ID2090 ASSIGNMENT-5 REPORT

Shwetha V

May 2024

1 Transform your chances

1.1 Introduction

Fourier Transform is a tool that breaks a waveform (a function or signal) into an alternate representation, characterized by the sine and cosine functions of varying frequencies. The Fourier Transform shows that any waveform can be re-written as the sum of sinusoids.

1.2 Applications

It is one of the most powerful tools in digital signal processing and digital image processing that is used for the the frequency analysis of signals. It is used in a wide range of applications, such as signal interpolation, signal smoothing, image filtering, image reconstruction and image compression. In fact any field of physical science that uses sinusoidal signals, will make use of Fourier series and Fourier transforms.

1.3 Fourier Transform of a function

It gives us a unique way of viewing any function as the sum of simple sinusoids. The Fourier Transform of a function $g(t)$ is defined by:

$$G(f) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi i f t} \tag{1}$$

In addition, g can be obtained from G via the inverse Fourier Transform:

$$g(t) = \int_{-\infty}^{\infty} G(F)\, e^{2\pi i f t} \tag{2}$$

1.4 Why use Complex Numbers?

In the time domain, we have our horizontal and vertical axis. If we were to place our samples on the x-axis, then the y-axis would show the amplitude of those

samples. Now, if we were to represent the samples in the frequency domain, our samples need to conserve two properties, the amplitude, and the phase; the latter is quite important for various applications of the transform such as data filtering. Complex numbers provide a nice way of storing the phase.

2 Discrete Fourier Transform

In Discrete Fourier Transform, we modify the above integral equations to Fourier transform the discrete finite samples. If we take a sample size of N, then there will be exactly N values generated after the transform, thus we can say that DFT maps N complex numbers from one domain, say time, to N complex numbers in another domain, say frequency. Finding the discrete sum of the continuous equations above gives us:

$$G_k = \sum_{n=0}^{N-1} g_n \cdot e^{\frac{-i2\pi}{N} \cdot kn} \tag{3}$$

2.1 Fourier Matrix

An N-point DFT is expressed as the multiplication $X = Wx$ where x is the original input signal and W is the N-by-N square DFT matrix.

$$\omega = e^{\frac{-2\pi i}{N}} \tag{4}$$

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix} \tag{5}$$

$$W = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{pmatrix} \tag{6}$$

$$DFT(x) = X = x \cdot W \tag{7}$$

```python
def compute_dft(x):
    N = len(x)
    X = [0 for _ in range(N)]
    for i in range(N):
        for j in range((N)):
            omega = np.exp(complex(0, -2) * np.pi / N)
            X[i] += x[j] * omega**(i * j)
    return X
```

Figure 1: A python program to compute DFT of a given signal

2.2 Script Overview

2.2.1 Bash Script

The Bash script is a facilitator for executing the Python script responsible for performing convolution through Fourier Transform. Here's what it does:

- **Shebang Line (!/bin/bash):** This line tells the system to use the Bash shell to interpret the script.
- **Argument Check (Lines 3-8):** It ensures that the correct number of arguments is provided; otherwise, it displays the correct usage.
- **Input File Path (Line 10):** Determines the path to the input file based on the provided argument.
- **Python Script Generation (Lines 13-40):** Generates a Python script named fourier.py, embedding the convolution computation within it.
- **Script Execution (Line 42):** Finally, it runs the Python script with the input file path.

2.2.2 Python Script

The Python script is where the actual convolution computation happens. Here's a simplified explanation:

- **Function Definitions (Lines 5-18):** Defines functions to handle reading, Fourier transforming, inversely Fourier transforming, and convolving functions.
- **Function Reading (Line 21):** Reads functions from the input file and converts them into SymPy expressions.
- **Fourier Transform (Lines 24-26):** Applies Fourier transform to each function.
- **Multiplication and Inverse Transform (Lines 29-32):** Multiplies the transformed functions and applies inverse Fourier transform to get the convolved function.

- **Output Display (Lines 35-37):** Prints the convolved function and its LaTeX representation.

2.2.3 Usage

To use the provided scripts, follow these steps:

1. Create a plaintext file named `functions.txt` containing the functions to be convolved, each on a separate line.
2. Run the Bash script using the command:

```
./question_1.sh functions.txt
```

3. The convolved function and its LaTeX representation will be printed as output.

3 Poised for Poiseuille flow

3.1 Introduction

Transport phenomena play a fundamental role in understanding the behavior of various physical, chemical, and biological systems. It's the study of momentum, heat, and mass transfer and the mechanisms by which these quantities are transported through different mediums. From the macroscopic flow of fluids in pipes to the microscopic diffusion of molecules across cell membranes, transport phenomena govern a wide range of natural and engineered processes.
This task is an application of Navier-Stokes equation which are partial differential equations governing flow of viscous fluid substances.

3.2 What are Navier-Stokes Equations?

- Equations which can be used to determine the velocity vector field that applies to a fluid, given some initial conditions.
- For almost all real situations, they result in a system of nonlinear partial differential equations.
- However, with certain simplifications (such as 1-dimensional motion) they can sometimes be reduced to linear differential equations.
- Usually, they remain nonlinear, which makes them difficult or impossible to solve; this is what causes the turbulence and unpredictability in their results.

The Navier-Stokes equations can be derived from the basic conservation and continuity equations applied to properties of fluids.

3.3 Continuity Equation

The basic continuity equation is an equation which describes the change of an intensive property L. An intensive property is something which is independent of the amount of material you have. The continuity equation derived can later be applied to mass and momentum.

The general form of the continuity equation

$$\frac{dL}{dt} + \nabla \cdot (L\vec{v}) + Q = 0.$$

In spherical coordinates, the continuity equation is given by:

$$\frac{\partial \rho}{\partial t} = \frac{1}{r^2}\frac{\partial \rho u_r r^2}{\partial r} + \frac{1}{r\sin\theta}\frac{\partial \rho u_\theta \sin\theta}{\partial \theta} + \frac{\partial \rho u_\phi}{\partial \phi} \tag{8}$$

3.4 General Form of the Navier-Stokes Equation

The most general form of the Navier-Stokes equation:

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \nabla \cdot T + \vec{f}.$$

$$\frac{\partial(\rho\vec{v})}{\partial t} + \nabla \cdot (\rho\vec{v}\vec{v}) = \rho\vec{g} - \nabla P + \mu\nabla^2\vec{v} \tag{9}$$

Navier-stokes equation in cylindrical coordinates is given by:

$$\frac{\partial \rho}{\partial t} + \frac{1}{r}\frac{\partial(\rho v_r)}{\partial r} + \frac{\partial(\rho v_z)}{\partial z} + \frac{1}{r}\frac{\partial(\rho v_\theta)}{\partial \theta} = 0 \tag{10}$$

3.5 Physical Explanation of the Navier-Stokes Equation

The left hand side of the equation,

$$\rho\frac{D\vec{v}}{Dt},$$

is the force on each fluid particle. The equation states that the force is composed of three terms:

- $-\nabla p$: A pressure term which prevents motion due to normal stresses. The fluid presses against itself and keeps it from shrinking in volume.
- $\nabla \cdot T$: A stress term which causes motion due to horizontal friction and shear stresses. The shear stress causes turbulence and viscous flows. Turbulence is the result of the shear stress.
- $\vec{f}$: The force term which is acting on every single fluid particle.

3.6 Observations

The script analyzes Poiseuille flow, a steady-state, incompressible fluid flow within a cylindrical pipe. To derive an analytical solution for the axial velocity profile, the script incorporates the following simplifying assumptions:

1. **Incompressible Flow (ρ is constant):** Fluid density remains constant, implying negligible volume changes.
2. **Fully Developed Flow (z-velocity not dependent on z):** The axial velocity (z-direction) is independent of the axial position (z) within the pipe.
3. **θ-symmetric flow (θ components and their changes can be neglected):** There's no variation in flow properties with respect to the angular direction (θ).
4. **Impenetrable Wall (Zero radial velocity at a radius equal to pipe radius):** Fluid doesn't penetrate the pipe walls, resulting in zero radial velocity at the pipe's radius.
5. **Continuous and Smooth (Differentiable) flow profile:** The velocity profile throughout the pipe's cross-section is continuous and differentiable.

3.7 Script Functionality

This combination of Bash and Python scripts automates the process of solving a fluid flow problem.

1. Input Validation

   The Bash script checks if the correct number of arguments is provided when running the script. If not, it displays instructions on proper usage and exits with an error.

2. Input Processing

   - The provided file containing the pressure profile (e.g., 'press.txt') is assigned to a variable for easy access. - The Bash script utilizes a here document to create a temporary Python script named 'poise.py'. This script contains the Python code needed for the calculations.

3. Python Script Execution

   The Bash script then executes the 'poise.py' script, passing the input file as an argument.

Python Script (poise.py)

1. Data Reading

   'poise.py' reads the pressure profile data from the provided input file.

2. Constant Definition

   Necessary constants like dynamic viscosity (u) and pipe radius (R) are defined.

3. Symbolic Processing

   - The pressure profile data is converted into a symbolic expression using the SymPy library. - The script symbolically computes the axial velocity profile using the given approach.

4. C++ Code Generation

   A C++ file named 'vel1.cpp' is created. This C++ code can later be used to calculate the axial velocity based on the radial distance from the axis.

3.8 Conclusion

The successful implementation of the script provides valuable insights into the behavior of fluid flows and offers practical applications in engineering and science. By solving the Navier-Stokes equation, engineers and researchers can optimize the design of aerodynamic systems, predict fluid behavior in industrial processes, and understand natural phenomena such as weather patterns and ocean currents.