ID2090 Assignment 5 Report

Arivoli Ramamoorthy ee23b008

May 2024

1 Transform your chances:

1.1 Context

1. The ability of Fourier Transform to approximate a signal as an infinite sum of sinusoidal waves of integral multiples of some frequency, the ability to transform a time-series data into the frequency domain, has a variety of applications in Solving of Dirichlet PDEs, Von Neumann Stability Analysis of CFD schemes, Analysis of Complicated Circuits, etc.

2. Its discrete version, discrete Fourier Transform DFT, is crucial to analyzing images in terms of noise, details, structure, filters required, etc.

Task Fourier Transform 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. Your task is to create a Sage / Sympy 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 You may assume that the functions given in the input are defined over x in R.

1.2 Solution

```
#!/bin/python3
import sys
from sympy import *
from latex2sympy2 import latex2sympy

if len(sys.argv) != 2:
    print("Usage: question_1.sh functions.txt")
    sys.exit(1)

with open(sys.argv[1], 'r') as file:
    t1 = latex2sympy(file.readline().strip())
    t2 = latex2sympy(file.readline().strip())
x = symbols('x')
k = symbols('k')
F1 = fourier_transform(t1, x, k)
```

```
F2 = fourier_transform(t2, x, k)

conv = F1*F2

out = latex(inverse_fourier_transform(conv, k, x))
print(out)
```

1.3 Explanation + Modules used

- Latex input is taken, converted to sympy using latex2sympy and stored in t1, t2.
- x and k are initialized as symbols
- Fourier transforms of the two functions with respect to 'x' are found and stored in F1 and F2 wrt input/equation (t1, t2).
- The code then computes the convolution of the Fourier transforms F1 and F2 by multiplying them element-wise and stores the result in conv.
- latex2sympy2 (to convert from sympy to latex and vice versa)
- sympy

1.4 Output

(The first output line is not present in the code, this is just for debugging and clarity)

```
nxtblazevfx@fedora:~/Documents/id2090/assignments/assignment5$ ./q1.sh functions.txt
exp(-(x - 6)**2/16)/(16*sqrt(pi))
\frac{e^{- \frac{\left(x - 6\right)^{2}}{16}}}{16 \sqrt{\pi}}
```

2 Poised for Poiseuille flow

2.1 Context

1. The Transport phenomenon and associated equations are derived from the continuum hypothesis, which governs the fluid flow and its related properties.
2. They are predominantly used in mechanical, chemical, metallurgy, ocean engineering, and mathematics, too, to predict fluid properties at a given time and location.

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

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

2.2 Getting the Equation

```
#!/bin/python3

import sys
import sympy as s
from sympy import *
from latex2sympy2 import latex2sympy
if len(sys.argv) != 2:
        print("Usage: question_2.sh press.txt")
        sys.exit(1)

with open(sys.argv[1], 'r') as file:
        exp = s.sympify(file.read().strip())

f = s.diff(exp, 'z')
x, R = s.symbols('x R')
y, a, b = s.Function('y')(x), 1/x, 0

eq = s.Eq(y.diff(x, x) + a * y.diff(x) + b * y, f)
boundary_conditions = {y.subs(x, 1): 0, y.diff(x).subs(x, 0): 0}
soln = s.dsolve(eq, y, ics=boundary_conditions)

velocityexp=soln.rhs
vcpp=s.ccode(velocityexp)
#print(velocityexp)
```

- using sympy.sympify, we take the input expression and store it into the variable exp.
- f stores differentiation of expression by z.
- The differential equation is defined using the symbols and functions defined earlier.
- Boundary conditions are specified for the differential equation.
- right hand side of the solution is extracted (after sympy.dsolve(solves an ODE)) and stored into velocityexp

2.3 Cpp part

Output is only the magnitude of velocity (asked TA), hence couting abs(velocity)

μ, R were taken to be unity for simplicity of question.

```
executable = f'''
#include <iostream>
#include <cmath>
double velocity(double x) {{
    return {vcpp};
}}
int main(int argc, char* argv[]) {{
    if (argc != 2) {{
```

```
        std::cerr << "Usage: " << argv[0] << " <value of x>" << std
    ::endl;
        return 1;
    }}
    double x = std::stod(argv[1]);
    double vcpp = velocity(x);
    std::cout << std::abs(vcpp) << std::endl;
    return 0;
}}
'''
with open('v.cpp','w') as f:
        f.write(executable)

import subprocess
compile_process = subprocess.run(["g++", "-o", "vel.cpp", "v.cpp"],
     stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
```

2.4 Output:

```
nxtblazevfx@fedora:~/Documents/id2090/assignments/assignment5$ ./q2.sh press.txt
nxtblazevfx@fedora:~/Documents/id2090/assignments/assignment5$ ./vel.cpp 0.5
0.375
```