC_1D_dx = Paso01_LC_1D_x/(Paso01_LC_1D_nx-1)
        Paso01_LC_1D_nt = int(self.ui.lineEdit_Paso01_LC_03_nt.text())
        Paso01_LC_1D_dt = float(self.ui.lineEdit_Paso01_LC_04_dt.text())
        Paso01_LC_1D_c = float(self.ui.lineEdit_Paso01_LC_05_c.text())
        Paso01_LC_1D_ue = float(self.ui.lineEdit_Paso01_LC_06_u.text())
        Paso01_LC_1D_x1 = float(self.ui.lineEdit_Paso01_LC_07_x1.text())
        Paso01_LC_1D_x2 = float(self.ui.lineEdit_Paso01_LC_08_x2.text())

        # Barra de progreso
        x = 0
        while x < 100:
            x += 0.0001
            self.ui.progressBar_Paso01_LC_01.setValue(x)

        # Ecuación de Convección - Función sombrero
        print("Ecuación de convección - Función sombrero")
        u = np.ones(Paso01_LC_1D_nx)
        u[int(Paso01_LC_1D_x1/Paso01_LC_1D_dx) : int(Paso01_LC_1D_x2/Paso01_LC_1D_dx+1)] = Paso

        # Dominio01:
        Paso01_LC_1D_x1 = np.linspace(0,Paso01_LC_1D_x, Paso01_LC_1D_nx)
        print("Dominio01 x1:")
        print(Paso01_LC_1D_x1)

        # Rango01:
        print("Rango01 u:")
        print(u)

        y1 = u
```

```
    print("Rango01 y1:")
    print(y1)

    # Gráfico
    plt.subplot(2,1,1)
    plt.plot(Paso01_LC_1D_x1, y1, 'o-')
    plt.title('Resolviendo la ecuación de convección')
    plt.ylabel('Velocidad (m/s) - Eje Y1')

    self.ui.label_Paso01_LC_response_01.setText("uini: " + str(y1))

    #Ecuación de Convección - Discretizada
    print("Ecuación de convección - Discretizada")
    un = np.ones(Paso01_LC_1D_nx)

    for n in range(Paso01_LC_1D_nt):
      un[:] = u[:]
      for i in range(1,Paso01_LC_1D_nx):
          u[i] = un[i]-Paso01_LC_1D_c*Paso01_LC_1D_dt/Paso01_LC_1D_dx*(un[i]-un[i-1])

    # Dominio02
    print("Dominio02 x2:")
    Paso01_LC_1D_x2 = np.linspace(0,Paso01_LC_1D_x, Paso01_LC_1D_nx)
    print(Paso01_LC_1D_x2)

    # Rango02
    print("Rango02 u:")
    print(u)
    y2 = u
    print("Rango02 y2:")
    print(y2)

    plt.subplot(2,1,2)
    plt.plot(Paso01_LC_1D_x2, y2, '.-')
    plt.xlabel('distancia (m)')
    plt.ylabel('Velocidad (m/s) - Eje Y2')

    plt.show()

    self.ui.label_Paso01_LC_response_02.setText("Ufin : " + str(y2))
def dispsolver_Paso01_NCL(self):
    print("Hello")
    Paso01_NLC_1D_nx = int(self.ui.lineEdit_Paso01_NLC_01_nx.text())
    Paso01_NLC_1D_x = float(self.ui.lineEdit_Paso01_NLC_02_x.text())
    Paso01_NLC_1D_dx = Paso01_NLC_1D_x/(Paso01_NLC_1D_nx-1)
    Paso01_NLC_1D_nt = int(self.ui.lineEdit_Paso01_NLC_03_nt.text())
    Paso01_NLC_1D_dt = float(self.ui.lineEdit_Paso01_NLC_04_dt.text())
    Paso01_NLC_1D_ue = float(self.ui.lineEdit_Paso01_NLC_06_ue.text())
```

```
Paso01_NLC_1D_x1 = float(self.ui.lineEdit_Paso01_NLC_07_x1.text())
Paso01_NLC_1D_x2 = float(self.ui.lineEdit_Paso01_NLC_08_x2.text())

# Barra de progreso
x = 0
while x < 100:
    x += 0.0001
    self.ui.progressBar_Paso01_NLC_01.setValue(x)

# Ecuación de COnvección - Función sombrero
print("Ecuación de convección - Función sombrero")
u = np.ones(Paso01_NLC_1D_nx)
u[int(Paso01_NLC_1D_x1/Paso01_NLC_1D_dx) : int(Paso01_NLC_1D_x2/Paso01_NLC_1D_dx+1)] =

# Dominio01:
Paso01_NLC_1D_x1 = np.linspace(0,Paso01_NLC_1D_x, Paso01_NLC_1D_nx)
print("Dominio x1:")
print(Paso01_NLC_1D_x1)

# Rango01:
print("Rango01 u:")
print(u)

y1 = u
print("Rango01 y1:")
print(y1)

# Gráfico
plt.subplot(2,1,1)
plt.plot(Paso01_NLC_1D_x1, y1, 'o-')
plt.title('Resolviendo la ecuación de convección')
plt.ylabel('Altura - Eje Y1')

self.ui.label_Paso01_NLC_response_01.setText("uini: " + str(y1))

# Ecuación de convección - Discretizada
print("Ecuación de convección - Discretizada")
un = np.ones(Paso01_NLC_1D_nx)

for n in range(Paso01_NLC_1D_nt):
    un[:] = u[:]
    for i in range(1,Paso01_NLC_1D_nx):
        u[i] = un[i]-un[i]*Paso01_NLC_1D_dt/Paso01_NLC_1D_dx*(un[i]-un[i-1])

# Dominio02
print("Dominio02 x2:")
Paso01_NLC_1D_x2 = np.linspace(0,Paso01_NLC_1D_x, Paso01_NLC_1D_nx)
print(Paso01_NLC_1D_x2)
```

```
        # Rango02
        print("Rango02 u:")
        print(u)
        y2 = u
        print("Rango02 y2:")
        print(y2)

        plt.subplot(2,1,2)
        plt.plot(Paso01_NLC_1D_x2, y2, '.-')

        plt.show()

        self.ui.label_Paso01_NLC_response_02.setText("Ufin : " + str(y2))


if __name__=="__main__":
    app = QApplication(sys.argv)
    w = MyForm()
    w.show()
    sys.exit(app.exec_())
```

Pruebe reemplazando valores distintos, corra el script de preferencia desde su terminal llamando al interprete de Python.

Acabamos de ver como se puede ver nuestro software GUI que resuelve la ecuació-1D linea y no lineal; veamos este último sin la GUI:

[ ]:
```
# Librerías
import numpy as np # Numpy contraída como np
import matplotlib.pyplot as plt # Matplotlib contraída como plt

x = 10. # Longitud en el eje X
nx = 61 # Número de divisiones que se le hace a x
dx = x/(nx-1)
nt = 300 # Tiempo t total
dt = 0.010 # Paso del tiempo t
ue = 2
x1 = 1
x2 = 3

# Condiciones iniciales
u = np.ones(nx)       # numpy función ones() creando un array de 1 con nx␣
 ↪elmentos
u[int(x1/dx) : int(x2/dx+1)]=ue
print("Esto es u:")
print(u)
print("Esto es nuestra función sombrero:")
```

```
plt.plot(np.linspace(1,x,nx), u) # Es esquema es para hacer una gráfica simple␣
↪plot(X,Y)

un = np.ones(nx)

for n in range(nt):
    un[:] = u[:]
    for i in range(1,nx):
        u[i] = un[i]-un[i]*(dt/dx)*(un[i]-un[i-1])
plt.plot(np.linspace(0,x,nx),u)
```

2.13.1 Condición Courant-Friedrichs-Lewys (CFL)

Se utiliza como una restricción para que exista una convergencia de las ecuaciones diferenciales parciales, es diferente a la estabilidad numérica.
El CFL hace que el paso del tiempo deba ser menos a un valor ya que si no, la solución falla.