SHRI RAMSWAROOP MEMORIAL UNIVERSITY

Department of Computer Science and Engineering

Signal & System Presentation on

Sampling Theorem Demonstrator

Using Python

Submitted To – Mr. Pappu Kumar
Assistant Professor
Department of Electrical & Electronics Engg.
Shri Ramswaroop Memorial University

Submitted By – Jasika Awasthi (202310101150**036**)
Course - B.Tech
Branch - CSE (DS + AI)
Group - 51 , 52

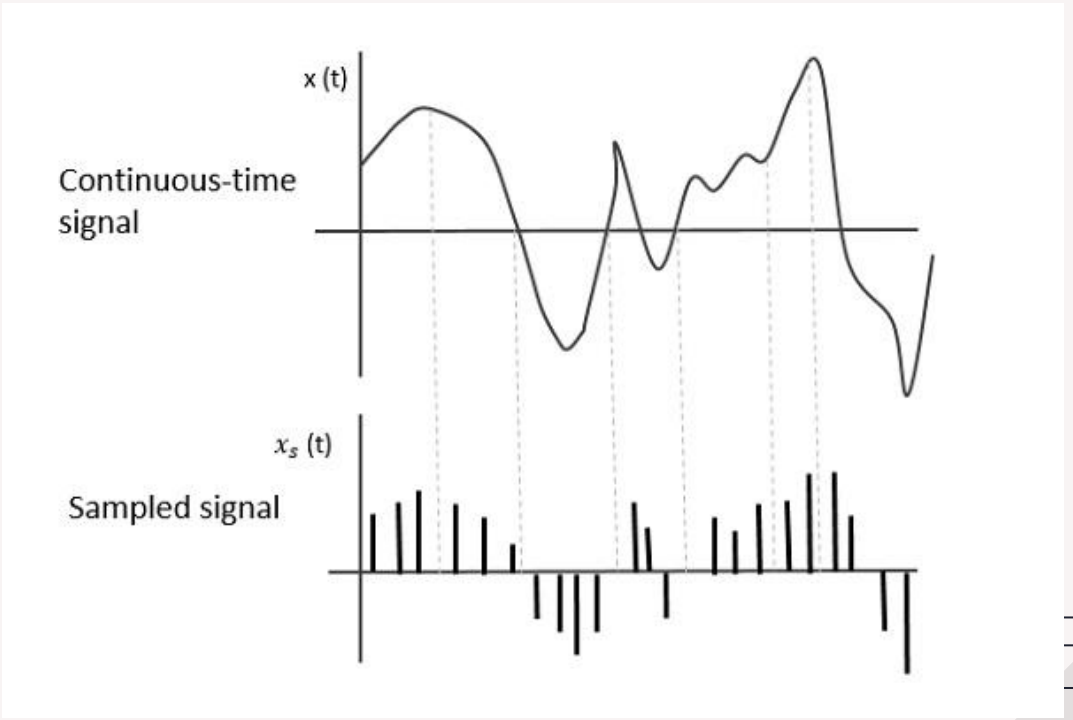
Table of Content

Introduction

- The process of converting an analog signal into digital signal known as Sampling.
- The Sampling Theorem states that a continuous-time signal can be completely represented by its sample if and only if sampling frequency (fs) must be greater than and twice its highest frequency component present in the signal (fm).

$$fs >= 2fm$$

Fig.I- Continuous-Time Signal and Its Sampled Signal



Overview of Demonstrator

This Python-based tool visualizes:

- Continuous Signal
- Sampled Signal
- Reconstructed Signal using sinc interpolation
- Determine the type of sampling (Over sampling, Critical sampling & Under sampling)
- Interactive sliders allow real-time adjustments.

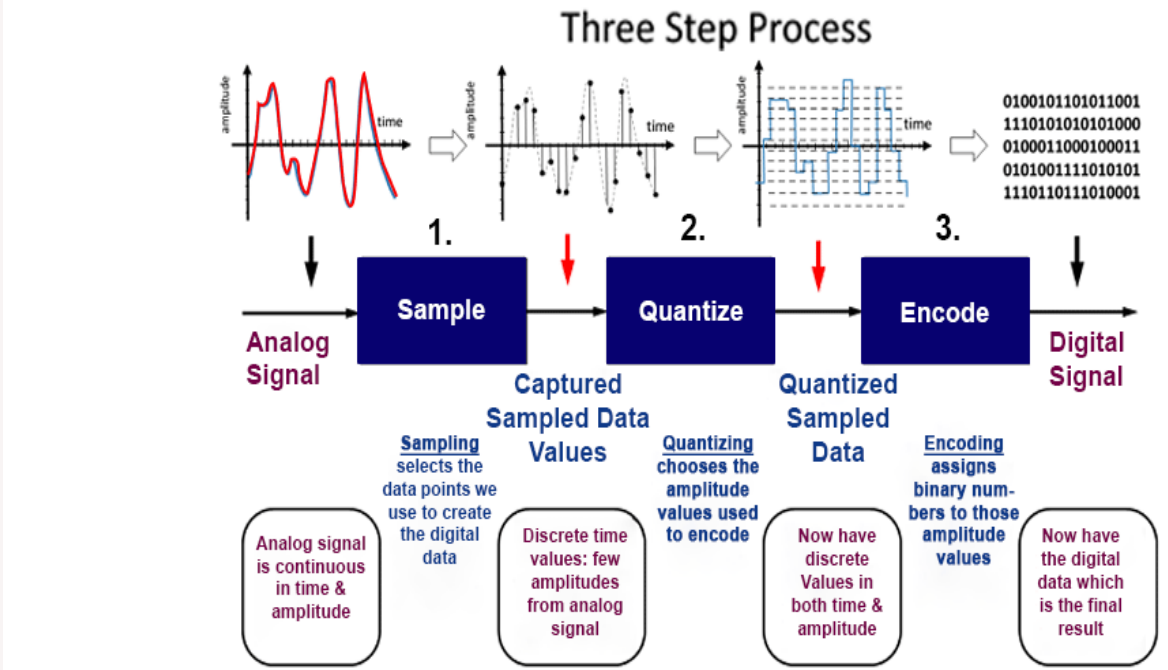


Fig.3- Analog-to-Digital Conversion (ADC) Process

Tools and Technology

- **Python:** Programming language used for implementing the demonstrator
- **NumPy:** Library for numerical computations and signal generation.
- **Matplotlib:** Library for plotting and visualizing signals
- **ipywidgets:** Library for creating interactive sliders and dropdowns in Google Colab
- **Google Colab:** Cloud-based Python environment, enable interactive without local setup
- **Powerpoint:** Used for creating presentation slides

Python Programming Language

Fig.2- Python Logo

What is Python ?

High-level, interpreted programming language used for general-purpose programming including scientific computing, data analysis and visualization.

Key Features :

Easy to read, Easy to learn, large library support, widely use in education, data science

Who discovered Python ?

Discovered by Guido Van Rossum in 1989

Why Python for this Demonstrator ?

Simple syntax, strong support for scientific libraries and works seamlessly with Google Colab for interactive visualizations.

Sampling in Frequency Domain

Over - Sampling

1. Sampling frequency is higher than twice the maximum signal frequency.

fs > 2fm

2. No overlap between spectral copies; reconstruction is perfect.

Perfect – Sampling

1. Sampling frequency equals to the twice of maximum signal frequency (Nyquist rate).

fs = 2fm

2. Spectral copies touch but do not overlap ; ideal reconstruction possible.

Under - Sampling

1. Sampling frequency is less than twice the maximum signal frequency

fs < 2fm

2. Spectral copies overlap → aliasing occurs → original signal cannot be perfectly reconstructed.

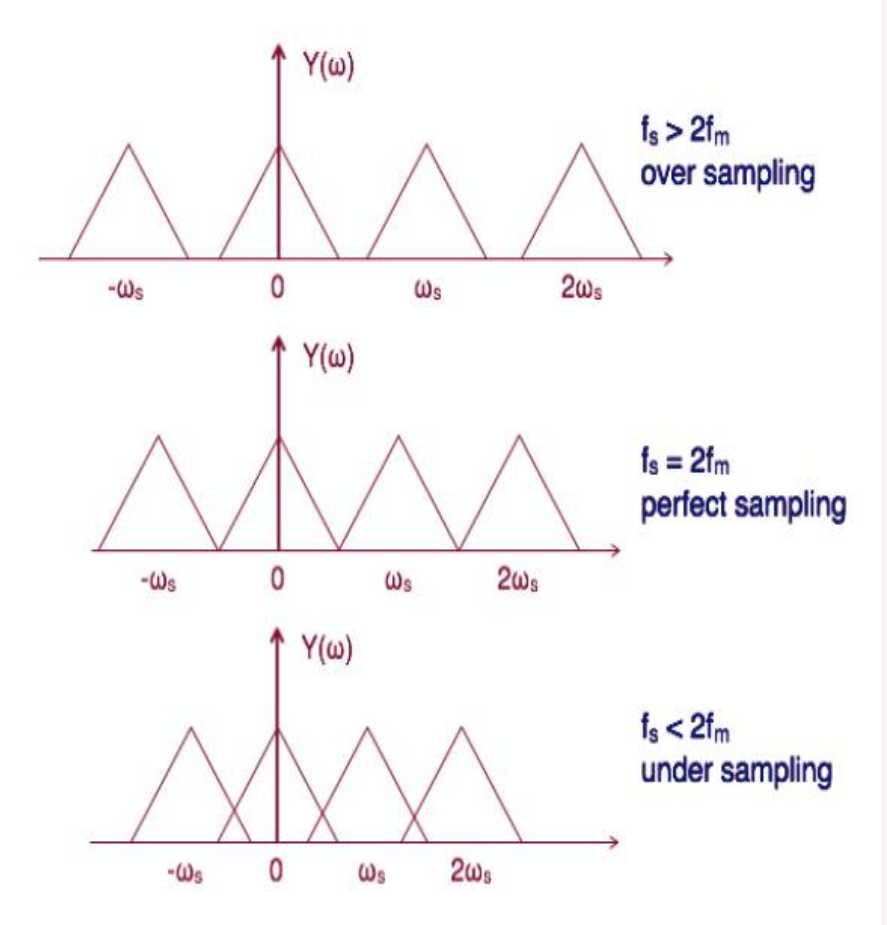


Fig.4- Types of Sampling Frequency

Advantages

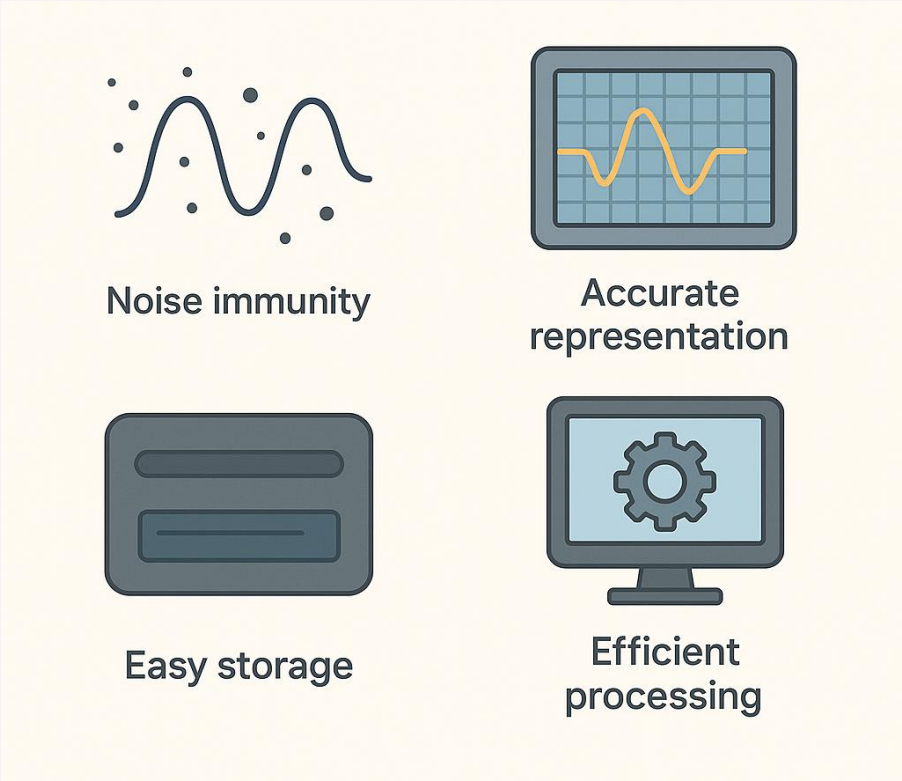


Fig. 5- Advantages of Sampling Frequency

1. **Accurate Reconstruction:** Allows perfect reconstruction when sampling ≥ Nyquist rate.

2. **Efficient Digital Processing:** Enables easy storage, transmission, and manipulation of signals

3. **Reduced Data Requirements:** Avoids oversampling, saving memory and processing time.

4. **Reliable Communication:** Ensures distortion-free transmission with proper sampling

5. **Foundation of Digital Signal Processing:** Forms the basis of digital audio, imaging, and communication systems.

Disadvantages

1. **Requires Band-Limited Signals:** Real-world signals are not perfectly band-limited.

2. **Aliasing Risk:** Sampling below Nyquist rate leads to distortion and false frequencies.

3. **High Resource Usage (at high rates):** Higher sampling demands more storage and faster processors.

4. **Ideal Filters Needed:** Perfect anti-aliasing and reconstruction filters can't be built practically.

5. **Noise Sensitivity:** Noisy signals require higher sampling rates to maintain accuracy.

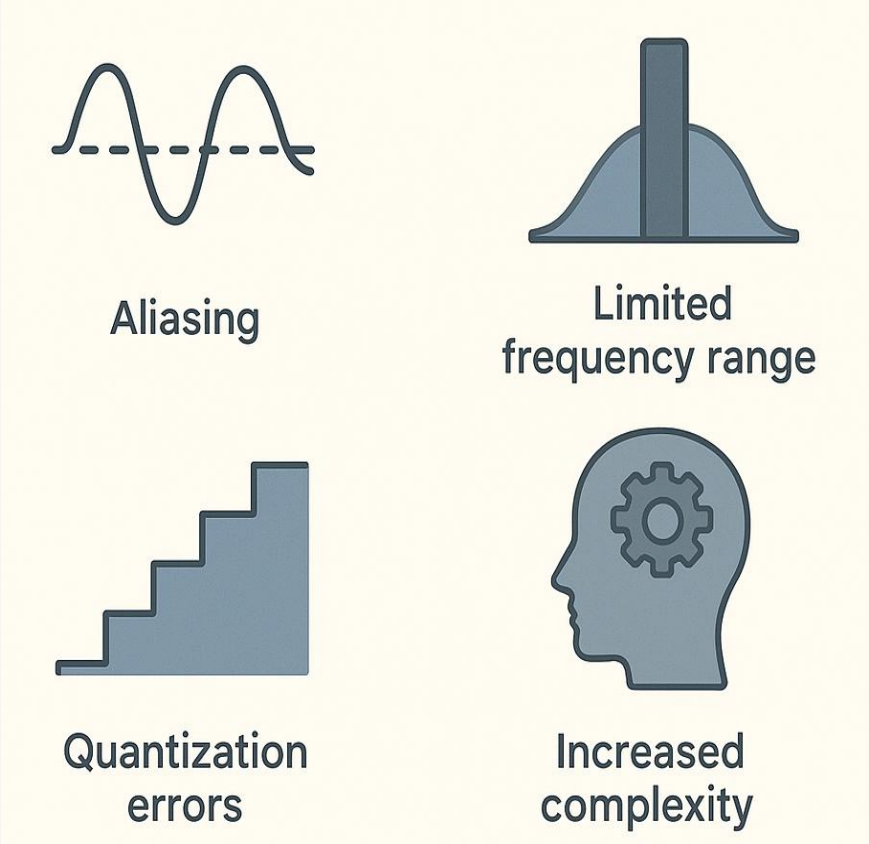


Fig. 6- Disadvantages of Sampling Frequency

Applications

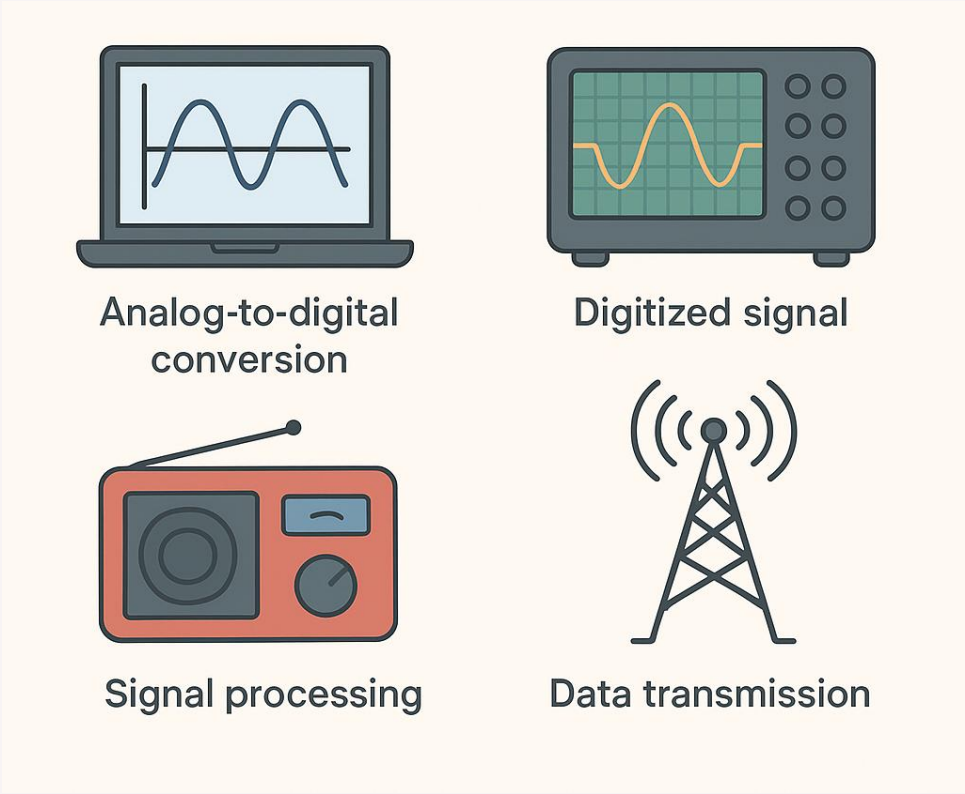


Fig. 7- Applications of Sampling Frequency

1. Digital audio and music processing
2. Telecommunication systems (4G/5G, modems)
3. Medical devices (ECG, MRI, CT)
4. Digital image acquisition (cameras, scanners)

Implementation

```
# 🎯 Sampling Theorem Demonstrator – Final Advanced Version
# ✅ Works directly in Google Colab
# 🧠 Features: Dual View (Time + Frequency), Aliasing Warning, RMS Error,

import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, FloatSlider, Dropdown, Checkbox
from IPython.display import clear_output

# ------------------------------------
# 1️⃣ Define the continuous-time signal
# ------------------------------------
def continuous_signal(t, freq, signal_type):
    if signal_type == "Sine":
        return np.sin(2 * np.pi * freq * t)
    elif signal_type == "Cosine":
        return np.cos(2 * np.pi * freq * t)
    elif signal_type == "Square":
        return np.sign(np.sin(2 * np.pi * freq * t))
    else:
        return np.sin(2 * np.pi * freq * t)

#
```

```
# ------------------------------------
# 2️⃣ Sampling function
# ------------------------------------
def sample_signal(t, signal_func, fs):
    Ts = 1/fs
    t_samples = np.arange(t[0], t[-1], Ts)
    y_samples = signal_func(t_samples)
    return t_samples, y_samples

# ------------------------------------
# 3️⃣ Vectorized sinc reconstruction
# ------------------------------------
def reconstruct_signal_fast(t, t_samples, y_samples, fs):
    sinc_matrix = np.sinc((t[:, None] - t_samples[None, :]) * fs)
    return np.dot(sinc_matrix, y_samples)

# ------------------------------------
# 4️⃣ Frequency spectrum (FFT)
# ------------------------------------
def compute_spectrum(signal, fs):
    N = len(signal)
    freq = np.fft.fftfreq(N, 1/fs)
    magnitude = np.abs(np.fft.fft(signal)) / N
    return freq[:N//2], magnitude[:N//2]
```

```python
# ------------------------------------
# 5️⃣ Main plotting function (dual view)
# ------------------------------------
def plot_sampling(signal_freq=5.0, sampling_freq=10.0, signal_type="Sine",
                  show_reconstruction=True, dual_view=True):

    clear_output(wait=True)
    t = np.linspace(0, 1, 1000)
    y_cont = continuous_signal(t, signal_freq, signal_type)
    t_samples, y_samples = sample_signal(t, lambda tt: continuous_signal
     (tt, signal_freq, signal_type), sampling_freq)
    y_recon = reconstruct_signal_fast(t, t_samples, y_samples, sampling_freq)
    if show_reconstruction
    else: np.zeros_like(y_cont)
    # --- Time Domain Plot ---
    if dual_view:
        fig, axes = plt.subplots(1, 2, figsize=(14, 5))
        ax1, ax2 = axes
    else:
        fig, ax1 = plt.subplots(figsize=(10, 5))

    ax1.plot(t, y_cont, label="Continuous Signal", color='blue')
    ax1.plot(t_samples, y_samples, 'o', label="Sampled Points", color='red')
    if show_reconstruction:
        ax1.plot(t, y_recon, '--', label="Reconstructed Signal", color='green')
    ax1.set_title("Time-Domain Representation")
    ax1.set_xlabel("Time (s)")
    ax1.set_ylabel("Amplitude")
    ax1.grid(True)
    ax1.legend()
```

```python
# Aliasing and parameter display
if sampling_freq < 2 * signal_freq:
    ax1.text(0.02, 0.9, "⚠️ Under-sampling (Aliasing!)", color='red',
             transform=ax1.transAxes, fontsize=11, fontweight='bold')
else:
    ax1.text(0.02, 0.9, "✅ Proper Sampling (No Aliasing)", color='green',
             transform=ax1.transAxes, fontsize=11, fontweight='bold')
ax1.text(0.02, 0.8, f"Signal Freq = {signal_freq}
Hz\nSampling Freq = {sampling_freq} Hz",
         transform=ax1.transAxes, fontsize=10,
         bbox=dict(facecolor='white', alpha=0.7))
# --- Frequency Domain Plot ---
if dual_view:
    freq_cont, mag_cont = compute_spectrum(y_cont, 1000)
    freq_recon, mag_recon = compute_spectrum(y_recon, sampling_freq)
    ax2.plot(freq_cont, mag_cont, label="Original Spectrum", color='blue')
    if show_reconstruction:
        ax2.plot(freq_recon, mag_recon, '--',
                 label="Reconstructed Spectrum", color='green')
    ax2.set_title("Frequency-Domain Representation")
    ax2.set_xlabel("Frequency (Hz)")
    ax2.set_ylabel("Magnitude")
    ax2.grid(True)
    ax2.legend()
plt.tight_layout()
plt.show()
# RMS Error (if reconstruction is enabled)
if show_reconstruction:
    error = np.sqrt(np.mean((y_cont - y_recon)**2))
    print(f"🔹 RMS Reconstruction Error: {error:.4f}")
```

```
# ------------------------------------
# 6 Interactive Controls
# ------------------------------------
interact(
    plot_sampling,
    signal_freq=FloatSlider(value=5, min=1, max=20, step=0.5,
                            description="Signal Freq (Hz)"),
    sampling_freq=FloatSlider(value=10, min=1, max=50, step=1,
                              description="Sampling Freq (Hz)"),
    signal_type=Dropdown(options=["Sine", "Cosine", "Square"],
                         value="Sine", description="Signal Type"),
    show_reconstruction=Checkbox(value=True, description="Show Reconstruction"),
    dual_view=Checkbox(value=True, description="Dual View (Time + Freq)")
)
```

Output

Signal Freq... 9.50

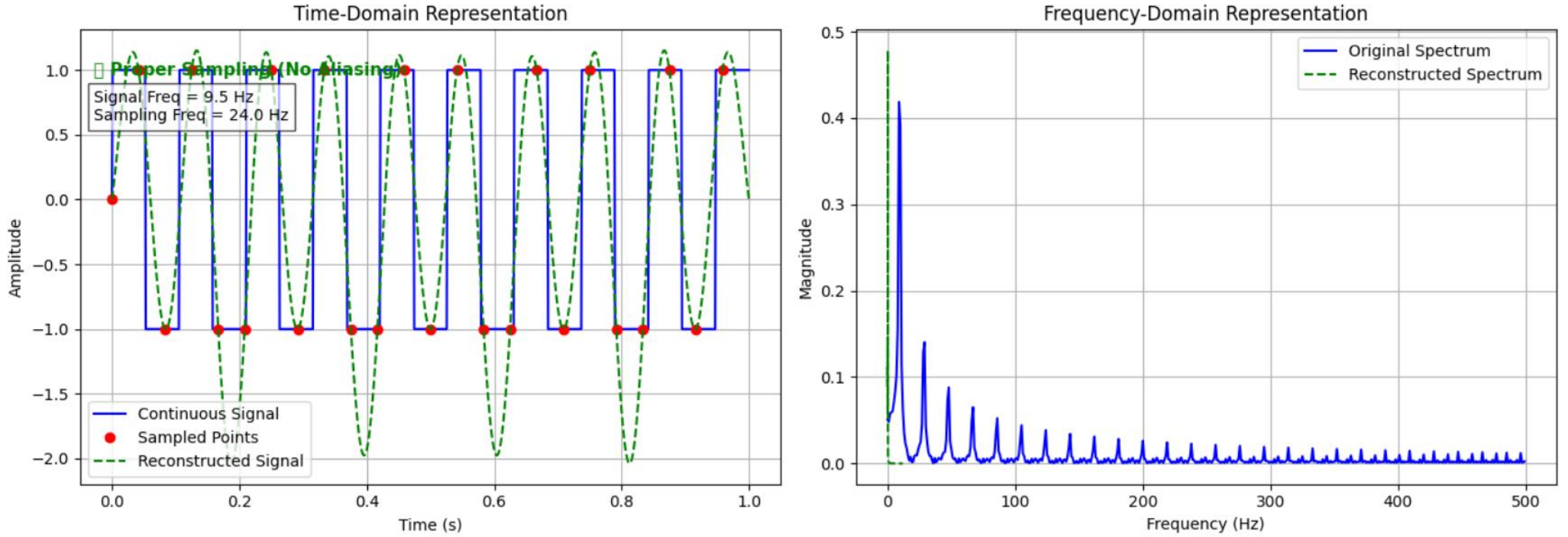
Sampling F... 24.00

Signal Type Square

Show Reconstruction

Dual View (Time + Freq)

```
/tmp/ipython-input-3800348745.py:98: UserWarning: Glyph 9989 (\N{WHITE HEAVY CHECK MARK}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
```



RMS Reconstruction Error: 0.5948

Conclusion

Project Impact & Significance :

The Sampling Theorem Demonstrator provides an effective and interactive platform for understanding one of the most fundamental principles in digital signal processing. By visually illustrating how sampling rate affects the accuracy of signal reconstruction, the tool enhances conceptual clarity and supports deeper learning for students, educators, and engineers.

Key Takeaways :

- **Educational Value** : The demonstrator simplifies complex signal theory by allowing users to observe real-time effects of sampling, quantization, and aliasing.
- **Practical Utility :** It serves as a powerful tool for analyzing and interpreting sampling behavior relevant to digital communication, audio processing, and embedded systems.
- **Real-World Relevance :** The visualization highlights the importance of the Nyquist rate in preventing aliasing, reinforcing principles used in modern DSP, telecommunications, and biomedical instrumentation.

References

- Brewer, K. R. W., & Hanif, M. (1983). Sampling with unequal probabilities. Springer-Verlag New York, Inc.
- G.R. Liu, Mesh Free Methods: Moving Beyond the Finite Element Method, Taylor and Francis Group, New York, 2010
- Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H. and Teller E. 1953 Equation of state calculations by fast computing machines J. Chem. Phys. 21 1087-1092
- Hastings W. K. 1970 Monte Carlo Sampling Methods Using Markov Chains and Their Applications Biometrika 57 97-109

Thank You !!