Universidad de La Laguna

Series de Polinomio de Taylor Cos(x)

Sergio Álvarez Fernández
Cirilo Fleitas Rufino
Rayco Hernández Delgado

Grupo (2H)

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

Lenguajes y Sistemas Informáticos

Facultad de Matemáticas

Universidad de La Laguna

La Laguna, 12 de mayo de 2014

Índice general

Índice de figuras

Capítulo 1

Motivación y objetivos

El objetivo de este trabajo ha sido desarrollar un experimento, para evaluar la serie del polinomio de Taylor de una función concreta. En nuestro caso el cos(x). Para ello hemos desarrollado un programa en Python, que a partir de recibir varias cotas superiores del error (resto de Taylor) calcula el polinomio de Taylor y lo muestra por pantalla. También hemos creado un programa para que nos represente en una gráfica los polinomios de Taylor anteriormente hallados junto con el cos(x). Y otra gráfica que muestra el tiempo empleado en calcular cada polinomio y mostrarlo.

Capítulo 2

Fundamentos teóricos

En cálculo, el teorema de Taylor, recibe su nombre del matemático británico Brook Taylor, quien lo enunció con mayor generalidad en 1712, aunque previamente James Gregory lo había descubierto en 1671. Este teorema permite obtener aproximaciones polinómicas de una función en un entorno de cierto punto en que la función sea diferenciable. Además el teorema permite acotar el error obtenido mediante dicha estimación.

2.1. Teorema de Taylor

Este teorema permite aproximar una función derivable en el entorno reducido alrededor de un punto $a \in$ (a, d) 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:

(1a)

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

O en forma compacta

(1b)

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

Donde $k!$ denota el factorial de $k, y R_n(f)$ es el resto, término que depende de x y es pequeño si x está próximo al punto a. Existen dos expresiones para R que se mencionan a continuación:

(2a)

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

donde a y x, pertenecen a los números reales, n a los enteros y ξ es un número real entre a y x:

(2b)

$$R_n(f) = \int_a^x \frac{f^{(n+1)}(t)}{n!}(x-t)^n\,dt$$

Si $R_n(f)$ es expresado de la primera forma, se le denomina Término complementario de Lagrange, dado que el Teorema de Taylor se expone como una generalización del Teorema del valor medio o Teorema de Lagrange, mientras que la segunda expresión de R muestra al teorema como una generalización del Teorema fundamental del cálculo integral.

Para algunas funciones $f(x)$, se puede probar que el resto, $R_n(f)$, se aproxima a cero cuando n se acerca al ∞; dichas funciones pueden ser expresadas como series de Taylor en un entorno reducido alrededor de un punto a y son denominadas funciones analíticas.

El teorema de Taylor con $R_n(f)$ expresado de la segunda forma es también válido si la función f tiene números complejos o valores vectoriales. Además existe una variación del teorema de Taylor para funciones con múltiples variables.

En LaTeX [1] es sencillo escribir expresiones matemáticas como $a = \sum_{i=1}^{10} {x_i}^3$ y deben ser escritas entre dos símbolos \$. Los superíndices se obtienen con el símbolo ^, y los subíndices con el símbolo _. Por ejemplo: $x^2 \times y^{\alpha+\beta}$.

2.1.1. Demostración

La demostración de la fórmula (1a), con el resto de la forma (2a), se sigue trivialmente del teorema de Rolle aplicado a la función:

$$F(y) = f(x) - f(y) - \frac{f'(y)}{1!}(x-y) - \ldots - \frac{f^{(n)}(y)}{n!}(x-y)^n$$

Un cálculo rutinario permite ver que la derivada de esta función cumple que:

$$F'(y) = -\frac{f^{(n+1)}(y)}{n!}(x-y)^n$$

Se define ahora la función G como:

$$G(y) = F(y) - \left(\frac{x-y}{x-a}\right)^{n+1} F(a)$$

Es evidente que esta función cumple $G(a)=G(x)=0$, y al ser esta función diferenciable, por el teorema de Rolle se sigue que:

$$\exists \xi \in (x,a) : G'(\xi) = 0$$

Y como:

$$0 = G'(\xi) = F'(\xi) + (n+1)\frac{(x-\xi)^n}{(x-a)^{n+1}}F(a)$$

Se obtiene finalmente que:

$$F(a) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}$$

Y sustituyendo en esta fórmula la definición de F(a), queda precisamente la fórmula (1a) con la forma del resto (2a).

2.1.2. Propiedades

$\alpha, \beta \in \mathrm{R}$, f y g funciones.

(1) $T_n(\alpha f + \beta g) = \alpha T_n(f) + \beta T_n(g)$

(2) $T_n(f \cdot g) = T_n(f) \cdot T_n(g)-$términos de orden $> n$

(3) $T_n(f/g) = \dfrac{T_n(f)}{T_n(g)}$ "haciendo división larga hasta n"

(4) $T_n(f \circ g) = T_n(f) \circ T_n(g)-$términos de orden $> n$

(5) $[T_n(f)]' = T_{n-1}(f')$

(6) $\displaystyle\int_a^x T_n(f)(t)dt = T_{n+1}(\int_a^x) f(t)dt$

(6)' $\displaystyle\int T_n(f) = T_{n+1}(\int f) + K$, $K \in \mathrm{R}$.

2.2. Coseno

En trigonometría el coseno (abreviado cos) de un ángulo agudo en un triángulo rectángulo se define como la razón entre el cateto adyacente a dicho ángulo y la hipotenusa:

$$cos(\alpha) = b/c$$

En virtud del Teorema de Tales, este número no depende del triángulo rectángulo escogido y, por lo tanto, está bien construido y define una función del ángulo α.

2.3. Aproximaciones

Es una representación inexacta que, sin embargo, es suficientemente fiel como para ser útil. Aunque en matemáticas la aproximación típicamente se aplica a números, también puede aplicarse a objetos tales como las funciones matemáticas, figuras geométricas o leyes físicas.

2.3.1. Series de Potencias

En matemáticas, una serie de Taylor es una representación de una función como una infinita suma de términos. Estos términos se calculan a partir de las derivadas de la función para un determinado valor de la variable (respecto de la cual se deriva), lo que involucra un punto específico sobre la función. Si esta serie está centrada sobre el punto cero, se le denomina serie de McLaurin.

Esta representación tiene tres ventajas importantes:

La derivación e integración de una de estas series se puede realizar término a término, que resultan operaciones triviales. Se puede utilizar para calcular valores aproximados de la función. Es posible demostrar que, si es viable la transformación de una función a una serie de Taylor, es la óptima aproximación posible. Algunas funciones no se pueden escribir como serie de Taylor porque tienen alguna singularidad. En estos casos normalmente se puede conseguir un desarrollo en serie utilizando potencias negativas de x (véase Serie de Laurent. Por ejemplo $f(x) = exp(-1/x^2)$ se puede desarrollar como serie de Laurent.

Capítulo 3

Procedimiento experimental

Para realizar los pertinentes experimentos hemos creado un programa en lenguaje Python, que para cada una de las cotas de error recibidas, halla el término en el cual el resto del polinomio de Taylor pasa a ser menor que el error. Una vez hallado el grado del resto utiliza este valor para representar el polinomio de Taylor que es de un grado menos que el grado del resto. También hemos creado un programa para que nos represente en una gráfica los polinomios de Taylor anteriormente hallados junto con el cos(x). Y otra gráfica que muestra el tiempo empleado en calcular cada polinomio y mostrarlo.

3.1. Descripción de los experimentos

Para realizar distintos experimentos debemos cambiar las cotas de error con lo que lograremos que se generen distintos polinomios de Taylor. Y gracias a la representación gráfica podemos ver cuanto se aproximan estos polinomios a la función cos(x). Otros experimentos posibles serían cambiar el valor del centro y también cambiar el valor del punto donde debe ser evaluado. En ambos casos con un objetivo similar al primer experimento.

3.2. Resultados obtenidos

Todos los experimentos fueron realizados para cinco cotas de error, estas son: [0.4789, 0.0456, 0.00005, 0.000000005, 0.000000000007].

En primer lugar ejecutamos el programa con el centro fijado en el punto 0 y con un valor de 2 para el punto.

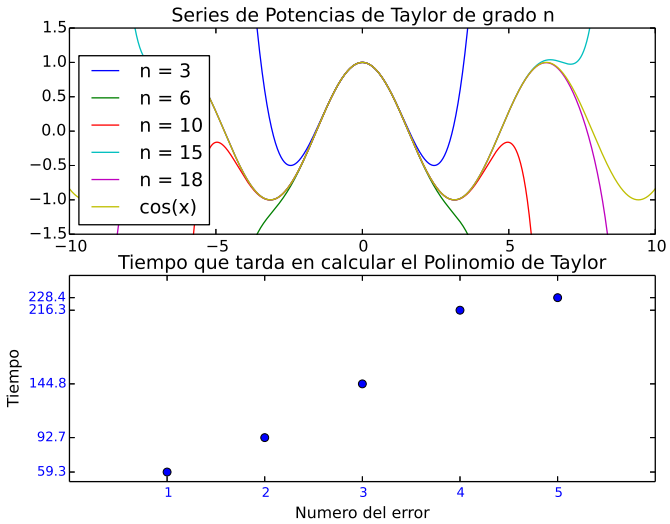

Figura 3.1: Grafico de Taylor

Podemos ver como cuanto mayor es el grado del polinomio la aproximación al coseno es mejor.

En segundo lugar hemos ejecutado el programa con el centro fijado en el punto 5 y con un valor de 2 para el punto.

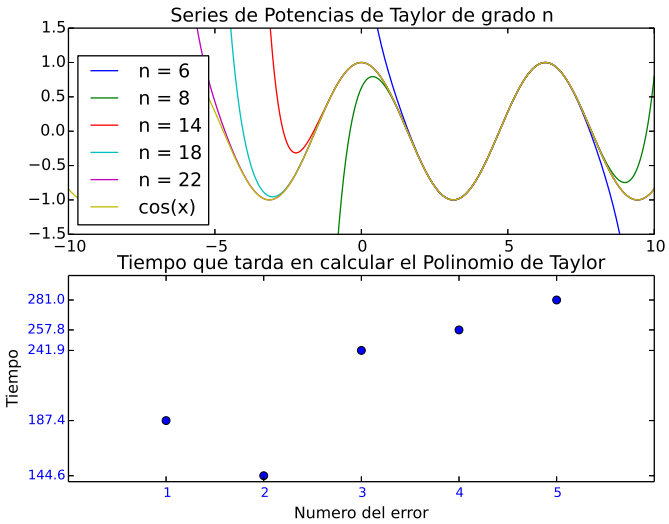

Figura 3.2: Valor del centro 5

En esta gráfica podemos ver como el polinomio aproxima al cos(x) pero siendo el punto 5 donde más se aproxima dado que es el centro.
Por último ejecutamos el programa con el centro fijado en el punto 0 y con un valor de 10 para el punto.

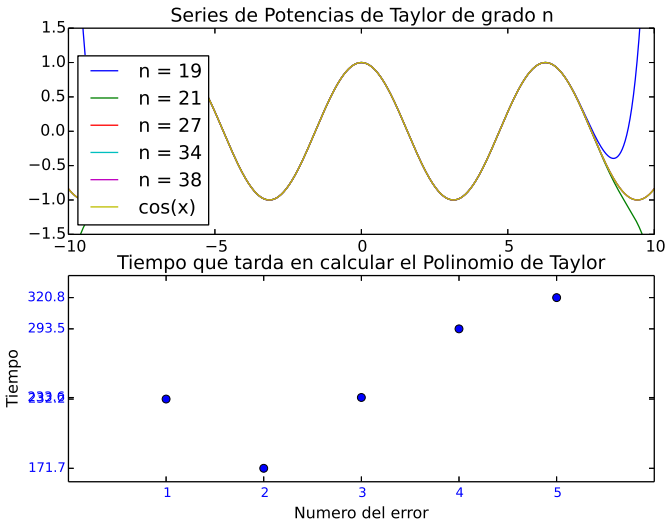

Figura 3.3: Valor del punto 10

Capítulo 4

Conclusiones

El desarrollo en series de Taylor de una función en un punto permite obtener resultados con una gran aproximación al valor real del la función en ese punto. Tiene muchas aplicaciones teóricas además de las prácticas.

Apéndice A

Código python

A.1. Programa python para hallar el polinomio de Taylor

```
#!/usr/bin/python
import random, sys
from sympy import *
import numpy as np
import math
import time
import timeit

Errores=[0.4789, 0.0456, 0.00005, 0.000000005, 0.000000000007]
x=2.0
c=0.0
f=(x+c)/2

valor_c = Symbol('c')
valor_a = Symbol('f')
funcion = cos(valor_c)
funcion_ = cos (valor_a)


def fac(n):
  if n == 0:
    return 1
  else:
    return n * fac(n-1)

def MostrarTaylor(c,n):
  for i in range(n + 1):
    derivada = eval(str(diff(funcion,valor_c,i)))
    if (i<1):
     sys.stdout.write((str(derivada)))
    if(i>=1):
     x=Symbol('x')
     a=(derivada*((x-c)**i))
     if (derivada != 0):
```

```
       if (derivada<0):
        a=-a
        sys.stdout.write((' - '+str(a) +' / '+str (i)+'!'))
       else:
        sys.stdout.write((' + '+str(a) +' / '+str (i)+'!'))
  print '\n'
  return 1

def graficaTaylor(c,n):
  v=0
  for i in range(n + 1):
    derivada = eval(str(diff(funcion,valor_c,i)))
    if (i<1):
     v=v+derivada
    if(i>=1):
     x=np.arange(-10,10,0.001)
     a=(derivada*((x-c)**i))
     if (derivada != 0):
       v=v+derivada*((x-c)**i)/fac(i)
  return v


def ErrorTaylor(x,c,error):
  i=0
  derivada = eval(str(diff(funcion_,valor_a,i)))
  polinomio = ((derivada/(fac(i)))*((x - c)**i))
  while (abs(polinomio)>=error):
    i+=1
    derivada = eval(str(diff(funcion_,valor_a,i)))
    polinomio = ((derivada/(fac(i)))*((x - c)**i))
  return i


if __name__=='__main__':
  for error in Errores:
    n=ErrorTaylor(x,c,error)
    print ('\n %3i iteraciones para dar un error <= %.15f') %(n,error)
    MostrarTaylor(c,n-1)
```

A.2. Programa para representar graficamente los polinomios hallados

```
#!/usr/bin/python
#!encoding: UTF-8

#import pylab as dibujo
import sys
from sympy import *
import math
import time
```

```
import timeit

import numpy as np
import matplotlib.pyplot as plt
import calculotaylor  #programa anterior


n=3



tiempo=[]
xtiempo=[]



graf1= plt.subplot(211)
print"Cargado el   0 por ciento de las Graficas"
i=0
for error in calculotaylor.Errores:
    start=time.time()
    i+=1
    n=calculotaylor.ErrorTaylor(calculotaylor.x,calculotaylor.c,error)
    x1=np.arange(-10,10,0.001)
    y=calculotaylor.graficaTaylor(calculotaylor.c,n)
    plt.plot(x1,y, label= 'n = %d' %(n-1))
    print"Cargado %3d por ciento de las Graficas" %(100*i/(len(calculotaylor.Errores))) # Calcula el % de lo
hace la espera mas amena.
    finish=time.time()-start
    tiempo=tiempo+[finish]
plt.plot(x1,np.cos(x1), label = 'cos(x)')
plt.title('Series de Potencias de Taylor de grado n')
plt.legend(loc = 3)
plt.ylim(-1.5,1.5)

for i in range (1,len(tiempo)+1):
  xtiempo=xtiempo+[i]

graf2=plt.subplot(212)
plt.title('Tiempo que tarda en calcular el Polinomio de Taylor')
plt.plot(xtiempo,tiempo, 'bo')
plt.xticks(xtiempo, size = 'small', color = 'b')
plt.yticks(tiempo, size = 'small', color = 'b')
plt.xlabel("Numero del error")
plt.ylabel("Tiempo")
plt.xlim(0,(len(tiempo)+1))

plt.savefig("Graficas.eps", dpi=100)
plt.show()
```

Bibliografía

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