UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA,INFORMÁTICA Y MECÁNICA

INGENIERÍA INFORMÁTICA Y DE SISTEMAS

Guia 4: Algoritmos FFT

Alumnos :
Ian Logan Will Quispe Ventura *211359*
Jhon Esau Pumachoque Choquenaira *210940*
Ciro Gabriel Callapiña Castilla *134403*
Luis Manuel Tinoco Ccoto *204807*
Jorge Enrique Zegarra Rojas *161534*

Docente:
Lauro Enciso Rodas

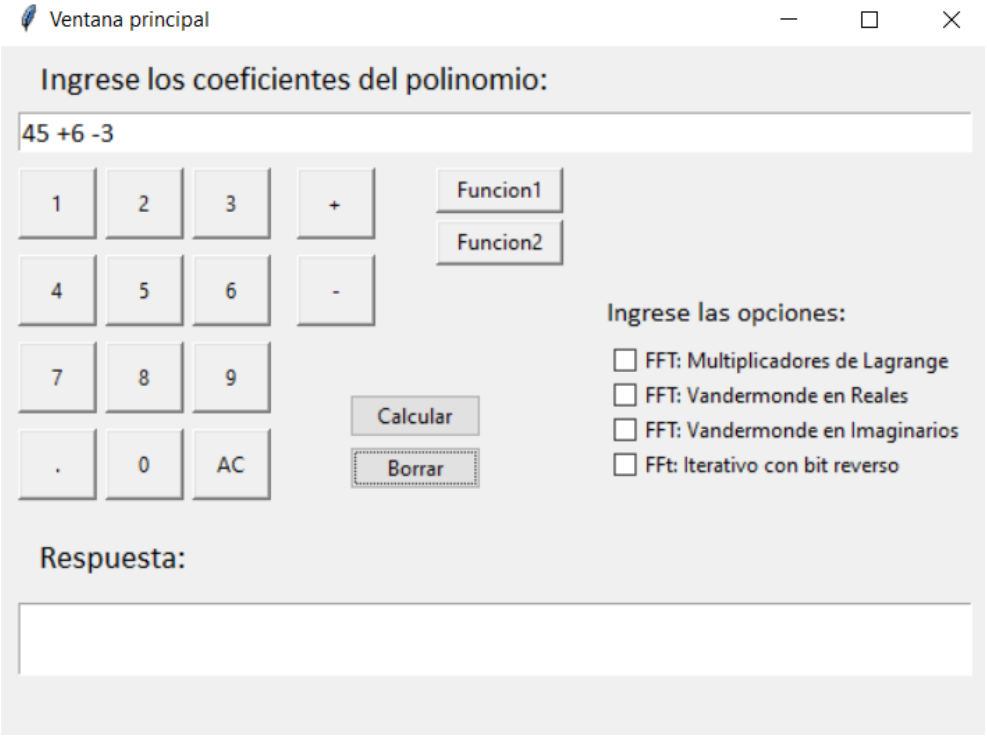
Curso:
Algoritmos Avanzados

Cusco - Perú
2024 - I

1 Guia de Usuario

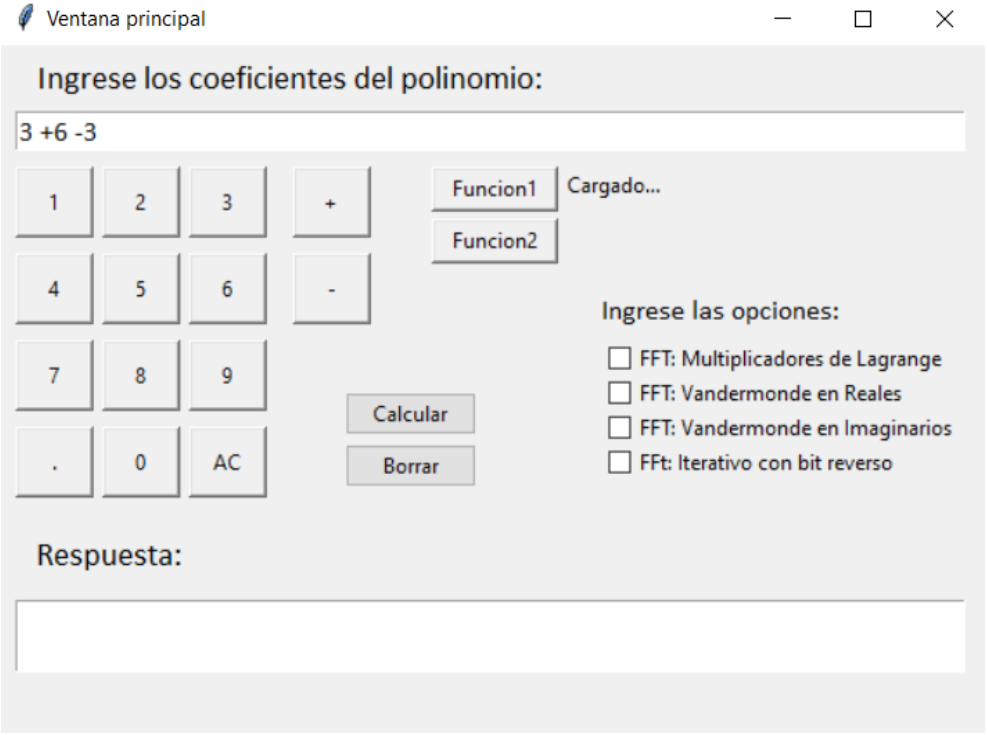
1.1 Ingresar Primera funcion

Ingresamos con el uso de solo los botones, los coeficientes del primer polinomio en orden ascendente.
Ejm: $45 + 6x - 3x^2$ -> $45 + 6 - 3$
Luego hacemos click en el boton "Funcion1" Para cargar la función. Cuando se carge la función, aparecera al costado del boton un texto que diga "cargado..." y desaparecera lo escrito.



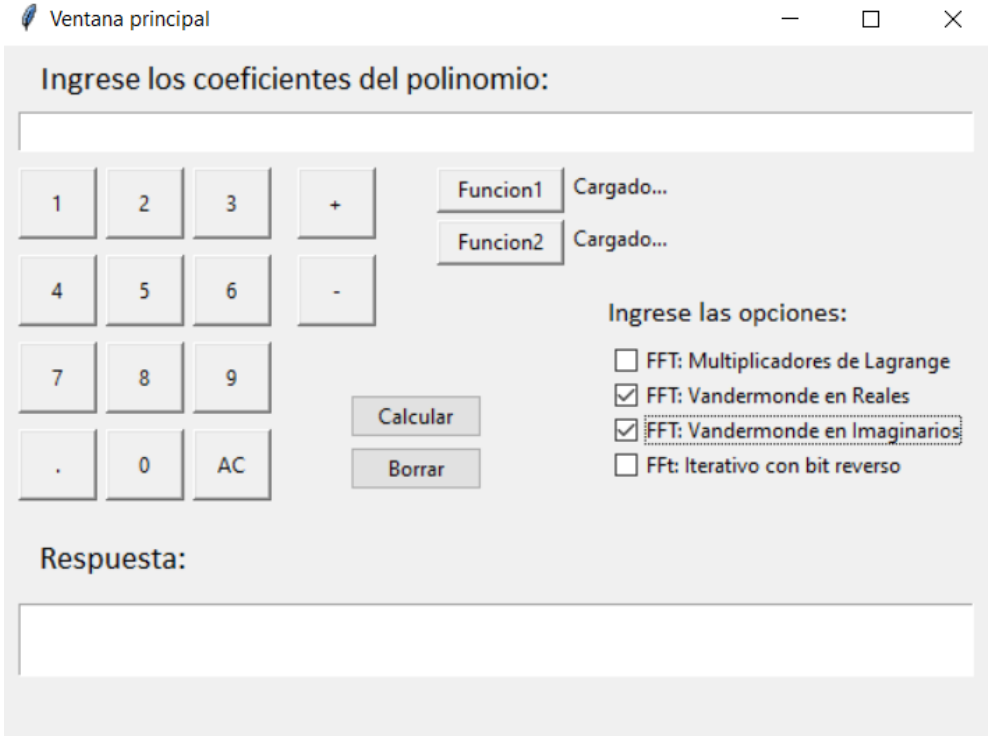
1.2 Ingresar la Segunda funcion

Escribimos con la ayuda de los botones numeros, "+" y "-" los coeficientes de nuestra segunda función, para luego hacer click en el boton "Funcion2" con lo cual la segunda fución ya estara cargada.



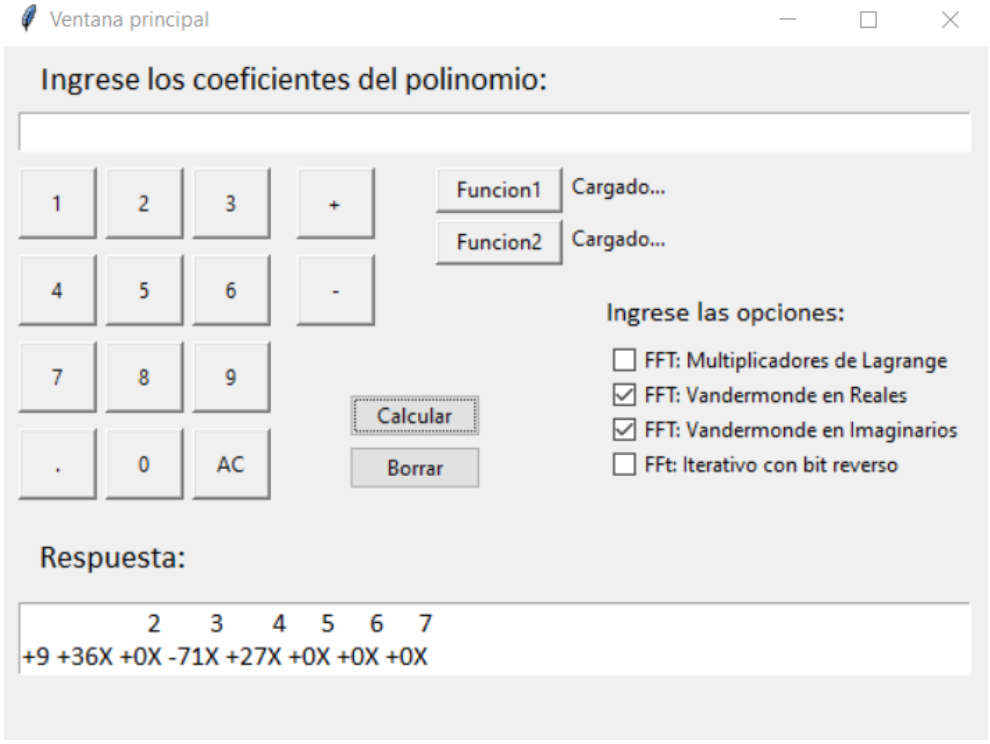
1.3 Seleciónar los metodos

Nos dirigimos a la parte derecha de nuestra ventana, en donde tenemos 4 celdas, para selecionar los metodos con los que queremos calcular la Transformada Rapida de Fourier.
Luego hacemos click en el boton que dice "Calcular", el cual se encuentra en el medio de la ventana.



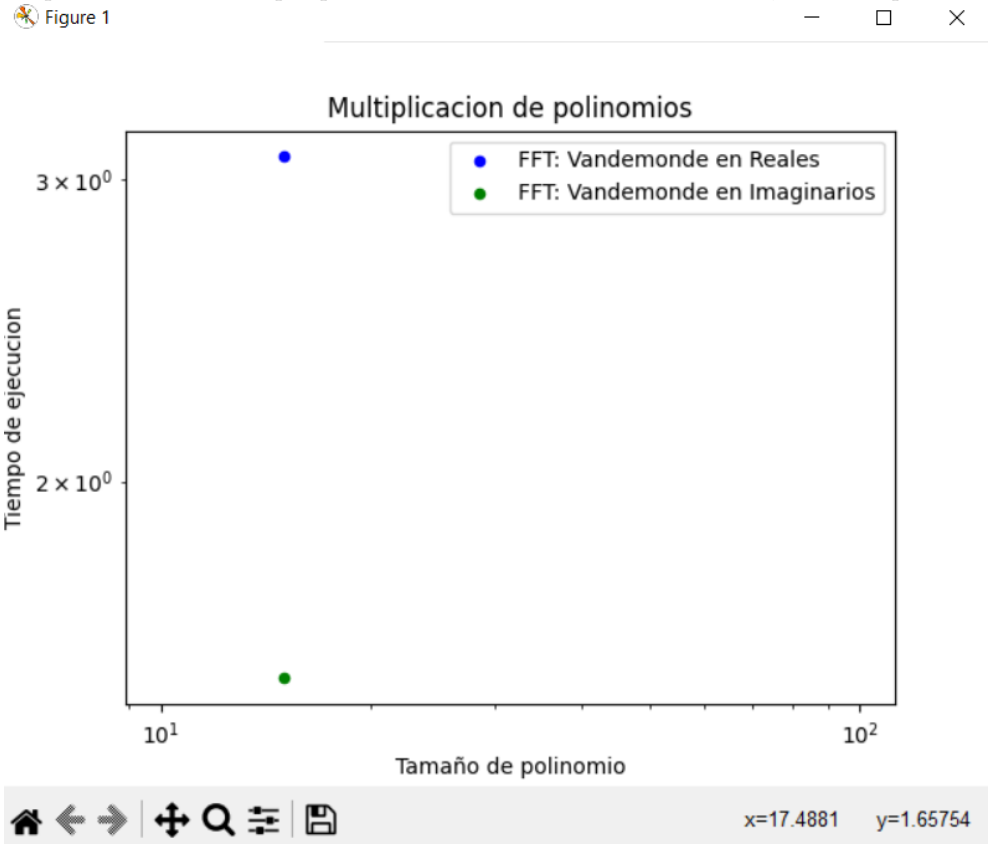
1.4 Respuesta

En la parte de abajo tendremos un recudro en el cual dice "Respuesta", debajo de el es donde aparecera el resultado de la Transformada Rápida de Fourier



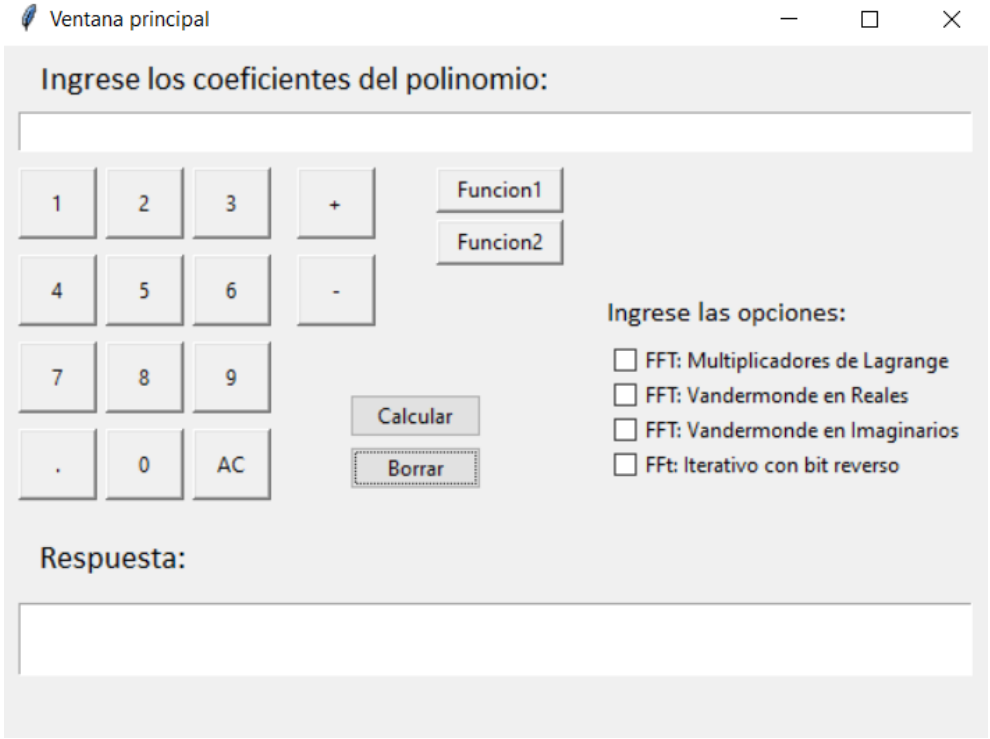
1.5 Grafica

Luego de hacer click en el boton ”Calcular” veremos que también se habre otra ventana, en está, apareceran los graficos conparativos de el tiempo que se demoro en calcular cada metodo, este tiempo esta dado en ms (milesimas de segundos)



1.6 Borrar

Para borrar todo lo que hemos hecho y volver ha hacer otro calculo, tendremos que hacer click en el boton ”Borrar” el resultado es que todo se limpia, como se muestra en la figura.



2 Implementación

2.1 Diseño Principal

Aqui mostramos el código de la parte grafica para mostrarse al usuario.

```
import tkinter as tk
from tkinter import ttk
from fft_imaginario import *
from fft_iterativo import *
from fft_lagrange import *
from fft_reales import *
#JEPC
    ------------------------------------------------------------------------------------------


from JEPC.Graficos import *
#
    ------------------------------------------------------------------------------------------

# Varibles globales
i = 0
Fun1 = []
Fun2 = []

def abrir_ventana_secundaria():
    # Crear una ventana secundaria.
    ventana_secundaria = tk.Toplevel()
    ventana_secundaria.title("Ventana secundaria")
    ventana_secundaria.config(width=300, height=200)

    # Crear un botón dentro de la ventana secundaria para cerrarla.
    boton_cerrar = ttk.Button(ventana_secundaria, text="Cerrar ventana", command=
        ventana_secundaria.destroy)
    boton_cerrar.place(x=75, y=75)

def click_boton(valor, e_texto, texto_var):
    global i
    # Insertar el valor en el Entry
    e_texto.insert(i, str(valor))
    i += len(valor)
    # Actualizar la variable de seguimiento
    texto_var.set(e_texto.get())
    punto = True

def click_borrar(e_texto, texto_var):
    global i
    # Borrar los valores
    e_texto.delete(0, i)
    i = 0
    # Actualizar la varible de seguimiento
    texto_var.set(e_texto.get())

def click_funcion1(e_texto, texto_var, label1_var):
    global i, Fun1
    # Sacar los valores de la función
    Aux1 = e_texto.get()
    Aux2 = [int(x) for x in Aux1.split(" ")]
    Fun1 = Aux2
    # Actualizar valores
    label1_var.set("Cargado...")
```

```
    texto_var.set("")
    i = 0
    #print(Fun1)

def click_funcion2(e_texto, texto_var, label2_var):
    global i, Fun2
    # Sacar los valores de la función
    Aux1 = e_texto.get()
    Aux2 = [int(x) for x in Aux1.split(" ")]
    Fun2 = Aux2
    # Actualizar valores
    label2_var.set("Cargado...")
    texto_var.set("")
    i = 0
    #print(Fun2)

def Calcular(Lagrange_v, Iterativo_v, VandermondeI_v, VandermondeR_v):
    global Fun1, Fun2
    Resp = ""
    # Tiempos = []
    Tiempos = [0, 0, 0, 0] #jepc
    Respuestas = []
    if (Lagrange_v.get()):
        TiempoMS, RespuestaL = appFFT_Lagrange(Fun1, Fun2)
        #Tiempos.append(TiempoMS)
        Tiempos[0] = TiempoMS #jepc
        Respuestas.append(RespuestaL)
    if (Iterativo_v.get()):
        TiempoMS, RespuestaI = appFFT_Iterativo(Fun1, Fun2)
        #Tiempos.append(TiempoMS)
        Tiempos[1] = TiempoMS #jepc
        Respuestas.append(RespuestaI)
    if (VandermondeI_v.get()):
        TiempoMS, RespuestaVI = appFFT_Imaginario(Fun1, Fun2)
       # Tiempos.append(TiempoMS)
        Tiempos[2] = TiempoMS #jepc
        Respuestas.append(RespuestaVI)
    if (VandermondeR_v.get()):
        TiempoMS, RespuestaVR = appFFT_Reales(Fun1, Fun2)
       # Tiempos.append(TiempoMS)
        Tiempos[3] = TiempoMS #jepc
        Respuestas.append(RespuestaVR)

    Aux = Respuestas[0]
    Respuesta = [str(k) if (k < 0) else "+"+str(k) for k in Aux]
    for k in range(0, len(Respuesta)):
        Resp += " "*len(Respuesta[k])*2
        if (k <= 1):
            if (k == 0):
                Resp += " "
            else:
                Resp += "  "
        else:
            Resp += " " + str(k)
    Resp += "\n"
    for k in range(0, len(Respuesta)):
        Resp += Respuesta[k]
        if (k == 0):
            Resp += " "
        else:
            Resp += "X "
```

```
    texto_respuesta.delete("1.0", "end")
    texto_respuesta.insert("1.0", Resp)

    tamP =  len(Fun1) - 1 if len(Fun1) >= len(Fun2) else len(Fun2) -1
    #print(bool(Lagrange_v))
    FC_Graficar(Lagrange_v,VandermondeR_v,VandermondeI_v,Iterativo_v,tamP,Tiempos)


def click_borrarTodo():
    label1_var.set("")
    label2_var.set("")
    texto_respuesta.delete("1.0", "end")
    Lagrange_v.set(False)
    VandermondeR_v.set(False)
    VandermondeI_v.set(False)
    Iterativo_v.set(False)

# Crear la ventana principal
ventana_principal = tk.Tk()
ventana_principal.title("Ventana principal")
ventana_principal.config(width=570, height=400)

# Variables auxiliares
texto_var = tk.StringVar()  # Variable de seguimiento entrada texto
label1_var = tk.StringVar()  # Variable de seguimiento para cargar funcion1
label2_var = tk.StringVar()  # Variable de seguimiento para cargar funcion2

# Texto que se muestra en pantalla
lblFuncion = ttk.Label(ventana_principal, text="Ingrese los coeficientes del polinomio: ",
    font=("Calibri 14"))
lblFuncion.place(x=20, y=5)
txtFuncion = tk.Entry(ventana_principal, font=("Calibri 12"), width= 68, textvariable=
   texto_var)
txtFuncion.place(x=10, y=38)

# Botones Númericos
boton1 = tk.Button(ventana_principal, text="1", width=5, height= 2, command=lambda:
   click_boton("1", txtFuncion, texto_var))
boton1.place(x=10, y=70)
boton2 = tk.Button(ventana_principal, text="2", width=5, height= 2, command=lambda:
   click_boton("2", txtFuncion, texto_var))
boton2.place(x=60, y=70)
boton3 = tk.Button(ventana_principal, text="3", width=5, height= 2, command=lambda:
   click_boton("3", txtFuncion, texto_var))
boton3.place(x=110, y=70)
boton4 = tk.Button(ventana_principal, text="4", width=5, height= 2, command=lambda:
   click_boton("4", txtFuncion, texto_var))
boton4.place(x=10, y=120)
boton5 = tk.Button(ventana_principal, text="5", width=5, height= 2, command=lambda:
   click_boton("5", txtFuncion, texto_var))
boton5.place(x=60, y=120)
boton6 = tk.Button(ventana_principal, text="6", width=5, height= 2, command=lambda:
   click_boton("6", txtFuncion, texto_var))
boton6.place(x=110, y=120)
boton7 = tk.Button(ventana_principal, text="7", width=5, height= 2, command=lambda:
   click_boton("7", txtFuncion, texto_var))
boton7.place(x=10, y=170)
boton8 = tk.Button(ventana_principal, text="8", width=5, height= 2, command=lambda:
   click_boton("8", txtFuncion, texto_var))
boton8.place(x=60, y=170)
```

```
boton9 = tk.Button(ventana_principal, text="9", width=5, height= 2, command=lambda:
    click_boton("9", txtFuncion, texto_var))
boton9.place(x=110, y=170)
boton0 = tk.Button(ventana_principal, text="0", width=5, height= 2, command=lambda:
    click_boton("0", txtFuncion, texto_var))
boton0.place(x=60, y=220)

# Botones de escritura auxiliares
boton_borrar = tk.Button(ventana_principal, text= "AC", width= 5, height= 2, command=
    lambda: click_borrar(txtFuncion, texto_var))
boton_borrar.place(x = 110, y = 220)
boton_punto = tk.Button(ventana_principal, text=".", width= 5, height= 2, command= lambda:
     click_boton(".", txtFuncion, texto_var))
boton_punto.place(x= 10, y=220)
boton_suma = tk.Button(ventana_principal, text= "+", width= 5, height= 2, command= lambda:
     click_boton(" +", txtFuncion, texto_var))
boton_suma.place(x= 170, y=70)
boton_resta = tk.Button(ventana_principal, text= "-", width= 5, height= 2, command= lambda
    : click_boton(" -", txtFuncion, texto_var))
boton_resta.place(x= 170, y=120)

# label para mostrar cargado de 1ra y 2da función
label_funcion1 = tk.Label(ventana_principal, text="...", textvariable= label1_var)
label_funcion1.place(x=325, y=70)
label_funcion2 = tk.Label(ventana_principal, text="...", textvariable= label2_var)
label_funcion2.place(x=325, y=100)
# botones para cargar las funciones
boton_funcion1 = tk.Button(ventana_principal, text= "Funcion1", width= 9, height= 1,
    command= lambda: click_funcion1(txtFuncion, texto_var, label1_var))
boton_funcion1.place(x= 250, y = 70)
boton_funcion2 = tk.Button(ventana_principal, text= "Funcion2", width= 9, height= 1,
    command= lambda: click_funcion2(txtFuncion, texto_var, label2_var))
boton_funcion2.place(x= 250, y = 100)

# Mostrar el ingreso de opciones
label_Opciones = tk.Label(ventana_principal, text="Ingrese las opciones:", font=("Calibri
    12"))
label_Opciones.place(x=345, y=140)
# Crear variables para almacenar los estados de los chekbox (bool)
Lagrange_v = tk.BooleanVar()
VandermondeR_v = tk.BooleanVar()
VandermondeI_v = tk.BooleanVar()
Iterativo_v = tk.BooleanVar()
# Crear los chekbox con texto
chkLagrange = ttk.Checkbutton(ventana_principal, text="FFT: Multiplicadores de Lagrange",
    variable=Lagrange_v)
chkLagrange.place(x= 350, y= 170)
chkVandermondeR = ttk.Checkbutton(ventana_principal, text="FFT: Vandermonde en Reales",
    variable=VandermondeR_v)
chkVandermondeR.place(x= 350, y= 190)
chkVandermondeI = ttk.Checkbutton(ventana_principal, text="FFT: Vandermonde en Imaginarios
    ", variable=VandermondeI_v)
chkVandermondeI.place(x= 350, y= 210)
chkIterativo = ttk.Checkbutton(ventana_principal, text="FFt: Iterativo con bit reverso",
    variable=Iterativo_v)
chkIterativo.place(x= 350, y= 230)

# Parte de respuesta
lblRespuesta = ttk.Label(ventana_principal, text="Respuesta: ", font=("Calibri 14"))
lblRespuesta.place(x=20, y=280)
texto_respuesta = tk.Text(ventana_principal, width=68, height=2, font=("Calibri", 12))
```

```
texto_respuesta.place(x=10, y=320)

# Crear un botón para calcular la respuesta
boton_abrir = ttk.Button(ventana_principal, text="Calcular", command= lambda: Calcular(
    Lagrange_v, Iterativo_v, VandermondeI_v, VandermondeR_v))
boton_abrir.place(x=200, y=200)

# Crear un botón para borrar todo
boton_borrarTodo = ttk.Button(ventana_principal, text="Borrar", command= lambda:
    click_borrarTodo())
boton_borrarTodo.place(x=200, y=230)

ventana_principal.mainloop()
```

2.2 Graficos Estadisticos

A continuación se muestra el código en python el cual sirve para mostrar un gráfico estadistico, en base a los datos dados

```
import matplotlib.pyplot as plt
import numpy as np
import time
import sys
import os
proyecto_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
sys.path.append(proyecto_dir)

from fft_lagrange import multiplicar_polinomios as fftLagrange
from fft_imaginario import multiplicar_polinomios as FFTImaginario
from fft_iterativo import multiplicar_polinomios as FFTIterativo
from fft_reales import multiplicar_polinomios as FFTReales

tiempos = [[],[],[],[]]
tamanios = [2**x for x in range(1,11)]

def JEPC_G_ML():
    for tamn in tamanios:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
        fftLagrange(A, B)
        fin = time.perf_counter()


        tiempos[0].append((float(fin) - float(inicio))*1000)
def JEPC_G_FFT_Reales():
    for tamn in tamanios:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
        FFTReales(A, B)
        fin = time.perf_counter()

        tiempos[1].append((float(fin) - float(inicio))*1000)
def JEPC_G_FFT_Imaginarios():
    for tamn in tamanios:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
```

```
        C = FFTImaginario(A, B)
        fin = time.perf_counter()

        tiempos[2].append((float(fin) - float(inicio))*1000)
def JEPC_G_FFT_Iterarivo_bitReverso():
    for tamn in tamanios:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
        C = FFTIterativo(A, B)
        fin = time.perf_counter()


        tiempos[3].append((float(fin) - float(inicio))*1000)

# Pruba de tiempos de ejecucion con valores aletorios
def FC_GraficarPruebaEstres(langrange, vandermonderR, vandermonderI,iterativo,
    tamanioPolinomio = 0,times = [],respuestas = []):
    JEPC_G_ML()
    JEPC_G_FFT_Reales()
    JEPC_G_FFT_Imaginarios()
    JEPC_G_FFT_Iterarivo_bitReverso()
    # Crear el gráfico de dispersión con puntos de diferentes colores
    plt.scatter( tamanios,tiempos[0], color='red', label='Multiplicadores de Lagrange',
        marker='o',s=20)
    plt.scatter( tamanios,tiempos[1], color='blue', label='FFT Reales', marker='o',s=20)
    plt.scatter( tamanios,tiempos[2], color='green', label='FFT Imaginarios', marker='o',s
        =20)
    plt.scatter( tamanios,tiempos[3], color='black', label='Iterativo bit reverso', marker
        ='o',s=20)

    # Agregar etiquetas y título
    plt.xlabel('Tamaño de polinomio')
    plt.ylabel('Tiempo de ejecucion')
    plt.title('Multiplicacion de polinomios')

    # Agregar leyenda
    plt.legend()

    plt.xscale('log')
    plt.yscale('log')
    # Mostrar el gráfico
    plt.show()

def FC_Graficar(Lagrange_v,VandermondeR_v,VandermondeI_v,Iterativo_v, tamanioPolinomio =
    0,times = []):

    # Crear el gráfico de dispersión con puntos de diferentes colores
    if(Lagrange_v.get()):
        plt.scatter( tamanioPolinomio,times[0], color='red', label='FFT: Multiplicadores
            de Lagrange', marker='o',s=20)
    if(VandermondeR_v.get()):
        plt.scatter( tamanioPolinomio,times[3], color='blue', label='FFT: Vandemonde en
            Reales', marker='o',s=20)
    if(VandermondeI_v.get()):
        plt.scatter( tamanioPolinomio,times[2], color='green', label='FFT: Vandemonde en
            Imaginarios', marker='o',s=20)
    if(Iterativo_v.get()):
        plt.scatter( tamanioPolinomio,times[1], color='black', label='FFT: Iterativo con
            bit reverso', marker='o',s=20)
```

```
    # Agregar etiquetas y título
    plt.xlabel('Tamaño de polinomio')
    plt.ylabel('Tiempo de ejecucion')
    plt.title('Multiplicacion de polinomios')

    # Agregar leyenda
    plt.legend()

    plt.xscale('log')
    plt.yscale('log')
    # Mostrar el gráfico
    plt.show()
```

2.3 FFT: Vandermont Imaginaria

Código pyhonn que sirve para calcular y tomar el tiempo de la Transformada Rapida de Fourier Usando Vandermont Imaginaria.

```
import numpy as np
import matplotlib.pyplot as plt
import time


def FFT(P):
    """
        Calcula la transformada rapida de Fourier de manera
        recursiva y usando el concepto de matriz de Vandermont en
        imaginarios para un polinomio P
        :param P: Lista de coeficientes del polinomio
        :return: Lista de coeficientes de la transformada de Fourier del polinomio
    """
    n = len(P)
    if n == 1:
        return P

    w = np.exp(-2 * np.pi * 1j / n)

    Pe, Po = P[::2], P[1::2]
    ye, yo = FFT(Pe), FFT(Po)

    y = [0] * n

    for j in range(n // 2):
        y[j] = ye[j] + (w**j) * yo[j]
        y[j + n // 2] = ye[j] - (w**j) * yo[j]

    return y


def IFFT(P):
    """
        Calcula la transformada rapida de Fourier inversa de un polinomio P
        usando el concepto de matriz de Vandermont en imaginarios
        :param P: Lista de coeficientes de la transformada de Fourier del polinomio
        :return: Lista de coeficientes del polinomio resultado original
    """
    n = len(P)
    if n == 1:
        return P
```

```
    w = np.exp(2 * np.pi * 1j / n)

    Pe, Po = P[::2], P[1::2]
    ye, yo = IFFT(Pe), IFFT(Po)

    y = [0] * n

    for j in range(n // 2):
        y[j] = ye[j] + (w**j) * yo[j]
        y[j + n // 2] = ye[j] - (w**j) * yo[j]

    return y


def multiplicar_polinomios(A, B):
    """
        Multiplica dos polinomios A y B utilizando la Transformada Rapida de Fourier (FFT)
            .
        :param A: Lista de coeficientes del primer polinomio
        :param B: Lista de coeficientes del segundo polinomio
        :return: Lista de coeficientes del polinomio resultado de la multiplicacion
    """
    m = len(A)
    n = len(B)
    k = 2 ** (int(np.log2(m + n - 1)) + 1)

    A.extend([0] * (k - m))
    B.extend([0] * (k - n))

    ya = FFT(A)
    yb = FFT(B)

    yc = [ya[i] * yb[i] for i in range(k)]

    C = [int((val / k).real + 0.5) for val in IFFT(yc)]

    return C

def appFFT_Imaginario(A, B):
    inicio = time.perf_counter()
    C = multiplicar_polinomios(A, B)
    fin = time.perf_counter()
    TiempoMS = (float(fin) - float(inicio))*1000
    return TiempoMS, C

def main():

    tiempos = []
    tamagnos = [2**i for i in range(1, 11)]

    for tamn in tamagnos:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
        C = multiplicar_polinomios(A, B)
        fin = time.perf_counter()

        tiempos.append(fin - inicio)
```

```
    plt.plot(tamagnos, tiempos, marker='o')
    plt.xlabel('Tamaño del polinomio')
    plt.ylabel('Tiempo de ejecucion (s)')
    plt.title('Tiempo de ejecucion para FFT con imaginarios')
    plt.xscale('log')
    plt.yscale('log')
    plt.show()


if __name__ == '__main__':
    main()
```

2.4 FFT: Vandermount Reales

Código en python, para calcular la Transformada Rapida de Fourier usando Vandermount Reales

```
import numpy as np
import time

def FFT(P):
    n = len(P)
    if n == 1:
        return P

    vandermonde_matrix = []
    for i in range(n):
        row = []
        for j in range(n):
            value = np.cos(2 * np.pi * i * j / n)
            row.append(value)
        vandermonde_matrix.append(row)

    W = np.array(vandermonde_matrix)

    y = np.dot(W, P)

    return y

def IFFT(P):
    n = len(P)
    if n == 1:
        return P

    vandermonde_matrix = []
    for i in range(n):
        row = []
        for j in range(n):
            value = np.cos(2 * np.pi * i * j / n)
            row.append(value)
        vandermonde_matrix.append(row)

    transposed_vandermonde_matrix = np.transpose(np.array(vandermonde_matrix))

    W_inv = transposed_vandermonde_matrix / n

    y = np.dot(W_inv, P)

    return y

def multiplicar_polinomios(A, B):
    m = len(A)
```

```
    n = len(B)
    k = 2 ** (int(np.log2(m + n - 1)) + 1)

    A.extend([0] * (k - m))
    B.extend([0] * (k - n))

    ya = FFT(A)
    yb = FFT(B)

    yc = np.fft.ifft(np.multiply(ya, yb)).real

    C = [int(round(val)) for val in yc[:m+n-1]]

    return C

# Programa que es llamado por otros modulos, para ejecutarse
def appFFT_Reales(A, B):
    inicio = time.perf_counter()
    C = multiplicar_polinomios(A, B)
    fin = time.perf_counter()
    TiempoMS = (float(fin) - float(inicio))*1000
    return TiempoMS, C

def main():
    A = [5, 10, 15]
    B = [20, 25, 30]

    C = multiplicar_polinomios(A, B)

    print(C)

if __name__ == "__main__":
    main()
```

2.5 Bit Reverso

Código en python, para calcular y medir el tiempo de la Transformada Rapida de Fourier usando el metodo iterativo con Bit reverso.

```
import numpy as np
import matplotlib.pyplot as plt
import time


def bit_reverso(n):
    """
        Realiza el bit reverso de un numero n
        :param n: Numero entero al que se le va a realizar el bit reverso
        :return: Lista de indices con los bits invertidos
        ...
    """
    num_bits = int(np.log2(n))
    indices_reversos = [0] * n
    for i in range(n):
        indices_reversos[i] = int(
            format(i, '0' + str(num_bits) + 'b')[::-1], 2)
    return indices_reversos


def FFT(P):
```

```
    """
        Calcula la transformada rapida de Fourier usando iteracion
        y el concepto de operacion butterfly o mariposa para un polinomio P
        :param P: Lista de coeficientes del polinomio
        :return: Lista de coeficientes de la transformada de Fourier del polinomio
        ...
    """

    n = len(P)
    indices = bit_reverso(n)
    P = [P[i] for i in indices]

    for s in range(1, int(np.log2(n)) + 1):
        m = 2 ** s
        w_m = np.exp(-2 * np.pi * 1j / m)
        for k in range(0, n, m):
            w = 1
            for j in range(m // 2):
                t = w * P[k + j + m // 2]
                u = P[k + j]
                P[k + j] = u + t
                P[k + j + m // 2] = u - t
                w = w * w_m

    return P


def IFFT(P):
    """
        Calcula la transformada rapida de Fourier inversa de un polinomio P
        usando iteracion y el concepto de operacion butterfly o mariposa
        :param P: Lista de coeficientes de la transformada de Fourier del polinomio
        :return: Lista de coeficientes del polinomio original
        ...
    """
    n = len(P)
    indices = bit_reverso(n)
    P = [P[i] for i in indices]

    for s in range(1, int(np.log2(n)) + 1):
        m = 2 ** s
        w_m = np.exp(2 * np.pi * 1j / m)
        for k in range(0, n, m):
            w = 1
            for j in range(m // 2):
                t = w * P[k + j + m // 2]
                u = P[k + j]
                P[k + j] = u + t
                P[k + j + m // 2] = u - t
                w = w * w_m

    P = [val / n for val in P]

    return P


def multiplicar_polinomios(A, B):
    """
        Multiplica dos polinomios A y B utilizando la Transformada Rapida de Fourier (FFT)
            .
        :param A: Lista de coeficientes del primer polinomio
```

```
        :param B: Lista de coeficientes del segundo polinomio
        :return: Lista de coeficientes del polinomio resultado de la multiplicacion
        ...
    """
    m = len(A)
    n = len(B)
    k = 2 ** (int(np.log2(m + n - 1)) + 1)

    A.extend([0] * (k - m))
    B.extend([0] * (k - n))

    ya = FFT(A)
    yb = FFT(B)

    yc = [ya[i] * yb[i] for i in range(k)]

    C = [int(val.real + 0.5) for val in IFFT(yc)]

    return C

# Módulo que es llamado por otros programas
def appFFT_Iterativo(A, B):
    # ------ INICIO de medicion de tiempo
    inicio = time.perf_counter()
    C = multiplicar_polinomios(A, B)
    fin = time.perf_counter()
    # ------ FIN
    TiempoMS = (float(fin) - float(inicio))*1000
    return TiempoMS, C

def main():
    # para registrar los tiempos de ejecucion para diferentes tamaños de polinomios
    tiempos = []
    tamagnos = [2**i for i in range(1, 11)]

    for tamn in tamagnos:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        # ------ INICIO de medicion de tiempo
        inicio = time.perf_counter()
        C = multiplicar_polinomios(A, B)
        fin = time.perf_counter()
        # ------ FIN

        tiempos.append(fin - inicio)

    plt.plot(tamagnos, tiempos, marker='o')
    plt.xlabel('Tamaño del polinomio')
    plt.ylabel('Tiempo de ejecucion (s)')
    plt.title('Tiempo de ejecucion para FFT iterativo con bit reverso')
    plt.xscale('log')
    plt.yscale('log')
    plt.show()


if __name__ == '__main__':
    main()
```

2.6 Interpolación Lagrange

Código en Python, para calcular y medir el tiempo de la Transformada Rápida de Fourer usando la Iterpolación de Lagrange

```
import numpy as np
import time

def vandermonde_eval(P, x):
    """
    Evaluar el polinomio en la matriz de Vandermonde.
    :param P: Lista de coeficientes del polinomio.
    :param x: Puntos en los que se evaluará el polinomio.
    :return: Lista de ys evaluados en la matriz de Vandermonde usando reales.
    """
    y = []
    for xi in x:
        valor = sum(coef * xi**i for i, coef in enumerate(P))
        y.append(valor)
    return y

def interpolacion_lagrange(x, y):
    """
    Calcula los coeficientes del polinomio interpolante de Lagrange.
    :param x: Lista de valores de x.
    :param y: Lista de valores de y.
    :return: Lista de coeficientes del polinomio interpolante.
    """
    n = len(x)
    P = np.zeros(n)

    for k in range(n):
        L = np.array([y[k]])
        for j in range(n):
            if j != k:
                L = np.convolve(L, np.array([-x[j], 1]), mode='full')
                if (x[k] - x[j] > 1e-9):
                    L = L / (x[k] - x[j])
        P = np.add(P, L)
    return P

def multiplicar_polinomios(A, B):
    """
    Multiplica dos polinomios A y B utilizando la interpolación de Lagrange.
    :param A: Lista de coeficientes del primer polinomio.
    :param B: Lista de coeficientes del segundo polinomio.
    :param x: Puntos en los que se evaluarán los polinomios.
    :return: Lista de coeficientes del polinomio resultado de la multiplicación.
    """
    x = np.linspace(-1, 1, len(A) + len(B) - 1)
    # Evaluar los polinomios en los puntos x
    y_A = vandermonde_eval(A, x)
    y_B = vandermonde_eval(B, x)

    # Multiplicar los valores resultantes punto a punto
    y_producto = [a * b for a, b in zip(y_A, y_B)]
    C = [int(c) for c in interpolacion_lagrange(x, y_producto)]

    # Interpolar para obtener los coeficientes del polinomio resultante
    return C

def appFFT_Lagrange(A, B):
    inicio = time.perf_counter()
```

```
    C = multiplicar_polinomios(A, B)
    fin = time.perf_counter()
    tiempoMS = (float(fin) - float(inicio))*1000
    return tiempoMS, C

def main():
    tiempos = []
    tamagnos = [2**i for i in range(1, 11)]
    for tamn in tamagnos:
        A = [np.random.rand() for _ in range(tamn)]
        B = [np.random.rand() for _ in range(tamn)]

        inicio = time.perf_counter()
        C = multiplicar_polinomios(A, B)
        fin = time.perf_counter()

        tiempos.append(fin - inicio)
        print("Tiempo: ", tiempos)
        print("Respuesta: ", C)
    #return tiempos, tamagnos

if __name__ == '__main__':
    main()
```