Universidad
de La Laguna

Series de potencias
Newton:f(x)=cos(x)

Nayra Kintan Díaz, Javier de León Morales, José Eduardo Lorenzo Pérez

Equipo (2_G)

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

Lenguajes y Sistemas Informáticos

Facultad de Matemáticas

Universidad de La Laguna

La Laguna, 9 de mayo de 2014

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Motivación y objetivos

Por el presente trabajo pretendemos ampliar los conocimientos acerca de las series de potencias hasta llegar a la aplicación del Método de Newton a la función f(x)=cos(x).
Para la realización de este trabajo nos basamos en el uso de LaTeX, la programación en Python, el uso del procesador de texto Kate, la utilización de Beamer y en general el trabajo con Linux.
LaTeX es un procesador de texto con un lenguaje de bajo nivel. Hemos aprendido a insertar gráficas para representar ecuaciones, fórmulas, etc.
Nos permite estructurar fácilmente el documento con capítulos, secciones, notas, bibliografía, índices analíticos, lo que nos ayuda en la realización de este trabajo.
Beamer es un tipo de documento que está diseñado para presentaciones que utilicen recursos de LaTeX, requiere la compilación a través de PDFLaTeX.
Este nos fue útil para introducir fórmulas matemáticas, gráficos, imágenes, etc.
En particular, podemos decir que realizamos un programa en Python para resolver el método de Newton aplicado a la función trigonométrica $cos(x)$.
Vamos a desarrollar los contenidos sobre las series de potencias y sus aplicaciones.

Capítulo 2

Fundamentos teóricos

En este capítulo profundizaremos sobre la series de potencias y el método de Newton. Una serie de potencias puede ser interpretada como una función de x:

$$f(x) = \sum_{n=0}^{n\infty} a_n * (x - c)^n$$

cuyo dominio es el conjunto de los x $\in$ R para los que la serie es convergente y el valor de $f(x)$ es, precisamente, la suma de la serie en ese punto x. Las series de potencias, vistas como funciones, tienen un comportamiento bueno, en el sentido de que son funciones continuas y derivables de cualquier orden. Más aún, su función derivada es, otra vez, una serie de potencias. Desde un punto de vista más práctico, las series de potencias aproximan a su función suma. Es decir, la suma parcial de orden n, que no es más que un polinomio de grado n a lo sumo, representa una aproximación a la función suma en su dominio de convergencia. En la siguiente gráfica, puede verse la función:

$$f(x) = e^x$$

junto con algunas aproximaciones mediante sumas parciales de su serie de potencias.

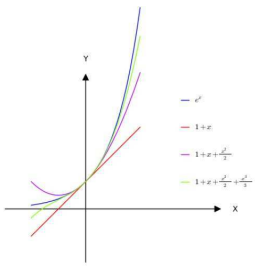

Figura 2.1: Aproximación a $f(x) = e^x$ por su serie de potencias

El método de Newton es una aplicación del cálculo diferencial que se utiliza para hallar los ceros de una función derivable de n-esimo grado. Los procedimientos para hallar las raíces o ceros de funciones lineales o cuadráticas a partir de los coeficientes de la ecuación son sencillos y exactos.

Antes, debemos desarrollar algo acerca de el polinomio de Taylor.

Este teorema permite aproximar una función derivable en el entorno reducido alrededor de un punto a mediante un polinomio cuyos coeficientes dependen de las derivadas de la función en ese punto. Más formalmente, si n $\geq$ 0 es un entero y f una función que es derivable n veces en el intervalo cerrado [a, x] y n+1 veces en el intervalo abierto (a,x), entonces se cumple que:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n(f)$$

o en forma compacta:

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_n(f)$$

Para llegar a la fórmula del método a estudiar se puede obtener si truncamos la fórmula del desarrollo de Taylor, igualando f(x)=0 se obtiene:

$$x = xo - \frac{f(xo)}{f'(xo)}$$

2.1. Obtención de la fórmula

El método de Newton tiene una interpretación geométrica sencilla, de hecho, este consiste en una linealización de la función, como lo cual quiere decir, que la función f se reemplazará por una recta tal que contiene al punto f(xo).La nueva aproximación a la raíz, x1, se obtiene de la intersección de la función lineal con el eje X de ordenadas. La ecuación de la recta que pasa por el punto(xo,f(xo)) y de la pendiente f'(xo) es:

$$y - f(xo) = f'(xo) * (x - xo)$$

Haciendo y=0 y despejando la x se obtiene la ecuación de Newton:

$$x = xo - \frac{f(xo)}{f'(xo)}$$

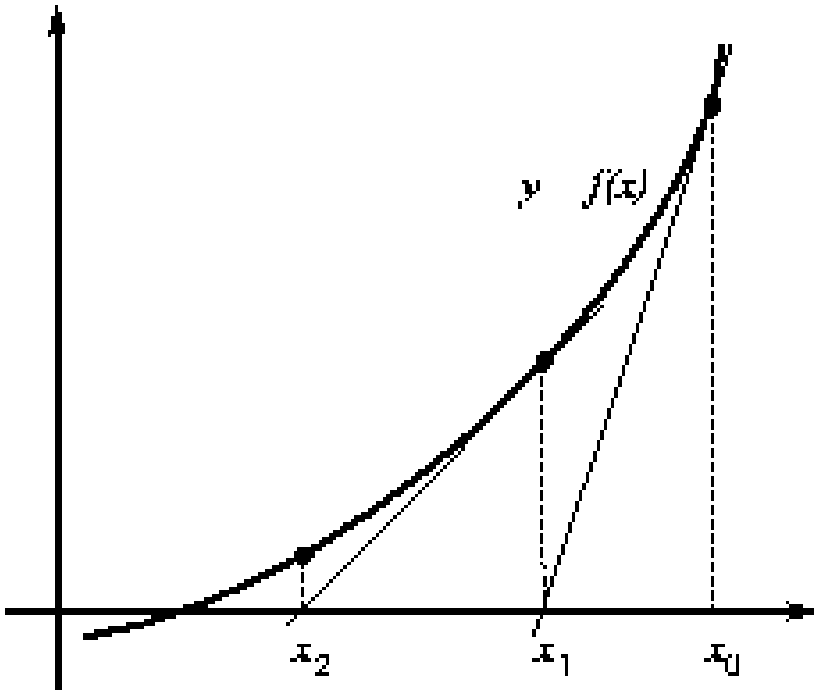

Figura 2.2: Método de Newton

Capítulo 3

Procedimiento experimental

3.1. Descripción de los experimentos

Usaremos la función f(x)=cos(x) para desarrollar nuestro experimento.

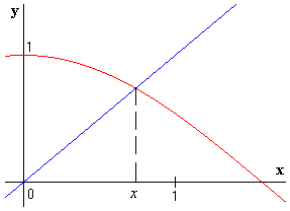

Figura 3.1: Gráfico 1

En el primer gráfico podemos observar las representaciones de las dos funciones que utilizaremos asi como la coordenada de la abscisa de su punto de intersección en el primer cuadrante. Para hallar la abscisa del punto de intersección basta con igualar las funciones de la siguiente manera: x=cos(x), entonces igualamos a cero la ecuación: $cos(x) - x = 0$.

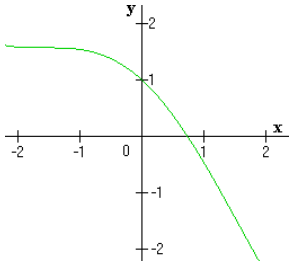

Figura 3.2: Gráfico 2

En el grafico 2 vemos la representación de la función $f(x) = cos(x) - x$ Ahora utilizaremos el Método de Newton para hallar, con una aproximación de 4 cifras decimales, el cero:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, f'(x_n)! = 0 \qquad (*)$$

Sea:

$$f(x) = cos(x) - x \quad (1) \qquad \rightarrow f'(x) = -sen(x) - 1 \qquad (2)$$

De (*), (1) y (2) se deduce que la segunda aproximación está dada por:

$$x2 = x1 - \frac{(cos(x)-x)}{(-sen(x)-1)} = x1 + \frac{(cos(x)-x)}{(sen(x)+1)} \qquad (3)$$

Fijándonos en la gráfica de f, escogemos a x1=1. Ahora:

$$f(1) = -0,4597 \qquad (2)$$
$$f'(1) = 1,8415 \qquad (6)$$

Sustituyendo (4),(5) y (6) en (3), se obtiene:

$$x2 = 1 + \frac{(-0,4597)}{1,8415} \rightarrow x2 = 0,7504$$

Para este nuevo valor de x, calculamos:

$$f(0,7504) = -0,0109$$
$$f'(0,7504) = 1,6819$$

De tal manera que:

$$x3 = 0,7504 + \frac{(-0,0109)}{1,6819} = 0{,}7391 \qquad (7)$$

Para este nuevo valor de x, calculamos:

$$f(0{,}7391) = 0$$
$$f'(0{,}7391) = 1{,}6736$$

De tal manera que:

$$x4 = 0{,}7391 - \frac{0}{1,6736} = 0{,}7391 \qquad (8)$$

Comparando (7) y (8), se tiene que:

$$x3 = x4 = 0{,}7391$$

La coordenada x del punto de intersección en el primer cuadrante de las gráficas es

x=0.7391

Por último, en este primer apartado de descripción experimental es necesario poner un código python que satisfaga nuestro experimento y constituye el siguiente:

```
while(x != tempo and i<100):
     tempo = x
     x = x- funcionDada(x)/derivadaFuncionDada(x)
     e = abs ((x-tempo)/x)

     print("x" + str(i) + "=" + str(x) + "error=" + str(e) + "\n")
     i=i+1
```

3.2. Descripción del material

En este apartado, hablaremos sobre el material en el cual hemos realizado nuestro experimento.

Nuestro experimento ha sido realizado en un PC con la versión del sistema operativo(S.O.):

'Linux-3.2.0-61-generic-i686-with-Ubuntu-12.04-precise'

El tipo de compilador que hemos utilizado para nuestro experimento es el del lenguaje de programación Python :

'2.7.3'

Y, por último, los datos que aparecen a continuación, hacen referencia al hardware de la máquina:

- Pentium(R) Dual-Core CPU E5200 @ 2.50GHz
- GenuineIntel
- 1200.000 Hz
- 2048 KB

Todos los lectores de este informe, se preguntaran de cómo hemos obtenido estos datos, pues se le ha de informar que estos datos han sido obtenidos gracias a un código Python(que guarda los datos en un fichero) que es el siguiente:

```
def SOFTinfo():
  softinfo=()
  softinfo={'Several':platform.uname(),'S.O':platform.platform(),
 'Pythons Version':platform.python_version(), 'Date':platform.python_build()}
  return softinfo
```

3.3. Resultados obtenidos

En este apartado de **Resultados obtenidos** debemos darle un valor al código Python mostrado en páginas anteriores y observar los distintos valores del error. A continuación mostraremos una tabla con los valores del punto x_0 y los errores obtenidos, dándole el valor 1 a la x.

x_0	error
0.496558178297	7.04158813835
2.13100384448	1.2330160875
0.689662720778	2.089972175491
0.739652997531	0.0675861206807
0.739085204376	0.000768237751393
0.739085133215	9.62821076424e-08
0.739085133215	1.50215851291e-15
0.739085133215	0.0

Cuadro 3.1: Tabla de errores

Seguidamente vamos a exponer la gráfica de los errores, realizada mediante el programa matplolib. Esta figura tiene dos gráficas, una ampliada con los errores al completo, y otra más reducida, sin los tres primeros x y errores.

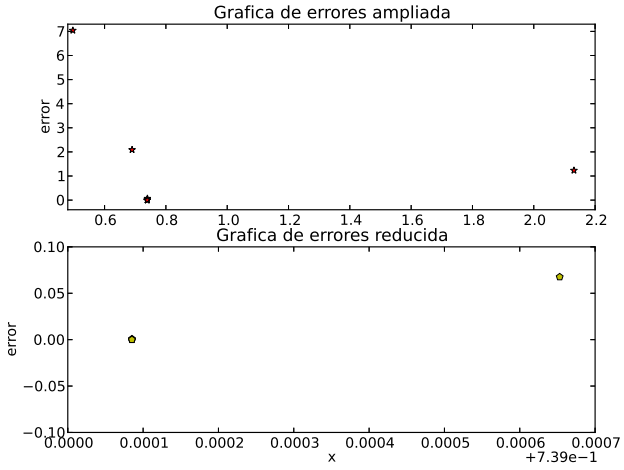

Figura 3.3: Gráfico de errores

Para realizar está gráfica, hemos empleado un coigo python, que es el siguiente:

```
hola.plot(x,y,"r*")
```

```
hola.title('Grafica de errores ampliada')
hola.ylabel('error')
hola.ylim(-0.4,7.3)
hola.xlim(0.48,2.2)
```

3.4. Análisis de resultados

Al realizar el programa en Python para resolver el método de Newton aplicado a la función $f(x) = cos(x)$ y obtener los resultados de la tabla anteriormente expuesta, pudimos apreciar que se obtienen una serie de errores según se le iban dando valores a la "x", pudimos ver una cierta curiosidad, la cual dándole casi el mismo valor a la x; 0.739652997531, 0.739085204376 y 0.739085133215(este se da 3 veces, pero es porque el programa redondea en ese numero), se producen errores muy diferentes ya que unos son de $\frac{1}{10^8}$ y otros de 0,000768237751393.
También se puede observar que cuanto mayor sea el valor de la "x", mayor es el error, excepto en el caso de $x_0 = 0{,}496558178297$ cuyo error es 7.0415881385 y $x_1 = 2{,}13100384448$ cuyo error es 1.2330160875.
También observamos que al insertar x mayores, es decir, $x = 6$, $x = 7$, $x = 8$, etc, se producen muchos más valores de x_n y por lo tanto, muchos más errores.
A partir de la gráfica podemos ver la gran cercanía de algunos puntos, que incluso ni se aprecian.
Según los datos que hemos obtenido, podemos llegar a la conclusión de que, este método hace que la función tienda a cero, esto es, que los valores de los errores se aproximen cada vez más a **cero**. Este hecho, constrasta lo dicho en el apartado de fundamentos teóricos donde se expuso que f(x)=0 debido a la aplicación del método de Newton.

Capítulo 4

Conclusiones

En este capítulo vamos a enumerar principalmente todos los objetivos que se han conseguido respecto al capítulo de **Motivación y Objetivos**.
Durante este informe que hemos trabajado en LaTeXhemos aprendido a:

1. Emplear fórmulas y demostraciones matemáticas.
2. Introducir texto en LaTeXy a realizar presentaciones en Beamer.
3. Introducir tablas que muestren los datos de nuestro experimento.
4. Introducir figuras y gráficas diseñadas en el progrma matplolib.

A parte, de aprender a realizar un informe en LaTeX, también hemos descubierto la solución a nuestro problema mediante el procedimiento experimental:

1. Diseñando una demostración matemática por medio de la demostración hacia delante.
2. Proponiendo un programa python para resolver nuestro problema.
3. Análisis de los resultados del programa python y opinión personal.

Apéndice A

Título del Apéndice 1

A.1. Algoritmo XXX

```
##############################################################################
# Fichero .py
##############################################################################
#
# AUTORES Nayra Kintan Díaz, Javier de León Morales, José Eduardo Lorenzo Pérez
#
# FECHA  11 de mayo de 2014
#
# DESCRIPCION Programa Python método de Newton: f(x)=cos(x)
#
##############################################################################

#!/usr/bin/python
#! enconding: utf-8
import math
import time

startubuntu=time.time()
startcpu=time.clock()
def funcionDada(x):
      return math.cos(x) - x

def derivadaFuncionDada(x):
      return -math.sin(x) - 1

i = int(0);

x = float(input("Ingrese el valor de x: "))

tempo = 0;
while(x != tempo and i<100):
     tempo = x
     x = x- funcionDada(x)/derivadaFuncionDada(x)
     e = abs ((x-tempo)/x)
```

```
     print("x" + str(i) + "=" + str(x) + "error=" + str(e) + "\n")
     i=i+1

if(i==100):
    print("\n\nNo converge")

else:
    print("\n\nSolucion x: " + str(x))

finishubuntu=time.time()
finishcpu=time.clock()
timecpu=finishcpu-startcpu
timeubuntu=finishubuntu-startubuntu
print 'Tiempo Ubuntu:%2.10f \n Tiempo CPU: %2.10f' %(timeubuntu,timecpu)
```

A.2. Algoritmo YYY

```
/##############################################################################
 # Fichero .h
 ##############################################################################
 #
 # AUTORES  Nayra Kintan Díaz, Javier de León Morales, José Eduardo Lorenzo Pérez
 #
 # FECHA    11 de mayo de 2014
 #
 # DESCRIPCION  Programa Python que imprime los datos del hardware y del software de la máquina
 #
 ##############################################################################
#!enconding:UTF-8
#!/usr/bin/python

import platform
import os


def SOFTinfo():
  softinfo=()
  softinfo={'Several':platform.uname(),'S.O':platform.platform(),
 'Pythons Version':platform.python_version(), 'Date':platform.python_build()}
  return softinfo

def CPUinfo():
  # infofile on Linux machines:
  infofile ='/proc/cpuinfo'
  cpuinfo = {}
  if os.path.isfile(infofile):
    f = open(infofile, 'r')
    for line in f:
      try:
name, value = [w.strip() for w in line.split(':')]
      except:
continue
      if name == 'model name':
```

```
cpuinfo['CPU type'] = value
      elif name == 'cache size':
cpuinfo['cache size'] = value
      elif name == 'cpu MHz':
cpuinfo['CPU speed'] = value + ' Hz'
      elif name == 'vendor_id':
cpuinfo['vendor ID'] = value
    f.close()
  return cpuinfo

if __name__ == '__main__':
  softinfo=SOFTinfo()
  for keys in softinfo.keys():
    print softinfo[keys]
```

Apéndice B

Título del Apéndice 2

B.1. Otro apendice: Seccion 1

```
#Program python para dibujar la gráfica de los errores que se encuentra en este informe
import matplotlib.pyplot as hola
hola.subplot(2,1,1)
x=[0.496558178297, 2.13100384448, 0.689662720778, 0.739652997531, 0.739085204376, 0.739085133215, 0.73908513
y=[7.04158813835, 1.2330160875, 2.089972175491, 0.0675861206807, 0.000768237751393,9.62821076424e-08, 1.5021
hola.plot(x,y,"r*")
hola.title('Grafica de errores ampliada')
hola.ylabel('error')
hola.ylim(-0.4,7.3)
hola.xlim(0.48,2.2)

hola.subplot(2,1,2)
x=[0.739652997531, 0.739085204376, 0.739085133215, 0.739085133215, 0.739085133215]
y=[0.0675861206807, 0.000768237751393,9.62821076424e-08, 1.50215851291e-15, 0.0]
hola.plot(x,y,"yp")
hola.title('Grafica de errores reducida')
hola.xlabel('x')
hola.ylabel('error')
hola.ylim(-0.1,0.1)
hola.xlim(0.7390,0.7397)
hola.savefig("Grafica_de_errores.eps",dpi=72)
hola.show(y)
```

B.2. Otro apendice: Seccion 2

Texto