# Code Overview: Fourier Transform in the Complex Plane

Shreyash Singh

## 1 Introduction

This document provides a concise overview of the Python (Jupyter Notebook) implementation used to compute and visualize the Fourier transform in the complex plane. The code is designed to:

- Define suitable test functions (Gaussian and exponential),
- Compute the complex Fourier transform numerically,
- Visualize the magnitude and phase over the complex plane,
- Generate animated spectral slices.

The full implementation is modular and emphasizes clarity over heavy optimization.

## 2 Core Libraries and Setup

The following libraries are used throughout the project:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

- `NumPy` is used for numerical computation and array handling.
- `Matplotlib` is used for all static plots and animations.
- `FuncAnimation` enables animated spectral slices.

## 3 Definition of the Test Function

The primary test function used is the Gaussian:

```
def f(x):
    return np.exp(-x**2)
```

This function is smooth, rapidly decaying, and belongs to the Schwartz class, ensuring convergence of the complex Fourier transform for all $z \in \mathbb{C}$.

# 4 Complex Grid Construction

The complex frequency grid is constructed using a tensor product mesh:

```
xi_vals  = np.linspace(-5, 5, 200)
eta_vals = np.linspace(-3, 3, 200)
XI, ETA = np.meshgrid(xi_vals, eta_vals)
Z = XI + 1j * ETA
```

Here, $\xi$ represents the real frequency and $\eta$ represents the exponential growth or decay rate.

# 5 Numerical Evaluation of the Transform

The Fourier transform is computed using the trapezoidal rule:

```
x = np.linspace(-20, 20, 6000)
FZ = np.zeros_like(Z, dtype=complex)

for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        z = Z[i, j]
        integrand = f(x) * np.exp(-1j * z * x)
        FZ[i, j] = np.trapz(integrand, x)
```

This double loop evaluates the Fourier integral at every point in the complex plane grid.

# 6 Magnitude and Phase Visualization

Once the transform is computed, the magnitude and phase are extracted as

```
absF = np.abs(FZ)
argF = np.angle(FZ)
```

These arrays are visualized using two-dimensional heatmaps with `imshow`.

# 7 Animated Spectral Slices

To visualize how the spectrum deforms along the imaginary direction, animated slices are generated:

```
fig, ax = plt.subplots()
line, = ax.plot(xi_vals, np.abs(FZ[0, :]))
```

The animation updates the plotted data for each fixed $\eta$ value, showing exponential decay and instability regions dynamically.

# 8 Numerical Stability and Accuracy

Accuracy of the computation depends on:

- The truncation of the real line to $[-L, L]$,
- The number of quadrature points in $x$,
- The resolution of the complex grid.

Increasing these parameters improves accuracy at the cost of greater computational time.

# 9 Conclusion

This codebase provides a clean and modular implementation of the complex Fourier transform. The numerical pipeline mirrors the theoretical development and enables high-quality visualizations suitable for research presentation, reports, and further PDE or spectral extensions.