Reflection A5

D V Anantha Padmanabh
ME23B012

April 2024

Task 1

Transform your chances

1.1 Introduction

1.1.1 Convolution

Let $x_1(t)$ and $x_2(t)$ be continuous functions defined for $-\infty < t < \infty$. Their convolution is defined as:

$$x_1(t) * x_2(t) = \int_{-\infty}^{\infty} x_1(\tau) \cdot x_2(t-\tau)\, d\tau$$

1.1.2 Fourier Transform

Fourier transform converts the time domain of the function(generally signals) into its frequency domain.

Let $f(t)$ be a continuous function defined for $-\infty < t < \infty$. The Fourier transform $\mathcal{F}\{f(t)\}$ of $f(t)$ is defined as:

$$\mathcal{F}\{f(t)\} = F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{1.1}$$

where ω is the angular frequency and $F(\omega)$ is the Fourier transform of $f(t)$.

1.1.3 Convolution using Fourier Transform

Convolution Theorem: Let $x_1(t)$ and $x_2(t)$ be two functions with Fourier transforms $X_1(\omega)$ and $x_2(\omega)$ respectively. Then the Fourier transform of their convolution $(x_1 * x_2)(t)$ is given by the product of their Fourier transforms:

$$\boxed{x_1(t) * x_2(t) \quad \underset{IFT}{\overset{FT}{\rightleftharpoons}} \quad X_1(\omega) \cdot X_2(\omega)}$$

1.2 Implementation

1.2.1 Convolution using Fourier Transform

```
#!/usr/bin/python3

import sys
import sympy as sp
from latex2sympy2 import latex2sympy

text_file = sys.argv[1]
with open(text_file, "r") as f:
    fns_lat = f.read().split("\n")

fns = [latex2sympy(i) for i in fns_lat]

x = sp.symbols("x")
k = sp.symbols("k")

ft_fns = [sp.fourier_transform_(i, x, k) for i in fns]

convolution = sp.inverse_fourier_transform(ft_fns[0] * ft_fns[1], k, x)
convolution = sp.simplify(convolution)

sp.pprint(convolution)
```

1.2.2 Traditional Way of Convolution

```
import sys
import sympy as sp
from latex2sympy2 import latex2sympy

text_file = sys.argv[1]
with open(text_file, "r") as f:
    fns_lat = f.read().split("\n")

fns = [latex2sympy(i) for i in fns_lat]

x = sp.symbols("x")
T = sp.symbols("T")

f1 = fns[0]
f2 = fns[1]

convolution = sp.integrate(f1.subs(x, T)*f2.subs(x,x-T),(T,sp.oo,sp.oo))
```

```
convolution = sp.simplify(convolution)

sp.pprint(convolution)
```

1.3 Observation and Conclusion

- The traditional method of convolution is approximately 1.55 times faster than the fourier transform method. This is **NOT** intended but occured because of inbuilt $sympy.fourier\_transform$ which does CFT(Continuous Fourier Transform). If we sample the continuous function as a discrete function and do FFT(Fast FT) method then it would be faster than the traditional convolution. See reference.
- The convolution of two gaussian functions are gaussian.

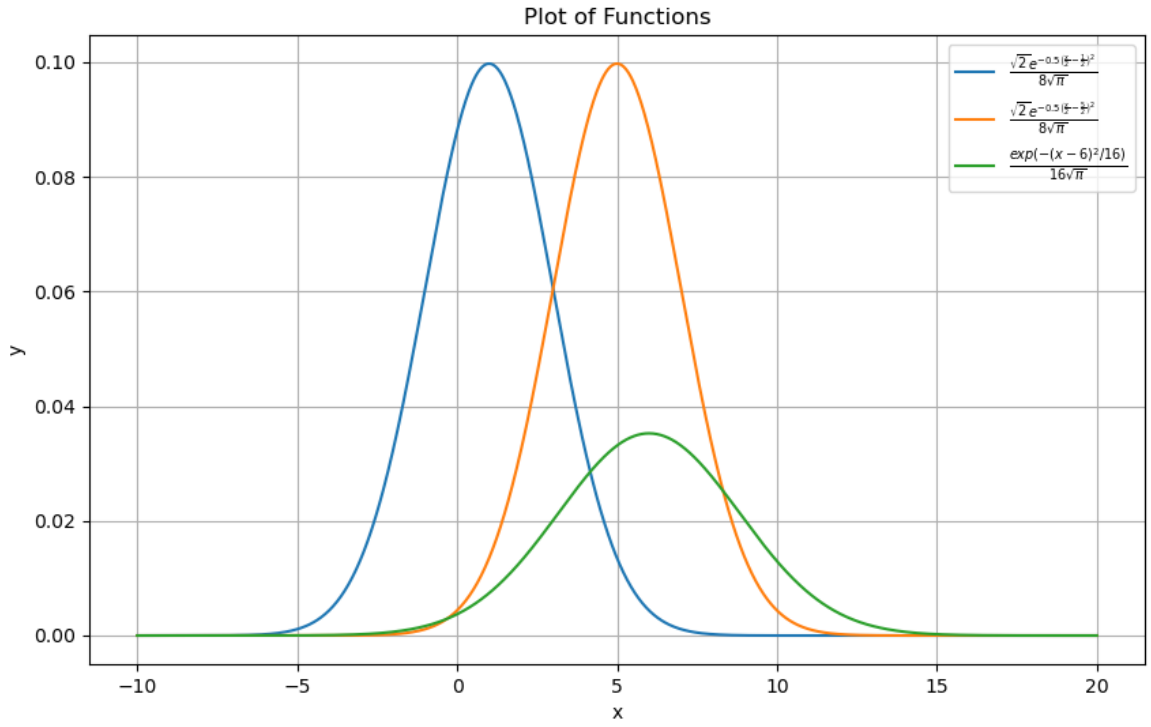

Figure 1.1: Plots

Task 2

Poised for Poiseuille flow

2.1 Introduction

2.2 Divergence and Laplacian in cylindrical coordintes

$$\nabla \cdot \mathbf{A} = \frac{1}{r}\frac{\partial (rA_r)}{\partial r} + \frac{1}{r}\frac{\partial A_\phi}{\partial \phi} + \frac{\partial A_z}{\partial z}$$

$$\nabla^2 \mathbf{A} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial \mathbf{A}}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 \mathbf{A}}{\partial \phi^2} + \frac{\partial^2 \mathbf{A}}{\partial z^2}$$

2.2.1 Navier-Stokes Equations

- Continuity Equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

  in cylindrical coordinates,

$$\frac{\partial \rho}{\partial t} + \frac{1}{r}\frac{\partial \left(\rho r v_r\right)}{\partial r} + \frac{1}{r}\frac{\partial \left(\rho v_z\right)}{\partial z} + \frac{1}{r}\frac{\partial \left(\rho v_\theta\right)}{\partial \theta} = 0 \tag{2.1}$$

- Momentum Equation

$$\frac{\partial (\rho \vec{v})}{\partial t} + \nabla \cdot (\rho \vec{v}\vec{v}) = \rho \vec{g} - \nabla P + \mu \nabla^2 \vec{v} \tag{2.2}$$

2.2.2 Assumptions

- 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.

2.2.3 Circular Poiseuille Flow in a Cylindrical Pipe

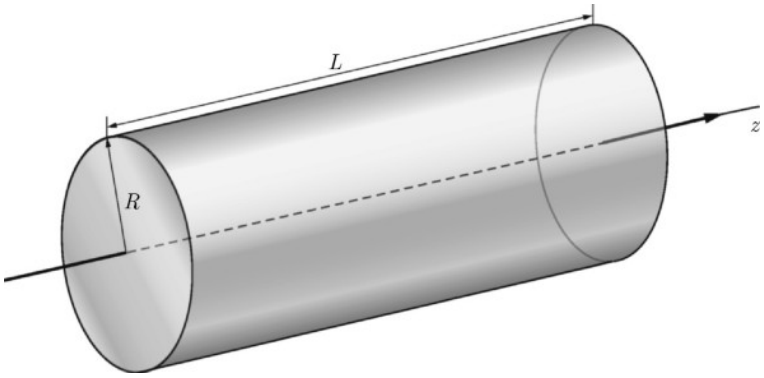

Figure 2.1: Cylindrical Pipe

By taking account into the assumptions and solving for cylindrical coordinates, Equation 2.1 simplifies to

$$\frac{1}{r}\frac{\partial\ (\rho r v_r)}{\partial r} = 0 \tag{2.3}$$

Integrating,

$$r v_r = f(z)$$

at $r = R$, $v_r = 0$ therefore $f(z) = 0$. This implies $v_r = 0$ everywhere

Now simplifying 2.2 all terms in LHS goes to zero as flow is time independent and the ssecond term becomes,

$$\nabla \cdot (\rho \vec{v}\vec{v}) = \rho\ (\nabla \cdot \vec{v})\vec{v} = \rho\ \left(\frac{1}{r}\frac{\partial (r v_r)}{\partial r}\right)\vec{v} = 0$$

Finally we neglect gravity and the equation becomes:

$$0\ =\ -\nabla P + \mu \nabla^2 \vec{v} \tag{2.4}$$

Expanding the divergence and laplacian keeping in mind that $\vec{v} = v_z \hat{z}$ and equation the non zero z components of each side,

$$\frac{\mu}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v_z}{\partial r}\right)\ =\ \frac{\partial P}{\partial z} \tag{2.5}$$

The right-hand side term only depends on z; on the left-hand side there is only dependence on r. Thus the two terms must be equal to a constant. Integrating, we obtain

$$v_z = \left(\frac{\partial P}{\partial z}\right)\frac{1}{\mu}\left(\frac{r^2}{4} + A \ln r + B\right). \tag{2.6}$$

The velocity must be finite on the axis $r = 0$. This leads to $A = 0$. Taking into account the condition $v_z(R) = 0$, we have

$$\boxed{v_z = -\left(\frac{\partial P}{\partial z}\right)\frac{R^2}{4\mu}\left(1 - \left(\frac{r}{R}\right)^2\right)} \tag{2.7}$$

2.3 Implementation

```
#!/usr/bin/python3

import sys
import sympy as sp
from latex2sympy2 import latex2sympy
from subprocess import run

press = sys.argv[1]
with open(press, "r") as f:
    pressure_function = f.read()

pressure_function = latex2sympy(pressure_function)

z = sp.symbols("z")

P_z = sp.diff(pressure_function, "z")
# print(P_z)

r = sp.symbols("r")
v_z = -P_z / 4 * (1 - r**2)
# print(v_z)

v_cpp = sp.ccode(v_z)

with open("vel.cpp", "w") as fcpp:
    fcpp.write("#include <iostream>\n")
    fcpp.write("#include <cmath>\n")
    fcpp.write("using namespace std;\n")
    fcpp.write("int main(int argc, char* argv[]) {\n")
    fcpp.write("    double r = stod(argv[1]);\n")
    fcpp.write(f"    double v_z = {v_cpp};\n")
    fcpp.write("    cout << abs(v_z) << endl;\n")
    fcpp.write("    return 0;\n")
    fcpp.write("}\n")

run(["g++", "vel.cpp", "-o", "vel.out"])
```

2.4 Observation and Conclusion

- The flow is said to be fully developed which implies v_z is independent of z therefore we infer that $\frac{\partial P}{\partial z}$ is a constant. This means pressure P will be a linear function of z

- From the final equation of v_z, it is seen that the velocity profile is parabolic for a particular z and remains constant for constant r.

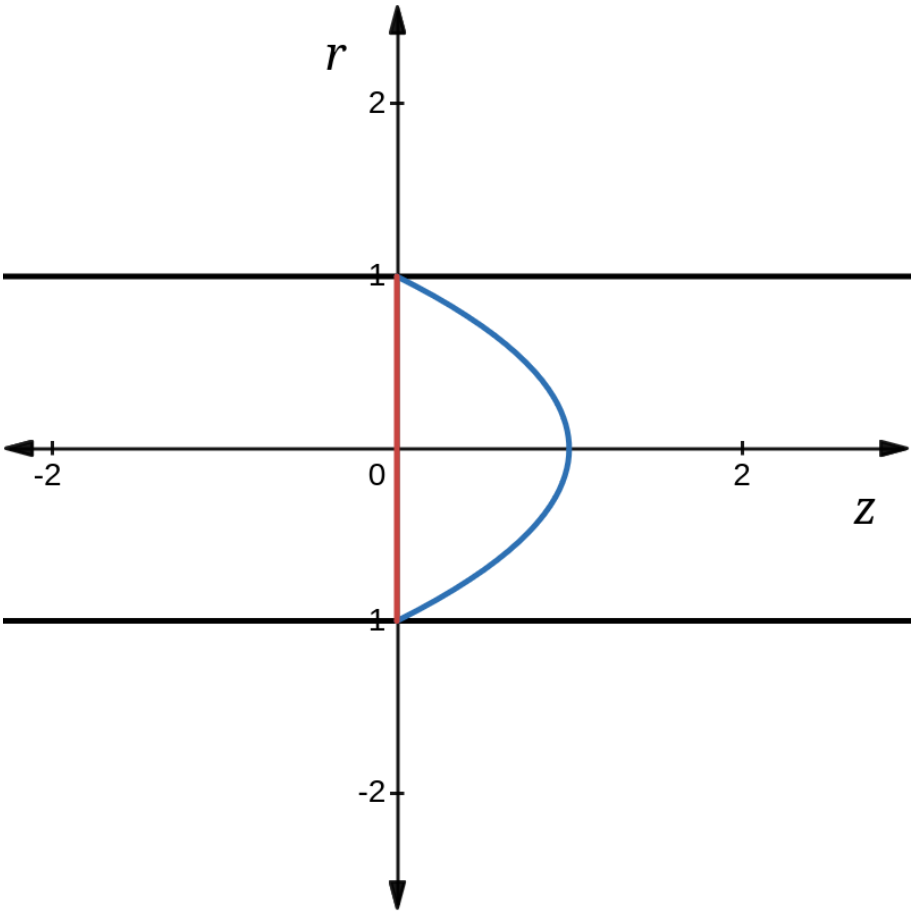

Figure 2.2: Velocity profile for $x = 0$

References

1. EE1101 Signals and Systems course reference textbook, Signals and Systems by Michael D Adams

2. Stack Overflow, FFT vs Traditional convolution, https://stackoverflow.com/questions/18384054/what-are-the-downsides-of-convolution-by-fft-compared-to-realspace-convolution

3. Exact solutions to $Navier - Stokes$ equations, https://link.springer.com/content/pdf/10.1007/978-3-031-04683-4_3.pdf