Universidad de Santiago de Chile

FACULTAD DE INGENIERÍA

TALLER 2 - FORMA A

ECUACIONES DIFERENCIALES Y MÉTODOS NUMÉRICOS

Sección: 13307-0-A-1

Profesores: Jarnishs Beltran Mejia / Eduardo Díaz Valenzuela

Autor: John Serrano Carrasco

Enero 2022

Contents

1 Introducción

El presente informe tiene como propósito dar respuestas a los problemas y preguntas planteados en el Taller N°2 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.

Se utilizará Sympy para poder trabajar con Transformadas de Laplace, la cual nos permite obtener soluciones de ecuaciones diferenciales.

Con estas herramientas, junto con los conceptos aprendidos en las Unidades 2 y 3 del curso, se pueden responder los dos problemas dados.

2 Problema 1

El número áureo satisface la ecuación

$$\phi = 1 + \frac{1}{\phi},$$

de donde se puede obtener de manera relativamente sencilla que

$$\phi = \frac{1+\sqrt{5}}{2}$$

2.1 Parte A

"Calcule directamente en Python el valor de ϕ."

Utilizando Python, podemos calcular el valor del número áureo. Para ello, se realiza el siguiente código:

[1]:
```
# 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
```

[2]:
```
# Calculando el numero aureo se tiene
aureo = (1 + (np.sqrt(5)))/2
print("El numero aureo es", aureo)
```

A través de esto, se obtiene que el valor del número áureo es **1,618033988749895**.

2.2 Parte B

"Defina una función adecuada para utilizar el método de la bisección y aproxime el valor de ϕ, justificando la elección del intervalo elegido, con una tolerancia de 10^-6."

Considerando que

$$\phi = \frac{1+\sqrt{5}}{2}$$

Se puede obtener que una función valida para ser utilizada con el método de la bisección es:

$$f(\phi, y) = \frac{1+\sqrt{5}}{2} - \phi = 0$$

Esto debido a que $\phi = \frac{1+\sqrt{5}}{2}$ es una raiz de la sección anterior. Ahora, para encontrar los intervalos, podemos jugar un poco con el número ϕ para encontrar los intervalos a utilizar en el método de la bisección. Considerando que:

$$\sqrt{4} < \sqrt{5} < \sqrt{9}$$

Es decir:

$$2 < \sqrt{5} < 3$$

Sumando 1:

$$2+1 < \sqrt{5}+1 < 3+1$$

Diviendo por 2:

$$\frac{3}{2} < \frac{\sqrt{5}+1}{2} < \frac{4}{2}$$

Reescribiendo lo anterior tenemos:

$$\frac{3}{2} < \phi < 2$$

Por lo que nuestro intervalo es:

$$[\frac{3}{2}, 2]$$

Podemos comprobar que el intervalo sea valido si se cumple el teorema de Bolzano:

$$f(a) * f(b) < 0$$

Comprobando:

$$f(\frac{3}{2}) = \frac{1+\sqrt{5}}{2} - \frac{3}{2}$$

$$f(\frac{3}{2}) = 0,11803..$$

$$f(2) = \frac{1+\sqrt{5}}{2} - 2$$

$$f(2) = -0,38196$$

Por lo que claramente, se cumple que

$$f(\frac{3}{2}) * f(2) < 0$$

Considerando lo anterior y que tolerancia = $0,000001$, utizaremos el método de la bisección para aproximar el valor de ϕ, desarrollando el código a continuación en Python.

[3]:
```
# Definimos la funcion a utilizar
def f(x): return (1 + (np.sqrt(5)))/2 - x

# Definimos el metodo de biseccion visto en clases
def biseccion (a,b,f,tolerancia):
    while (np.abs(a-b)>= tolerancia):
        xi = (a+b)/2
        prod = f(a)  * f(xi)
        if prod < 0:
            b = xi
        else:
            if prod > 0:
                a = xi
            else:
                if prod == 0:
                    a = xi
                    b = xi
    return xi

solucion = biseccion(1.5, 2, f, 0.000001)
print("El valor aproximado de x con el metodo de la biseccion es:", solucion)
```

A traves del método de la Bisección, se obtiene que el valor aproximado del número aureo es **1.6180334091186523**.

2.3 Parte C

"El método de la secante es una variación del método de Newton Rhapson, donde en vez de calcular la derivada de la función, ésta se sustituye por una aproximación de la pendiente de la recta tangente $f'(X_n)\frac{f(X_n)-f(Xn-1)}{X_n-X_n-1}$ resultando en el siguiente algoritmo
* Datos de entrada: X_0, X_1, tolerancia, función
* Para n = 1,2,3 calcular

$$X_n + 1 = X_n - [\frac{X_n - X_n - 1)}{f(X_n) - f(X_n - 1)}]f(X_n)$$

hasta que $|X_n + 1 - X_n| <$ tolerancia
Elabore un código en Python para dicho algoritmo."

Basado en lo visto en clases, se puede realizar el método de la secante si se aplica la formula utilizada junto a un While True. "Break" será usado para romper el ciclo una vez que se cumpla la condición mencionada en el algoritmo. Cabe destacar de que la función utilizada es la misma de la parte B. Dicho esto, se tiene el siguiente código:

[4]:
```
# Definimos el metodo de la secante basado en el enunciado anterior
def secante(x0,x1,tolerancia,f):
    while True:
        # Se aplica la formula dada
        x2 = x1 - (f(x1)*(x1-x0))/(f(x1)-f(x0))
        x0 = x1
        x1 = x2
        # Se comprueba si se debe acabar el ciclo
        if np.abs(x1-x0)< tolerancia:
            # Se rompe el ciclo
            break
    # Se retorna el valor final de x2
    return x2
```

Por lo tanto, ahora podemos utilizar el método de la Secante para aproximar valores.

2.4 Parte D

"Utilice el código del item (c) con una tolerancia de 0,000001 para encontrar una aproximación de ϕ."

Se llama la funcion secante con x0 = 1 y x2 = 2, ambos valores cercanos al numero aureo. Se utiliza la función f de la parte B.

[5]:
```
# Llamamos a la funcion secante con x0 = 1 y x2 = 2, ambos valores cercanos al␣
 ↪numero aureo
aureo_secante = secante(1,2,0.000001, f)
# Se imprime el resultado
print("La aproximacion de x con el metodo de la secante es:", aureo_secante)
```

Con el método de la Secante obtenemos el valor **1.618033988749895**, lo cual es exactamente igual al valor que habiamos obtenido en la parte A. Esto indica que el Método de la Secante es un método mucho mas eficiente que el Método de la Bisección, por lo menos para este caso en especifico.

2.5 Parte E

"Suponga que el valor máquina encontrado en (a) es el valor real de . Calcule el error cometido con cada uno de los métodos y concluya cuál de ellos entrega una mejor aproximación."

Para esta parte, se puede realizar un breve código para calcular los errores. El código es el siguiente:

[6]:
```
print("El numero aureo es:", aureo)
print("La aproximacion con el metodo de la biseccion es:", solucion)
print("La aproximacion con el metodo de la secante es:", aureo_secante)
print("\n")

# Procedemos a calcular los errores:

error_biseccion = np.abs(solucion - aureo)
error_aureo_secante = np.abs(aureo_secante - aureo)
print("El error absoluto con el metodo de la biseccion es:", error_biseccion)
```

```
print("El error absoluto con el metodo de la secante es:", error_aureo_secante)
```

De lo anterior se tiene:

1. El numero aureo es: 1,618033988749895
2. La aproximacion con el metodo de la biseccion es: 1,6180334091186523
3. La aproximacion con el metodo de la secante es: 1,618033988749895
4. El error absoluto con el metodo de la biseccion es: **5,796312425587757e-07**
5. El error absoluto con el metodo de la secante es: **0,0**

Por lo tanto, la mejor aproximación es a traves del **Método de la Secante**, ya que el método no da error, si no que da exactamente el mismo valor obtenido en la parte A (asumiendo que ese es el valor real del número áureo).

3 Problema 2

Considere la función por partes definida por

$$f(t) = \begin{cases} 3 & si \quad t < 1 \\ t-1 & si \quad 1 \leq t < 2 \\ t & si \quad t \geq 2 \end{cases}$$

3.1 Parte A

"Reescriba la función en términos de la función de Heaviside, utilizando la notación de Jupyter para esta función."

Lo primero que podemos hacer es partir graficando la función anterior. El gráfico es el siguiente

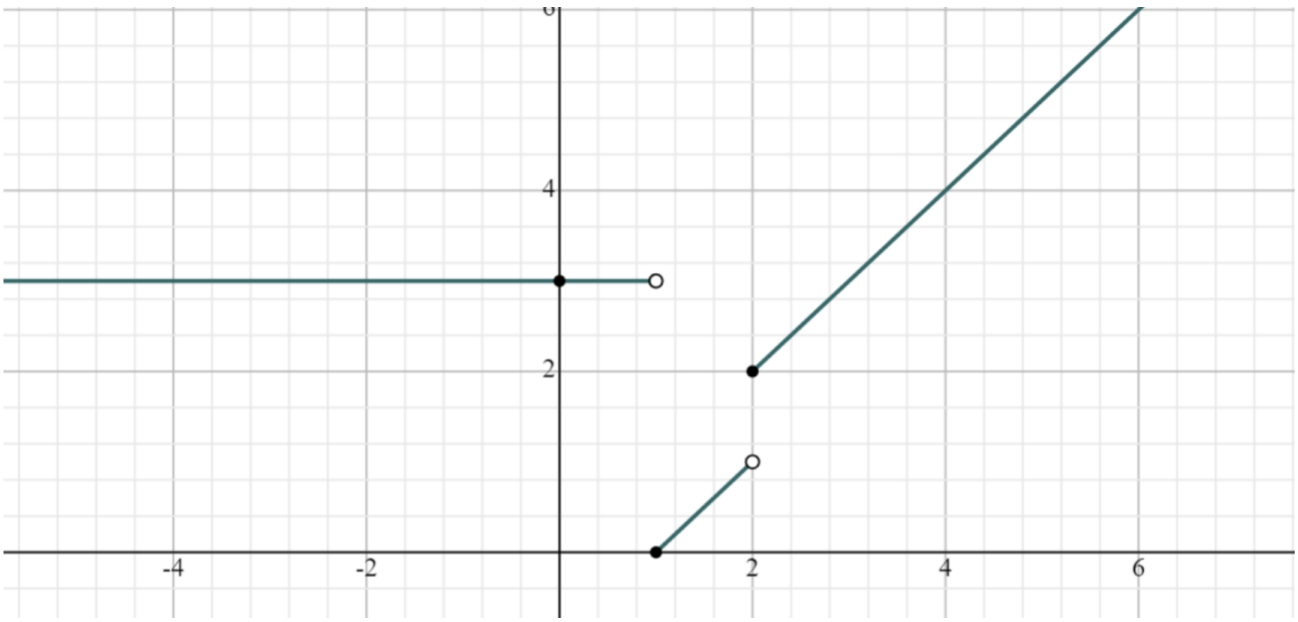

Figura 1: Grafico de la función por partes del Problema 2

Por lo tanto, considerando lo anterior, podemos guiarnos del gráfico para transformar la función en terminos de la función de Heaviside. Tenenemos las siguientes funciones basandonos en el gráfico anterior y en las traslaciones posibles:

$$\theta(t-1) = \begin{cases} 0 & si \quad t < 1 \\ 1 & si \quad 1 \leq t < 2 \end{cases}$$

$$\theta(t-2) = \begin{cases} 0 & si \quad 1 \leq t < 2 \\ 1 & si \quad t \geq 1 \end{cases}$$

A traves de lo anterior, junto con el gráfico y basandono en la métodologia utilizada en clases, podemos comenzar a transformar la función en terminos de la función de Heaviside. Tendriamos que inicialmente, la función seria:

$$f(t) = 3 + [(t-1) - 3]\theta(t-1) + [t - (t-1)]\theta(t-2)$$

Reescribiendo lo anterior:

$$3 - 3\theta(t-1) + (t-1)\theta(t-1) - (t-1)\theta(t-2) + t\theta(t-2)$$

Para comprobar que esto esté correcto, podemos gráficar la función. Para ello, vamos a utilizar sympy

[1]:
```
# Importamos sympy y Matplotlib.pyplot
import sympy as sp
import matplotlib.pyplot as plt
```

[2]:
```
# Definimos los simbolos que seran utiles para todo el problema
s = sp.Symbol('s')
t = sp.Symbol('t')
```

[3]:
```
# Definimos f, que es la funcion encontrada, donde sp.Heaviside es
f = 3-3*sp.Heaviside(t-1)+(t-1)*sp.Heaviside(t-1)-(t-1)*sp.Heaviside(t-2)+t*sp.
 ↪Heaviside(t-2)
# Graficaremos x entre -2 y 5 e y entre 0 y 20.
sp.plot(f,xlim=(-2,5),ylim=(0,20), xlabel = 't', ylabel = 'f(t)')
```

Gracias al código anterior, se obtiene el siguiente gráfico:

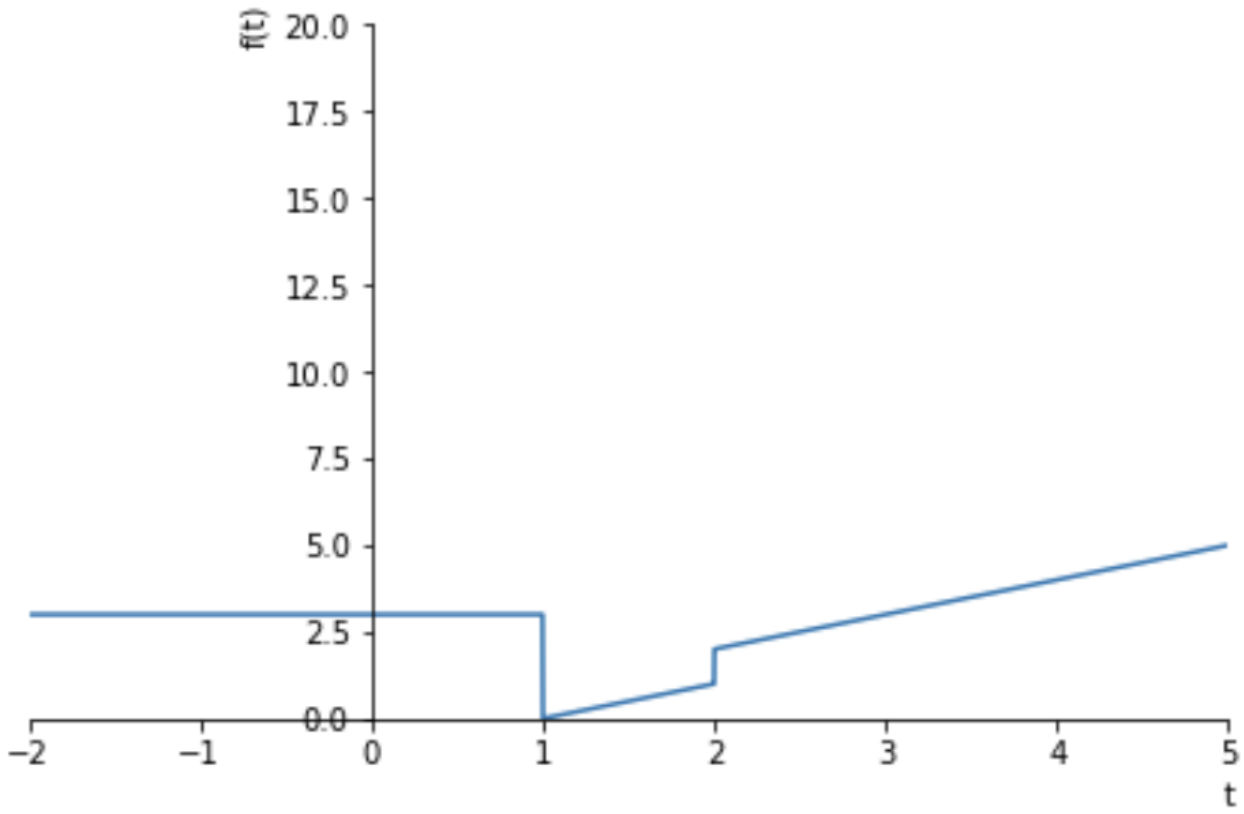

Figura 2: Grafico de la función en terminos de la función Heaviside

Podemos notar que coincide bastante con el gráfico mostrado anteriormente, por lo que se ha logrado exitosamente transforma la función por partes a una función en terminos de la función Heaviside.

3.2 Parte B

"Utilizando Transformada de Laplace y con ayuda de los comandos de Python, resuelva laecuación:

$$y''' + 2y'' = 1 + f(t)$$

con condiciones y(0) = y′(0) = 0, y″(0) = 1, donde f(t) es la función por partes definida en el apartado anterior."

Primero, podemos utilizar la siguiente formula para obtener las Transformadas de Laplace del lado izquierdo de la ecuación

$$\mathcal{L}\{f^{(n)}(t)\} = s^n\mathcal{L}\{f(t)\} - \sum_{i=1}^{n} s^{n-i}f^{i-1}(0) = s^n\mathcal{L}\{f(t)\} - s^{(n-1)}f(0) - ... - f^{(n-1)}(0)$$

Al aplicar esto, tenemos:

$$\mathcal{L}\{y'''\} = s^3\mathcal{L}\{y\}(s) - s^2y(0) - sy'(0) - y''(0)$$

$$\mathcal{L}\{y''\} = s^2\mathcal{L}\{y\}(s) - sy(0) - y'(0)$$

Reemplazando las condiciones iniciales:

$$\mathcal{L}\{y'''\} = s^3\mathcal{L}\{y\}(s) - 1$$

$$\mathcal{L}\{y''\} = s^2\mathcal{L}\{y\}(s)$$

Y con esto, podemos comenzar a resolver la ecuación utilizando Python y Transformada de Laplace.

[4]:
```
# Calculamos la transformada de Laplace de 1
transformada_1 = sp.laplace_transform(1,t,s,noconds = True)
# Calculamos la transformada de Laplace de la funcion obtenida en la parte A
transformada_f = sp.laplace_transform(f,t,s,noconds = True)
# Sumamos ambas transformadas de Laplace, lo que corresponde al lado derecho de la␣
 ↪ecuación
lado_derecho = transformada_1 + transformada_f
# Pasamos el 1 de la transformada de Laplace de y''' sumando hacia el lado derecho
lado_derecho = lado_derecho + 1
# Pasamos dividiendo lo que queda tras factoriazar la transformada de Laplace de Y
lado_derecho = lado_derecho / (s**3+2*s**2)
# Ahora, para obtener la solucion de la ecuacion aplicamos transformada inversa de␣
 ↪Laplace
transformada_final = sp.inverse_laplace_transform(lado_derecho,s,t,noconds = True)
# Mostramos el resultado final por pantalla
transformada_final
```

Del código anterior, se obtiene que la solución de la ecuación es la siguiente:

$$\frac{\left(12\left(\left(4t^2 - 2t + 1\right)e^{2t} - 1\right)\theta(t) + \left(6\left(2t^2 - 10t - e^{4-2t} + 13\right)\theta(t-2) + \left(4t^3 - 54t^2 + 138t + 21e^{2-2t} - 109\right)\theta(t-1)\right)e^{2t}\right)e^{-2t}}{48}$$

3.3 Parte C

Grafique la solución de la ecuación diferencial y estudie el comportamiento de la solución y(t)conforme t → ∞."

Utilizando sympy.plot, podemos gráficar la solución anterior, teniendo en cuenta que está guardada en la variable "tranformada_final".

[6]:
```
sp.plot(transformada_final, xlim=(-2,15),ylim=(0,20))
```

Se obtiene el siguiente gráfico:

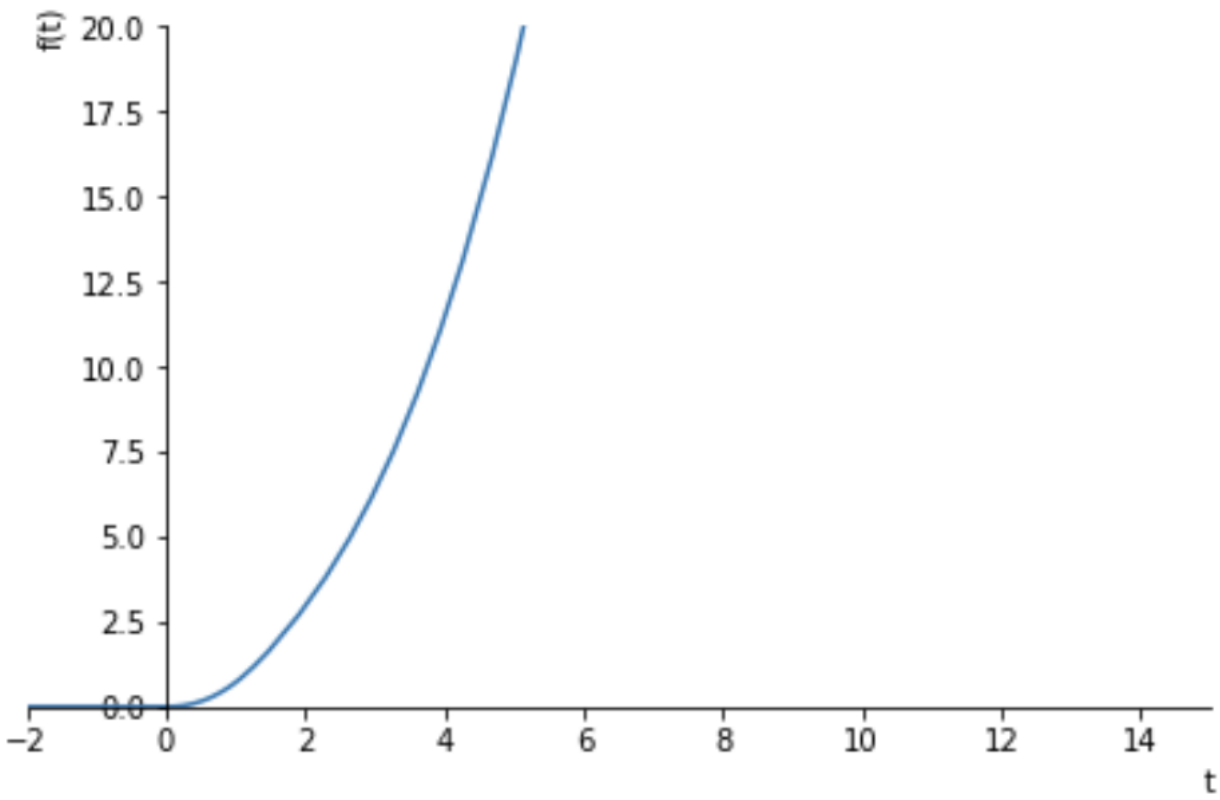

Figura 3: Grafico de la solución obtenida en la parte B

De este gráfico es posible notar dos cosas, a medida que $t \to \infty$.

1. El gráfico llega solo hasta t = 10 y luego se corta. Esto es debido a que en el proceso de la creación de la solución esta involucrada una función por partes
2. Su comportamiento parece ser el de una función exponencial, lo cual tiene sentido considerando que hay varias exponenciales involucradas en la función.

4 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 que lo aprendido en este taller sirva de practica para realizar la PEP 2.

Al ser este el último taller, se puede rescatar de que en el curso se aprendieron herramientas bastante utiles que no se tenian conocimiento de su existencia antes, como Sympy y se espera que en algún futuro estos conocimientos sean de utilidad para resolver problemas no solo de la Informática, si no que también relacionados con la matemática.