Universidad
de La Laguna

Búsqueda de raíces
Método de Newton

Rebeka Luis Hernández y Anabel Estévez Carrillo

Grupo (2 | C)

Técnicas Experimentales. 1^{er} *curso.* 2^{do} *semestre*

Lenguajes y Sistemas Informáticos

Facultad de Matemáticas

Universidad de La Laguna

La Laguna, 11 de mayo de 2014

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Motivación y objetivos

El objetivo propuesto es obtener los conocimientos necesarios para desarrollar un informe tecnico-científico usando latex, así como lograr un código en python que nos permita obtener la raíz de una función haciendo uso del método de Newton.El método de Newton-Raphson es un método iterativo qara aproximar la solución de una ecuación del tipo $f(x) = 0$. En el caso propuesto, la función tratada será $f(x) = log(x)$.

- Objetivo principal: Implementación con Python del método de Newton.
- Objetivo específico: Cómo se comporta el método de Newton aplicado a la función $f(x) = log(x)$.

El procedimiento será descrito de en un artículo realizado con el procesador de texto Latex y Beamer, una clase de documento especialmente diseñada para presentaciones que utilicen recursos Latex.

1.1. LaTeX

Latex es un sistema de que permite crear documentos con un aspecto completamente profesional de una forma sencilla. La idea principal es que el autor se centre en el contenido y no en la forma del documento. Para lograr esto,Latex está provisto de una serie de macros y estilos predefinidos.

Hay disponible una gran variedad de paquetes para facilitar la representación de fórmulas matemáticas, notación musical, fórmulas químicas, circuitos electrónicos y mucho más. Concretamente, Latex está orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas.

1.2. Beamer

Beamer es una clase de LaTeX para la creación de presentaciones (vía PDFLATEX). Es un cógido abierto y el nombre viene del vocablo alemán "beamer", un pseudo-anglicismo que significa videoproyector.

Las razón por las que se pretende adquirir conocimientos y mejorar nuestras habilidades en su uso se basa en algunas de sus características. Por ejemplo:

- Software libre y gratuito, con una amplísima comunidad de soporte.
- Generación de presentaciones con formato estándar y portable.
- Especialmente útil para preparar presentaciones en las que es necesario mostrar gran cantidad de expresiones matemáticas.

Capítulo 2

Fundamentos teóricos

2.1. Fundamentos Teóricos

El método de Newton-Raphson es un método iterativo con el que se pueden encontrar aproximaciones de soluciones de ecuaciones no lineales. El método parte de un valor inicial que se introduce en una expresión relacionada con la ecuación, obteniendo así un resultado. Ese resultado se introduce en la misma expresión, obteniendo un nuevo resultado, y así sucesivamente. Si la elección del valor inicial es buena, cada vez que se introduzca uno de los resultados obtenidos en esa expresión (es decir, cada vez que se haya realizado una iteración del método) el método proporciona una aproximación a la solución real mejor que la que se haya tenido anteriormente. Este procedimiento se realiza utilizando la siguiente fórmula de recurrencia:

$$x_n + 1 = x_n - \frac{f(x_n)}{f'(x_n)}$$

Aunque el principal problema de este método consiste en elgir de forma correcta el valor inicial, debido a que si esto no ocurre no es seguro que los resultados obtenidos por el método sean de verdad buenas aproximaciones de la solución. Con lo que ¿ cómo se debería elegir el valor inicial? Debemos partir de una función $f(x)$ y un intervalo [a,b] en el cual la función debe ser al menos dos veces derivable en dicho intervalo y la segunda derivada continua en él. En principio, se debe escoger un intervalo en el que $f(x) = 0$ cumpla el teorema de Bolzano para poder asegurar así que hay al menos una raíz en dicho intervalo. Es decir, $f(x)$ debe ser continua en dicho intervalo y debe cumplirse también que $f(a)$ y $f(b)$ tengan signos distintos. Además, la primera de derivada de f debe ser distinta de cero en [a,b] y también se debe cumplir que la segunda derivada no cambie de signo en dicho intervalo (es decir, que sea siempre positiva o siempre negativa). Con lo que con todo esto se cumplirá que:

$$f(a) \times f'(a) > 0, \text{ o que } f(b) \times f'(b) > 0$$

Eligiendo como valor inicial el extremo del intervalo que cumpla que ese producto es positivo está asegurado que el método converge a la única solución de la ecuación en [a,b].

Capítulo 3

Procedimiento experimental

En este capítulo se enumerarán los experimentos realizados en la verificación del cálculo de las raíces de la función tratada.

3.1. Descripción de los experimentos

En este capítulo se enumerarán los experimentos realizados en la verificación del cálculo de las raíces de la función tratada. Se ha redactado un código en el lenguaje de programación Python con tal objetivo y se ha analizado su funcionamiento. El código cuenta con una función a la que hemos llamado newton que calcula el resultado de aplicar la fórmula de Newton siempre que el valor introducido sea mayor que cero. Esta función evalúa el error cometido como la diferencia entre el valor obtenido y el valor de partida. El proceso se ejecutará tantas veces como sea necesario para que el error sea menor que un valor prefijado. Si el punto de partida es cero o negativo, no se podrá aplicar el método de newton y el programa advertirá al usuario. El programa se ejecutará con los valores que el usuario introduce como parámetro, pero en el caso de que estos se omitan, tomará valores por defecto. En el apéndice A.1 se puede cosultar el código descrito.

3.2. Descripción del material

Para la realización de este experimento se ha utilizado un ordenador que cuenta con un un procesador Intel(R) Pentium(R) CPU B960 @ 2.20GHz. El computador cuenta con 400Gb de disco duro y 6Gb de memoria RAM. Así mismo, dispone de una velocidad de CPU de 800.000Hz y un caché de CPU de 2048 KB.

El sistema operativo empleado es una adaptación de la distribución de Linux *kubuntu* a las necesidades en cuanto a Software de los miembros de la comunidad universitaria ULL: Bardinux, en concreto se ha utilizado Linux-3.2.0-37-generic-i686-with-Ubuntu-12.04-precise. Se ha interpretado el código descrito con el intérprete de Python que cuenta este sistema operativo, la versión utilizada ha sido la 2.7.3.

3.3. Resultados obtenidos

Tras la implementación del código, se ha comprobado su precisión dando valores que serán las aproximaciones iniciales de la raíz y se han obtenido los resultados plasmados en la siguiente tabla.

Aproximación	Raiz	f(raiz)	Iteraciones necesarias
1	1.000	0.000	1
2	1.000	0.000	6
3	1.000	0.000	7
5	1.000	0.000	8
6	1.000	0.000	10
7	1.000	0.000	11

Cuadro 3.1: Resultados experimentales

3.4. Análisis de los resultados

Al ejecutar el programa con distintos valores como aproximación inicial de la raíz se observa que en todos los casos se calcula la raíz situada en el punto x=1 con precisión, y por tanto el valor de la función en ese punto es cero.

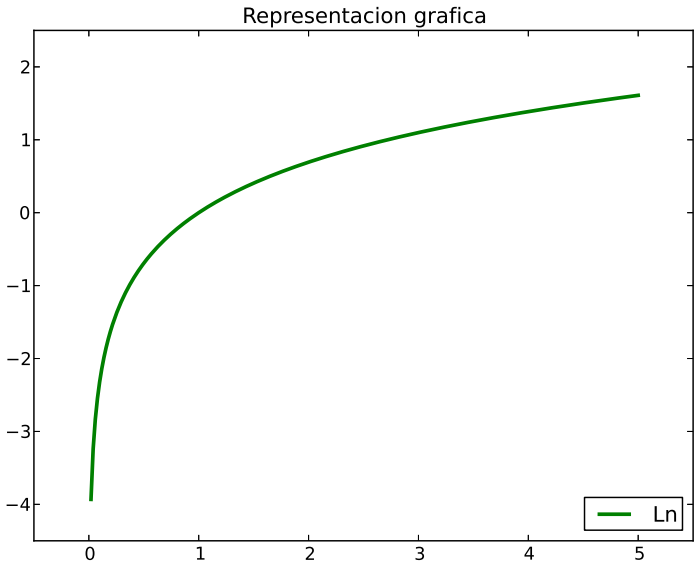

Figura 3.1: Ejemplo de figura

Esta figura se ha realizado con mathplotlib. En el apéndice B.1 se puede consultar el código.

Por otro lado, cabe destacar que el número de veces que se aplica la fórmula del método de newton, es decir, las iteraciones necesarias para obtener los resultados expuestos crecen cuando la aproximación inicial que introducimos se aleja de la raíz. Así, por ejemplo, si introducimos como aproximación inicial la propia raíz, bastaría con una iteración para obtener los resultados buscados.

Si se traza una función que represente el número de iteraciones necesarias para calcular las raíces mediante el método de newton para la función $f(x) = log(x)$, se observa que ésta tiene una mayor pendiente cuando nos aproximamos cero. A medida que nos alejamos, aunque se mantiene creciente, el valor la pendiente será menor. Esto ocurre ya que el método de Newton calcula la raíz de una función utilizando su derivada, que no es más que el valor de la pendiente de la recta tangente en ese punto. Pues bien, como vemos, la pendiente de la recta tangente en el caso de la función $f(x) = log(x)$ es superior a medida que nos acercamos a cero.

El tiempo de CPU aumenta a medida que se incrementa el valor inicial, al alejarse de la raíz y necesitar que se desarrolle el método de Newton un mayor número de veces. Sin embargo, para cualquier valor inicial que se introduzca el tiempo de CPU registrado será cero. Así que, para obtener datos al respecto, se ha ejecutado el programa 100000 para cada valor empleando el módulo timeit. En el apéndice A.2 se puede observar este código. De este modo, se puede observar por ejemplo que se tarda más tiempo en ejecutar el método partiendo de x = 7.389 que desde x = 2.0 (2 segundos y pico frente a 0.66 segundos). Los resultados se han almacenado en la siguiente tabla:

Valor inicial	Tiempo de CPU al ejecutarse 100000
1	0.1154410839
2	0.5907168388
3	0.6773560047
4	0.5563027859
5	0.7719118595
6	0.9161560535
6.5	0.8663120270
7	0.9962458611
7.35	1.3717629910

Cuadro 3.2: Resultados tiempo CPU

Nótese que se ha introducido como valor máximo al explicar el tiempo de ejecución el valor de 7.389. Esto es debido a que a partir de este valor, las aproximaciones en vez de acercarse al 1.00000 se alejan, por lo que llega un momento en el que el valor de x es tan grande, que se produce una división por cero al evaluar la expresión $\frac{log(x)}{(1/x)}$. Por lo tanto, como x es muy grande el cociente $\frac{1}{x}$ tiende a 0 y la división principal provoca el error.

Estos resultados se han representado en la siguiente gráfica:

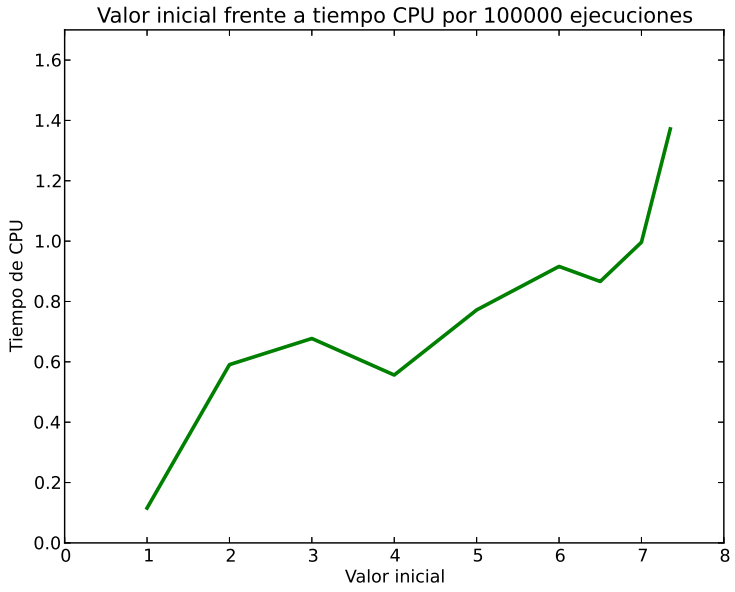

Figura 3.2: Tiempo de CPU

Esta figura se ha realizado con mathplotlib. En el apéndice B.2 se puede consultar el código.

Capítulo 4

Conclusiones

- Se ha conseguido implementar un código en el lenguaje de programacion Python que consigue resolver de manera precisa la raiz de la función tratada aplicando el método de Newton, si bien el algoritmo diseñado puede ser extrapolado a cualquier otra función, lograndose los objetivos descritos al comienzo de este informe.
- En el caso de que el valor de partida que tomamos como aproximación inicial de la raiz disté de este, se necesitará aplicar el método de Newton una mayor cantidad de veces para obtener resultados precisos. La pendiente de la función tiene un papel importante a la hora de determinar el número de veces que se debe aplicar la fórmula del método de Newton.
- El programa diseñado actúa con eficiencia, sin embargo, el tiempo de CPU necesarió será mayor cuando la aproximación inicial de la raíz se aleje de la misma.
- La elección de un buen valor como aproximación inicial de la raíz es importante para que el programa funcione correctamente. Se han desarrollado los puntos a tener en cuenta para elegir de forma correcta el valor inicial, consiguiendose resultados más precisos.
- A partir del valor 7.389, las aproximaciones en vez de acercarse al 1.00000 se alejan, por lo que llega un momento en el que el valor de x es tan grande, que se produce una división por cero al evaluar la expresión $\frac{log(x)}{(1/x)}$. Por lo tanto, como x es muy grande el cociente $\frac{1}{x}$ tiende a 0 y la división principal provoca el error.

Apéndice A

Códigos en Python

A.1. Cálculo de la raíz por el método de Newton

```
################################################################################
# metodonewton.py
################################################################################
#
# REBEKA LUIS Y ANABEL ESTÉVEZ
#
# 8 DE MAYO DE 2014
#
# #!/src/bin/python
#-*- encoding: utf-8 -*-

import sys
import math
error = 0.000000001

# PRINCIPIO DE LA FUNCIÓN.

def newton(x):

  if(x>0):
    i=1
    while 1:
      aprox= abs(x - (math.log(x)/float(1/float(x)))) # Para que si la aproximación sea siempre positiva y s
      e = abs(aprox-x)
      if (e<error):
print "La raiz de la función ln(x) es: %.10f" %aprox
print "El valor de la función ln(x) es: %.10f" %math.log(aprox)
print "Se ha aplicado %d veces el método de Newton." %i
break
      i = i+1
      x=aprox

  elif (x<0):
    print "Los números negativos nos pertenecen al dominio de la función. "
  else:
```

```
    print "No se puede aplicar el método de Newton en el punto x=0."

# FINAL DE LA FUNCIÓN.

if __name__== "__main__":

  if (len(sys.argv)==2):
    n = int(sys.argv[1])
  else:
    print "No se ha especificado el valor de partida. Tomaremos el valor por defecto."
    n=2

  newton(n)

##############################################################################################
```

A.2. Tiempo de CPU

```
/#############################################################################################
 # time.py
 #############################################################################################
 #
 # REBEKA LUIS Y ANABEL ESTÉVEZ
 #
 # 8 DE MAYO DE 2014
 #
 ##-*- encoding: utf-8 -*-
 #!/src/bin/python

 import timeit

t = timeit.Timer('newton(7.389)', setup='from metodo2 import newton')
print "Al ejecutarse el código 100000 veces desde el valor  se tarda %.10f" % t.timeit(100000)
 #
 #############################################################################################
```

Apéndice B

Códigos en Mathplotlib

B.1. Códigos en mathplotlib: gráfico $log(x)$

```
import matplotlib.pylab as pl
import numpy as np
from math import log

pl.figure(figsize=(8,6), dpi=80)



X = np.linspace(0, 5, 256, endpoint=True)
C =np.log(X)


pl.plot(X,C, color="green", linewidth=2.5, linestyle="-", label="Ln")
pl.legend(loc='lower right')
pl.xlim(-0.5,5.5)

pl.ylim(-4.5,2.5)
pl.title("Representacion grafica")

pl.savefig("ln.eps", dpi=72)

pl.show()
```

B.2. Código en mathplotlib:grafico tiempo CPU

```
import matplotlib.pylab as pl
import numpy as np
from math import log

# Lista con los valores de partida
x = [1, 2, 3,  4, 5, 6, 6.5, 7, 7.35, ]

# Lista con los valores de l tiempo
```

```
y = [0.1154410839, 0.5907168388,  0.6773560047, 0.5563027859, 0.7719118595, 0.9161560535, 0.8663120270, 0.99

# Utilizar la funcion plot para trazar el grafico
pl.plot(x,y, color="green", linewidth=2.5, linestyle="-")

# Incluir un titulo
pl.title('Valor inicial frente a tiempo CPU por 100000 ejecuciones')

# Poner etiquetas en los ejes
pl.xlabel('Valor inicial')
pl.ylabel('Tiempo de CPU')

# Poner limites a los ejes
pl.xlim(0.0, 8)
pl.ylim(0.0, 1.7)

# mostrar por la consola el resultado
pl.savefig("time.eps", dpi=72)
pl.show()
```

Bibliografía

[1] Daniel Francisco Campos Aranda. *MÃ©todo de Newton*. Writing guides. Reference. Universidad de Cadiz. Servicio de Publicaciones., 2003.

[2] J. Gibaldi. *Tutorial de Python*. Writing guides. Reference. Modern Language Association of America, 2009.

[3] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison–Wesley Pub. Co., Reading, MA, 1986.

[4] Juan Antonio Navarro Perez. *Tutorial de Latex*. Writing guides. Reference. Paquillo Dubois, 2009.