Manual Técnico

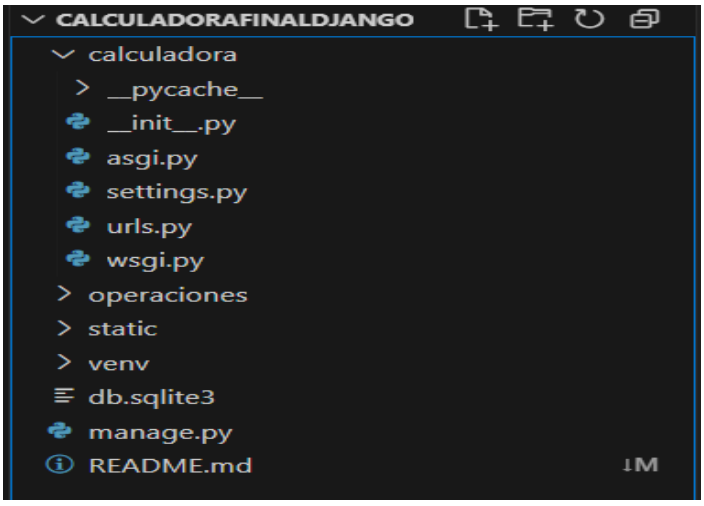
Calculadora DJANGO

La calculadora pretende poder resolver por distintos métodos, las funciones que el usuario le presente, mostrando cada iteración como su gráfica correspondiente.

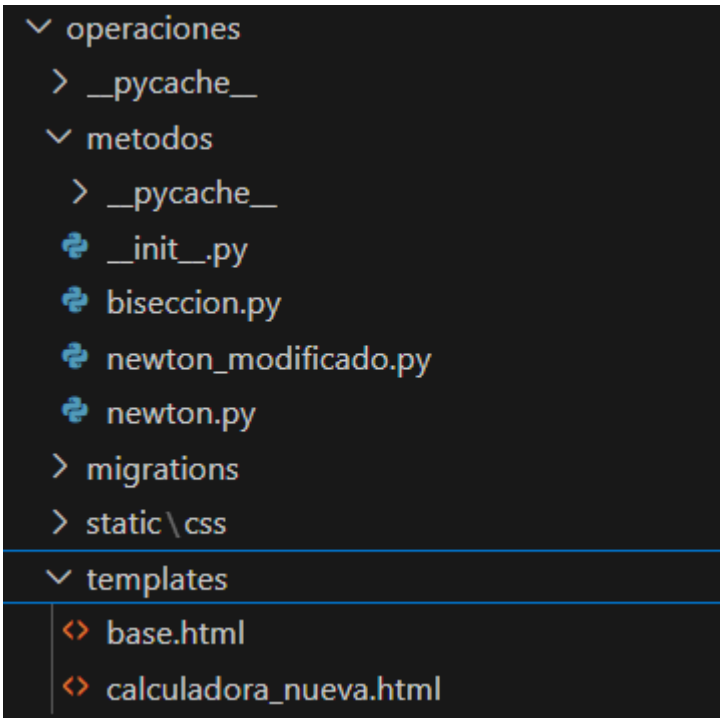
-Estructura del proyecto:

Todo el proyecto está montado sobre un entorno virtual y se utiliza DJANGO para poder generar todo de una manera más práctica y rápida

Diagrama o listado de carpetas y archivos importantes:



-Operaciones



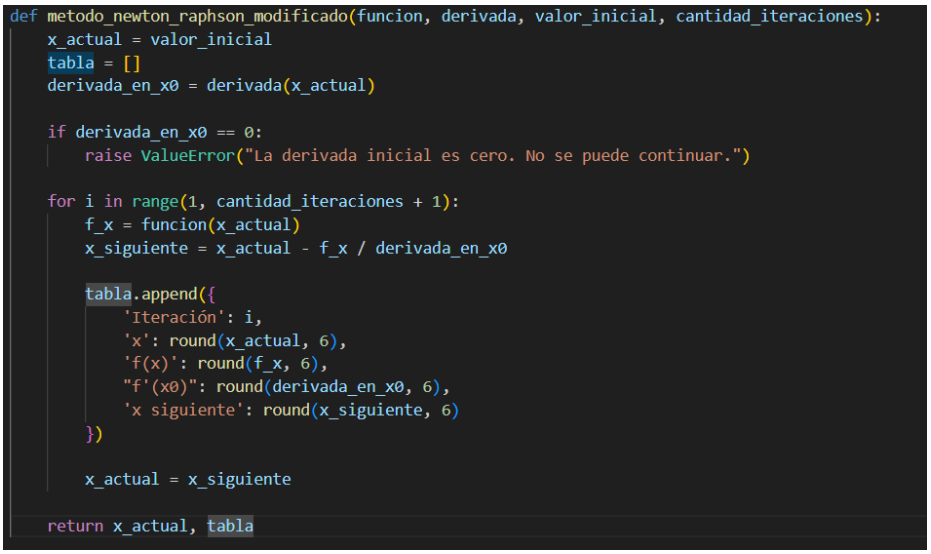
Bisección.py

Acá funciona la lógica matemática del método bisección, su funcionamiento, las iteraciones correspondientes, etc.

```python
def metodo_biseccion(funcion, limite_inferior, limite_superior, cantidad_iteraciones):
    tabla_iteraciones = []
    for numero_iteracion in range(1, cantidad_iteraciones + 1):
        punto_medio = (limite_inferior + limite_superior) / 2
        valor_funcion_medio = funcion(punto_medio)
        valor_funcion_inferior = funcion(limite_inferior)

        tabla_iteraciones.append({
            "Iteración": numero_iteracion,
            "a": round(limite_inferior, 6),
            "b": round(limite_superior, 6),
            "pm": round(punto_medio, 6),
            "f(pm)": round(valor_funcion_medio, 6)
        })

        if valor_funcion_inferior * valor_funcion_medio < 0:
            limite_superior = punto_medio
        else:
            limite_inferior = punto_medio

    return punto_medio, tabla_iteraciones

```

Newton Raphson modificado

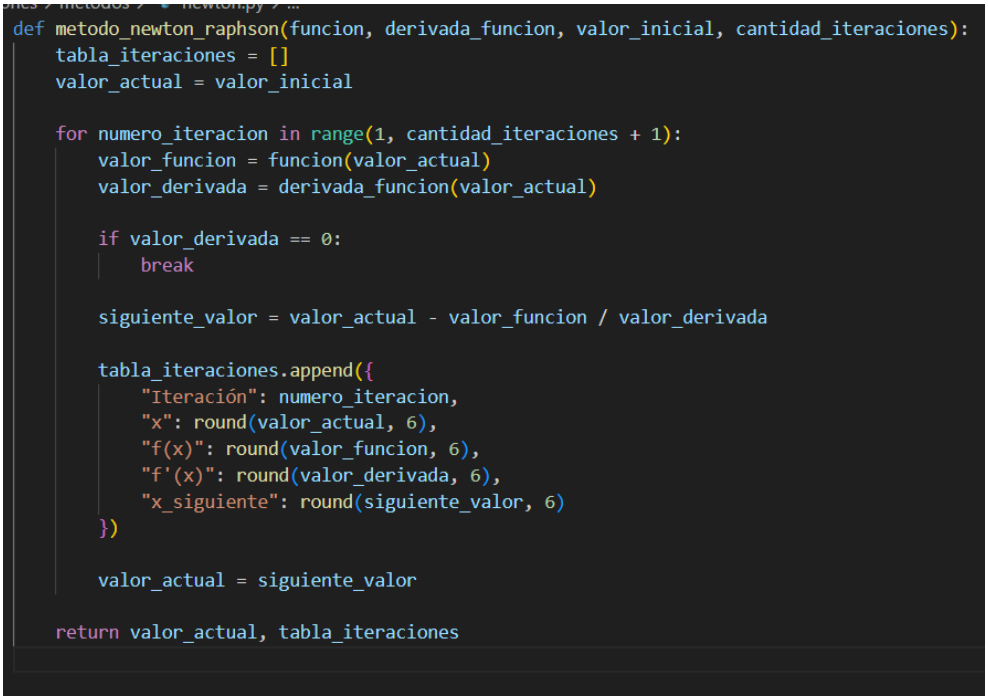
Acá funciona la lógica matemática del método Newton Rapshon modificado, su funcionamiento, las iteraciones correspondientes, etc.



```python
def metodo_newton_raphson_modificado(funcion, derivada, valor_inicial, cantidad_iteraciones):
    x_actual = valor_inicial
    tabla = []
    derivada_en_x0 = derivada(x_actual)

    if derivada_en_x0 == 0:
        raise ValueError("La derivada inicial es cero. No se puede continuar.")

    for i in range(1, cantidad_iteraciones + 1):
        f_x = funcion(x_actual)
        x_siguiente = x_actual - f_x / derivada_en_x0

        tabla.append({
            'Iteración': i,
            'x': round(x_actual, 6),
            'f(x)': round(f_x, 6),
            "f'(x0)": round(derivada_en_x0, 6),
            'x siguiente': round(x_siguiente, 6)
        })

        x_actual = x_siguiente

    return x_actual, tabla
```

Newton Raphson

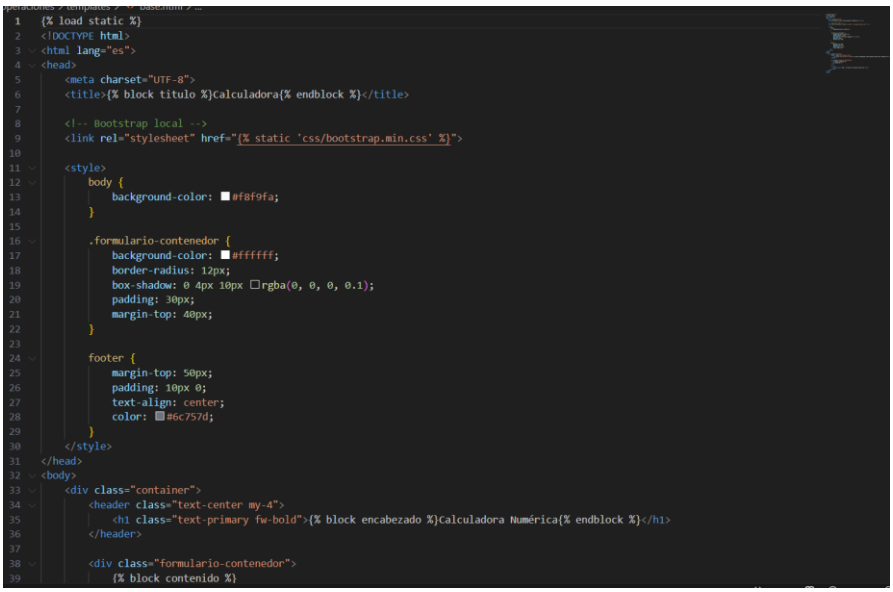
Acá funciona la lógica matemática del método Newton Rapshon, su funcionamiento, las iteraciones correspondientes, etc.



```python
def metodo_newton_raphson(funcion, derivada_funcion, valor_inicial, cantidad_iteraciones):
    tabla_iteraciones = []
    valor_actual = valor_inicial

    for numero_iteracion in range(1, cantidad_iteraciones + 1):
        valor_funcion = funcion(valor_actual)
        valor_derivada = derivada_funcion(valor_actual)

        if valor_derivada == 0:
            break

        siguiente_valor = valor_actual - valor_funcion / valor_derivada

        tabla_iteraciones.append({
            "Iteración": numero_iteracion,
            "x": round(valor_actual, 6),
            "f(x)": round(valor_funcion, 6),
            "f'(x)": round(valor_derivada, 6),
            "x_siguiente": round(siguiente_valor, 6)
        })

        valor_actual = siguiente_valor

    return valor_actual, tabla_iteraciones
```

Base.html

En este apartado se ve toda la parte gráfica, estilos, etc de la página.



```
{% load static %}
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block titulo %}Calculadora{% endblock %}</title>

    <!-- Bootstrap local -->
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">

    <style>
        body {
            background-color: #f8f9fa;
        }

        .formulario-contenedor {
            background-color: #ffffff;
            border-radius: 12px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
            padding: 30px;
            margin-top: 40px;
        }

        footer {
            margin-top: 50px;
            padding: 10px 0;
            text-align: center;
            color: #6c757d;
        }
    </style>
</head>
<body>
    <div class="container">
        <header class="text-center my-4">
            <h1 class="text-primary fw-bold">{% block encabezado %}Calculadora Numérica{% endblock %}</h1>
        </header>

        <div class="formulario-contenedor">
            {% block contenido %}
```

Calculadora_Nueva.html

Acá es donde cargamos toda la información que se debe mostrar en la página html

```
{% extends 'base.html' %}
{% load static %}
{% block titulo %}
Calculadora Numérica
{% endblock %}

{% block encabezado %}
Métodos: Bisección, Newton-Raphson y Newton-Raphson Modificado
{% endblock %}

{% block contenido %}
<form method="POST" class="row g-3" id="formularioMetodo">
    {% csrf_token %}

    <div class="col-12">
        <label class="form-label">Función f(x):</label>
        <input type="text" class="form-control" name="funcion" placeholder="Ej: x^3 - 2x + 1" required>
    </div>

    <div class="col-md-4">
        <label class="form-label">Método:</label>
        <select name="metodo" class="form-select" id="selectorMetodo" required>
            <option value="biseccion">Bisección</option>
            <option value="newton">Newton-Raphson</option>
            <option value="newton_modificado">Newton-Raphson Modificado</option>
        </select>
    </div>

    <div class="col-md-4" id="grupo_x0">
        <label class="form-label">x₀:</label>
        <input type="number" step="any" class="form-control" name="x0">
    </div>

    <div class="col-md-4" id="grupo_a">
        <label class="form-label">Límite inferior (a):</label>
        <input type="number" step="any" class="form-control" name="a">
    </div>

    <div class="col-md-4" id="grupo_b">
```

Gráficas

Acá es donde se genera la gráfica que mostrará la calculadora, y dónde es que albergará dicha imagen en la pc.

```python
import matplotlib.pyplot as plt
import numpy as np
import os
from django.conf import settings

def generar_grafica(funcion, raiz=None):
    # Valores de x para la gráfica
    x = np.linspace(-10, 10, 400)
    y = []

    for valor in x:
        try:
            y.append(funcion(valor))
        except:
            y.append(np.nan)  # Ignorar errores de dominio

    # Crear la gráfica
    plt.figure()
    plt.plot(x, y, label='f(x)', color='blue')
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)

    # Si hay raíz, marcarla en la gráfica
    if raiz is not None:
        try:
            y_raiz = funcion(raiz)
            plt.plot(raiz, y_raiz, 'ro', label='Raíz')
            plt.annotate(f"Raíz: {raiz:.4f}", xy=(raiz, y_raiz), xytext=(raiz, y_raiz+1),
                         arrowprops=dict(arrowstyle='->', color='red'), color='red')
        except:
            pass

    plt.grid(True)
    plt.title('Gráfica de f(x)')
    plt.legend()

    # Crear la carpeta si no existe
    ruta_carpeta = os.path.join(settings.BASE_DIR, 'static', 'graficas')
    os.makedirs(ruta_carpeta, exist_ok=True)
```

Views.py

En este apartado es donde se genera toda la lógica backend del programa es donde se reciben los datos del usuario, llaman a las funciones correspondientes para poder operar matemáticamente, luego envía el resultado para poder visualizarlo via HTML y que así el usuario pueda obtener lo que espera ver

```python
from django.shortcuts import render
import sympy as sp
import re

from .metodos.biseccion import metodo_biseccion
from .metodos.newton import metodo_newton_raphson
from .metodos.newton_modificado import metodo_newton_raphson_modificado
from .graficas import generar_grafica

def corregir_expresion(expresion):
    expresion = expresion.replace('^', '**')
    expresion = re.sub(r'(\d)([a-zA-Z])', r'\1*\2', expresion)
    expresion = re.sub(r'([a-zA-Z])\(', r'\1*(', expresion)
    expresion = re.sub(r'\)([a-zA-Z])', r')*\1', expresion)
    return expresion

def convertir_funcion(expresion):
    x = sp.symbols('x')
    funcion_simbolica = sp.sympify(expresion)
    return sp.lambdify(x, funcion_simbolica, 'math'), sp.lambdify(x, funcion_simbolica.diff(x), 'math')

def calculadora(request):
    resultado = None
    mensaje_error = None
    imagen_grafica = None
    tabla_iteraciones = None

    if request.method == 'POST':
        try:
            expresion_funcion = request.POST.get('funcion')
            expresion_funcion = corregir_expresion(expresion_funcion)

            metodo = request.POST.get('metodo')
            cantidad_iteraciones = int(request.POST.get('iteraciones', 10))

            funcion_evaluable, derivada_funcion = convertir_funcion(expresion_funcion)

            if metodo == 'newton':
```