Applications of Real Analysis in Computer Science

Geoffrey Wang

August 2024

1 Introduction

Real Analysis has profound applications in various fields of computer science, including signal processing, numerical analysis, probability theory, and data analysis. This document provides Python implementations for several key algorithms that are grounded in real analysis, showcasing their practical applications.

2 Fourier Transform and Signal Processing

Fourier Transform is a fundamental tool in real analysis used for converting time-domain signals into their frequency-domain representation. The Fourier Transform of a signal $f(t)$ is given by:

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

This transformation is crucial in signal processing for analyzing the frequency components of signals.

Listing 1: Fourier Transform and Signal Processing in Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

# Generate a composite signal with two frequency
    components
sampling_rate = 1000
t = np.linspace(0, 1, sampling_rate)
signal = np.sin(2 * np.pi * 50 * t) + np.sin(2 * np.pi
    * 120 * t)

# Compute Fourier Transform
```

```
fft_signal = fft(signal)

# Frequency axis
frequencies = np.fft.fftfreq(len(fft_signal), 1/
    sampling_rate)

# Plot signal and spectrum
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(t, signal)
plt.title("Original␣Signal")
plt.xlabel("Time␣(s)")
plt.ylabel("Amplitude")

plt.subplot(2, 1, 2)
plt.plot(frequencies[:len(frequencies)//2], np.abs(
    fft_signal)[:len(frequencies)//2])
plt.title("Frequency␣Spectrum")
plt.xlabel("Frequency␣(Hz)")
plt.ylabel("Magnitude")

plt.tight_layout()
plt.show()
```

3 Numerical Integration

Numerical integration methods, such as the trapezoidal rule and Simpson's rule, are commonly used in real analysis to approximate the area under a curve, particularly when an analytical solution is difficult or impossible to obtain. For example, the integral of a function $f(x)$ from a to b is given by:

$$\int_a^b f(x)\,dx$$

Listing 2: Numerical Integration of a Function in Python

```
import numpy as np
from scipy.integrate import quad

# Define the function
def f(x):
    return np.sin(x) / x if x != 0 else 1.0  # Sinc
        function

# Compute numerical integration
```

```
integral, error = quad(f, 0, np.inf)

print(f"Integral of sinc function from 0 to infinity:
    {integral}")
```

4 Error Function and Probability Distribution

The error function is a special function in real analysis frequently used in probability theory and statistics. The error function $\text{erf}(x)$ is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

It is closely related to the cumulative distribution function of the normal distribution.

Listing 3: Plotting the Error Function in Python

```
import numpy as np
from scipy.special import erf
import matplotlib.pyplot as plt

# Generate data
x = np.linspace(-3, 3, 100)
y = erf(x)

# Plot the error function
plt.plot(x, y, label="erf(x)")
plt.title("Error Function")
plt.xlabel("x")
plt.ylabel("erf(x)")
plt.grid(True)
plt.legend()
plt.show()
```

5 Data Interpolation and Approximation

In real-world data analysis, interpolation is a technique used to construct new data points within the range of known data points. Real analysis provides the theoretical foundation for these interpolation techniques. Given a set of points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, interpolation constructs a function $f(x)$ such that:

$$f(x_i) = y_i \quad \text{for all } i$$

Listing 4: Data Interpolation Example in Python

```
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

# Generate sparse data points
x = np.linspace(0, 10, 10)
y = np.sin(x)

# Linear interpolation
f_linear = interp1d(x, y)

# Fit new data points
x_new = np.linspace(0, 10, 100)
y_linear = f_linear(x_new)

# Plot original data and interpolation result
plt.plot(x, y, 'o', label='Data␣Points')
plt.plot(x_new, y_linear, '-', label='Linear␣
    Interpolation')
plt.title("Linear␣Interpolation␣Example")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

6 Wavelet Transform and Signal Processing

The wavelet transform is another powerful tool in real analysis, particularly useful for analyzing non-stationary signals in the time-frequency domain. A discrete wavelet transform decomposes a signal into its wavelet coefficients:

$$W_\psi(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-b}{a}\right) dt$$

Listing 5: Wavelet Transform in Python

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

# Generate a signal
t = np.linspace(0, 1, 400)
signal = np.sin(2 * np.pi * 50 * t) + np.sin(2 * np.pi
    * 120 * t)
```

```
# Perform wavelet transform
coeffs = pywt.wavedec(signal, 'db1', level=4)

# Plot wavelet coefficients
plt.figure(figsize=(10, 8))
for i, coef in enumerate(coeffs):
    plt.subplot(len(coeffs), 1, i + 1)
    plt.plot(coef)
    plt.title(f"Wavelet Coefficients Level {i + 1}")
plt.tight_layout()
plt.show()
```

7 Conclusion

These examples demonstrate how real analysis is applied in various computational tasks using Python. From signal processing to numerical integration, real analysis provides the theoretical underpinning for these practical techniques.