Analisis de Fourier

Alumno: *Ángel López Manríquez*
Tema: *Graficador de series de Fourier*
Fecha: *17 de junio de 2019*
Grupo: *2CM4*

PROF. MIGUEL OLVERA ANDANA

Instituto Politecnico Nacional
Escuela Superior de Cómputo

1. Marco teorico

Definición 1.1. Producto interno Se dice que una funcion (,): $V \to \mathbb{K}$ es un producto interno si para $u,\ v,\ w\ \in V$, con V un espacio vectorial, $\alpha \in \mathbb{K}$ si se cumple que

1. $(u, v) = (v, u)$
2. $(\alpha u, v) = \alpha(u, v)$
3. $(u + v, w) = (u, w) + (v, w)$
4. $(u, u) > 0$ si $u \neq 0$
5. $(u, u) = 0$ si $u = 0$

Definición 1.2. El producto interno de funciones El producto interno de dos funciones f y g en el intervalo $[a, b]$ es

$$\int_a^b f(x)g(x)dx$$

Teorema 1.1 (Expansion ortogonal en series). *Suponga que* $\{\phi_k(x)\}_{k=0}^{\infty}$ *es un conjunto ortogonal en* $[a, b]$, *entonces una funcion puede ser escrita como*

$$f(x) = \sum_{k=0}^{\infty} \frac{(f, \phi_k)}{\|\phi_k\|^2} \phi_k$$

Definición 1.3. Las series de Fourier Una funicion f definida en un un intervalo $(-T/2, T/2)$ esta dada por

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(nw_0 t) + b_n \sin(nw_0 t)]$$

donde

$$a_0 = \frac{2}{T} \int_{-T/2}^{T/2} f(t)dt, \quad a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(nw_0 t)dt, \quad b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(nw_0 t)dt,$$

Teorema 1.2 (Convergencia de las series de Fourier). *Sea f una funcion continua a trozos. La serie de fourier convergera en los tramos continuos a f y convergera a los puntos discontinuos a la semisuma*

$$\frac{f(t+) + f(t-)}{2}$$

Definición 1.4. Paridad de funciones Se dice que una funcion f es par si $f(-x) = f(x)$ (simetrica respecto al eje y) e impar si $f(-x) = -f(x)$ (simetrica respecto al origen.

Teorema 1.3. *Propiedades de funciones pares e impares*

1. *El producto de dos funciones, ya sean ambas pares o impares, es par.*
2. *El producto de una funcion par con una impar es impar.*
3. *La suma de dos funciones pares da una funcion par.*
4. *La suma de dos funciones impares da una funcion impar.*
5. *Si f es par, entonces* $\int_{-p}^{p} f(x)dx = 2\int_0^p f(x)dx$
6. *Si f es impar, entonces* $\int_{-p}^{p} f(x)dx = 0$

Teorema 1.4 (Serie de cosenos)**.** *La series de Fourier para una funcion par en (-p, p) es*

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nw_0 t)$$

donde

$$a_0 = \frac{4}{T}\int_0^{T/2} f(t)dx, \quad a_n = \frac{4}{T}\int_0^{T/2} f(t)\cos(nw_0 t)dt$$

Teorema 1.5 (Serie de senos)**.** *La series de Fourier para una funcion impar en (-p, p) es*

$$f(t) = \sum_{n=1}^{\infty} b_n \sin(nw_0 t)$$

donde

$$b_n = \frac{4}{T}\int_0^{T/2} f(t)\sin(nw_0 t)dt$$

2. Planteamiento del problema

Dados los coeficientes $a_0, a_n y b_n$ y un periodo T, hacer un programa que grafique la serie de Fourier finita $S_5(t), S_{50}(t) y S_{100}(t)$.

3. Diseño y funcionamiento de la solución

Como bien es sabido, las series de Fourier nos permiten expresar una funcion periodica en suma de senos y cosenos. Como vemos en el marco teorico, solo es requerido el periodo y los coeficientes de Fourier para graficar la serie de Fourier.

El programa se hizo en el lenguaje python con la biblioteca matplotlib junto con una clase que evalua una expresion matematica para poder leer por teclado los valores con display inline (parecido a WolframAlpha).

El programa se hizo asumiendo la base ortogonal para la serie de Fourier $\{\cos(nwt), \sin(nwt)\}_{n=1}^{\infty}$. Como se ve en la implementacion, puede que el programa pueda fallar si la suma empieza en 2, pues estariamos haciendo una division entre 0. Ademas, los argumentos de las funciones seno y coseno son fijos, por lo que el programa no acepta algunas optimizaciones tales como que a_n solo tiene valores para n par o impar. Quitando estas excepciones, el programa funciona correctamente, incluzo es capaz de hallar la serie finita con mas terminios en un tiempo aceptable. Aqui se muestra el codigo para ejecutarse:

```python
import matplotlib.pyplot as plt
import numpy as np
from NumericStringParser import *

def am(n):
    # Se puede usar expran sin modificar
    # sin hacer global expran
    expr = expran.replace("m", '('+str(n)+')')
    print(expr)
    return nsp.eval(expr)

def bm(n):
    expr = exprbn.replace("m", '('+str(n)+')')
    print(expr)
    return nsp.eval(expr)
```

```

nsp = NumericStringParser()
period = nsp.eval(input("T: "))
a0 =  nsp.eval(input("a0: "))
expran = input("a_m: ")
exprbn = input("b_m: ")

w = 2 * np.pi / period
t = np.arange(-period, period, 0.001)

cos_n = lambda n: np.cos(n * w * t)
sin_n = lambda n: np.sin(n * w * t)
f_n = lambda n: (a0 / 2 + sum([am(k) * cos_n(k) + \
    bm(k) * sin_n(k) for k in range(2, n + 1)], 0))

for k in [5, 50, 100, 1000]:
    plt.plot(t, f_n(k), label = 'n = %d' % k)
plt.legend() # Habilita los labels
plt.title('Serie de Fourier')
plt.xlabel('Tiempo (t)')
plt.ylabel('Amplitud')
plt.show()
```

Con la clase de *NumericStringParser*

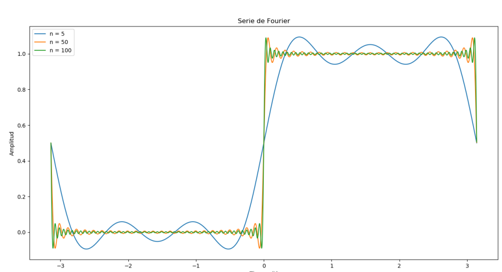
```
from __future__ import division
from pyparsing import (Literal, CaselessLiteral, Word, Combine, Group, Optional,
                       ZeroOrMore, Forward, nums, alphas, oneOf)
import math
import operator

__author__ = 'Paul McGuire'
__version__ = '$Revision: 0.0 $'
__date__ = '$Date: 2009-03-20 $'
__source__ = '''http://pyparsing.wikispaces.com/file/view/fourFn.py
http://pyparsing.wikispaces.com/message/view/home/15549426
'''
__note__ = '''
All I've done is rewrap Paul McGuire's fourFn.py as a class, so I can use it
more easily in other places.
'''


class NumericStringParser(object):
    '''
    Most of this code comes from the fourFn.py pyparsing example

    '''

    def pushFirst(self, strg, loc, toks):
        self.exprStack.append(toks[0])

    def pushUMinus(self, strg, loc, toks):
        if toks and toks[0] == '-':
            self.exprStack.append('unary -')

    def __init__(self):
```

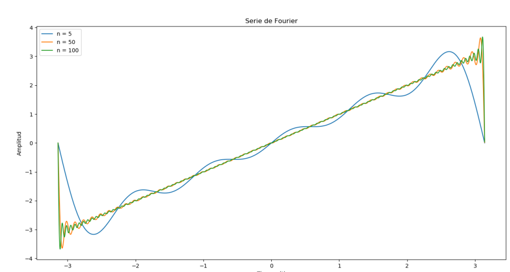
```
        """
        expop   :: '^'
        multop  :: '*' | '/'
        addop   :: '+' | '-'
        integer :: ['+' | '-'] '0'..'9'+
        atom    :: PI | E | real | fn '(' expr ')' | '(' expr ')'
        factor  :: atom [ expop factor ]*
        term    :: factor [ multop factor ]*
        expr    :: term [ addop term ]*
        """
        point = Literal(".")
        e = CaselessLiteral("E")
        fnumber = Combine(Word("+-" + nums, nums) +
                          Optional(point + Optional(Word(nums))) +
                          Optional(e + Word("+-" + nums, nums)))
        ident = Word(alphas, alphas + nums + "_$")
        plus = Literal("+")
        minus = Literal("-")
        mult = Literal("*")
        div = Literal("/")
        lpar = Literal("(").suppress()
        rpar = Literal(")").suppress()
        addop = plus | minus
        multop = mult | div
        expop = Literal("^")
        pi = CaselessLiteral("PI")
        expr = Forward()
        atom = ((Optional(oneOf("- +")) +
                 (ident + lpar + expr + rpar | pi | e |
                 ↪  fnumber).setParseAction(self.pushFirst))
                | Optional(oneOf("- +")) + Group(lpar + expr + rpar)
                ).setParseAction(self.pushUMinus)
        # by defining exponentiation as "atom [ ^ factor ]..." instead of
        # "atom [ ^ atom ]...", we get right-to-left exponents, instead of left-to-right
        # that is, 2^3^2 = 2^(3^2), not (2^3)^2.
        factor = Forward()
        factor << atom + \
            ZeroOrMore((expop + factor).setParseAction(self.pushFirst))
        term = factor + \
            ZeroOrMore((multop + factor).setParseAction(self.pushFirst))
        expr << term + \
            ZeroOrMore((addop + term).setParseAction(self.pushFirst))
        # addop_term = ( addop + term ).setParseAction( self.pushFirst )
        # general_term = term + ZeroOrMore( addop_term ) | OneOrMore( addop_term)
        # expr <<  general_term
        self.bnf = expr
        # map operator symbols to corresponding arithmetic operations
        epsilon = 1e-12
        self.opn = {"+": operator.add,
                    "-": operator.sub,
                    "*": operator.mul,
                    "/": operator.truediv,
                    "^": operator.pow}
        self.fn = {"sin": math.sin,
                   "cos": math.cos,
                   "tan": math.tan,
```

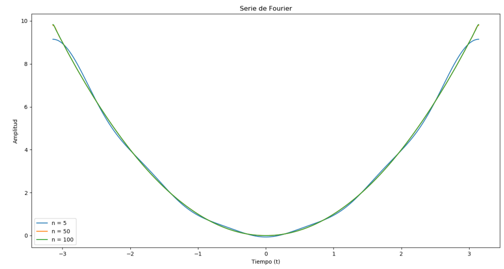
```
                    "sinh": math.sinh, #Agregado
                    "cosh": math.cosh,
                    "tanh": math.tanh,
                    "exp": math.exp,
                    "abs": abs,
                    "trunc": lambda a: int(a),
                    "round": round,
                    "sgn": lambda a: abs(a) > epsilon and cmp(a, 0) or 0}

    def evaluateStack(self, s):
        op = s.pop()
        if op == 'unary -':
            return -self.evaluateStack(s)
        if op in "+-*/^":
            op2 = self.evaluateStack(s)
            op1 = self.evaluateStack(s)
            return self.opn[op](op1, op2)
        elif op == "PI":
            return math.pi  # 3.1415926535
        elif op == "E":
            return math.e  # 2.718281828
        elif op in self.fn:
            return self.fn[op](self.evaluateStack(s))
        elif op[0].isalpha():
            return 0
        else:
            return float(op)

    def eval(self, num_string, parseAll=True):
        self.exprStack = []
        results = self.bnf.parseString(num_string, parseAll)
        val = self.evaluateStack(self.exprStack[:])
        return val
```

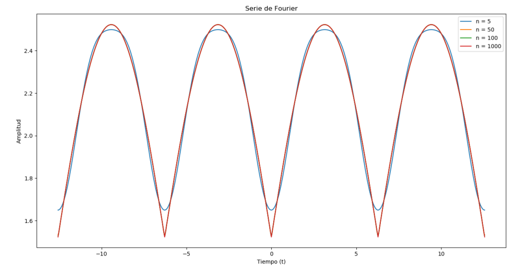
4. Graficas

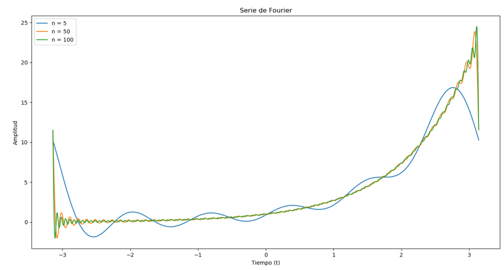
Aqui podemos ver una demostracion del programa con los ejercicios de la pagina 21 del libro Analisis de Fourier de HSU,











Aqui vemos como se ejecuta el programa desde la cmd

```
C:\Users\ANGEL\Documents\codes\python\FourierSeries
λ make
python fourier.py
T: 1
a0: 2*(1-exp(-1))
a_m: ((2*exp(-1))/((m*2*pi)^2+1))*(exp(1)-1)
b_m: ((2*exp(-1)*m*2*pi)/((m*2*pi)^2+1))*(exp(1)-1)
```

y los ejercicios hechos en clase