ANALISIS DE METODOS NUMERICOS

Y. F. Santos

Universidad Industrial de Santander, escuela de física, cll 9 con 27

Resumen

En el presente documento se presentan diferentes métodos de raíces como lo son el método de Newton – Raphson, el de Posición falsa, el de la secante, y el método de Punto fijo, para un polinomio en particular, con lo cual se busca verificar la veracidad de cada método y su eficiencia por medio de un análisis de tolerancia o error en cada interacción por medio del lenguaje de python.

1. Introducción.

En un mundo tan cambiante como el nuestro, es de gran utilidad y eficiencia el análisis computacional tanto en la ciencia como la ingeniería. la información aumenta rápidamente en nuestros medios de comunicación puesto que se requieren de aparatos o dispositivos electrónicos para encontrar o dar información con rapidez.

Esta herramienta está incorporada en prácticamente todas las actividades de la vida cotidiana. Por ello nace la necesidad de las personas aprendan a utilizar esta herramienta para aprovechar sus ventajas, aumentar la productividad y eficacia en las tareas realizadas, en el caso de los ingenieros y científicos la computación es el pan de cada día.

En la formación escolar de un científico o un ingeniero la búsqueda de soluciones de una ecuación es fundamental y en la mayoría de ocasiones no es posible su solución por métodos analíticos por tal razón requieren el método numérico, el cual es un procedimiento en el que se obtiene, casi siempre de manera aproximada, la solución de ciertos problemas realizando cálculos puramente aritméticos y lógicos (operaciones aritméticas elementales, cálculo de funciones, consulta de una tabla de valores, cálculo preposicional, etc.). Un tal procedimiento consiste de una lista finita de instrucciones precisas que especifican una secuencia de operaciones algebraicas y lógicas (algoritmo), que producen o bien una aproximación de la solución del problema (solución numérica) o bien un mensaje. La eficiencia en el cálculo de dicha aproximación depende, en parte, de la facilidad de implementación del algoritmo y de las características especiales y limitaciones de los instrumentos de cálculo (los computadores). En general, al emplear estos instrumentos de cálculo se introducen errores llamados de redondeo.

En el presente documento se estudian y analizan cuatro diferentes métodos de solución o búsqueda de raíces de un polinomio en particular, primero se hallarán las raíces en cada método, luego se graficará según el caso el método y la función, y por último se mostrará un análisis de errores verificando su eficiencia y velocidad al momento de su implementación.

2. Métodos de estudio.

Para la búsqueda de raíces se estudiarán los diferentes métodos de acuerdo a la siguiente función.

$$f(x) = x^3 + 4x^2 - 10 \qquad (1)$$

Cuya grafica asociada se presenta en la fig 1. En la cual se observa que presenta una raíz en la posición $x = 1.3652$.

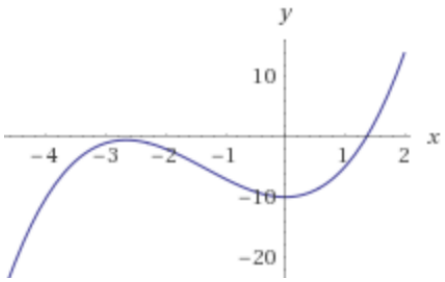

Fig 1. Grafica de la función (1).

2.1. Método de Newton – Raphson.

El algoritmo de este método consiste en una relación de recurrencia. Donde de acuerdo a la función (1) su derivada es.

$$f'(x) = 3(x^2) + 8x \qquad (2)$$

Y partiendo de un valor inicial (3) muy cercano a la raíz para garantizar la convergencia rápida.

$$x_i = 2 \qquad (3)$$

Este valor es obtenido por medio de la fig. 1.

Ahora definimos para cada número natural n

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (4)$$

Donde f' denota la derivada de f.

Y determinar cuando $|x_{n+1} - x_n| < tolerancia$.

El error de este método es del orden $O(h^2)$. Para visualizar esto retomamos la forma

$$\propto + \epsilon_{n+1} = \emptyset(\alpha + \epsilon_n)$$
$$= \propto + \epsilon_n - \frac{F(\propto + \epsilon_n)}{F'(\alpha + \epsilon_n)} \qquad (5)$$

Y procedemos de manera análoga al caso del método de Euler.

2.2. Método de la secante.

Un problema potencial en la implementación del método de Newton es la evaluación de la derivada. Aunque esto no es un inconveniente para los polinomios no para muchas otras funciones, existen algunas funciones cuyas derivadas en ocasiones resultan muy difíciles de calcular. En dichos casos, la derivada se puede aproximar mediante el siguiente cociente incremental:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \qquad (6)$$

Entonces, la fórmula para calcular el valor aproximado de la raíz en la iteración n+1 es:

$$x_{n+1} = x_n - \frac{f(x_n) * (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \qquad (7)$$

El algoritmo que se obtiene al aplicar esta fórmula se conoce como el método de la secante. Si bien este método requiere de dos valores iniciales, no es necesario que f(x) cambie de signo entre los valores dados.

2.3. Método de posición falsa.

El método de la regula falsi (regla del falso) o falsa posición es un método iterativo de resolución numérica de ecuaciones no lineales. El método combina el método de bisección y el método de la secante.
Como en el método de bisección, se parte de un intervalo inicial $[a_o, b_0]$ con $f(a_0)$ y $f(b_0)$ de signos opuestos, lo que garantiza que en su interior hay al menos una raíz (véase Teorema de Bolzano). El algoritmo va obteniendo sucesivamente en cada paso un intervalo más pequeño $[a_k, b_k]$ que sigue incluyendo una raíz de la función f.

A partir de un intervalo $[a_k, b_k]$ se calcula un punto interior c_k:

$$c_k = \frac{f(b_k)a_k - f(a_k)b_k}{f(b_k) - f(a_k)} \qquad (8)$$

Dicho punto es la intersección de la recta que pasa por $(a, f(a_k))$ y $(b, f(b_k))$ con el eje de abscisas (igual a como se hace en el método de la secante).

Se evalúa entonces $f(c_k)$. Si es suficientemente pequeño, c_k es la raíz buscada. Si no, el próximo intervalo $[a_{k+1}, b_{k+1}]$ será:

- $[a_k, c_k$ si $f(a_k)$ y $f(c_k)$ tienen signos opuestos;
- $[c_k, b_k]$ en caso contrario.

2.4. Método del punto fijo.

es un método iterativo que permite resolver sistemas de ecuaciones no necesariamente lineales. En particular se puede utilizar para determinar raíces de una función de la forma $f(x)$, siempre y cuando se cumplan los criterios de convergencia.

también denominado método de aproximación sucesiva, requiere volver a escribir la ecuación $f(x) = 0$ en la forma de $x = g(x)$

El procedimiento empieza con una estimación o conjetura inicial de x, que es mejorada por iteración hasta alcanzar la convergencia. Para que converja, la derivada dg/dx debe ser menor que 1 en magnitud (al menos para los valores x que se encuentran durante las iteraciones). La convergencia será establecida mediante el requisito de que el cambio en x de una iteración a la siguiente no sea mayor en magnitud que alguna pequeña cantidad ε.

3. Analisis de resultados.

Los codigos estudiados se presentan a continuacion.

3.1. Algoritmo de newton raphson para encontrar raices es:

```
import numpy as np
import matplotlib as plt

funcionx=lambda x:x**3+4*x**2-10
fxderiva=lambda x:3*(x**2)+8*x

#ingresamos datos
xi=2
tolera=0.001

#busqueda de raices
tabla=[]
tramo=abs(2*tolera)
while(tramo>=tolera):
    xnuevo=xi-funcionx(xi)/fxderiva(xi)
    tramo=abs(xnuevo-xi)
    tabla.append([xi,xnuevo,tramo])
    xi=xnuevo

tabla=np.array(tabla)
n=len(tabla)

#imprimimos
print(['xi','xnuevo','tramo'])
np.set_printoptions(precision=4)
for i in range(0,n,1):
    print(tabla[i])
print('raiz en: ',xi)

error=abs(xnuevo-xi)/xi
```

Cuyo resultado en la salida muestra:

```
['xi', 'xnuevo', 'tramo']
[2.  1.5 0.5]
[1.5    1.3733 0.1267]
[1.3733 1.3653 0.0081]
[1.3653e+00 1.3652e+00 3.2001e-05]
raiz en:  1.3652300139161466
```

Donde se puede observar que el valor exacto de la raiz es 1.36523 con 4 interacciones

3.2. Algoritmo de la secante para encontrar raices es:

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


funcionx=lambda x:x**3+4*x**2-10
x=np.linspace(0,9,100)
plt.plot(x,x**3+4*x**2-10)       # graficamos la 


def secante_tabla(funcionx,xa,tolera):
    dx=4*tolera              #delta de x
    xb=xa+dx                 #un paso mas que xa
    tramo=dx
    tabla=[]
    while (tramo>=tolera):
        fa=funcionx(xa)
        fb=funcionx(xb)
        xc=xa-fa*(xb-xa)/(fb-fa)
        tramo=abs(xc-xa)

        tabla.append([xa,xb,xc,tramo])
        xb=xa
        xa=xc
    tabla=np.array(tabla)
    return(tabla)
```

```
    #ingresamos datos
a=1
b=2
xa=1.5
tolera=0.001
tramos=100

tabla=secante_tabla(funcionx,xa,tolera)
n=len(tabla)
raiz=tabla[n-1,2]

np.set_printoptions(precision=4)
print('[xa,\t xb,\t xc,\t tramo]')
for i in range(0,n,1):
    print(tabla[i])
print('raiz en: ',raiz)


# añadiendo la grafica
xi=np.linspace(a,b,tramos+1)
fi=funcionx(xi)
dx=(b-xa)/2
m=(funcionx(xa+dx)-funcionx(xa))/(xa+dx-xa)
b0=funcionx(xa)-m*xa
tangentei=m*xi+b0
fxa=funcionx(xa)
xb=xa+dx
fxb=funcionx(xb)
```

```
plt.plot(xi,fi,label='f(x)')
plt.plot(xi,tangentei,label='secante')
plt.plot(xa,funcionx(xa),'go',label='xa')
plt.plot(xa+dx,funcionx(xa+dx),'ro',label='xb')
plt.plot((-b0/m),0,'yo',label='xc')
plt.plot([xa,xa],[0,fxa],'pendiente')
plt.plot([xb,xb],[0,fxb],'pendiente')
plt.axhline(0,color='k')
plt.title('Metodo de la secante')
plt.show()
```

Cuyo resultado obtenido es:

```
[xa,     xb,     xc,     tramo]
[1.5    1.504  1.3736 0.1264]
[1.3736 1.5    1.3658 0.0078]
[1.3658e+00 1.3736e+00 1.3652e+00 5.2085e-04]
raiz en:  1.3652321429167764
```

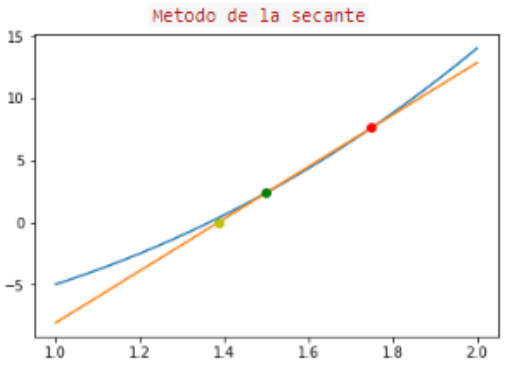

Fig.2. metodo de la secante.

En la figura 2 el punto rojo hace referencia al punto xa y el punto xb es el de color verde oscuro y el color amarillo.

3.3. Algoritmo de la posicion falsa para encontrar raices es:

```
import numpy as np

funcionx=lambda x:x**3+4*x**2-10

#ingresamos datos
a=1
b=2
tolera=0.001

#busqueda de raices
tabla=[]
fa=funcionx(a)
fb=funcionx(b)
c=b-fb*(a-b)/(fa-fb)
fc=funcionx(c)
tramo=abs(c-a)
tabla=[[a,b,c,fa,fb,fc,tramo]]

while(tramo>tolera):
    cambia=np.sign(fa)*np.sign(fc)
    if(cambia>0):
        a=c
    else:
        b=c
    fa=funcionx(a)
    fb=funcionx(b)
    c=b-fb*(a-b)/(fa-fb)
    fc=funcionx(c)
    tramo=abs(c-a)
    tabla.append([a,c,b,fa,fb,fc,tramo])
```

```
tabla=np.array(tabla)
resultado=c

#buscando la respuesta
fa=funcionx(a)
fb=funcionx(b)
cambia=np.sign(fa)*np.sign(fb)
if(cambia>0):
    resultado=np.nan

#imprimimos
print('[a,\t b, \t c, \t f(a), \t f(b), \t f(c),tramo]')
n=len(tabla)
np.set_printoptions(precision=4)
for i in range(0,n,1):
    print(tabla[i])
print('la raiz es:', resultado)
```

Cuyo resultado es:

```
[a,      b,      c,      f(a),   f(b),   f(c),tramo]
[ 1.      2.      1.2632 -5.     14.     -1.6023  0.2632]
[ 1.2632  1.3388  2.     -1.6023 14.     -0.4304  0.0757]
[ 1.3388  1.3585  2.     -0.4304 14.     -0.11    0.0197]
[ 1.3585e+00  1.3635e+00  2.0000e+00 -1.1001e-01  1.4000e+01 -2.776
  5.0011e-03]
[ 1.3635e+00  1.3648e+00  2.0000e+00 -2.7762e-02  1.4000e+01 -6.983
  1.2596e-03]
[ 1.3648e+00  1.3651e+00  2.0000e+00 -6.9834e-03  1.4000e+01 -1.755
  3.1669e-04]
la raiz es: 1.3651237178843778
```

3.4. **Algoritmo de punto fijo para encontrar raices es:**

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

funcionx=lambda x:np.exp(-x)-x
funciongx=lambda x:np.exp(-x)

#ingresamos datos
a=0
b=1
tolera=0.001
tramos=101

def puntofijo(gx,a,tolera,n=15):
    i=1
    b=gx(a)
    tramo=abs(b-a)
    while(tramo>=tolera and i<=n):
        a=b
        b=gx(a)
        tramo=abs(b-a)
        i=i+1
    respuesta=b

#buscando la respuesta

    if(i>=n):
        respuesta=np.nan
    return(respuesta)
```

```
respuesta=puntofijo(funciongx,a,tolera)


#buscando la respuesta

print('la raiz es:',respuesta)

#graficamos

xi=np.linspace(a,b,tramos)
fi=funcionx(xi)
gi=funciongx(xi)
yi=xi
plt.plot(xi,fi,label='f(x)',color='r')
plt.plot(xi,gi,label='g(x)',color='b')
plt.plot(xi,yi,label='y=x',color='y')
plt.axvline(respuesta)
plt.title('Punto Fijo')
plt.show()
```

Cuyo resultado es:

```
la raiz es: 0.5669089119214953
```

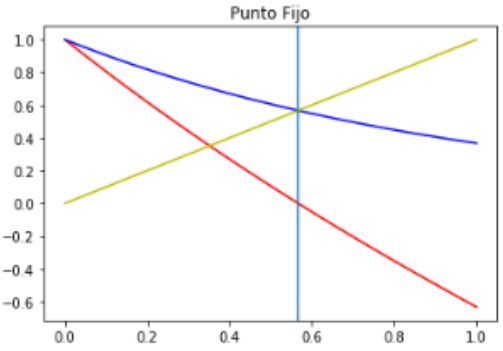

Fig. 3. Metodo de punto fijo.

Donde la funcion de color rojo es la funcion f(x), y g(x) es la azul y la funcion de color amarillo representa f(x)=x.

4. Conclusión.

Se puede observar que el método de newton raphson y el método de la secante son muy cercanos en su precisión al encontrar la raíz, pero el método de la posición falsa difiere de 1 milésima en comparación con los otros dos métodos.

Analizaremos las diferencias entre los métodos.

Diferencia entre los métodos de la secante y de regla falsa.

Como se puede observar, las expresiones que permiten calcular cada aproximación en los métodos de la secante y Regula - Falsi son idénticas en

todos los términos. Ambas usan dos valores iniciales para calcular una aproximación de la pendiente de la función que se utiliza para proyectar hacia el eje x una nueva aproximación de la raíz. Sin embargo, existe una diferencia importante entre ambos métodos. Esta diferencia radica en la forma en que uno de los valores iniciales se reemplaza por la nueva aproximación. En el método de Regula - Falsi, la última aproximación x_i de la raíz reemplaza cualquiera de los valores iniciales que dé un valor de la función con el mismo signo que $f(x_i)$. En consecuencia, las dos aproximaciones siempre encierran a la raíz. Por lo tanto, para todos los casos, el método siempre converge debido a que la raíz se encuentra dentro del intervalo.

En el caso del método de newton es eficiente en la solución de sistemas de ecuaciones no lineales, converge muy rápidamente y proporciona una muy buena precisión en los resultados. el método se emplea en la solución de problemas académicos y propios del mundo real.

En el caso del metodo del punto fijo, no es necesario poner o definir un punto cerca a la raiz, aunque en este trabajo no se puede comparar con los metodos de newton y la secante debido a que la funcion utilizada es diferente por facilidad de encontrar la inversa de la funcion. Posee condiciones para asegurar la convergencia. Es condición necesaria que $|g'(x)| < 1$ en cercanías de la raíz

En el método de la Secante los dos puntos iniciales no necesariamente encierran a la raíz buscada lo que puede provocar divergencia del método.