

El ejercicio consiste en transformar un filtro analógico (continuo) en un filtro digital (discreto) utilizando dos técnicas distintas: la transformación de Euler y la transformación bilineal¹. El objetivo es analizar cuál de estos métodos produce un resultado más fiel al sistema original, dada una frecuencia de muestreo específica.

1. Determinación de la Frecuencia de Corte y Muestreo

El primer paso es caracterizar el sistema continuo para definir los parámetros de la discretización.

Argumentación del Paso

Para convertir un sistema continuo a discreto, necesitamos definir una **frecuencia de muestreo (f_s)**. El enunciado nos pide basar esta frecuencia en la **frecuencia de corte (f_c)** del filtro analógico. La frecuencia de corte es un parámetro fundamental del filtro, ya que define el límite de su banda de paso. Es la frecuencia en la que la magnitud de la respuesta del sistema cae 3 decibelios (dB) con respecto a su valor máximo. Según el Teorema del Muestreo, para poder reconstruir una señal, la frecuencia de muestreo debe ser al menos el doble de la frecuencia máxima presente en la señal $(f_s \geq 2f_{\max})^2$. El ejercicio propone usar un factor de 4 ($f_s = 4f_c$), lo cual es una elección común para asegurar un buen margen.

Cálculo y Código

La función de transferencia del sistema continuo es:

$$H(s) = 44s^2 + 60625s + 625 \cdot 10412500s$$

Para encontrar f_c , analizamos la magnitud de la respuesta en frecuencia, $|H(j\omega)|$, donde $s = j\omega$. Buscamos la frecuencia donde la magnitud es $\frac{1}{\sqrt{2}}$ (equivalente a -3 dB) de la magnitud máxima.

El siguiente código Python realiza este cálculo numéricamente:

1. Define los coeficientes de $H(s)$.
2. Calcula la magnitud de $H(j\omega)$ para un rango de frecuencias.
3. Encuentra la magnitud máxima y el valor correspondiente a -3 dB.
4. Identifica la frecuencia (f_c) donde ocurre esta caída.
5. Calcula la frecuencia de muestreo ($f_s = 4f_c$) y el período de muestreo ($T = 1/f_s$).

<!-- end list -->

Python

```
import numpy as np
import scipy.signal as sig
import matplotlib.pyplot as plt
```

```

import pandas as pd

# Coeficientes de H(s)
num_s = [12500, 0]
den_s = [44, 60625, 6250000]

# Análisis para encontrar fc
w = np.logspace(0, 4, 2000) # Rango de frecuencias angulares
frecuencias_continuas = w / (2 * np.pi)
s = 1j * w
H_s = np.polyval(num_s, s) / np.polyval(den_s, s)
H_s_abs = np.abs(H_s)

# Cálculo de la frecuencia de corte
max_mag_lineal = np.max(H_s_abs)
mag_3db_lineal = max_mag_lineal / np.sqrt(2) # Valor en -3 dB
indiceFc = np.argmin(np.abs(H_s_abs - mag_3db_lineal))
fc = frecuencias_continuas[indiceFc]

# Cálculo de Parámetros de Discretización
fs = 4 * fc
T = 1 / fs

```

Los resultados obtenidos son:

- **Frecuencia de Corte (f_c):**
- **Frecuencia de Muestreo (f_s):**
- **Período de Muestreo (T):**

2. Aplicación de las Transformaciones Conformes

Ahora, con el período de muestreo T definido, se procede a obtener la función de transferencia discreta $H(z)$ mediante los dos métodos solicitados³³³³.

A. Transformación de Euler

Argumentación del Método

La transformación de Euler (hacia atrás) aproxima la derivada del tiempo continuo por una diferencia finita en el tiempo discreto⁴.

$$\text{dtdy}(t) \approx T y[n] - y[n-1]$$

Esto conduce a una relación de mapeo del plano s al plano z :

$$s = T^{-1}z - 1 = Tzz^{-1} - 1$$

Para obtener la función de transferencia discreta, se sustituye esta expresión de s en la

función de transferencia analógica $H(s)$.

Cálculo y Resultado

Se reemplaza s en $H(s)$:

$$H_{\text{Euler}}(z) = H(s) \Big|_{s = \frac{1-z}{Tz-1}}$$

Tras una extensa simplificación algebraica, se llega a la forma de cociente de polinomios en z .

El resultado es:

$$H_{\text{Euler}}(z) = \frac{z^2 - 1.487z + 0.5594}{0.003926z - 0.003926} = \frac{1 - 1.487z^{-1} + 0.5594z^{-2}}{1 - 1.487z^{-1} + 0.5594z^{-2}}$$

B. Transformación Bilineal

Argumentación del Método

La transformación bilineal se basa en una aproximación trapezoidal de la integración, lo que resulta en un mapeo diferente y, en general, más robusto⁷⁷⁷⁷. La sustitución correspondiente es⁸:

$$s = \frac{1-z^{-1}}{T(1+z^{-1})} = \frac{Tz-1}{Tz+1}$$

Este método tiene la ventaja de mapear de forma unívoca todo el eje imaginario ($j\omega$) del plano s sobre el círculo unitario del plano z ⁹⁹⁹⁹. Además, mapea el semiplano izquierdo de s (región de estabilidad para sistemas continuos) al interior del círculo unitario de z (región de estabilidad para sistemas discretos)¹⁰¹⁰¹⁰¹⁰.

Cálculo y Resultado

Se reemplaza s en $H(s)$:

$$H_{\text{Bilineal}}(z) = H(s) \Big|_{s = \frac{Tz-1}{Tz+1}}$$

La simplificación algebraica de esta expresión da como resultado:

$$H_{\text{Bilineal}}(z) = \frac{z^2 - 1.139z + 0.6355}{0.1878z^2 - 0.1878} = \frac{1 - 1.139z^{-1} + 0.6355z^{-2}}{1 - 1.139z^{-1} + 0.6355z^{-2}}$$

3. Análisis Comparativo de la Respuesta en Frecuencia

El último paso es evaluar qué tan bien los filtros discretos obtenidos imitan al filtro analógico original. Para ello, se grafican las respuestas en frecuencia de los tres sistemas.

Argumentación del Análisis

La calidad de una transformación se juzga por la similitud entre la respuesta en frecuencia del sistema discreto y la del sistema continuo original¹¹. Si la transformación es exitosa, la forma y las características clave (ganancia máxima, frecuencia de corte) del filtro digital deben ser casi idénticas a las del analógico.

Resultados Gráficos y Conclusión

El código genera una gráfica comparativa de las magnitudes en dB.



Python

(Código para calcular H_bilineal y H_euler en un rango de frecuencias)

Gráfica Comparativa

```
plt.figure(figsize=(12, 7))
plt.semilogx(frecuencias_continuas, 20 * np.log10(np.abs(H_s)), label='Original H(s)',
linewidth=2.5)
plt.semilogx(frecuencias_digitales, 20 * np.log10(np.abs(H_euler)), linestyle='--',
label='Discreto (Euler)')
plt.semilogx(frecuencias_digitales, 20 * np.log10(np.abs(H_bilineal)), linestyle=':',
label='Discreto (Bilineal)', linewidth=2)
plt.axvline(fc, color='red', linestyle='-.', label=f'fc = {fc:.2f} Hz')
plt.title('Comparación de Respuestas en Frecuencia')
plt.xlabel('Frecuencia [Hz]')
plt.ylabel('Magnitud |H(f)| [dB]')
plt.legend()
plt.grid(True, which="both", ls="-")
plt.ylim(-40, 5)
plt.show()
```

Análisis de los resultados:

-  **Transformación Bilineal (Línea de Puntos):** La respuesta del sistema obtenido mediante la transformación bilineal es **excelente**. Sigue casi perfectamente la curva del sistema continuo original en todo el rango de frecuencias, preservando la ganancia y la frecuencia de corte.
-  **Transformación de Euler (Línea Discontinua):** La respuesta del sistema de Euler es **deficiente**. La curva está notablemente distorsionada: la ganancia máxima es mucho menor y la forma del filtro se ha alterado significativamente.

¿Fue adecuada la frecuencia de muestreo ($f_s=4f_c$)?

- **Para la Transformación Bilineal: Sí.** El método mapea correctamente las frecuencias, y la elección de $f_s=4f_c$ es más que suficiente para obtener una representación fiel del filtro analógico.
- **Para la Transformación de Euler: NO.** El mal resultado tiene una justificación teórica directa en el apunte. La transformación de Euler solo funciona bien si las frecuencias involucradas son bajas en relación con la frecuencia de muestreo. Específicamente, el apunte indica que la aproximación es aceptable solo si el ángulo digital normalizado θ es pequeño ($\theta < \pi/6$ radianes). Calculemos el ángulo en la frecuencia de corte de nuestro sistema:
$$\theta_c = f_s 2\pi f_c = 4f_c 2\pi f_c = 2\pi \text{ radianes}$$

Como $\frac{2\pi}{6} > \frac{2\pi}{6}$, la condición de validez no se cumple en absoluto en la región más importante del filtro. Esta es la razón teórica por la que la transformación de Euler falla en este caso y produce una distorsión tan grande. Para que Euler funcionara, se

necesitaría una frecuencia de muestreo mucho mayor ($f_s \gg f_c$)¹³.