UNIVERSIDAD
DE SANTIAGO
DE CHILE

INFORME DE TALLER 1; ECUACIONES DIFERENCIALES Y MÉTODOS NUMÉRICOS FORMA A

Nombre: John Serrano Carrasco

Sección: 13307-0-A-1

Fecha: 16 de Diciembre de 2021

Profesor de Teoría: Jarnishs Beltran

Profesor de Laboratorio: Eduardo Díaz

1. Introducción:

El presente informe tiene como propósito dar respuestas a los problemas y preguntas planteados en el Taller N°1 del curso de Ecuaciones Diferenciales y Métodos Numéricos, el cual se realiza con el apoyo de las herramientas Python y Jupyter Notebook (Anaconda).

Python es un lenguaje de programación, el cual permite resolver una gran cantidad de problemas no solo matemáticos si no que, de toda índole, a través de un conjunto de instrucciones dadas por el usuario.

Jupyter Notebook es un interfaz de web que trabaja con Python y permite realizar un conjunto de programas que se relacionan entre sí.

Con estas dos herramientas, junto con los conceptos aprendidos en la Unidad 1 del curso, se pueden responder los dos problemas dados.

2. DESARROLLO

2.1 Problema 1:

"La EDO autónoma

$$m\frac{dy}{dt} = mg - ky^2,$$

modela la velocidad y(t) de un cuerpo de masa m (en kg) que está cayendo bajo la influencia de la gravedad. Debido a que el término –ky² representa la resistencia del aire, la velocidad de un cuerpo que cae de una gran altura no aumenta sin límite conforme pasa el tiempo t.

Suponga que el cuerpo tiene masa 2 kg y k = g donde g = 9, 8 m/s²"

Considerando los datos anteriores y trabajando con la ecuación:

Primero, corresponde hacer el despeje de $\frac{dy}{dx}$:

$$m\frac{dy}{dt} = mg - ky^2$$

$$\frac{dy}{dt} = \frac{mg - ky^2}{m}$$

Reemplazando los datos: m = 2, k = g = 9,8

$$\frac{dy}{dt} = 9,8 - 4,9y^2$$

Igualando a 0 para encontrar los puntos de equilibrio:

$$0 = 9,8 - 4,9y^2$$

$$4,9y^2 = 9,8$$

$$y^2 = \frac{9,8}{4,9}$$

$$y = \sqrt{2}$$

$$y = -\sqrt{2}$$

Con esto, ahora es posible resolver las preguntas a continuación.

2.1.1 Pregunta A: Construya un campo de direcciones para la EDO autónoma.

Utilizando NumPy y Matplotlib, se puede construir un código para construir un Campo de direcciones para la EDO dada, considerando los puntos de equilibrio obtenidos anteriormente.

El código es el siguiente:

```
# Se importa Matplotlib.pyplot como plt para trabajar con graficos
import matplotlib.pyplot as plt
# Se importa numpy como np para trabajar con arreglos
import numpy as np
```

```
# Reemplazamos los valores del enunciado para trabajar con la formula dada
# dy/dt = (mg - k^y)/ m
def f(X,Y): return 9.8-4.9*(Y**2) # Definimos la funcion con la que trabajaremos
X,Y=np.meshgrid(np.linspace(-2,2,10), np.linspace(-2,2,10)) # Utilizamos meshgrid para obtener una matriz de puntos
U = 1.0  # Se define la direccion horizontal de las flechas
V = f(X,Y) # Se define la direccion vertical de las flechas
N = np.sqrt(U**2+V**2) # Se prepara para normalizar los vectores U y V
U = U/N # Se normaliza el vector U
V = V/N # Se normaliza el vector N
plt.quiver(X, Y, U, V)  # Se realiza el grafico del campo de direccion
plt.xlabel('t') # Se define el eje X como "t"
plt.ylabel('y') # Se fine el eje Y como "y"
plt.grid() # Se agregan una red al fondo del grafico
```

Figura N°1: Código realizado en Jupyter Notebook para realizar un campo de direcciones para el problema 1.

El código anterior da como resultado el siguiente campo de direcciones:

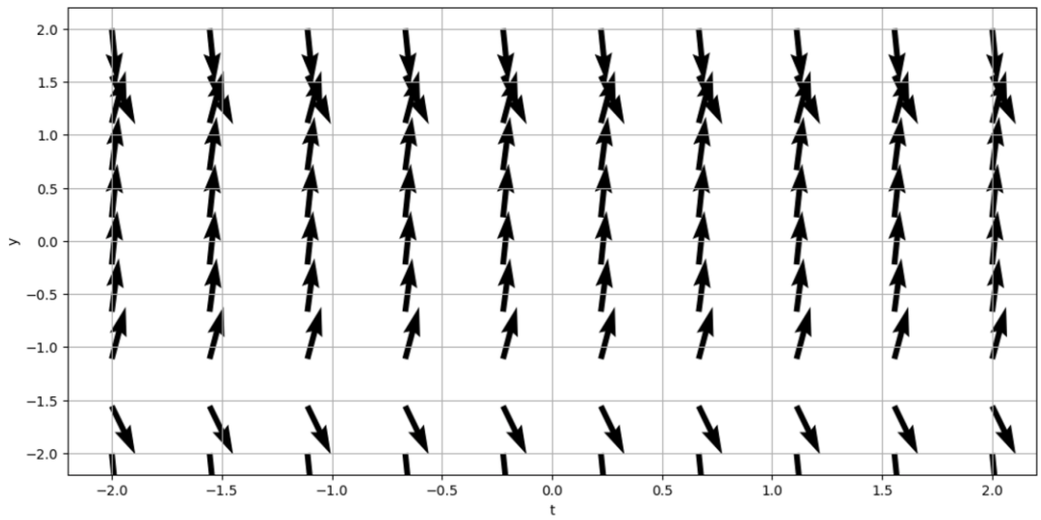

Figura N°2: Campo de direcciones obtenido para el problema 1

Sin embargo, se puede notar de que el campo no permite ver de manera clara el comportamiento de las flechas en los puntos de equilibrio, por lo que debemos hacer un “zoom” en ambas partes del gráfico. Para ello, se modifica el segundo linspace, para el caso de $\sqrt{2}$, utilizando los puntos 1.2 y 1.6 en el eje Y y para el caso de -$\sqrt{2}$, utilizando los puntos -1.6 y -1.2 en el eje Y.

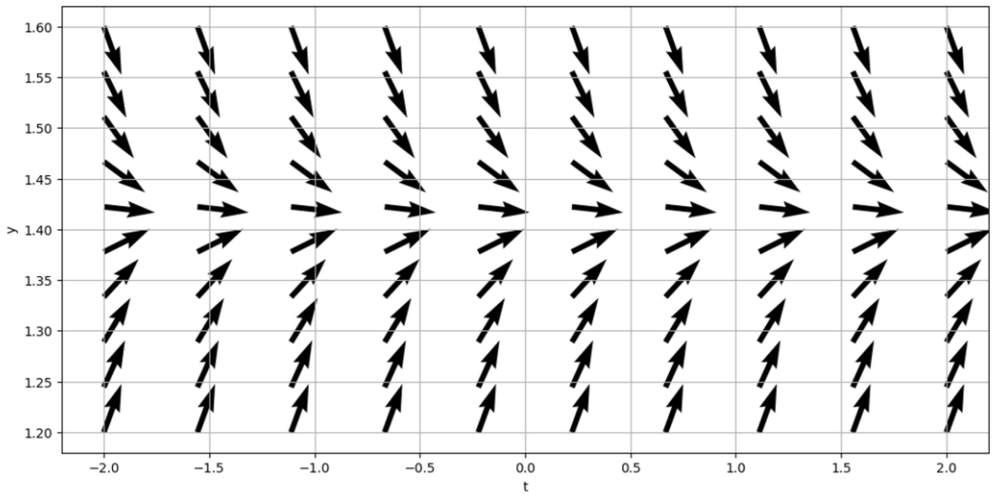

Figura N°3: Campo de direcciones claro para y = $\sqrt{2}$

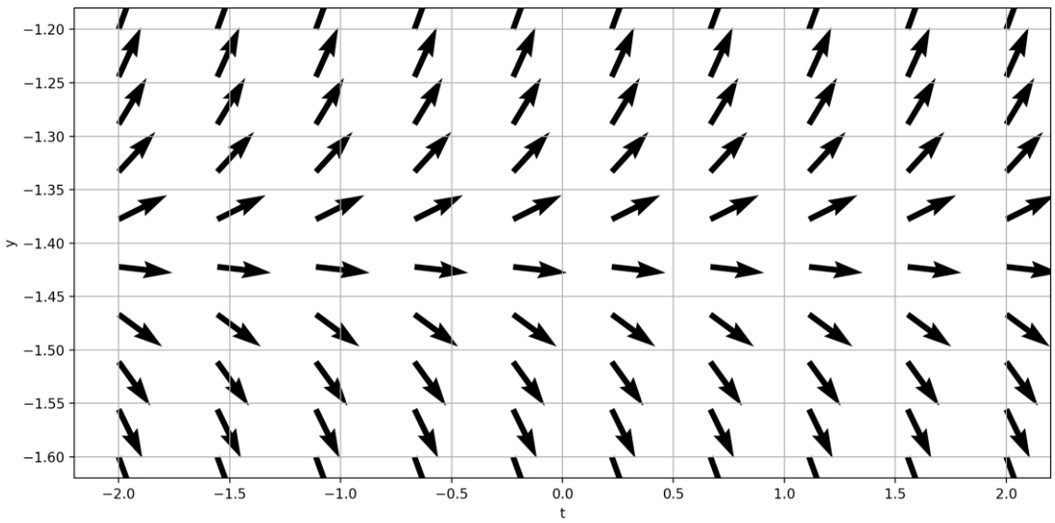

Figura N°4: Campo de direcciones claro para y = -$\sqrt{2}$

Con esto, tenemos los comportamientos claros para las rectas de Y cercanas a $\sqrt{2}$ y -$\sqrt{2}$.

2.1.2 Pregunta B: Determine y clasifique los puntos de equilibrio en atractor, repulsor o nodo

Ya sabemos que los puntos de equilibrios son $\sqrt{2}$ y -$\sqrt{2}$. Considerando los campos de direcciones obtenidos, podemos clasificarlos de la siguiente manera:

$y = \sqrt{2}$: **Atractor o sumidero**, pues cualquier solución cercana a $y = \sqrt{2}$ tiende a $\sqrt{2}$ a medida que t aumenta.

$y = -\sqrt{2}$: **Fuente o repulsor**, pues cualquier solución cercana a $y = -\sqrt{2}$ tiende a -$\sqrt{2}$ a medida que t decrece.

2.1.3 Pregunta C: Utilice la información anterior para encontrar la velocidad límite conforme t → ∞.

Como tenemos tres intervalos de y distintos, debemos considerar que la velocidad límite conforme t → ∞ puede ser distinta en esos tres intervalos. Considerando esto, las gráficas realizadas en la parte A y las clasificaciones de la parte B:

$\sqrt{2}$ < y < ∞: La velocidad límite conforme t → ∞ es **$\sqrt{2}$,** pues las flechas cada vez se atraen hacia y = $\sqrt{2}$ a medida que t aumenta.

-$\sqrt{2}$ < y < $\sqrt{2}$: La velocidad límite conforme t → ∞ es **$\sqrt{2}$,** pues las flechas cada vez se atraen hacia y = $\sqrt{2}$ a medida que t aumenta.

-∞ < y < -$\sqrt{2}$: La velocidad límite conforme t → ∞ es **-∞,** pues las flechas cada vez se van hacia y = - ∞ a medida que t aumenta.

2.2 Problema 2:

"Considere el PVI dado por:

$$y' - \frac{y}{x} = 1 \quad ; \quad y(1) = 2.$$

Trabajando con la expresión, queda de la siguiente forma:

$$\frac{dy}{dx} = 1 + \frac{y}{x}$$

2.2.1 Pregunta A: Determine una aproximación de y(10) utilizando el método de Euler mejorado considerando h = 0, 2.

Primero, se utiliza el código para el Método de Euler Mejorado desarrollado en clases:

```
# Se importa Matplotlib.pyplot como plt para trabajar con graficos
import matplotlib.pyplot as plt
# Se importa numpy como np para trabajar con arreglos
import numpy as np
```

```
# Primero, debemos definir el Metodo de Euler (o Metodo de Runge Kutta de Orden 2) para poder trabajar con este.
# Se utiliza el codigo desarrollado en clases, el cual traduce el algoritmo visto en clases.
def RK2(f, x0, xn, y0, n):
    X=np.linspace(x0, xn, n+1)
    Y=np.linspace(x0, xn, n+1)
    Y[0]=y0
    h=(xn - x0)/n

    for i in range(n):
        Y[i+1] = Y[i]+(h/2)*(f(X[i],Y[i])+f(X[i+1], Y[i]+h*f(X[i], Y[i])))

    salida = dict() # Se crea un diccionario, esto hara que sea mas facil graficar la informacion obtenida.
    salida['x'] = X
    salida['y'] = Y

    return salida
```

Fígura N°5: Código desarrollado en clases para el Método de Euler Mejorado.

Sabemos que f corresponde a la función dada por enunciado, x0 es el x dada por el PVI (1), xn es el x final (10), y0 es el y dado por el PVI (2), pero obtener n. Para obtener n, solo basta con despejarlo utilizando:

$$h = \frac{Xn - X_0}{n}$$

Reemplazando los datos y despejando n, obtenemos que n es 45. Por lo tanto, ahora podemos llamar a la función con los datos para obtener aproximaciones hasta y(10):

```
def f_1(X,Y): return (1 + (Y/X)) # Se define la funcion del enunciado
RK2(f_1, 1, 10, 2, 45) # Se llama al metodo de Euler Mejorado.
# Notar que: x0 = 1 ; xn = 10 ; y0 = 2; n = 45 (n = 45 debido a h= 0.2, pues se despeja n de h = (xn-x0)/n)
```

Figura N°6: Código donde se llama al Método de Euler Mejorado

El llamado de la función da como resultado el siguiente diccionario, el cual será esencial para poder graficar esta información posteriormente.

```
{'x': array([ 1. ,  1.2,  1.4,  1.6,  1.8,  2. ,  2.2,  2.4,  2.6,  2.8,  3. ,
         3.2,  3.4,  3.6,  3.8,  4. ,  4.2,  4.4,  4.6,  4.8,  5. ,  5.2,
         5.4,  5.6,  5.8,  6. ,  6.2,  6.4,  6.6,  6.8,  7. ,  7.2,  7.4,
         7.6,  7.8,  8. ,  8.2,  8.4,  8.6,  8.8,  9. ,  9.2,  9.4,  9.6,
         9.8, 10. ]),
 'y': array([ 2.        ,  2.61666667,  3.26706349,  3.94628685,  4.65068382,
         5.37742646,  6.12426002,  6.88934426,  7.67114859,  8.4683798 ,
         9.27993074, 10.10484279, 10.94227781, 11.79149677, 12.65184308,
        13.52272956, 14.40362794, 15.29406044, 16.19359283, 17.10182876,
        18.01840495, 18.94298731, 19.87526744, 20.81495989, 21.76179959,
        22.7155398 , 23.67595027, 24.6428156 , 25.61593389, 26.59511549,
        27.58018191, 28.57096489, 29.5673055 , 30.56905345, 31.57606633,
        32.58820905, 33.6053533 , 34.62737702, 35.65416396, 36.68560329,
        37.72158922, 38.76202067, 39.80680096, 40.8558375 , 41.9090416 ,
        42.96632816])}
```

Figura N°7: Diccionario resultante del llamado al Método de Euler Mejorado. El ultimo número de 'y' es y(10).

Por lo tanto, con esto se tiene que la aproximación de y(10) para la EDO autónoma utilizando el Método de Euler Mejorado es **42,966328162513..**

2.2.2 Pregunta B: Determine una aproximación de y(10) utilizando el método de Runge Kutta de orden 4 considerando h = 0, 5.

Primero, se utiliza el código para el Método Runge Kutta de Orden 4 desarrollado en clases:

```
# Primero, debemos definir el Metodo de Runge Kutta de Orden 4 para poder trabajar con este.
# Se utiliza el codigo desarrollado en clases, el cual traduce el algoritmo visto en clases.
def RK4(f, x0, xn, y0, n):
    X=np.linspace(x0, xn, n+1)
    Y=np.linspace(x0, xn, n+1)
    Y[0]=y0
    h=(xn - x0)/n

    for i in range(n):
        K1 = f(X[i],Y[i])
        K2 = f(X[i]+ (h/2),Y[i]+(h/2)*K1)
        K3 = f(X[i]+(h/2), Y[i]+(h/2)*K2)
        K4 = f(X[i]+h, Y[i]+h*K3)
        Y[i+1]=Y[i]+(h/6)*(K1+2*K2+2*K3+K4)

    ret = dict() # Se crea un diccionario. Esto hara que sea mas facil graficar la informacion obtenida.
    ret['x'] = X
    ret['y'] = Y

    return ret
```

Figura N°8: Código para el Método de Runge Kutta de Orden 4 desarrollado en clases.

Nuevamente se debe despejar n. Esta vez, considerando que h = 0,5 se tiene que n es igual a 18. Además, como ya está definida la función solo queda llamar al método

```
RK4(f_1, 1, 10, 2, 18) # Se llama al Metodo de Runge Kutta de Orden 4
# Notar que: x0 = 1 ; xn = 10 ; y0 = 2; n = 18 (n = 18 debido a h= 0.5, pues se despeja n de h = (xn-x0)/n)
```

Figura N°9: Código donde se llama al Método de Runge Kutta de Orden 4

El llamado de la función da como resultado el siguiente diccionario, el cual será esencial para poder graficar esta información posteriormente.

```
{'x': array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,
         6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. ]),
 'y': array([ 2.        ,  3.60777778,  5.38562547,  7.28985077,  9.29476766,
        11.38341378, 13.54373612, 15.76672364, 18.04538238, 20.37412525,
        22.74838577, 25.16436151, 27.61883693, 30.10905716, 32.63263552,
        35.1874843 , 37.77176187, 40.38383154, 43.02222919])}
```

Figura N°10: Diccionario resultante del llamado al Método de Runge Kutta de Orden 4. El ultimo elemento de 'y' corresponde a y(10).

Por lo tanto, con esto se tiene que la aproximación de y(10) para la EDO autónoma utilizando el Método de Runge Kutta de Orden 4 es **43,02222919019276..**

2.2.3 Pregunta C: Encuentre la solución exacta explicita del PVI.

Para resolver el PVI, podemos utilizar el Método de Variable Separables:

$$\frac{dy}{dx} = 1 + \frac{y}{x}$$

Utilizamos el cambio de variable:

$$z = \frac{y}{x}$$

Derivamos para obtener el equivalente a $\frac{dy}{dx}$:

$$\frac{dy}{dx} = x\frac{dz}{dx} + z$$

Ahora la ecuación queda de la siguiente manera:

$$x\frac{dz}{dx} + z = 1 + z$$

Dejando solo un tipo de variable en cada lado:

$$dz = \frac{dx}{x}$$

Integrando en ambos lados:

$$\int dz = \int \frac{dx}{x}$$

$$z = \ln|x| + C$$

Volviendo a las variables originales:

$$\frac{y}{x} = \ln|x| + C$$
$$y = x(\ln|x| + C)$$

Ahora, debemos obtener la constante C, utilizando la información dada por el PVI (y(1) = 2). Reemplazando:

$$2 = 1(\ln|1| + C)$$

Considerando que ln|1| = 0:

$$2 = C$$

Por lo tanto, la solución exacta explicita de la EDO considerando la información dada por el PVI es:

$$y = x(\ln|x| + 2)$$

2.2.4 Pregunta D: Grafique en un mismo plano, la solución explicita y cada una de las aproximaciones encontradas en (a) y (b). Establezca a partir de este, cual de las aproximaciones tiene un error global más pequeño.

Gracias a los diccionarios obtenidos en las preguntas A y B, se puede fácilmente graficar las aproximaciones. Para ello, se desarrolla el siguiente código:

```
eulerM = RK2(f_1, 1, 10, 2, 45) # Se llama al Metodo de Euler Mejorado
X1 = eulerM['x'] # Se guarda el arreglo X
Y1 = eulerM['y'] # Se guarda el arreglo Y
plt.plot(X1,Y1, label = "Euler Mejorado") # Se grafica el Metodo de Euler Mejorado

runge = RK4(f_1, 1, 10, 2, 18)  # Se llama al Metodo de Runge Kutta de Orden 4
X2 = runge['x'] # Se guarda el arreglo X
Y2 = runge ['y'] # Se guarda el arreglo Y
plt.plot(X2,Y2, label = "Runge Kutta 4") # Se grafica el Metodo de Runge Kutta de Orden 4

x = np.linspace (1,10,100) # Se crea un arreglo de 1 hasta 10 con 100 numeros
y = x * (np.log(x) +2) # Se define la solucion exacta

plt.plot(x,y, label = "Sol. Exacta") # Se grafica la solucion exacta

plt.legend() # Se aplican las labels para identificar las graficas realizadas
plt.grid() # Se agrega una malla de cuadros al fondo del grafico
```

Figura N°11: Código para graficar la solución exacta y las aproximaciones

Del cual se obtiene el siguiente gráfico:

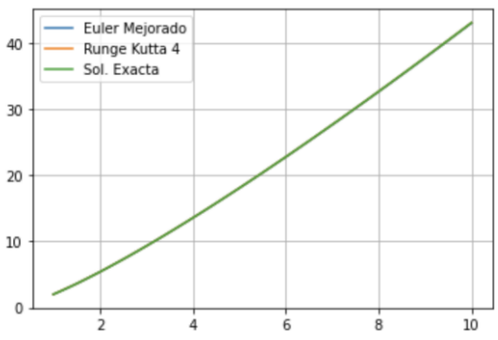

Figura N°12: Gráfico de la solución exacta y las aproximaciones

Nótese que el gráfico anterior no permite ver con claridad el error de los métodos utilizados con la solución exacta. Por lo tanto, es necesario hacer un zoom, con el siguiente código:

```
plt.plot(X1,Y1, label = "Euler Mejorado") # Se grafica el Metodo de Euler Mejorado
plt.plot(X2,Y2, label = "Runge Kutta 4") # Se grafica el Metodo de Runge Kutta de Orden 4
plt.plot(x,y, label = "Sol. Exacta") # Se grafica la solucion exacta

plt.axis([9.9, 10.1, 42.8, 43]) # Se realiza un "zoom" en el grafico anterior
plt.legend() # Se aplican las labels para identificar las graficas realizadas
plt.grid() # Se agrega una malla de cuadros al fondo del grafico
```

Figura N°13: Código para realizar un zoom cercano a y= 10 en el gráfico anterior

Con esto ahora se obtiene un gráfico mucho mas claro de visualizar:

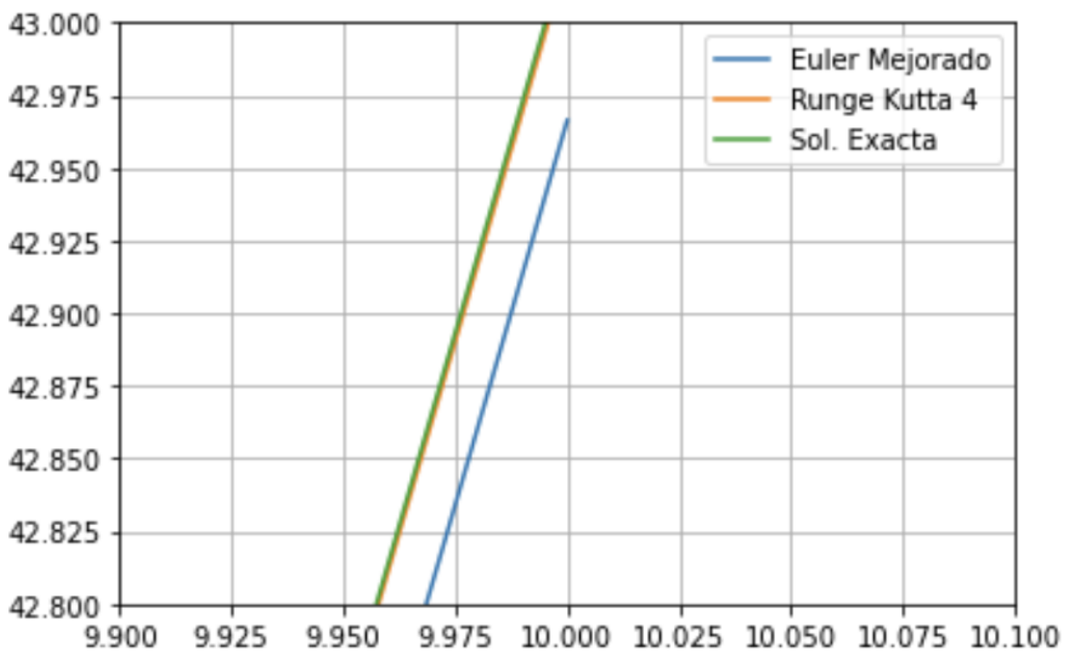

Figura N°14: Gráfico de la solución exacta y aproximaciones, con y = 10 más claro

Con este zoom realizado, claramente se puede ver que el Método que tiene menos error global es el **Metodo de Runge Kutta de Orden 4.** Esto es debido a que sus valores están muchos más cercanos a los valores reales de la solución exacta, comparado con el Método de Euler Mejorado cuyos valores están más alejados.

2.2.5 Pregunta E: Calcule el error absoluto cometido en cada una de las aproximaciones y concluya cual de ellas es una mejor aproximación de y(10)

Para resolver esto, debemos recordar que el error absoluto es:

$$Error = |Y(x) - y(x)|$$

Con Y(x) siendo la solución real e y(x) siendo la solución aproximada.

Por lo tanto, con el siguiente código se ahorra el proceso:

```
print("Considerando que: ")
print("El valor real de y(10) es: ",(10 * (np.log(10) +2)))
print("La aproximacion de y(10) con el Metodo de Euler Mejorado es: ", Y1[-1])
print("La aproximacion de y(10) con el Metodo de Runge Kutta de Orden 4 es: ", Y2[-1])
print("\n")

error_eulerM = np.abs(Y1[-1] - (10 * (np.log(10) +2))) # Se calcula el error absoluto del Metodo de Euler Mejorado
print("El error absoluto del Metodo de Euler Mejorado es: ", error_eulerM)
error_RK4 = np.abs(Y2[-1] - (10 * (np.log(10) +2))) # Se calcula el error absoluto del Metodo de Runge Kutta de Orden 4
print("El error absoluto del Metodo de Runge Kutta de Orden 4 es: ", error_RK4)
```

Figura N°15: Código para obtener los errores absolutos

Del código y de las preguntas anteriores se obtiene lo siguiente:

Considerando que:
El valor real de y(10) es: 43.02585092994046
La aproximacion de y(10) con el Metodo de Euler Mejorado es:
42.966328162513
La aproximacion de y(10) con el Metodo de Runge Kutta de Orden 4 es:
43.02222919019276

El error absoluto del Metodo de Euler Mejorado es: 0.059522767427459655
El error absoluto del Metodo de Runge Kutta de Orden 4 es:
0.0036217397477003033

Por lo tanto, se puede concluir que la mejor aproximación para y(10) corresponde al Método de Runge Kutta de Orden 4 ya que su error es bastante pequeño y por lo tanto, la aproximación es bastante cercana al valor real.

3. CONCLUSIÓN

En conclusión, se logró resolver los problemas del taller, utilizando los conceptos y algoritmos aprendidos en clase, junto a las herramientas de Python y Jupyter Notebook.

Se cumplió el objetivo sin complicaciones y se espera lo aprendido en este taller sirva de practica para realizar la PEP 1 que se viene la próxima semana.