# Tarea 2: Procesamiento Digital de Señales

Daniel Estuardo, Suy Fuentes, 202000443

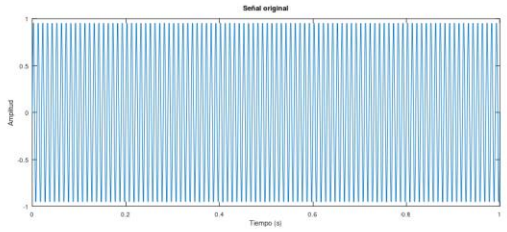
Proyectos de computación aplicados a I.E

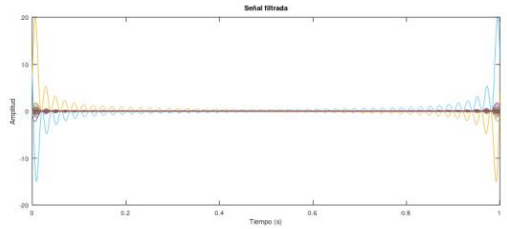
Escuela de Mecánica Eléctrica, Facultad de Ingeniería, Universidad de San Carlos de Guatemala

El código proporcionado en Octave se utiliza para generar una señal senoidal, aplicar la Transformada de Fourier, y luego aplicar un filtro pasa-bajo en el dominio de la frecuencia para filtrar la señal.

```
%Generar señal senoidal
fs = 1000; %frecuencia de muestreo
t = 0:1/fs:1; %vector de tiempo
f = 100; %frecuencia de la señal
x = sin(2*pi*f*t); %Definimos la variable x para la señal senoidal

%Aplicar Transformada de Fourier
xf = fft(x);

%Generar filtro pasa-bajo
n = length(x);
fcutoff = 50; %frecuencia de corte
h = ones(n, 1); %vector de ceros
h(round(n*fcutoff/fs)+1:end) = 0; % aplicar filtro pasa-bajo

% Aplicar filtro a la señal en el dominio de la frecuencia
xf_filtered = xf .* h;

% Convertir señal filtrada a dominio del tiempo
x_filtered = ifft(xf_filtered);

%Graficar señal original y señal filtrada
figure;
subplot (2,1,1);
plot(t, x);
title('Señal original');
xlabel('Tiempo (s)');
ylabel ('Amplitud');
subplot(2,1,2);
plot(t, real(x_filtered));
title('Señal filtrada');
xlabel('Tiempo (s)');
ylabel('Amplitud');
```

Graficas de la señal original y la señal resultante



Código en Python utilizando la biblioteca NumPy para las operaciones matemáticas y matplotlib para graficar:

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

# Generar señal senoidal
fs = 1000  # frecuencia de muestreo
t = np.arange(0, 1, 1/fs)  # vector de tiempo
f = 100  # frecuencia de la señal
x = np.sin(2 * np.pi * f * t)  # Definir la variable x para la señal senoidal

# Aplicar Transformada de Fourier
xf = np.fft.fft(x)

# Generar filtro pasa-bajo
n = len(x)
fcutoff = 50  # frecuencia de corte
h = np.ones(n)  # vector de unos
h[int(np.round(n * fcutoff / fs)):] = 0  # aplicar filtro pasa-bajo

# Aplicar filtro a la señal en el dominio de la frecuencia
xf_filtered = xf * h

# Convertir señal filtrada al dominio del tiempo
x_filtered = np.fft.ifft(xf_filtered)

# Graficar señal original y señal filtrada
plt.figure()

plt.subplot(2, 1, 1)
plt.plot(t, x)
plt.title('Señal original')
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')
```

```python
plt.subplot(2, 1, 2)
plt.plot(t, np.real(x_filtered))
plt.title('Señal filtrada')
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')

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

Graficas de la señal original y la señal resultante en Python