An Insight Into Assignment 5

Krishna Murari Chivukula, ME23B233

May 4, 2024

Abstract

This is a report which contains my insights for Assignment-5 of the course ID2090.

1 Transform Your Choices

1.1 Problem Abstract

This problem dealt with **Task Fourier Transform**. It is often used to model convolution operation between two functions, where each function is applied a Fourier Transform and multiplied, to get the Fourier Transform representation of the convoluted function. The language used was **Sympy** to write a script that takes the functions to be convolved, as arguments from a plaintext file, performs convolution using Fourier Transform, and outputs the final convolved function.

1.2 A Little About Convolution

- Convolution is a fundamental operation in signal processing and image processing. It describes the blending of two functions, often denoted as f and g, to produce a third function, typically denoted as $f * g$. In the context of continuous functions, convolution can be defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{1}$$

  where $*$ denotes convolution, and $f(\tau)$ and $g(\tau)$ are the functions being convolved.

- The convolution theorem states that the Fourier transform of the convolution of two functions is equal to the pointwise multiplication of their Fourier transforms:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{2}$$

  where $\mathcal{F}\{\cdot\}$ denotes the Fourier transform.

- In other words, given two functions $f(t)$ and $g(t)$ with Fourier transforms $F(\omega)$ and $G(\omega)$ respectively, the Fourier transform of their convolution $(f * g)(t)$ is the product of their Fourier transforms.

- Convolution finds applications in some of the following domains:

  1. Signal processing for filtering and modulation
  2. Image processing for feature extraction and restoration, Computer vision for object detection and pose estimation
  3. Neural networks for feature learning and transfer learning.

1.3 Convolution With Fourier Transform

- Essentially , what this code does is , each function is applied a Fourier Transform and multiplied, to get the Fourier Transform representation of the convoluted function. This is then outputted.

```
# Function to perform convolution using Fourier Transform
convolve_fourier_transform() {
    f1="$1"
    f2="$2"

    python3 - <<END
import sympy as sp


x, xi = sp.symbols('x xi')

# Parse functions from strings
f1 = sp.sympify("$f1")
f2 = sp.sympify("$f2")

# Apply Fourier Transform on the two functions
F1 = sp.fourier_transform(f1, x, xi)
F2 = sp.fourier_transform(f2, x, xi)

# Convolution in Fourier domain
convolved_F = F1 * F2

# Inverse Fourier Transform to get the final convolved function
convolution = sp.inverse_fourier_transform(convolved_F, xi, x)

# Print the simplified final convolved function
print(sp.latex(convolution.simplify())) #Using sp.latex simplifies
    the given output to it's latex expression, using sp.pprint
   makes it appear in a nice mathematical fashion

END
}

```

Listing 1: Function for Performing Convolution

- Here, we use the function `sp.inverse_fourier_transform` to obtain the final convolved function from the convolution domain to a normal output.

1.4 Output Code Snippet

- The output is again printed in a fairly simple fashion, by declaring a main function that calls the `convolve_fourier_transform` function, with some caution cases which can be read below.
- The caution deals with either the file not existing and ensuring that there are only 2 lines corresponding to each of the particular functions that are begin convolved. The code can be seen below.

```
     # Defining a Main
main() {
    # Read the functions from plaintext file
    functions_file="functions.txt"
    if [[ ! -f "$functions_file" ]]; then
        echo "Error: $functions_file not found"
        exit 1
    fi

    # Read the functions from the file
    functions=()
    while IFS= read -r line; do
        functions+=("$line")
    done < "$functions_file"

    # Ensure there are exactly two lines (two functions)
    if [[ ${#functions[@]} -ne 2 ]]; then
        echo "Error: $functions_file must contain exactly two
   lines (two functions)"
        exit 1
    fi

    # Perform convolution using Fourier Transform
    convolve_fourier_transform "${functions[0]}" "${functions[1]}"
}

# Run the main function
main
}

```

Listing 2: Output The Convolution

1.5 Sample Run

- The code was run against a text file titled `functions.txt`, that contained 2 Gaussian functions in 2 separate lines.These functions were

$$\frac{\sqrt{2}e^{-0.5\left(\frac{x}{2}-\frac{1}{2}\right)^2}}{8\sqrt{\pi}}$$

$$\frac{\sqrt{2}e^{-0.5\left(\frac{x}{2}-\frac{5}{2}\right)^2}}{8\sqrt{\pi}}$$

- They were taken in LaTeXformat, the output was the final convolved function.

$$\frac{0.0625e^{-0.0625(x-6)^2}}{\sqrt{\pi}}$$

- This output was also given in LaTeXformat.

```
me23b233@ID2090:~/assignment_5$ ./question_1.sh functions.txt
\frac{0.0625 e^{- 0.0625 \left(x - 6\right)^{2}}}{\sqrt{\pi}}
me23b233@ID2090:~/assignment_5$
```

Figure 1: Output

1.6 Graphical Anlaysis

- I wrote a small code block to implement the graphs of the 2 given Gaussian functions as follows.

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

def func(x, mu):
  return np.sqrt(2) * np.exp(-0.5 * (x - mu)**2) / (8 * np.sqrt(np
    .pi))

x = np.linspace(-100, 100, 1000)

# Plot the first function
plt.plot(x, func(x, 0.5), label=r"$\frac{\sqrt{2} e^{- 0.5 \left(\
    frac{x}{2} - \frac{1}{2}\right)^{2}}}{8 \sqrt{\pi}}$")

# Plot the second function
plt.plot(x, func(x, 2.5), label=r"$\frac{\sqrt{2} e^{- 0.5 \left(\
    frac{x}{2} - \frac{5}{2}\right)^{2}}}{8 \sqrt{\pi}}$")

# Set labels and title
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Two Gaussian functions")

# Add legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()

```

Listing 3: Graph The Functions

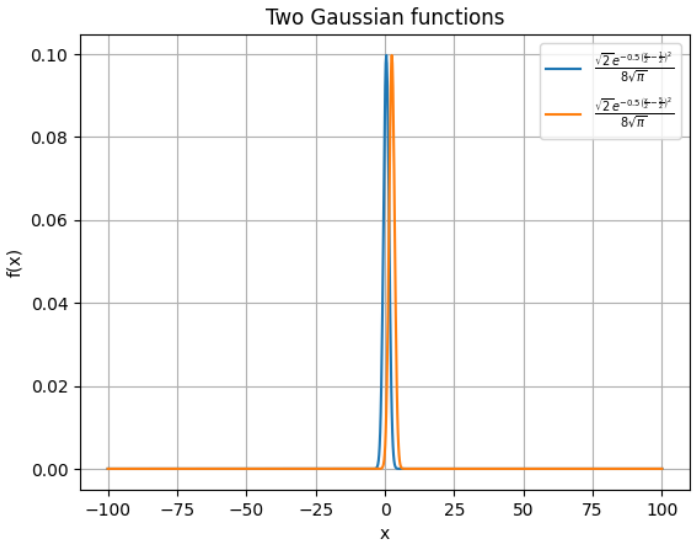

Figure 2: Graphs

-

- The final convolved function was also plotted in a similar fashion.Notice how the convolution of 2 Gaussian Functions also returned a Gaussian function !

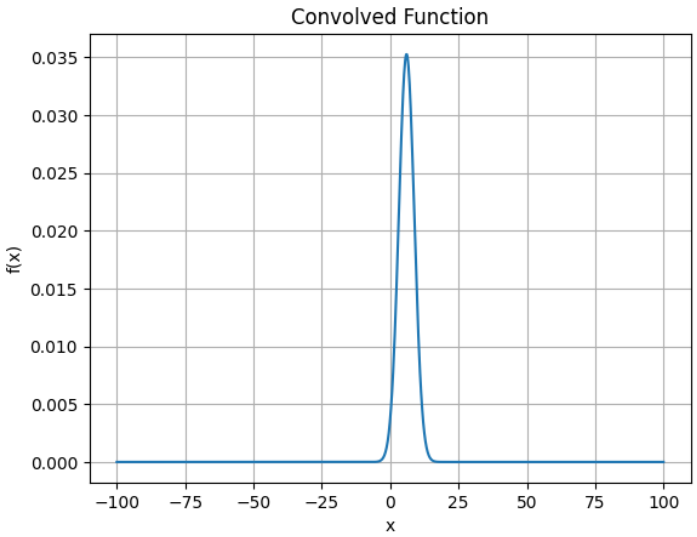

Figure 3: Convolved Gaussian

2 Poised for Poiseuille Flow

2.1 Problem Abstract

This question dealt with solving of differential equations pertaining to boundary conditions using the **Continuity Equation** as well as the **Navier Stokes Equation** for determination of velocity, given a pressure profile equation.
The **Continuity Equation** is given by

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

The **Navier Stokes Equation** is given by

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

The language used was **Sympy** for writing the script. The script calculates the velocity v symbolically and creates a C++ file titled vel.cpp which takes in argument as the radial distance from the axis and outputs the corresponding z-velocity to the output stream.

2.2 Reducing the Differential Equation

Solving the coupled differential equations stated above required some logical physical assumptions regarding the fluid flow as follows:

- Incompressible flow (ρ is constant)
- Fully Developed Flow (z-velocity not dependent on z)
- θ symmetric flow (θ components and their changes can be neglected)
- Impenetrable wall (Zero radial velocity at a radius equal to pipe radius
- Continuous and Smooth (Differentiable) flow profile.

The **Continuity** and **Navier Stokes** equations are simplified based off the above assumptions as follows. We treat the entire system in the cylindrical coordinate system and assume all constants as unity. We are also concerned only with the z direction. Furthermore , our boundary conditions are:

- Velocity at radial distance unity is 0.
- The velocity is maximum at radial distance 0 i.e, $\frac{\partial V_z}{\partial r}$ at $r = 0$, is 0.

By simplifying with the given physical assumptions, our equation reduces to

$$\frac{\partial^2 V_z}{\partial^2 r} + 1/r \frac{\partial V_z}{\partial r} = \frac{\partial P}{\partial z}.$$

2.3 Solving the Differential Equation

- First, a text file titled `press.txt` is taken in as input.This file contains the equation for the pressure profile in the z direction, in a single line.
- The `.txt` file is processed to convert it to a **Sympy** readble format and passed as an argument to the remaining **Sympy** code which solves the differential equation symbolically.

```
import sys
from sympy import *
import subprocess
file_name = sys.argv[1]
with open(file_name) as line:
    pressure_gradient = line.readline()
z, r= symbols("z r")
p_symp = sympify(pressure_gradient)
p = p_symp.diff(z)
P = Function('P')(z)
v_z = Function('v_z')(r)
d_e1 = Eq(p  , v_z.diff(r,2) + 1/r*(diff(v_z , r)))
initial_conditions = {diff(v_z,r).subs(r, 0): 0,
                      v_z.subs(r, 1): 0

}
solution = dsolve(d_e1, v_z ,ics = initial_conditions)
```

Listing 4: Sympy Portion Of Script

2.4 C++ Code To Output Velocity

```
#converting the solution to cpp format
cpp_vel = ccode(solution)
cpp_vel = cpp_vel.split("=")[-1]
#CPP Code
cpp = f'''
#include <iostream>
#include <cmath>

double solve(double r) {{
    return {cpp_vel};
}}

int main(int argc, char* argv[]) {{
    double radial = std::stod(argv[1]);
    double velocity = std::abs(solve(radial));
    std::cout << velocity << std::endl;
}}
'''

#writing the cpp code
with open('vel_temp.cpp', 'w') as f:
```

```
    f.write(cpp)

#compiling the CPP code through Python
compile_process = subprocess.run(["g++", "-o", "vel.cpp", "
    vel_temp.cpp"], stdout=subprocess.PIPE)
```

Listing 5: C++ Code

- The above code snippet first converts the solution of the differential equation to a C++ format and then computes the velocity. This portion takes in the radial distance r as an argument and calculates the velocity , outputs it to the output stream.
- Both the above code blocks were put into a single Python script titled `question_2.sh`. A sample run that I had executed initially, without using the `std::abs()` for the velocity is shown below.

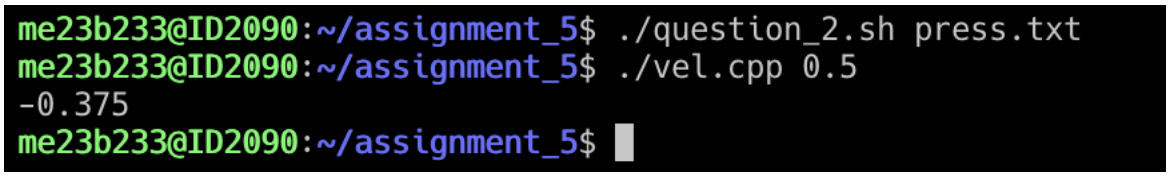

Figure 4: Test Run

- The final solution of the differential equation obtained is $v(r) = \frac{r^2-1}{2}$. The pressure profile taken for the test run is $P = 2z$ and the velocity is calculated at a radial distance $r = 0.5$. For this v came out as -0.375 not 0.375 as given in the sample case in the assignment file , since the pressure increases with increasing z. Fluid flow is from high pressure to low pressure , so velocity is downward(along $-z$ axis), hence the negative sign in my output.
- Since the output required was only the magnitude of the velocity, I modified my inital code using the `std::abs()` to get the output as 0.375.

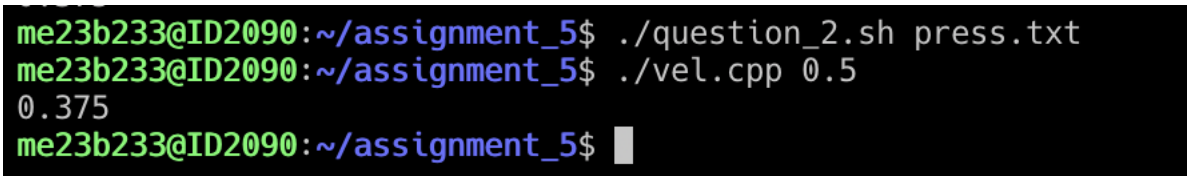

Figure 5: Final Output

2.5 Additional Learnings

- The **Navier Stokes Equation** can be applied to model ocean currents, weather, air flow around wings, and the flow of water in pipes. They are also applied in the examination of liquid flow, the study of pollution, the design of power, and other processes related to fluids. Along with Maxwell's equations, these equations can be applied to study and model magnetohydrodynamics as well.
- In the most general form, there are no analytical solutions to the **Navier-Stokes equations**. In other words, it is only possible to get some form of an analytical solution in particular approximate scenarios, i.e with some logical assumptions as executed above. The outcomes may not ever be realised in a real system.