Método de Punto Fijo

Fernando González Alejandro Madrazo

Noviembre 2019

1 Resumen

El objetivo de este documento científico es demostrar el método de Punto Fijo como una aproximación bastante precisa a una función no lineal de forma $f(x) = 0$, aplicando el método iterativo a código. Este teorema solo requiere que la función sea continua, y el valor numérico de la solución esta determinado por un proceso iterativo, en el cual, la función iterativa podrá divergir o converger según la función.

2 Palabras Clave

Teorema de Punto Fijo, Función no lineal, Teorema del valor medio.

3 Introducción

En una función $g : R \to R$, un valor x tal que: $x = g(x)$, cuando x es la solución de $f(x) = 0$, se le llama Punto Fijo de la función g, ya que x permanece constante cuando se le aplica g. Este método nos ayuda a modificar una función no lineal y adaptarla como un problema de punto fijo para una función no lineal relacionada. Cuando x es la solución de $f(x)$, ambos lados de la ecuación $x = g(x)$ son iguales. Gráficamente, podemos observar que al trazar $y = x$ y $y = g(x)$, nos da el punto de intersección de las dos gráficas, llamado punto fijo, que es la solución.

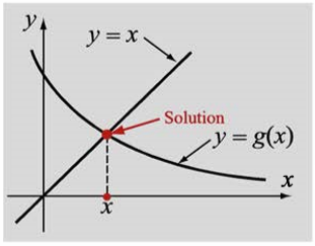

Figure 1: Método iterativo de punto fijo — Gilar, Amos

4 Contenido

El valor numérico de la solución está determinado por un proceso iterativo. Este proceso iterativo comienza tomando un valor de x cerca del punto fijo, como estimación de la solución, y sustituyéndola en $g(x)$. El valor de $g(x)$ que se obtiene, es la nueva estimación (segundo valor) de la solución. Este segundo valor, se sustituye por x en la función $g(x)$, que luego dará la nueva estimación (tercer valor) de la solución, y así sucesivamente, hasta que el método converge. La formula de iteración esta dada por:

$$x_{i+1} = g(x_i)$$

Donde x_{i+1} es el nuevo valor de estimación, y $g(x_i)$ es el valor viejo o el actual de estimación. La función $g(x)$ es llamada la función de iteración.

Cuando el método funciona, los valores de x que se obtienen son iteraciones sucesivas que progresivamente convergen hacia la solución.

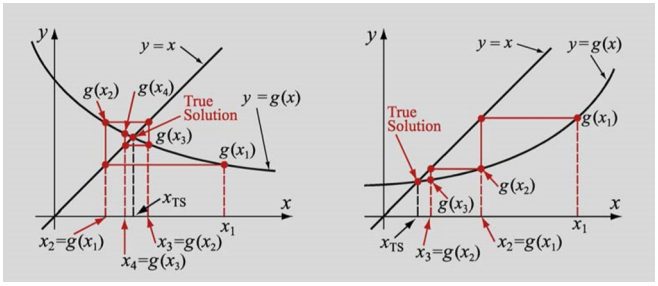

Figure 2: Convergencia del Método iterativo de punto fijo — Gilar, Amos

Por otro lado, es posible que las iteraciones no converjan hacia el punto fijo, sino que diverjan de la solución. En la Figura 3, se puede observar como los puntos se empiezan a mover lejos de la solución.

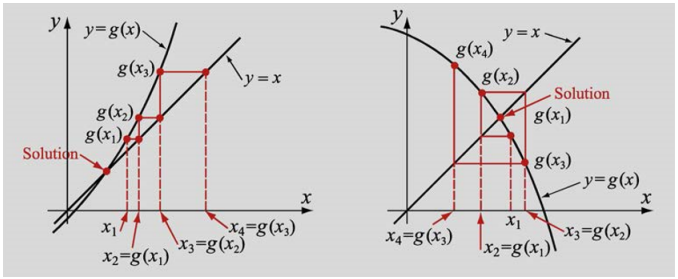

Figure 3: Divergencia del Método iterativo de punto fijo — Gilar, Amos

El proceso de solución es el siguiente:

1. Comienza seleccionando el punto x_1 en el eje x y dibujando una línea vertical que intersecta la curva $y = g(x)$ en el punto $g(x_1)$
2. Como $x_2 = g(x_1)$, se dibuja una línea horizontal desde el punto $(x_1, g(x_1))$ hacia la línea $y = x$. El punto de intersección da la ubicación de x_2.
3. Desde x_2 de dibuja una línea vertical hacia la curva $y = g(x)$. El punto de intersección es ahora $(x_2, g(x_2))$, y $g(x_2)$ también pasa a ser el valor de x_3.
4. Desde el punto $(x_2, g(x_2))$ se dibuja una línea horizontal nuevamente hacia $y = x$, y el punto de intersección da la ubicación de x_3.

A medida que el proceso continúa, los puntos de intersección convergen hacia el punto fijo, o la verdadera solución x_{vs}.

Hay casos, en la que la forma $f(x) = 0$ no se presta para derivar una formula de iteración de la forma $x = g(x)$. En tal caso, siempre se puede sumar y restar x a $f(x)$ para obtener $x + f(x) - x = 0$. Esta última ecuación se puede reescribir en la forma que se puede usar en el método de iteración de punto fijo como:

$$x = x + f(x) = g(x)$$

Al escoger una función iterativa $g(x)$, es importante tomar en cuenta ciertos factores que pueden ser útiles para la ejecución del método.

- Existen varias formas de cambiar la ecuación a la forma $x = g(x)$.
- La función de iteración no es única.
- Se pueden escribir varias funciones de iteraciones $g(x)$ para la misma ecuación.
- Una $g(x)$ que debe usarse en $x_{i+1} = g(x)$ para el proceso de iteración es uno para el cual las iteraciones converjan hacia el punto.

Es posible determinar de antemano si las función de iteración que escogimos convergerá o divergerá de la solución. El método de iteración de punto fijo converge si, en la vecindad del punto fijo, la derivada de la función $g(x)$ tiene un valor absoluto menor a 1.

$$\mid g'(x) \mid < 1$$

Las iteraciones se detienen cuando la tolerancia en $f(x)$ es mayor a un valor predeterminado.

$$\mid f(x_1) \mid > Tolerancia$$

5 Código

```
'''
Esta es una implementación del método de Punto Fijo en Python.
En el código se encuentran las librerias necesarias, las funciones
que son necesarias para aplicar el método, junto con la función principal
de punto fijo, la cúal recibe como parametros la función (f), la función
de iteración (g), la función derivada de g (g'(x)), el punto inicial (x),
y el grado de tolerancia.
Las funciones tienen una función de ejemplo, la cúal converje.
'''
#Imports
import math
import numpy as np  #Librerias necesarias

#input

x = 1           #Punto de input (ejemplo)
tol = 0.0001    #grado de tolerancia


def f(x):  # Funcion f(x)

    efe = x*math.e**(0.5*x)+1.2*x-5 #Función de ejemplo

    return efe


def g(x):  # g(x) funcion de iteración

    fun = ((5)/(math.e**(0.5*x)+1.2))  #Función de ejemplo
    #fun = ((5)-(math.e**(0.5*x)/1.2))
    return fun


def funDer(x):                      # g'(x), g(x) derivada

    # Función derivada ejemplo
    der = ((-5*math.e**(0.5*x))/(2*(math.e**(0.5*x)+1.2)**(2)))
    #der = (-(math.e**(0.5*x)+0.5*x*math.e**(0.5*x))/1.2)

    return der


def fixedpoint(f, g, fprim, x1, tol):  # Función de punto fijo
    it = 0                              #variable de conteo de iteraciones
```

```
    x2 = x1 + 1                         #Punto x2 para evaluar convergencia
    gp1 = np.abs(fprim(x1))             # g'(x1), g prima evaluado en x1
    gp2 = np.abs(fprim(x2))             #g'(x2), g prima evaluado en x2
    if(gp1 and gp2 < 1):                # |g'(x)|<1 (Converge o Diverge)

        while(np.abs(f(x1)) > tol):  # Condición de iteración (|f(x1)|> tolerancia)
            x_n = g(x1)  # x_n -> x nueva
            x1 = x_n    #x1 pasa a ser el nuevo valor de estimación
            it += 1     #Conteo de iteraciones

        return print('Resultado:\n', x_n, '\nIteraciones:\n', it)
    else:
        print('Diverge.')           #No se puede llegar a la solución


fixedpoint(f, g, funDer, x, tol)  # Llamada a la función con los paramentros
```

6 Implementación

Considere la siguiente función:

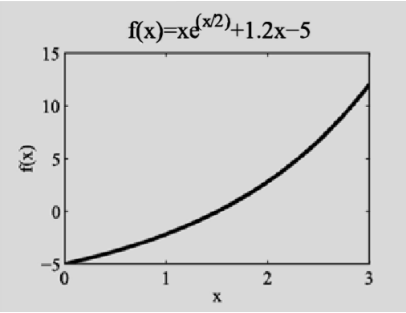

Figure 4: Gráfica $f(x) = x * e^{x/2} + 1.2x - 5$ — Gilar, Amos

En esta gráfica se puede observar que la ecuación tiene solución entre los puntos $x = 1$ y $x = 2$. Igualaremos la función a 0 para poder hacer la implementación del método, $x * e^{x/2} + 1.2x - 5 = 0$. Esta ecuación se puede reescribir en la forma $x = g(x)$ en varias maneras diferentes.

En este caso, haremos la demostración de dos casos, una en la que la función de iteración diverja y otra que converja a la solución.

Caso No.1

$$x = \frac{5 - xe^{0.5x}}{1.2}$$

En este caso, $g(x) = \frac{5 - xe^{0.5x}}{1.2}$ y $g'(x) = \frac{-(e^{0.5x} + 0.5xe^{0.5x}}{1.2}$. Utilizaremos los valores de $x = 1$ y $x = 2$, ya que son los puntos próximos de la solución. Evaluaremos estos valores en la función $g'(x)$, para ver sí la función de iteración diverge o converge.

$$g'(1) = \frac{-(e^{0.5*1} + 0.5*1*e^{0.5*1}}{1.2} = -2.06$$
$$g'(2) = \frac{-(e^{0.5*2} + 0.5*2*e^{0.5*2}}{1.2} = -4.53$$

Caso No.2

$$x = \frac{5}{xe^{0.5x} + 1.2}$$

En este caso, $g(x) = \frac{5}{xe^{0.5x} + 1.2}$ y $g'(x) = \frac{-5e^{0.5x}}{2(e^{0.5x} + 1.2)^2}$. Utilizaremos los valores de $x = 1$ y $x = 2$, ya que son los puntos próximos de la solución. Evaluaremos estos valores en la función $g'(x)$, para ver sí la función de iteración diverge o converge.

$$g'(1) = \frac{-5e^{0.5*1}}{2(e^{0.5*1} + 1.2)^2} = -0.50$$
$$g'(2) = \frac{-5e^{0.5*2}}{2(e^{0.5*2} + 1.2)^2} = -0.44$$

Dados los resultados, el **Caso No.2** es la función iterativa $g(x)$ que se debe de usar, ya que $\mid g'(1) \mid < 1$ y $\mid g'(2) \mid < 1$.

Ahora, en el código se sustituye la función que queremos aproximar, en este caso $f(x) = x * e^{x/2} + 1.2x - 5$, la función de iteración, $g(x) = \frac{5}{xe^{0.5x} + 1.2}$, con el punto inicial $x = 1$, y una $tolerancia = 0.0001$.

Al llamar la función con estos parámetros, se empezara el método iterativo hasta llegar a una aproximación bastante acertada a la solución real, que en este caso es $x = 1.5050$. El proceso de iteración en el código, se vería de esta manera, paso a paso:

$$x_2 = \frac{5}{e^{0.5 \cdot 1} + 1.2} = 1.7552 \qquad x_3 = \frac{5}{e^{0.5 \cdot 1.7552} + 1.2} = 1.3869$$

$$x_4 = \frac{5}{e^{0.5 \cdot 1.3869} + 1.2} = 1.5622 \qquad x_5 = \frac{5}{e^{0.5 \cdot 1.5622} + 1.2} = 1.4776$$

$$x_6 = \frac{5}{e^{0.5 \cdot 1.4776} + 1.2} = 1.5182 \qquad x_7 = \frac{5}{e^{0.5 \cdot 1.5182} + 1.2} = 1.4986$$

Figure 5: Proceso iterativo paso a paso — Gilar, Amos

Al correr el código, da una aproximación casi igual a la solución real, y con un error casi nulo. Así como también, que se requirió de 14 iteraciones para llegar a la aproximación.

```
Resultado:
1.5049
Iteraciones:
14
```

7 Implementación No. 2

Considere la siguiente movimiento:

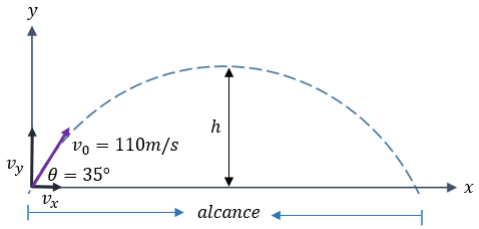

Figure 6: Gráfica Tiro Parabólico — FISIMAT

Una máquina lanza un proyectil a una velocidad inicial de $110m/s$, con ángulo de 35°. La función de movimiento $\Delta y(t)$ puede usarse para aproximar el tiempo de vuelo si se le resta una constante de rozamiento de viento $w = 32$.

Sabiendo que:

$$\Delta y(t) = v_{0y}t + \frac{1}{2}gt^2 - w$$

$$v_{0x} = 110m/s$$

$$v_{0y} = v_{0x}sen(\theta)$$

$$v_{0y} = 63.09m/s$$

$$g = -9.8m/s^2$$

Tenemos:

$$\Delta y(t) = 63.09t - 4.9t^2 - 32$$

Para obtener el tiempo de vuelo, debemos de igualar nuestra función $\Delta y(t)$ a 0, para saber cuando el proyectil llego al suelo.

Por lo tanto:

$$f(t) = 0$$
$$63.09t - 4.9t^2 - 32 = 0$$

Despejando t para nuestra función de iteración:

$$4.9t^2 = -32 + 63.09t$$
$$t = \frac{-32+63.09t}{4.9t}$$

Aplicando el problema al código:

```
def f(x):  # Funcion f(x)

    efe = 63.09*x-4.9*x**2-32

    return efe


def g(x):  # g(x) funcion de iteración

    fun = 63.09/4.9-32/(4.9*x)

    return fun


def funDer(x):                    # g'(x), g(x) derivada

    der = 6.53/x**(2)

    return der
```

```
Resultado:
12.35
Iteraciones:
5
```

Al corre nuestro código con las funciones y el punto de estimación, nos da un resultado de 12.345 de tiempo de vuelo en 5 iteraciones, y la solución real es de 12.355. Por lo que podemos concluir, que este método es bastante eficiente y preciso a la hora de hacer aproximaciones a funciones no lineales, siempre y cuando, se escoja una buena función de iteración $g(x)$.

8 Punto fijo con vectores columnas nx1

Para calcular el punto fijo para vectores columna $nx1$, es muy bastante parecido al proceso ya antes descrito. En este caso, x_0 sera un vector inicial de dimensión $nx1$, y la función de iteración sigue la misma linea que la anterior, $x_{k+1} = f(x_k)$. Por otra parte, en esta función, podremos ingresar dos vectores $nx1$ para poder encontrar el punto fijo de la función. Este algoritmo sigue iterando hasta cumplir con la tolerancia o bien, cuando llegue al máximo de iteraciones ingresada por el usuario. Otra diferencia que podemos observar, es la función auxiliar que ingresa el vector x_0 y el vector resultante de la función entre los dos vectores ingresados y el vector v_0, para hacer un diferencial sobre el valor actual menos el valor esperado, partido el valor esperado. El resultado de esto, nos ayudara a decidir si la aproximación es lo suficiente precisa para cumplir con la tolerancia.

9 Código No.2

```
#Libreriasa necesarias
from scipy import optimize
import numpy as np
import decimal
from scipy._lib._util import _asarray_validated, _lazywhere


def sp_helper(actual, desired):
    return (actual - desired) / desired   #Funcion auxiliar para calcular diferencial

def fixedPoint(function, x0, args, tol, maxit):
    p0 = x0
    for i in range(maxit):    #Iterar hasta le maximo de iteraciones

        p1 = function(p0, *args)  #Vector resultante

        p=p1          #Variable temporal
```

```
        relerr = _lazywhere(p0 != 0, (p, p0), f=sp_helper, fillvalue=p) #Funcion aux.
        if(np.all(np.abs(relerr) < tol)):  #Condición de convergencia
            return p

        p0 = p

    msg = "Failed to converge after %d iterations, value is %s" % (maxit, p)
    raise RuntimeError(msg)


def auxFixedPoint(function, x0, maxit, tol, args=()):
    """
    Encontrar el punto fijo de un vector columna nx1.
    Dada una función de una o más varaiables y un punto de inicio, encontrar
    el punto fijo de la función. Dondé ``func(x0) == x0``.

    Parametros
    ----------
    func : function
        Función a evaluar.
    x0 : vector
        Punto fijo de la función-
    args : tuple
        Argumentos para la función.
    tol : float
        Tolerancia de convergencia, recomendado: 1e-08.
    maxit : int
        Maximo de iteraciones.
    References
    ----------
    .. [1] Burden, Faires, "Numerical Analysis", 5th edition, pg. 80
    --------
    """

    x0 = _asarray_validated(x0, as_inexact=True) #Verifica el input es valido

    return fixedPoint(function, x0, args, tol, maxit)

def function(x,c1,c2):        #Vector coumna con n funciones

    z = np.sqrt(c1/(x+c2))

    return z
```

```
c1 = np.array([1, 12])                  #Vector columna
c2 = np.array([1, 5])                   #Vector columna
x0 = np.array([10, 20])                 #Vector inicial


mit = input("Ingrese el maximo de iteraciones deseado: \n")
tole = input("Ingrese el nivel de tolerancia deseado (Recomendadp: 1e-8): \n")

maxit = int(mit)
tol = decimal.Decimal(tole)


print("Resultado: \n", auxFixedPoint(function, x0, maxit, tol, args=(c1, c2)), "\n")
```

References

[1] Michael T. Heath. *Scientific Computing an Introduction Survey*. University of Illinois

[2] Amos Gilar. *Numerical Methods for engineers and scientists*. 3rd edition, Vish Subramian.

[3] Amos Gilar. *Numerical Analysis*. Burden, Faires, 5th edition, pg. 80.