

# Tarea 2

## Métodos Numéricos

Benjamin Rivera

13 de septiembre de 2020

### Índice

<b>1. Tarea 2</b>	<b>2</b>
1.0.1. Como ejecutar . . . . .	2
1.1. Ejercicio 1 . . . . .	3
1.1.1. Como ejecutar . . . . .	4
1.2. Ejercicio 2 . . . . .	5
1.3. Ejercicio 3 . . . . .	6
1.3.1. Como ejecutar . . . . .	7
1.4. Ejercicio 4 . . . . .	9
1.4.1. Como ejecutar . . . . .	10
1.5. Ejercicio 5 . . . . .	11
1.5.1. Ejercicio 5 . . . . .	12

## 1. Tarea 2

Tarea 2 de Benjamín Rivera para el curso de **Métodos Numéricos** impartido por Joaquín Peña Acevedo. Fecha limite de entrega **13 de Septiembre de 2020**.

### 1.0.1. Como ejecutar

**Requerimientos** Este programa se ejecuto en mi computadora con la version de **Python 3.8.2** y con estos [requerimientos](#)

**Jupyter** En caso de tener acceso a un *servidor jupyter* ,con los requerimientos antes mencionados, unicamente basta con ejecutar todas las celdas de este *notebook*. Probablemente no todas las celdas de *markdown* produzcan el mismo resultado por las *Nbextensions*.

**Consola** Habrá archivos e instrucciones para poder ejecutar cada uno de los ejercicios desde la consola.

**Si todo sale mal** En caso de que todo salga mal se puede ejecutar esta version modificada para python6 en [Google Colab](#)

```
[69]: import sys

import numpy as np
import matplotlib.pyplot as plt
import warnings

from math import atan, cos, exp, log, pi, sin

# Para probar que todo sale bien
if __name__ == "__main__":
    import doctest
    doctest.testmod()

    if len(sys.argv) >= 2:
        ejercicio = sys.argv[1].lower()

        if ejercicio == 'ejercicio1':
            print(Ejercicio1())
        elif ejercicio == 'ejercicio3':
            print(Ejercicio3())
        elif ejercicio == 'ejercicio4':
            print(Ejercicio4())
        elif ejercicio == 'ejercicio5':
            print(Ejercicio5())
        else:
            print('= ... =')
```

## 1.1. Ejercicio 1

Considere la función  $f(x) = x + \ln \sqrt{x} - 2,5$ . Verifique que la función  $g(x) = 2,5 - \ln \sqrt{x}$  cumple con las condiciones del teorema de punto fijo en el intervalo  $I = [1, 3]$ , de modo que puede usarse para encontrar una raíz de  $f(x)$ . Estime el número de iteraciones que requiere el algoritmo de punto fijo para aproximar a la raíz de  $f(x)$  con un error menor a  $10^{-6}$ , partiendo de cualquier punto en el intervalo  $I$ .

**Respuesta.** Suponemos que

$$\ln x = \log_e x$$

Podemos ver que la función  $g$  es constante, por lo que únicamente falta corroborar que

$$a \leq g(x) \leq b, \quad \forall x \in [a, b]$$

Empezamos por verificar los valores para los que  $a \leq g(x)$ , de manera que

$$a \leq g(x)$$

$$\begin{aligned} 1 &\leq 2,5 - \ln \sqrt{x} \\ 0 &\leq 1,5 - \ln \sqrt{x} \\ \ln \sqrt{x} &\leq 1,5 \\ \Downarrow \\ \sqrt{x} &\leq e^{1,5} \\ x &\leq e^3 \sim 20,08 \end{aligned}$$

Por otro lado, se da que

$$g(x) \leq b$$

$$\begin{aligned} 2,5 - \ln \sqrt{x} &\leq 3 \\ -\ln \sqrt{x} &\leq 0,5 \\ \ln \sqrt{x} &\geq -0,5 \\ \Downarrow \\ \sqrt{x} &\geq e^{-0,5} \\ x &\geq e^{-1} \sim 0,36 \end{aligned}$$

Por los dos resultados anteriores tenemos que, para todo el intervalo  $[1, 3]$ , se cumple que  $a \leq g(x) \leq b$ . De manera que la función  $g$  cumple con el **Teorema de punto fijo** y se puede usar para encontrar una raíz de  $f(x)$ .

[70]:

```
def metodo_punto_fijo(g, x, n, t=0,/,v=True):
    """ Implementacion del metodo de punto fijo, esta
    funcion recibe tres parametros obligatorios y uno
    opcional.
    Input:
        g := La derivada de la funcion g del metodo
        x := La aproximacion de la raiz de f
        n := numero de iteraciones requeridas
        t := Tolerancia de la variable. Default a 0
        v := [opcional] Indica si se quiere imprimir
            las iteraciones en pantalla
    Output:
        (x,k)
        x := aproximacion de la raiz
        k := ultima iteracion realizada
    return (x,_)
# ejercicio
def Ejercicio1():
    g = lambda x : 2.5 - log(x**(1/2))
    return metodo_punto_fijo(g, 2, 20, 1.0e-6, False)
print(Ejercicio1())
```

(2.1234737040050793, 8)

De manera práctica podemos ver en la celda anterior que se puede encontrar una raíz, con la tolerancia indicada, en 8 iteraciones.

**Estimación** La diferencia entre dos iteraciones del metodo, especificamente con la funcion  $g$ , esta dada por

$$\begin{aligned} d(x) = |g(x) - g(g(x))| &= \left| 2,5 - \ln \sqrt{x} - \left( 2,5 - \ln \sqrt{2,5 - \ln \sqrt{x}} \right) \right| \\ &= \left| 2,5 - \ln \sqrt{x} - 2,5 + \ln \sqrt{2,5 - \ln \sqrt{x}} \right| \\ &= \left| \ln \sqrt{2,5 - \ln \sqrt{x}} - \ln \sqrt{x} \right| \\ &= \left| \frac{\ln(2,5 - \ln \sqrt{x})}{2} - \frac{\ln x}{2} \right| \\ &= \left| \frac{1}{2} \ln \frac{5 - \ln x}{2x} \right| \end{aligned}$$

Podemos ver que, si esta funcion no tuviera el valor absoluto ( $h$ ), ser'ia una funci'on descendente en el rango  $I$ . Donde  $h(a) = \frac{\ln(5/2)}{2} \sim 0,45$ ,  $h(b) = \frac{\ln(5/6 - 1/6 \ln(3))}{2} \sim -0,21$ , por lo que el rango de la funci'on  $d$  es  $[0, 0,45]$ . De esto vemos que el mayor *paso* que podemos dar es de 0,45. Suponemos que estamos trabajando en una maquina con 64bits , por lo que el *epsilon* de la maquina es del orden  $\epsilon_m \sim e - 16$

Sabemos que la raíz de  $f$  en el intervalo  $I$  es  $\sim 2,1234$ . Adem'as, el valor m'as alejado de este es 1. Partiendo de 1 necesitamos primero 3 iteraciones para llegar a 2,1.

Se necesitan más de 3 iteraciones.

### 1.1.1. Como ejecutar

Para ejecutar este ejercicio en **consola** es importante ubicarse en la misma carpeta del [archivo Tarea2.py](#) y ejecutar el siguiente comando en consola

```
python3 Tarea2.py Ejercicio1
```

Este programa no espera recibir argumento alguno.

## 1.2. Ejercicio 2

Explique como usar el **Teorema 1**, junto con algún método de cálculo de raíces visto en clase para tratar de encontrar todas las raíces de un polinomio, además mencione la conveniencia de usarlo en función de  $\eta$ .

**Teorema 1.** Todos los ceros del polinomio  $p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  de grado  $n$  están contenidos en el círculo  $\Omega$  del plano complejo definido por

$$\Omega = \{z \in \mathbb{C} : |z| \leq 1 + \eta\} \quad \text{con } \eta = \max_{0 \leq k \leq n-1} \left| \frac{a_k}{a_n} \right|$$

**Respuesta** Con inspiración en el *ejercicio 4*, tratamos de encontrar una manera de usar el **Teorema 1** con el *método de la secante*.

De manera general podemos considerar que únicamente nos interesan los puntos del círculo  $\Omega$  cuya parte imaginaria es 0, esto porque estamos buscando raíces del polinomio en los  $\mathbb{R}$ . De manera que, siendo  $z$  un número complejo,  $z.r$  la parte su parte real y  $z.i$  su parte imaginaria, tenemos que

$$\begin{aligned} |z| &\leq 1 + \eta \\ \sqrt{z.r^2 + z.i^2} &\leq 1 + \eta \\ \sqrt{z.r^2} &\leq 1 + \eta \\ |z.r| &\leq 1 + \eta \end{aligned}$$

por lo que todos los  $x$  del dominio de la función  $f$  que sean raíces deben de estar en el intervalo  $[-(1 + \eta), 1 + \eta]$ .

**Aquí podemos notar que, mientras más chico sea  $\eta$ , habrá menos valores donde debemos buscar las raíces**

Para poder implementar esta idea con algún método, este requeriría ir encontrando las raíces en orden. Dicho de otra manera, un método que nos asegure encontrar la raíz más cercana que exista al punto inicial que le otorguemos. Esto nos permitiría poder explorar todo el intervalo creado por el **Teorema 1** y encontrar todas las raíces del polinomio.

### 1.3. Ejercicio 3

#### Respuesta

```
[71]: def metodo_secante(f, x0, x1, t, N,/, v=True, tDato=np.  
↳float64):  
    """ Implementacion del metodo de la secante.␣  
    ↳Pseudocodigo obtenido de las notas de clase. Argumentos:  
        Input:  
        f := Funcion fx del metodo.  
        x0,x1 := Valores de los puntos iniciales.  
        t := Valor de la tolerancia del metodo.  
        N := Maximo numero de iteraciones.  
        [Opcionales]  
        v := [True] Verdadero si queremos imprimir en␣  
↳plantalla  
        tDato := [np.float64] Espera uno de los tipos de␣  
↳datos de numpy para manejar la presicion dentro del metodo.  
        Output:  
        (xk, f(xk), k, res)  
        xk := Ultimo punto generado por el algoritmo  
        f(xk) := Valor del xk evaluado en f  
        k := Iteraciones realizadas  
        res := Variable formateada como se solicita  
            0 -> abs(fxk) < t  
            1 -> fxk - fxk1 casi cero  
            2 -> ninguna de los anteriores  
    """  
    return ret  
  
[72]: """ En el diccionario 'funciones' se guardan cada uno de los␣  
↳incisos de la parte 3 del ejercicio, esto se hace en el formato␣  
↳[f,x0,x1]. Donde f es la funcion en formato lambda y x0,x1 son␣  
↳los puntos iniciales del metodo.  
    """  
    def Ejercicio3(v=True):  
        ret = []  
        funciones = {  
            'a': [(lambda x: exp(2*x)-x-6), 3, 15],  
            'b': [(lambda x: 3*cos(3*pi*x) - 4*x), -0.75, -0.  
↳8],  
            'c': [(lambda x: atan(x)), 0.5, 0.6],  
            'd': [(lambda x: atan(x)), 2.0, 2.8] }  
        t = 2.3e-16**(1/2) # Tolerancia de la Tarea1  
        N = 30 # Por iluminacion divina  
        for f, x0, x1 in funciones.values():  
            ret.append(metodo_secante(f,x0,x1,t,N, v=v))  
            if v: print('-'*45 + '\n')  
        return ret  
  
    print(Ejercicio3())
```

```

k= 1 | xk=15.000000 | f(xk)= 1.0686475e+13
k= 2 | xk= 3.000000 | f(xk)= 3.9442879e+02
k= 3 | xk= 3.000000 | f(xk)= 3.9442879e+02
k= 4 | xk= 2.510548 | f(xk)= 1.4306670e+02
k= 5 | xk= 2.231968 | f(xk)= 7.8596661e+01
k= 6 | xk= 1.892347 | f(xk)= 3.6129816e+01
k= 7 | xk= 1.603404 | f(xk)= 1.7096730e+01
k= 8 | xk= 1.343858 | f(xk)= 7.3542093e+00
k= 9 | xk= 1.147938 | f(xk)= 2.7851891e+00
k=10 | xk= 1.028508 | f(xk)= 7.9408843e-01
k=11 | xk= 0.980878 | f(xk)= 1.3092111e-01
k=12 | xk= 0.971474 | f(xk)= 7.8274317e-03
k=13 | xk= 0.970877 | f(xk)= 8.3965349e-05
k=14 | xk= 0.970870 | f(xk)= 5.4658317e-08
Resultado final
    x0= 3.0000,
    x1=15.0000, f(x1)= 1.068647458e+13,
    k =14, xk=0.9709, f(xk)=3.8192e-13,
    res =0
-----

k= 1 | xk=-0.800000 | f(xk)= 4.1270510e+00
k= 2 | xk=-1.007542 | f(xk)= 1.0377431e+00
k= 3 | xk=-1.077258 | f(xk)= 2.0697954e+00
k= 4 | xk=-0.937441 | f(xk)= 1.2562805e+00
k= 5 | xk=-0.721527 | f(xk)= 5.4939572e+00
k= 6 | xk=-1.001450 | f(xk)= 1.0060801e+00
k= 7 | xk=-1.064202 | f(xk)= 1.7894625e+00
k= 8 | xk=-0.920859 | f(xk)= 1.4799830e+00
k= 9 | xk=-0.235364 | f(xk)= -8.6802909e-01
k=10 | xk=-0.488782 | f(xk)= 1.6385517e+00
k=11 | xk=-0.323123 | f(xk)= -1.6936284e+00
k=12 | xk=-0.407322 | f(xk)= -6.7040135e-01
k=13 | xk=-0.462487 | f(xk)= 8.1126572e-01
k=14 | xk=-0.432282 | f(xk)= -5.8181942e-02
k=15 | xk=-0.434304 | f(xk)= -3.8747730e-03
k=16 | xk=-0.434448 | f(xk)= 2.4348491e-05
Resultado final
    x0=-0.7500,
    x1=-0.8000, f(x1)= 4.127050983e+00,
    k =16, xk=-0.4344, f(xk)=-9.9678e-09,
    res =0
-----

k= 1 | xk= 0.600000 | f(xk)= 5.4041950e-01
k= 2 | xk=-0.103929 | f(xk)= -1.0355708e-01
k= 3 | xk= 0.009269 | f(xk)= 9.2688075e-03
k= 4 | xk=-0.000030 | f(xk)= -3.0298567e-05
-----

Resultado final
    x0= 0.5000,
    x1= 0.6000, f(x1)= 5.404195003e-01,
    k = 4, xk=0.0000, f(xk)=8.6485e-10,
    res =0
-----

k= 1 | xk= 2.800000 | f(xk)= 1.2277724e+00
k= 2 | xk=-5.342829 | f(xk)= -1.3857703e+00
k= 3 | xk=-1.025283 | f(xk)= -7.9788123e-01
k= 4 | xk= 4.834477 | f(xk)= 1.3668253e+00
k= 5 | xk= 1.134544 | f(xk)= 8.4834676e-01
k= 6 | xk=-4.919372 | f(xk)= -1.3702509e+00
k= 7 | xk=-1.180351 | f(xk)= -8.6792665e-01
k= 8 | xk= 5.280012 | f(xk)= 1.3836198e+00
k= 9 | xk= 1.309991 | f(xk)= 9.1879710e-01
k=10 | xk=-6.537395 | f(xk)= -1.4190068e+00
k=11 | xk=-1.774166 | f(xk)= -1.0575375e+00
k=12 | xk=12.161436 | f(xk)= 1.4887538e+00
k=13 | xk= 4.013633 | f(xk)= 1.3266170e+00
k=14 | xk=-62.652410 | f(xk)= -1.5548366e+00
k=15 | xk=-26.679316 | f(xk)= -1.5333316e+00
k=16 | xk=2538.249192 | f(xk)= 1.5704024e+00
k=17 | xk=1240.467304 | f(xk)= 1.5699902e+00
k=18 | xk=-4942056.218483 | f(xk)= -1.5707961e+00
k=19 | xk=-2469773.635260 | f(xk)= -1.5707959e+00
k=20 | xk=19172755786885.507812 | f(xk)= 1.
↪5707963e+00
    k= 21 | xk=9586375423042.201172 | f(xk)= 1.
↪5707963e+00
    k= 22 | xk=-28858014414977399511252992.000000 | ↪
↪f(xk)= -1.5707963e+00
    k= 23 | xk=-144290072074879130568491008.000000 | ↪
↪f(xk)= -1.5707963e+00
Resultado final
    x0= 2.0000,
    x1= 2.8000, f(x1)= 1.227772386e+00,
    k =22, xk=-28858014414977399511252992.
↪0000, f(xk)=-1.5708e+00,
    res =1
-----

[(0.9708700202758297, 3.8191672047105385e-13, 14, ↪
↪0), (-0.43444687459757464,
-9.967753733519658e-09, 16, 0), (8.
↪648514335884798e-10, 8.648514335884798e-10,
4, 0), (-2.885801441497774e+26, -1.5707963267948966, ↪
↪22, 1)]

```

### 1.3.1. Como ejecutar

Para ejecutar este ejercicio en **consola** es importante ubicarse en la misma carpeta del [archivo Tarea2.py](#) y ejecutar el siguiente comando en consola

```
python3 Tarea2.py Ejercicio3
```

Este programa no espera recibir argumento alguno. La salida debe ser similar a la siguiente imagen

```
bench@bench-dell: ~/Documents/Acade...  
bench@bench-dell: ~/Docu... x bench@bench-dell: ~/Docu... x  
bench@bench-dell:~/Documents/academico/UG/Licenciatura/DENAT/GitHubRepo/NN/Tareas/T2$ python3 Tarea2.py Ejercicio3  
k= 1 | xk=15.000000 | f(xk)= 1.0686475e+13  
k= 2 | xk= 3.000000 | f(xk)= 3.9442879e+02  
k= 3 | xk= 3.000000 | f(xk)= 3.9442879e+02  
k= 4 | xk= 2.518548 | f(xk)= 1.4306670e+02  
k= 5 | xk= 2.231968 | f(xk)= 7.8596661e+01  
k= 6 | xk= 1.892347 | f(xk)= 3.6129816e+01  
k= 7 | xk= 1.603404 | f(xk)= 1.7096730e+01  
k= 8 | xk= 1.343858 | f(xk)= 7.3542093e+00  
k= 9 | xk= 1.147938 | f(xk)= 2.7851891e+00  
k= 10 | xk= 1.028508 | f(xk)= 7.9488843e-01  
k= 11 | xk= 0.980878 | f(xk)= 1.3092111e-01  
k= 12 | xk= 0.971474 | f(xk)= 7.8274317e-03  
k= 13 | xk= 0.970877 | f(xk)= 8.3965349e-05  
k= 14 | xk= 0.970870 | f(xk)= 5.4658317e-08  
Resultado final  
x0= 3.0000,  
x1=15.0000, f(x1)= 1.0686475e+13,  
k =14, xk=0.9709, f(xk)=3.8192e-13,  
res =0  
-----  
k= 1 | xk=-0.800000 | f(xk)= 4.1270510e+00  
k= 2 | xk=-1.007542 | f(xk)= 1.0377431e+00  
k= 3 | xk=-1.077258 | f(xk)= 2.0697954e+00  
k= 4 | xk=-0.937441 | f(xk)= 1.2562805e+00  
k= 5 | xk=-0.721527 | f(xk)= 5.4939572e+00  
k= 6 | xk=-1.001450 | f(xk)= 1.0060801e+00  
k= 7 | xk=-1.064202 | f(xk)= 1.7894625e+00  
k= 8 | xk=-0.920859 | f(xk)= 1.4799830e+00  
k= 9 | xk=-0.235364 | f(xk)= -8.6802909e-01  
k= 10 | xk=-0.488782 | f(xk)= 1.6385517e+00  
k= 11 | xk=-0.323123 | f(xk)= -1.6936284e+00  
k= 12 | xk=-0.407322 | f(xk)= -6.7040135e-01  
k= 13 | xk=-0.462487 | f(xk)= 8.1126572e-01  
k= 14 | xk=-0.432282 | f(xk)= -5.8181942e-02  
k= 15 | xk=-0.434304 | f(xk)= -3.8747730e-03  
k= 16 | xk=-0.434448 | f(xk)= 2.4348491e-05  
Resultado final  
x0=-0.7500,  
x1=-0.8000, f(x1)= 4.1270509e+00,  
k =16, xk=-0.4344, f(xk)=-9.9678e-09,  
res =0  
-----  
k= 1 | xk= 0.600000 | f(xk)= 5.4041950e-01  
k= 2 | xk=-0.103029 | f(xk)= -1.0355700e-01  
k= 3 | xk= 0.009269 | f(xk)= 9.2688075e-03  
k= 4 | xk=-0.000030 | f(xk)= -3.0298567e-05  
Resultado final  
x0= 0.5000,  
x1= 0.6000, f(x1)= 5.4041950e-01,  
k = 4, xk=0.0000, f(xk)=8.6485e-10,  
res =0  
-----  
k= 1 | xk= 2.800000 | f(xk)= 1.2277724e+00  
k= 2 | xk=-5.342820 | f(xk)= -1.3857703e+00  
k= 3 | xk=-1.025283 | f(xk)= -7.9788123e-01  
k= 4 | xk= 4.834477 | f(xk)= 1.3668253e+00  
k= 5 | xk= 1.134544 | f(xk)= 8.4834676e-01  
k= 6 | xk=-4.919372 | f(xk)= -1.3702590e+00  
k= 7 | xk=-1.180351 | f(xk)= -8.6792665e-01  
k= 8 | xk= 5.280012 | f(xk)= 1.3836198e+00  
k= 9 | xk= 1.309991 | f(xk)= 9.1879710e-01  
k= 10 | xk=-6.537395 | f(xk)= -1.4190068e+00  
k= 11 | xk=-1.774166 | f(xk)= -1.0575375e+00  
k= 12 | xk=12.161436 | f(xk)= 1.4887538e+00  
k= 13 | xk= 4.013633 | f(xk)= 1.3266170e+00  
k= 14 | xk=-62.652410 | f(xk)= -1.5548366e+00  
k= 15 | xk=-26.679316 | f(xk)= -1.5333316e+00  
k= 16 | xk=-2538.249192 | f(xk)= 1.5704024e+00  
k= 17 | xk=1240.467304 | f(xk)= 1.5699902e+00  
k= 18 | xk=-4942056.218483 | f(xk)= -1.5707961e+00  
k= 19 | xk=-2469773.635260 | f(xk)= -1.5707959e+00  
k= 20 | xk=19172755786885.507812 | f(xk)= 1.5707963e+00  
k= 21 | xk=9586375423042.201172 | f(xk)= 1.5707963e+00  
k= 22 | xk=-288580144149777399511252992.000000 | f(xk)= -1.5707963e+00  
k= 23 | xk=-144290072074879130568491008.000000 | f(xk)= -1.5707963e+00  
Resultado final  
x0= 2.0000,  
x1= 2.8000, f(x1)= 1.227772386e+00,  
k =22, xk=-288580144149777399511252992.0000, f(xk)=-1.5708e+00,  
res =1  
-----  
[(0.9708700202758297, 3.8191672047105385e-13, 14, 0), (-0.43444687459757464, -9.96775373519658e-09, 16, 0), (8.648514335884798e-10, 8.648514335884798e-10, 4, 0), (-2.885801441497774e+26, -1.5707963267948966, 22, 1)]  
bench@bench-dell:~/Documents/academico/UG/Licenciatura/DENAT/GitHubRepo/NN/Tareas/T2$
```

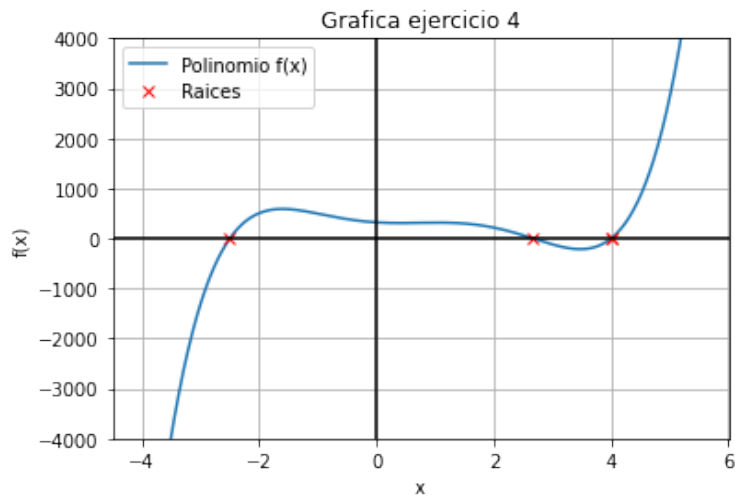


## 1.4. Ejercicio 4

**Respuesta** Es posible que este metodo encuentre puntos que no corresponden a las raices, pero estos son facilmente identificables en la grafica. Este error se da porque no se filtra el resultado res del metodo de la secante.

```
[73]: def f_polinomio(coef):  
    """ Funcion que genera una funcion polinomial de una  
    ↪variable dados sus coeficientes """  
    return f  
    def eta_Teorema1(coef):  
        """ Funcion que calcula la variable eta del T1 """  
        return max([abs(ak/coef[0]) for ak in coef[1:]])  
    def Teorema1(f, eta,/, metodo=metodo_secante):  
        """ Impementacion de los conceptos del T1 """  
        t = 2.2e-16**(1/2) # Tolerancia  
        p = 1.0e-2 # Paso entre pruebas; definido  
        ↪arbitrariamente en funcio de la visualizacion de la grafica  
        N = 40 # Por iluminacion divina  
  
        I = [-int(eta+1), int(eta+1)] # Limite del Teorema 1  
        r_min = metodo(f, I[0], I[0]+2, t, N, False)[0]  
        r_max = metodo(f, I[1]-2, I[1], t, N, False)[0]  
        ret = [r_min]  
  
        if not (r_max <= r_min): # Buscamos mas  
            xk = r_min+p  
            fa = f(r_min+p) # Evaluacion de raiz anterior  
  
            while xk < r_max: # Iteracion principal  
                xk += p # Avanzamos un paso entre rmin y rmax  
                if f(xk)*fa < 0: # Buscamos cambio de signo  
                    ret.append(metodo(f,xk-p,xk,t, N,  
        ↪False)[0])  
                    fa = f(xk)  
                    ret.append(r_max)  
  
        return ret
```

```
[74]: def Ejercicio4():  
    coef = [6, -25, -24, 110, -72, 320]  
    eta = eta_Teorema1(coef)  
    f = f_polinomio(coef)  
    raices = Teorema1(f, eta)  
  
    """ Grafica del resultado"""  
  
    return raices  
  
Ejercicio4()
```

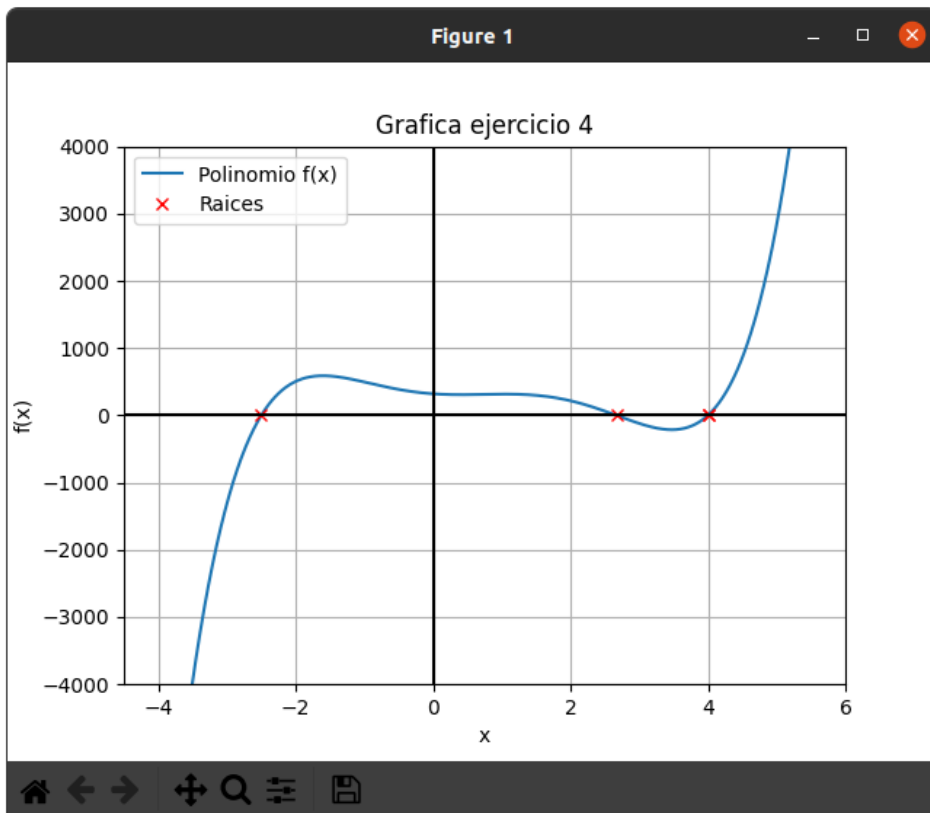


#### 1.4.1. Como ejecutar

Para ejecutar este ejercicio en **consola** es importante ubicarse en la misma carpeta del [archivo Tarea2.py](#) y ejecutar el siguiente comando en consola

```
python3 Tarea2.py Ejercicio4
```

Este programa no espera recibir argumento alguno. La salida debe ser similar a la siguiente imagen



## 1.5. Ejercicio 5

### Respuesta

```
[75]: def metodo_horner(n, a, x0,/,v=True, td=np.float64):  
    """ Implementacion del metodo de horner, basado  
    en el pseudocodigo del documento de la tarea.  
    Input  
        n := Grado del polinomio  
        a := Arreglo de coeficientes  
        x0 := Punto inicial del metodo  
    Output  
        (y, b)  
        y := evaluacion final  
        b := coeficientes de polinomio asociado  
    """  
    x0 = td(x0)  
    a = [td(c) for c in a] # Normalizar el tipo de dato  
    b = a[:] # Reservar memoria xP  
    for k in range(1,n):  
        b[k] = a[k] - b[k-1]*x0  
    return a[-1]-b[-2]*x0  
  
[76]: def Ejercicio5(v=True):  
    """ Ejercicio 6 de Tarea 2 """  
    dt = np.float64  
    coef = [6, -25, -24, 110, -72, 320]  
    coef_dx = [30, -100, -72, 220, -72]  
    muestras = 20  
  
    Ep = 0; Ed = 0  
    xs = np.linspace(-5, 5, num=muestras, dtype=dt)  
    f = f_polinomio(coef)  
    df = f_polinomio(coef_dx)  
  
    for k in range(20):  
        px = metodo_horner(len(coef), coef, xs[k])  
        dpx = metodo_horner(len(coef)-1, coef, xs[k])  
        dif_p = (f(xs[k]) - px)  
        dif_d = (df(xs[k]) - dpx)  
  
        Ep += abs(dif_p)  
        Ed += abs(dif_d)  
        if v: print(...)  
    if v: print(f'\n---FIN---\n'  
               f'Ep={Ep/20} | Ed={Ed/20}')  
    return (Ep/20, Ed/20)  
  
print(Ejercicio5())
```

```

k= 0
px= 2835.000000 | dif_p=-30780.000000
dpx= 2835.000000 | dif_d=25443.000000
k= 1
px= 788.425780 | dif_p=-16561.514891
dpx= 788.425780 | dif_d=17684.557377
k= 2
px= -45.844732 | dif_p=-7979.828634
dpx= -45.844732 | dif_d=11417.915526
k= 3
px= -212.626585 | dif_p=-3208.668531
dpx= -212.626585 | dif_d= 6658.427630
k= 4
px= -84.690012 | dif_p= -857.931844
dpx= -84.690012 | dif_d= 3304.650249
k= 5
px= 108.161574 | dif_p= 84.470201
dpx= 108.161574 | dif_d= 1167.421318
k= 6
px= 250.013584 | dif_p= 310.768406
dpx= 250.013584 | dif_d= -1.061857
k= 7
px= 309.762253 | dif_p= 251.491600
dpx= 309.762253 | dif_d= -478.164598
k= 8
px= 312.036645 | dif_p= 133.622630
dpx= 312.036645 | dif_d= -541.736862
k= 9
px= 308.120655 | dif_p= 38.754355
dpx= 308.120655 | dif_d= -441.035243
k= 10
px= 346.875010 | dif_p= -38.754355
dpx= 346.875010 | dif_d= -367.644970
k= 11
px= 445.659275 | dif_p= -133.622630
dpx= 445.659275 | dif_d= -426.401918
k= 12
px= 561.253853 | dif_p= -251.491600
dpx= 561.253853 | dif_d= -606.314603
k= 13
px= 560.781990 | dif_p= -310.768406
dpx= 560.781990 | dif_d= -751.486190
k= 14
px= 192.631775 | dif_p= -84.470201
dpx= 192.631775 | dif_d= -532.036491
k= 15
px= -942.621856 | dif_p= 857.931844
dpx= -942.621856 | dif_d= 584.976028
k= 16
px=-3421.295116 | dif_p= 3208.668531
dpx=-3421.295116 | dif_d= 3364.632245
k= 17
px=-8025.673366 | dif_p= 7979.828634
dpx=-8025.673366 | dif_d= 8833.230383
k= 18
px=-15773.089111 | dif_p=16561.514891
dpx=-15773.089111 | dif_d=18307.364002
k= 19
px=-27945.000000 | dif_p=30780.000000
dpx=-27945.000000 | dif_d=33423.000000

---FIN---
Ep=6020.705109125281 | Ed=6716.752874460997
(6020.705109125281, 6716.752874460997)

```

```

[77]: # alternativas
      """ Alternativa a np.linspace """
def linspace(start, stop, n):
    """ Programa que entrega un arreglo de numeros
    equitativamnte espaciado de n elementos empe_
    zando en start y hasta stop.

    _Doctest
    >>> linspace(-5,5,3)
    [-5.0, 0.0, 5.0]
    """
    step = (stop-start)/(n-1)
    return [start+step*(i) for i in range(n)]

```

### 1.5.1. Ejercicio 5

Para ejecutar este ejercicio en **consola** es importante ubicarse en la misma carpeta del [archivo Tarea2.py](#) y ejecutar el siguiente comando en consola

```
python3 Tarea2.py Ejercicio5
```

Este programa no espera recibir argumento alguno. La salida debe ser similar a la siguiente imagen

```
bench@bench-dell: ~/Documents/Acade...
python3 Tarea2.py Ejercicio5
k= 0
px= 2835.000000 | dif_p=-30780.000000
dpx= 2835.000000 | dif_d=25443.000000
k= 1
px= 788.425780 | dif_p=-16561.514891
dpx= 788.425780 | dif_d=17684.557377
k= 2
px= -45.844732 | dif_p=-7979.828634
dpx= -45.844732 | dif_d=11417.915526
k= 3
px= -212.626585 | dif_p=-3208.668531
dpx= -212.626585 | dif_d= 6658.427630
k= 4
px= -84.690012 | dif_p= -857.931844
dpx= -84.690012 | dif_d= 3304.650249
k= 5
px= 108.161574 | dif_p= 84.470201
dpx= 108.161574 | dif_d= 1167.421318
k= 6
px= 250.013584 | dif_p= 310.768406
dpx= 250.013584 | dif_d= -1.061857
k= 7
px= 309.762253 | dif_p= 251.491600
dpx= 309.762253 | dif_d= -478.164598
k= 8
px= 312.036645 | dif_p= 133.622630
dpx= 312.036645 | dif_d= -541.736862
k= 9
px= 308.120655 | dif_p= 38.754355
dpx= 308.120655 | dif_d= -441.035243
k= 10
px= 346.875010 | dif_p= -38.754355
dpx= 346.875010 | dif_d= -367.644970
k= 11
px= 445.659275 | dif_p= -133.622630
dpx= 445.659275 | dif_d= -426.401918
k= 12
px= 561.253853 | dif_p= -251.491600
dpx= 561.253853 | dif_d= -606.314603
k= 13
px= 560.781990 | dif_p= -310.768406
dpx= 560.781990 | dif_d= -751.486190
k= 14
px= 192.631775 | dif_p= -84.470201
dpx= 192.631775 | dif_d= -532.036491
k= 15
px= -942.621856 | dif_p= 857.931844
dpx= -942.621856 | dif_d= 584.976028
k= 16
px=-3421.295116 | dif_p= 3208.668531
dpx=-3421.295116 | dif_d= 3364.632245
k= 17
px=-8025.673366 | dif_p= 7979.828634
dpx=-8025.673366 | dif_d= 8833.230383
k= 18
px=-15773.089111 | dif_p=16561.514891
dpx=-15773.089111 | dif_d=18307.364002
k= 19
px=-27945.000000 | dif_p=30780.000000
dpx=-27945.000000 | dif_d=33423.000000
---FIN---
Ep=6020.705109125281 | Ed=6716.752874460997
(6020.705109125281, 6716.752874460997)
bench@bench-dell: ~/Documents/Academico/UG/LicenciaturaDEMAT/GitHubRepo/MN/Tareas/T2S
```