

PARCIAL 1

ANALISIS NUMERICO

DAVID ALFONSO ANTELIZ BONILLA-1004924436

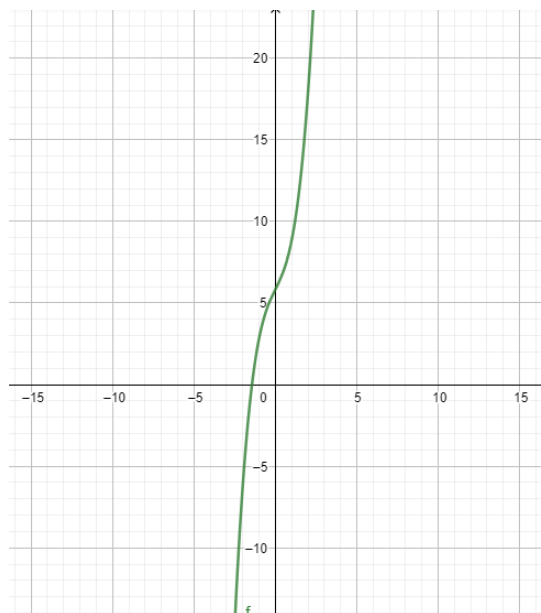
Punto 1B

The screenshot shows the Spyder Python IDE interface. The editor on the left contains a Python script for the secant method. The console on the right shows the output of running the script, displaying a table of iterations with columns for iteration number (k), current estimate (x), function value (f(x)), and error (e). The script defines a function  $f(x) = x^3 + 2x + 6 - 1/6$  and uses the secant method to find the root, starting with  $x_0 = 1$  and  $x_1 = 100$ , and iterating until the error is less than  $1e-32$ .

```
1 import numpy as np
2 import scipy.optimize as optimize
3 def f(x):
4     return x*x*x+2*x+6-1/6
5
6 def secante (f, x0, x1,N=100, emax=1e-32):
7
8     for k in range (N):
9         fp=(f(x1)-f(x0))/(x1-x0)
10        x=x1-f(x1)/fp
11        e=abs((x-x1)/x)
12        if e<emax:
13            break
14        x0=x1
15        x1=x
16        print(k,x,f(x),e)
17
18 secante(f,0,1)
```

Console output:

```
In [48]: runfile('C:/Users/david/.spyder-py3/temp.py', wdir='C:/Users/david/.spyder-py3')
0 -1.9444444444444444 -5.407235938643338 1.5142857142857145
1 -0.8264199106573076 3.616073614120551 1.3528528528528503
2 -1.2744660507657002 1.214332270118617 0.35155596325159555
3 -1.5010003091905765 -0.550424576588775 0.15092223829937604
4 -1.4363447240026324 0.046321589823654716 0.04039764733408064
5 -1.4358292724105635 0.001560976985787249 0.0038197769145996355
6 -1.4360205399534809 -4.67045894204035e-06 0.000133192762640858
7 -1.4360199693862075 4.607195909180986e-10 3.97325450610168e-07
8 -1.436019969443463 -5.828670879262872e-16 3.9870967079475334e-11
In [44]:
```



## Punto 2B

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\david\spyder-py3\temp.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import math
4 from scipy.interpolate import approximate_taylor_polynomial
5
6 def f(x):
7     return (x-1)*np.log(x)
8
9
10
11
12 def taylor(function, valorX, x0, x1):
13     x = np.linspace(x0, x1, num=1)
14     plt.plot(x, f(valorX), label="Function Original")
15
16     history = []
17     for degree in np.arange(1, 15):
18         n_taylor = approximate_taylor_polynomial(f, 0, degree, 1,)
19         plt.plot(x, n_taylor(x), label=f"degree = {degree}")
20         history.append(n_taylor)
21
22     valoresY = []
23     for xi in x:
24         valoresY.append(function(xi))
25
26     plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.0, sha
27     plt.tight_layout()
28     plt.axis([x0, x1, 1, 10])
29     plt.show()
30
31     return np.array(history)
32
33
34 polinomios = taylor(f, 1, 1, 10)
35
36
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in **Preferences > Help**.

New to Spyder? Read our [tutorial](#)

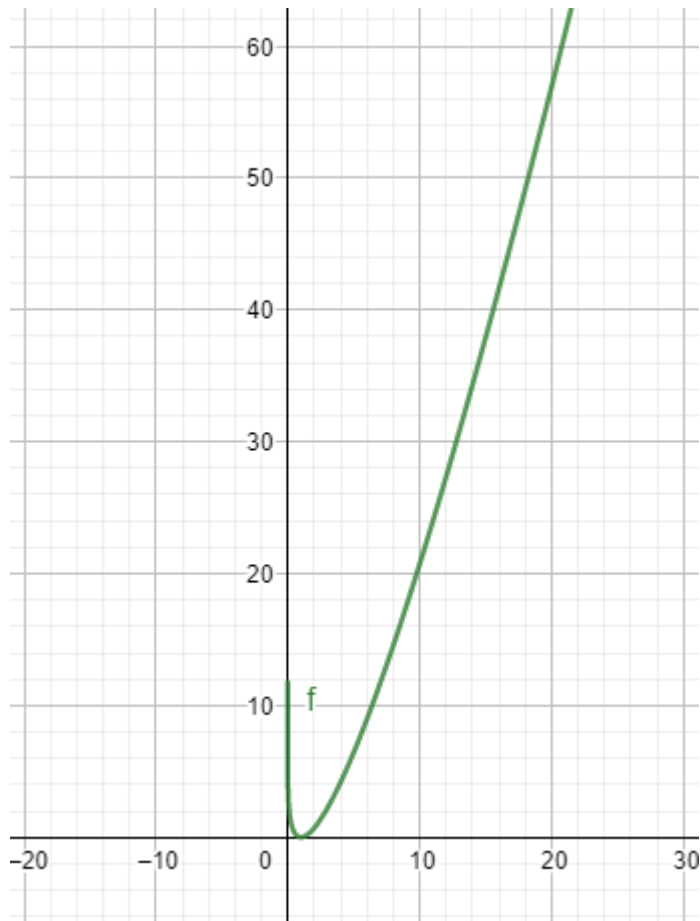
Variable explorer Help Plots Files

Console 1/A

```
In [28]: runfile('C:/Users/david/.spyder-py3/temp.py', wdir='C:/Users/david/.spyder-py3')
Traceback (most recent call last):
  File "C:/Users/david/.spyder-py3/temp.py", line 10, in <module>
    polinomios = taylor(f, 1, -10, 10)
  File "C:/Users/david/.spyder-py3/temp.py", line 10, in taylor
    n_taylor = approximate_taylor_polynomial(f, 0, degree, 1,)
  File "C:/Users/david/.spyder-py3/temp.py", line 10, in approximate_taylor_polynomial
    P = KroghInterpolator(xs, f(xs))
  File "C:/Users/david/.spyder-py3/temp.py", line 10, in f
    return (x-1)*math.log(x)
TypeError: only size-1 arrays can be converted to Python scalars
```

Run until current function or method returns

LSP Python: ready conda: base (Python 3.8.8) Line 34, Col 1 ASCII CRLF RW Mem 69%



## Punto 5

The image shows the Spyder Python IDE interface. The main editor window displays a Python script named `temp.py` located at `C:\Users\david\spyder-py3\temp.py`. The script implements the Newton-Raphson method to find the root of a function. It includes imports for `numpy`, `math`, and `matplotlib.pyplot`, and uses `sympy` for symbolic differentiation. The function `funcA(x)` is defined as  $f(x) = e^x - x - 1$ . The derivative `derivFunc(x)` is also defined as  $f'(x) = e^x - 1$ . The `newtonRaphson(x, f, e, n)` function uses a while loop to iteratively refine the root until the absolute value of the function is less than a specified tolerance `e` or the maximum number of iterations `n` is reached. The initial guess `x0` is set to 1. The script prints the final root value and calls `newtonRaphson(x0, funcA, 10e-12)`.

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4
5
6 from sympy import *
7 #funcion de entrada
8 def funcA( x ):
9     return math.e ** x - x - 1
10
11 #Derivada de la funcion
12 def derivFunc( x ):
13     return math.e ** x - 1
14
15 #Metodo de Newton
16 def newtonRaphson( x,f,e,n=0):
17     h = f(x) / derivFunc(x)
18     while abs(h) >= e:
19         h = f(x)/derivFunc(x)
20
21         x = x - h
22
23     print("El valor de la raiz es : ", "%.4f"% x)
24
25
26 #Asignacion x0
27 x0 = 1
28
29 #Llamado al metodo de Newton
30 newtonRaphson(x0,funcA,10e-12)
31
32
33
34
35
36
37
```

The right-hand side of the IDE shows the 'Console' tab with the following output:

```
In [45]: runfile('C:/Users/david/.spyder-py3/temp.py', wdir='C:/Users/david/.spyder-py3')
El valor de la raiz es : -0.0000
In [46]:
```

The bottom status bar indicates the current line and column (Line 31, Col 1) and the memory usage (Mem 69%).