Trabajo práctico 1

Análisis Numérico - Curso Sassano - Grupo 13

Octubre 2018

Integrantes

Nombre	Padrón	Correo electrónico
Manuel Anderson	101230	manuel121097@gmail.com
Martin Cohen	100812	martincohen98@gmail.com
Maximiliano Ruiz Vallejo	100485	maxieco73@gmail.com

Ejercicio

1. Para las siguientes funciones continuas y con raíz única en el intervalo $[0, 2]$:

$$f_1(x) = x^2 - 2$$

$$f_2(x) = x^5 - 6.6x^4 + 5.12x^3 + 21.312x^2 - 38.016x + 17.28$$

$$f_3(x) = (x - 1.5)\exp(-4(x - 1.5)^2)$$

se pide:

(a) Graficar las funciones en el intervalo de interés

(b) Halle para cada una de ellas la raíz en el intervalo indicado mediante los métodos vistos en clase (Bisección, Newton-Raphson, Newton-Raphson modificado, Secante). Use para todos los métodos como criterio de parada las siguientes cotas: $1 \cdot 10^{-5}$, $1 \cdot 10^{-13}$, para Newton-Raphson use semilla $x_0 = 1.0$, para secante use como semillas los extremos del intervalo.

(c) Halle la raíz mediante la función de búsqueda de raíces de un lenguaje o paquete orientado a cálculo numérico (e.g. Octave: `fzero`, Python+SciPy: `scipy.optimize.brentq`).

(d) Compare los resultados obtenidos para los distintos métodos y cotas. Discuta ventajas y desventajas. ¿Son las que esperaba en base a la teoría?

Ayuda

Si necesita derivadas (–revisar–):

$$f_1^{'}(x) = 2x$$

$$f_1^{''}(x) = 2$$

$$f_2^{'}(x) = 5x^4 - 26.4x^3 + 15.36x^2 + 42.624x - 38.016$$

$$f_2^{''}(x) = 20x^3 - 79.2x^2 + 30.72x + 42.624$$

$$f_3^{'}(x) = (-8x + 12.0)(x - 1.5)\exp(-4(x - 1.5)^2) + \exp(-4(x - 1.5)^2)$$

$$f_3^{''}(x) = (-24x + (x - 1.5)(8x - 12.0)^2 + 36.0)\exp(-4(x - 1.5)^2)$$

1. Introducción

1.1. Objetivo:

Tomar tres funciones, de diferente complejidad, para poder comparar la efectividad de los métodos de búsqueda de raíces, Bisección, Newton-Raphson, Newton-Raphson modificado y Secante, usando como criterio los pasos efectuados por sus algoritmos desarrollados en Python, para obtener una raíz, con una cierta cota de error máxima.

1.2. Descripción de los métodos:

1.2.1. Bisección:

Si se tiene una función F continua en un intervalo cerrado [a,b], donde el producto de F(a) y F(b) es menor a 0 (Condición 1), entonces podemos decir que en dicho intervalo se halla una raíz, un punto c donde F(c)=0. El método se basa en la reducción del intervalo [a,b] tanto como se quiera, proponiendo nuevos límites para el mismo, dividiendo el intervalo a la mitad y asegurándose que en el nuevo intervalo a tomar, se cumpla la condicion 1. [2]

1.2.2. Newton-Raphson

La forma más sencilla de interpretar este método es como una derivación del polinomio de Taylor. Tomando una función F (continua en un intervalo [a,b]), la cual debe cumplir que su derivada sea no nula en [a,b], y un punto p que suponemos es su raíz. [2]

Si planteamos el polinomio de Taylor (P(X)) de dicha función en x (punto talque la distancia entre p y x sea muy pequeña), sabiendo que P(p)=0, ya que p es raíz de F, y despreciando el término de orden dos del mismo, podemos deducir que:

$$p = X - \frac{F(x)}{Q(x)} \tag{1}$$

Siendo, p la raíz aproximada de F, F(x) la función a la que se quiere hallar las raíces, x una semilla y Q(x) la derivada de orden 1 de F(x).

Para el método de **Newton-Raphson modificado**, se define la función:

$$\mu(x) = \frac{f(x)}{f'(x)} \tag{2}$$

Luego se aplica el método con esta función en lugar de con f, para que el método converja cuadraticamente para raíces múltiples.

1.2.3. Secante:

Este método permite una aproximación de la raíz a buscar mediante la traza de rectas secantes entre un punto x0 y x1, cuya condición es que F(x1).F(x0) sea menor a 0. Una vez trazada la secante se utiliza la intersección de la misma con el eje x como un punto x2, para el cual F(x2) es menor a 0, por lo que, x1 puede usarse en reemplazo de x0, para repetir la secuencia descripta hasta alcanzar una cota de error deseada. [2]

La siguiente fórmula es la utilizada para llevar lo dicho a una forma analítica, donde x0,x1 y x2, cambian conforme avanzan la iteraciones, como se describió con anterioridad:

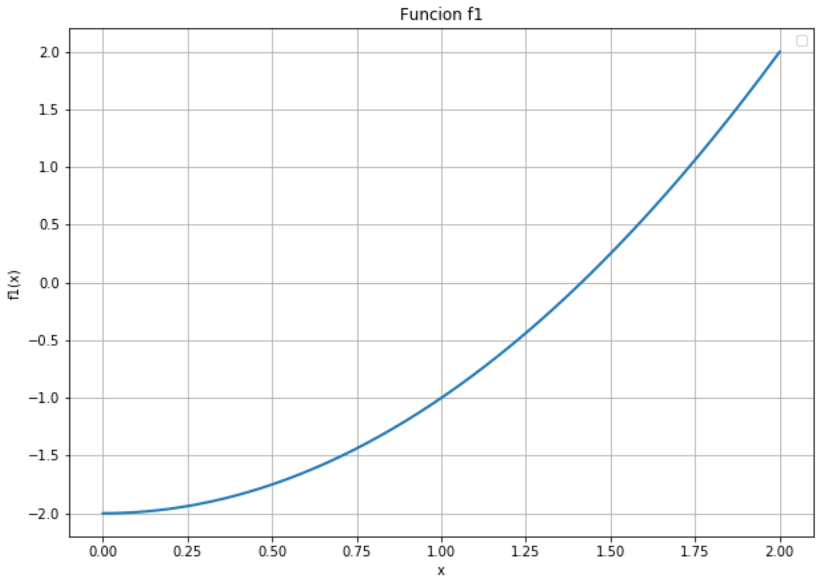
$$X2 = X1 - \frac{F(x1)(x1 - x0)}{F(x1) - f(x0)} \tag{3}$$

2. Desarrollo

Vamos a dividir esta sección en partes, analizando los 4 métodos usados para las 3 funciones presentadas en el enunciado.

2.1. Función f_1

2.1.1. Gráfico de la función

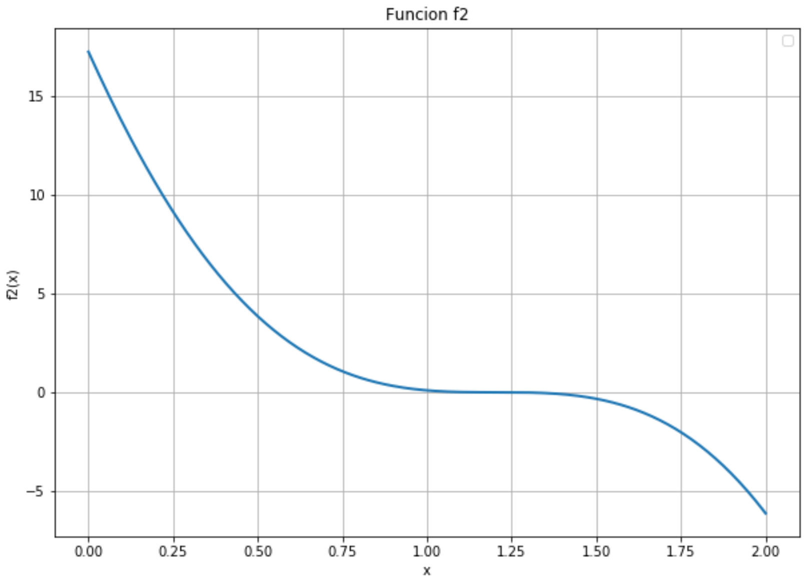


2.1.2. Tabla de resultados

Función	Algoritmo	Cota de error	Pasos necesarios	Raiz Encontrada
f_1	Brent(SciPy)	—	8	1.4142135623731364
f_1	Bisección	$1 \cdot 10^{-5}$	17	1.4142074584960938
f_1	Secante	$1 \cdot 10^{-5}$	6	1.4142135626888697
f_1	Newton-Raphson	$1 \cdot 10^{-5}$	4	1.4142135623746899
f_1	Newton-Raphson modificado	$1 \cdot 10^{-5}$	4	1.4142135623715002
f_1	Bisección	$1 \cdot 10^{-13}$	44	1.4142135623730496
f_1	Secante	$1 \cdot 10^{-13}$	8	1.4142135623730950
f_1	Newton-Raphson	$1 \cdot 10^{-13}$	6	1.4142135623730950
f_1	Newton-Raphson modificado	$1 \cdot 10^{-13}$	6	1.4142135623730951

2.2. Función f_2

2.2.1. Gráfico de la función

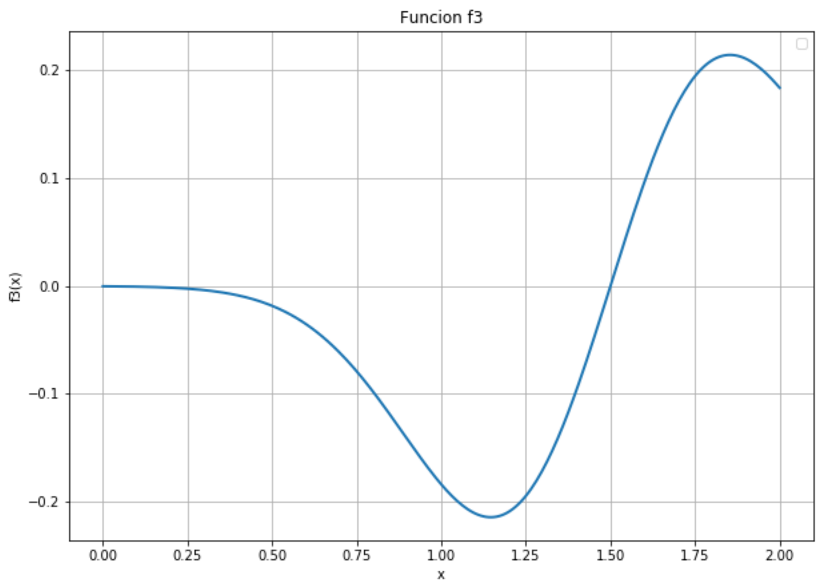


2.2.2. Tabla de resultados

Función	Algoritmo	Cota de error	Pasos necesarios	Raíz Encontrada
f_2	Brent(SciPy)	—	49	1.2000081652661798
f_2	Bisección	$1 \cdot 10^{-5}$	17	1.2000045776367188
f_2	Secante	$1 \cdot 10^{-5}$	34	1.2000297021775468
f_2	Newton-Raphson	$1 \cdot 10^{-5}$	23	1.1999826569716792
f_2	Newton-Raphson modificado	$1 \cdot 10^{-5}$	3	1.1999999399606254
f_2	Bisección	$1 \cdot 10^{-13}$	44	1.2000040998499912
f_2	Secante	$1 \cdot 10^{-13}$	41	1.2000070867064747
f_2	Newton-Raphson	$1 \cdot 10^{-13}$	35	1.2000073749969030
f_2	Newton-Raphson modificado	$1 \cdot 10^{-13}$	8	1.1999997119874050

2.3. Función f_3

2.3.1. Gráfico de la función



2.3.2. Tabla de resultados

Función	Algoritmo	Cota de error	Pasos necesarios	Raíz Encontrada
f_3	Brent(SciPy)	—	8	1.5000000000000198
f_3	Bisección	$1 \cdot 10^{-5}$	17	1.4999923706054688
f_3	Secante	$1 \cdot 10^{-5}$	No Converge	—
f_3	Newton-Raphson	$1 \cdot 10^{-5}$	No Converge	—
f_3	Newton-Raphson modificado	$1 \cdot 10^{-5}$	No Converge	—
f_3	Bisección	$1 \cdot 10^{-13}$	44	1.4999999999999432
f_3	Secante	$1 \cdot 10^{-13}$	No Converge	—
f_3	Newton-Raphson	$1 \cdot 10^{-13}$	No Converge	—
f_3	Newton-Raphson modificado	$1 \cdot 10^{-13}$	No Converge	—

3. Conclusión

Todos los algoritmos de búsqueda de raíces tienen sus ventajas y sus desventajas en comparación con las otras, y los resultados entran dentro del marco de posibilidades que se esperaba.

El método de la Bisección, por ejemplo, tiene como ventaja que siempre llega a un resultado, y a este siempre se llega en la misma cantidad de iteraciones para la misma cota de error sin importar la función. Al mismo tiempo es un método que suele requerir una mayor cantidad de iteraciones que los demás en aquellas funciones en las que puede aplicarse otro método.

El método de la Secante en cambio, es en general mucho más rápido que el método de la Bisección, pero no siempre llega a una raíz para todas las funciones, como puede verse en f_3. Esto mismo ocurre con el método de Newton-Raphson, con la diferencia que aunque el método de Newton-Raphson muestra ser algo más rápido que el método de la Secante, es importante tener en cuenta que debe utilizar la derivada de la función. El anterior no es un dato que siempre se puede obtener fácilmente.

El algoritmo de Newton-Raphson modificado tiene las mismas ventajas y desventajas con respecto a los otros métodos que el mismo Newton-Raphson, pero estos tienen una diferencia entre sí que puede verse en f_2. Cuando el algoritmo de Newton-Raphson se aplica para una raíz múltiple (caso de f_2), este no converge cuadraticamente (es decir que llega al resultado más lento). El algoritmo de Newton-Raphson modificado no tiene este problema, por lo que suele llamarse Newton-Raphson para raíces múltiples. Sin embargo, no muestra ninguna ventaja para encontrar raíces simples, y siempre presenta la desventaja de tener que utilizar la derivada segunda de la función.

Referencias

[1] Cheney, W.; Kincaid, D. *Numerical Mathematics and Computing.* 6ta ed. EE.UU.: Thomson Brooks/Cole, 2008.

[2] Burden, R. L,; Faires, J.D. *Análisis Numéirco.* 2da ed. México: Iberoamérica, 1996.

Anexo - Código

El programa principal usado:

```python

#Imports
from scipy.optimize import brentq
import numpy as np
import matplotlib.pylab as plt

#Funciones f1, f2, f3 y derivadas
def f1(x):
    return x**2-2
def df1(x):
    return 2*x
def ddf1(x):
    return 2

def f2(x):
    return x**5 - (6.6)*x**4 + (5.12)*x**3 + (21.312)*x**2 - (38.016*x) + 17.28
def df2(x):
    return (5)*x**4 - (26.4)*x**3 + (15.36)*x**2 + (42.624*x) - 38.016
def ddf2(x):
    return (20)*x**3 - (79.2)*x**2 + (30.72*x) + 42.624

def f3(x):
    return (x - 1.5)*(np.exp((-4*(x - 1.5)**2)))
def df3(x):
    return ((-8*x + 12)*(x - 1.5))*np.exp((-4*(x - 1.5)**2)) + np.exp((-4*(x - 1.5)**2))
def ddf3(x):
    return (-24*x + (x-1.5)*(8*x - 12)**2 + 36)*np.exp((-4*(x - 1.5)**2))

#Funciones busqueda de raices
def bisec(f, a, b, a_tol, n_max):

    x = a+(b-a)/2
    delta = (b-a)/2

    print('{0:^4} {1:^17} {2:^17} {3:^17}'.format('i', 'x', 'a', 'b'))
    print('{0:4} {1: .14f} {2: .14f} {3: .14f}'.format(0, x, a, b))

    for i in range(n_max):
        if f(a) * f(x) > 0:
            a = x
        else:
            b = x
        x_old = x
        x = a+(b-a)/2
        delta = np.abs(x - x_old)

        print('{0:4} {1: .14f} {2: .14f} {3: .14f}'.format(i+1, x, a, b))

        if delta <= a_tol: #Hubo convergencia
            print('Hubo convergencia, n_iter = ' + str(i+1))
            return x, delta, i+1

    #Si todavia no salio es que no hubo convergencia:
    raise ValueError('No hubo convergencia')
    return x, delta, i+1

def secante(f, x0, x1, a_tol, n_max):

```

```
    delta = 0

    print('{0:^4} {1:^17} {2:^17} {3:^17}'.format('i', 'x', 'x_-1', 'delta'))
    print('{0:4} {1: .14f} {2: .14f} {3: .14f}'.format(0, x1, x0, delta))

    for i in range(n_max):
        x = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0))
        delta = np.abs(x - x1)
        x0 = x1
        x1 = x

        print('{0:4} {1: .14f} {2: .14f} {3: .14f}'.format(i+1, x1, x0, delta))

        #Chequear convergencia
        if delta <= a_tol: #Hubo convergencia
            print('Hubo convergencia, n_iter = ' + str(i+1))
            return x, delta, i+1

    #Si todavia no salio es que no hubo convergencia:
    raise ValueError('No hubo convergencia')
    print ('No hubo convergencia')
    return x, delta, i+1

def newtonRaphson(f, df, x0, x1, toleranciaError, maximoIteraciones):

    delta = 0

    x = 1

    print('{0:^4} {1:^17} {2:^17}'.format('i', 'x','delta'))
    print('{0:4} {1: .14f} {2: .14f}'.format(0, x, delta))

    for i in range(maximoIteraciones):

        xSiguiente = x - (f(x)/df(x))
        delta = np.abs(xSiguiente - x)

        print('{0:4} {1: .14f} {2: .14f}'.format(i+1, xSiguiente, delta))

        x = xSiguiente

        #Chequear convergencia
        if delta <= toleranciaError: #Hubo convergencia
            print('Hubo convergencia, n_iter = ' + str(i+1))
            return x, delta, i+1

    #Si todavia no salio es que no hubo convergencia:
    raise ValueError('No hubo convergencia')
    print ('No hubo convergencia')
    return x, delta, i+1

def newtonRaphsonModificado(f, df, ddf, x0, x1, toleranciaError, maximoIteraciones):

    delta = 0

    x = 1

    print('{0:^4} {1:^17} {2:^17}'.format('i', 'x','delta'))
    print('{0:4} {1: .14f} {2: .14f}'.format(0, x, delta))

    for i in range(maximoIteraciones):

```

```
        xSiguiente = x - ((f(x)*df(x))/(df(x)**2-(f(x)*ddf(x))))
        delta = np.abs(xSiguiente - x)

        print('{0:4} {1: .14f} {2: .14f}'.format(i+1, xSiguiente, delta))

        x = xSiguiente

        #Chequear convergencia
        if delta <= toleranciaError: #Hubo convergencia
            print('Hubo convergencia, n_iter = ' + str(i+1))
            return x, delta, i+1

    #Si todavia no salio es que no hubo convergencia:
    #raise ValueError('No hubo convergencia')
    print ('No hubo convergencia')
    return x, delta, i+1

#Imprime en consola los datos obtenidos de cada uno de los metodos de busqueda, acompaado del grafico de la funcion
def imprimirDatos (nombreFuncion,funcion, dFuncion, ddFuncion, xMin, xMax, error1, error2, maximasIteraciones):
    print('----------------')
    print('Metodo biseccion')
    print('----------------')
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error1))
    r, delta, n_iter = bisec(funcion, xMin, xMax, error1, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error2))
    r, delta, n_iter = bisec(funcion, xMin, xMax, error2, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')

    print('----------------')
    print('Metodo secante')
    print('----------------')
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error1))
    r, delta, n_iter = secante(funcion, xMin, xMax, error1, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error2))
    r, delta, n_iter = secante(funcion, xMin, xMax, error2, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')

    print('----------------')
    print('Metodo Newton-Raphson')
    print('----------------')
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error1))
    r, delta, n_iter = newtonRaphson(funcion, dFuncion, xMin, xMax, error1, maximasIteraciones)
    print('raiz = ' +str(r))
```

```
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error2))
    r, delta, n_iter = newtonRaphson(funcion, dFuncion, xMin, xMax, error2, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')

    print('----------------')
    print('Metodo Newton-Raphson para races multiples')
    print('----------------')
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error1))
    r, delta, n_iter = newtonRaphsonModificado(funcion, dFuncion, ddFuncion, xMin, xMax,
        error1, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')
    print('Funcion ' + nombreFuncion +', a_tol = '+str(error2))
    r, delta, n_iter = newtonRaphsonModificado(funcion, dFuncion, ddFuncion, xMin, xMax,
        error2, maximasIteraciones)
    print('raiz = ' +str(r))
    print('delta= ' +str(delta))
    print('n_ite= ' +str(n_iter))
    print('')

    print('----------------')
    print('Metodo brent')
    print('----------------')
    print('')
    print('Funcion ' + nombreFuncion +', a_tol por defecto para la funcion')
    r, results = brentq(funcion, xMin, xMax, full_output=True)
    print('raiz = ' +str(r))
    print('Resultados: ')
    print(results)

    #Grafica de las funciones
    xx = np.linspace(xMin, xMax, 256+1)
    yy = funcion(xx)
    nombre = nombreFuncion
    plt.figure(figsize=(10,7))
    plt.plot(xx, yy, lw=2)
    plt.legend(loc='best')
    plt.xlabel('x')
    plt.ylabel(nombre +'(x)')
    plt.title('Funcion '+ nombre)
    plt.grid(True)
    plt.savefig(nombre + '.png')
    plt.show()

#Intervalo para buscar raiz
xMin = 0.0
xMax = 2.0

#Parametros para el algoritmo
error1 = 1e-5
error2 = 1e-13
maximasIteraciones = 100
```

```
imprimirDatos("f1", f1, df1, ddf1, xMin, xMax, error1, error2, maximasIteraciones)
imprimirDatos("f2", f2, df2, ddf2, xMin, xMax, error1, error2, maximasIteraciones)
imprimirDatos("f3", f3, df3, ddf3, xMin, xMax, error1, error2, maximasIteraciones)
```