Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Síntesis de Redes Activas

Trabajo Práctico de Laboratorio N° 4:

Filtros Activos

Nombre	DNI
Mangín, Matías Eduardo	35.596.112
Tardón, Damián David	36.733.088
Tito, Ricardo Clemente	35.308.739

Profesor Titular: Dr. Ing. Ferreyra Pablo
Profesor Adjunto: Ing. Reale César

Córdoba, República Argentina

2025

Índice

1. Introducción. **2**

2. Desarrollo **3**
2.1. Aproximación de función de transferencia por Python. 3
2.2. Análisis Topológico. 4
2.2.1. Realimentación Positiva (Sallen-Key) 5
2.2.2. Realimentación Negativa (Sallen-Key) 6
2.2.3. Definición de Componentes 7
2.2.3.1. Realimentación Positiva 7
2.2.3.2. Realimentación Negativa 8

1. Introducción.

En base a la planilla de requerimiento suministrada, por el docente. Se sintetizo un circuito basado en amplificadores operacionales que satisfaga esos requisitos.

Abordamos el diseño de un filtro activo utilizando el polinomio de Chebyshev para aproximar su función de atenuación. Comenzamos con una plantilla de requerimientos que proporcionaba frecuencias clave y funciones de transferencia, las cuales analizamos para extraer especificaciones como la frecuencia de corte, el rizo en la banda de paso y la atenuación en la banda de rechazo. Seleccionamos un filtro Chebyshev Tipo I y calculamos su orden, determinando que un filtro de segundo orden sería adecuado para cumplir con los requisitos. Calculamos los polos del filtro y construimos su función de transferencia. Se utilizó amplificadores operacionales, resistencias y capacitores para implementar la función de transferencia en un circuito activo, considerado configuraciones como Sallen-Key o filtro de estado variable.

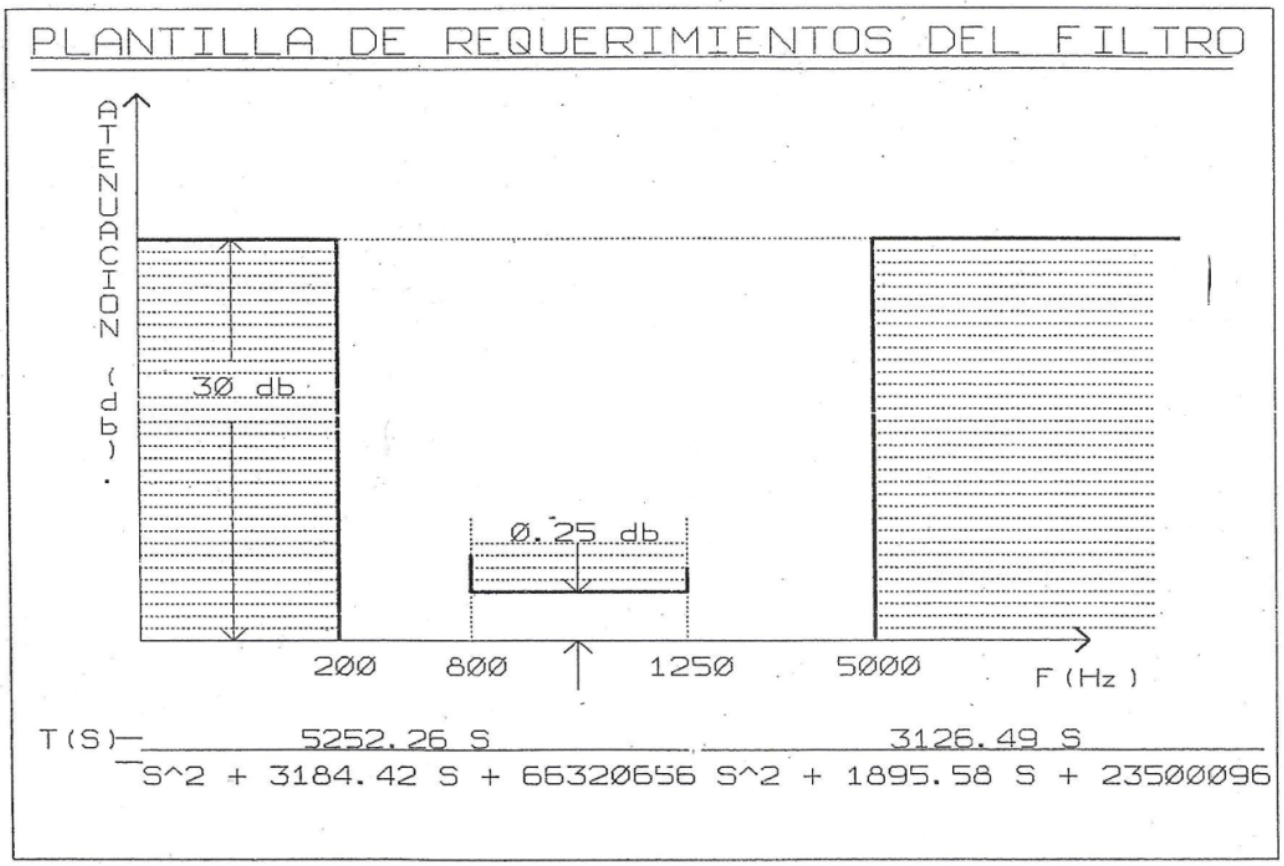

Figura 1: Plantilla de requerimientos del filtro.

2. Desarrollo

En base a la planilla de requerimientos de la Fig. 1. se pide:

2.1. Aproximación de función de transferencia por Python.

Aproximar a la función de atenuación mediante polinomios de Chebychev, utilizando Python. La plantilla proporciona la siguiente información: Frecuencias de interés:

- Banda de Rechazo:
$$200\ Hz\ ;\ 5000\ Hz$$
- Banda de Paso:
$$800\ Hz\ ;\ 1250\ Hz$$
- Atenuación en la Banda de Paso:
$$A_p = 0,25\ dB$$
- Atenuación en la Banda de Rechazo:
$$A_s = 30\ dB$$

Utilizamos la biblioteca scipy de python para poder obtener la función de transferencia del filtro correspondiente.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy import signal

fp=[800, 1250]     # BANDA DE PASO [Hz]
fs=[200, 5000]     # BANDA DE RECHAZO [Hz]

As=30      # Atenuacion en la BANDA DE RECHAZO [dB]
Ap=0.25    # Atenuacion en la BANDA DE PASO [dB]

# Convertir frecuencias a radianes por segundo
Wp = 2 * np.pi * np.array(fp)  # Banda de paso [rad/s]
Ws = 2 * np.pi * np.array(fs)  # Banda de rechazo [rad/s]

# Calcular el orden del filtro y frecuencia normalizada
n, Wp_norm = signal.cheb1ord(Wp, Ws, Ap, As, analog=True)
print(f'El orden obtenido es n = {n}')

# Disenar el filtro Chebyshev Tipo I
num, den = signal.cheby1(n, Ap, Wp_norm, btype='bandpass', analog=True, fs=None)

# Mostrar la funcion de transferencia
print("funcion de transferencia del filtro Chebyshev:")
print("Numerador: ", num)
print("Denominador: ", den)

# Crear la funcion de transferencia
filtro = signal.TransferFunction(num, den)

# Convertir la funcion de transferencia en secciones de segundo orden (SOS)
sos = signal.tf2sos(num, den)
print("\nSecciones de segundo orden (SOS):")
print(sos)
```

Este programa nos permitió descomponer una función de transferencia de orden superior en secciones bicuadráticas, lo que es útil para implementar filtros en hardware o software.

```
Secciones bicuadraticas:
Sección 1:
Numerador: [16420899.35090685 0. 0.]
```

```
Denominador: [1.00000000e+00 3.18442632e+03 6.63208202e+07]

Sección 2:
Numerador: [1. 0. 0.]
Denominador: [1.00000000e+00 1.89557535e+03 2.35000932e+07]
```

Para poder ver lo generado por código en python, se graficó la magnitud y fase de la función de transferencia sumando las siguientes líneas:

```
FiltroGraf= ctrl.TransferFunction(den,num)

omega = np.logspace(3,5,100)
plt.figure(figsize=(12,6))
ctrl.bode_plot(FiltroGraf,omega,dB=True,Hz=True)
plt.show()
```

Resultando en el siguiente gráfico:

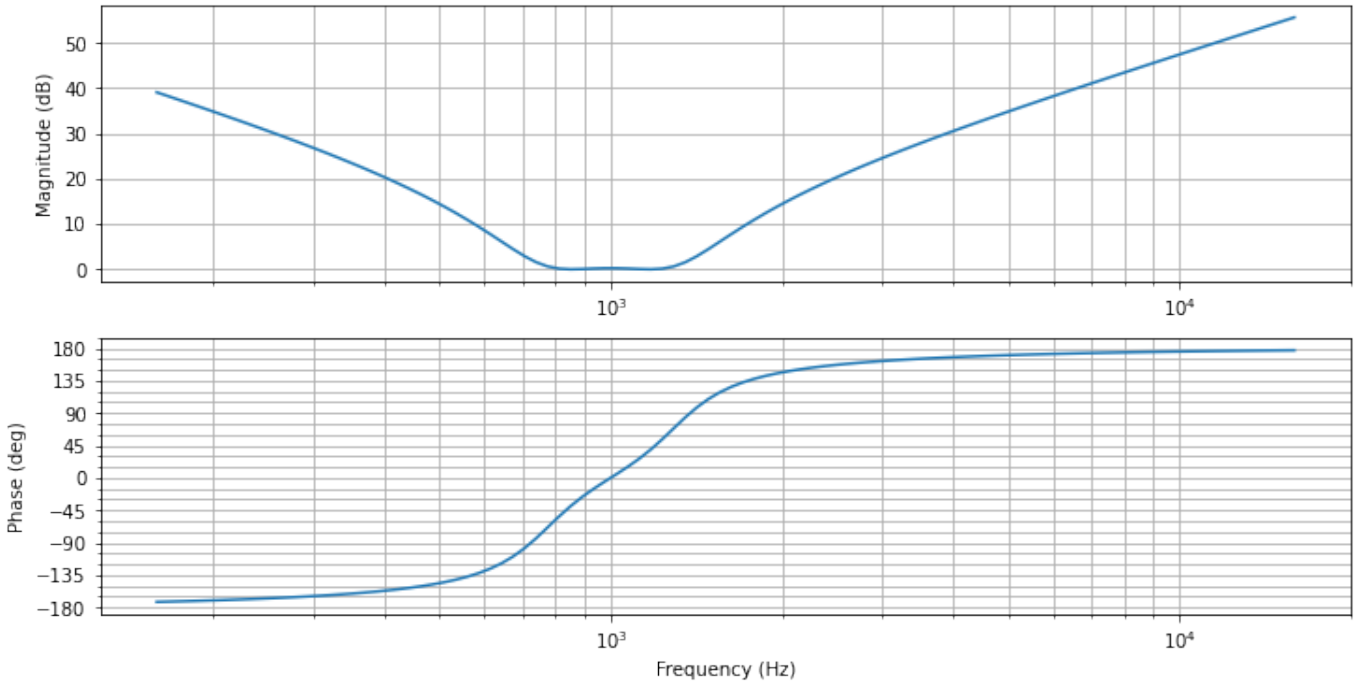

Figura 2: Magnitud y Fase de la Función de Transferencia

```
La Funcion de Transferencia es:


                        1.642e+07 s^2
-----------------------------------------------------------
s^4 + 5080 s^3 + 9.586e+07 s^2 + 2.006e+11 s + 1.559e+15
```

2.2. Análisis Topológico.

Se sintetizó un circuito que satisfaga los requerimientos del punto anterior utilizado topologías bicuadráticas de realimentación positiva y negativa. Para poder implementar el filtro se analizaron ambas topologías canónicas. Haciendo uso del modulo SymPy de Python se consiguieron las funciones de transferencia para cada una.

2.2.1. Realimentación Positiva (Sallen-Key)

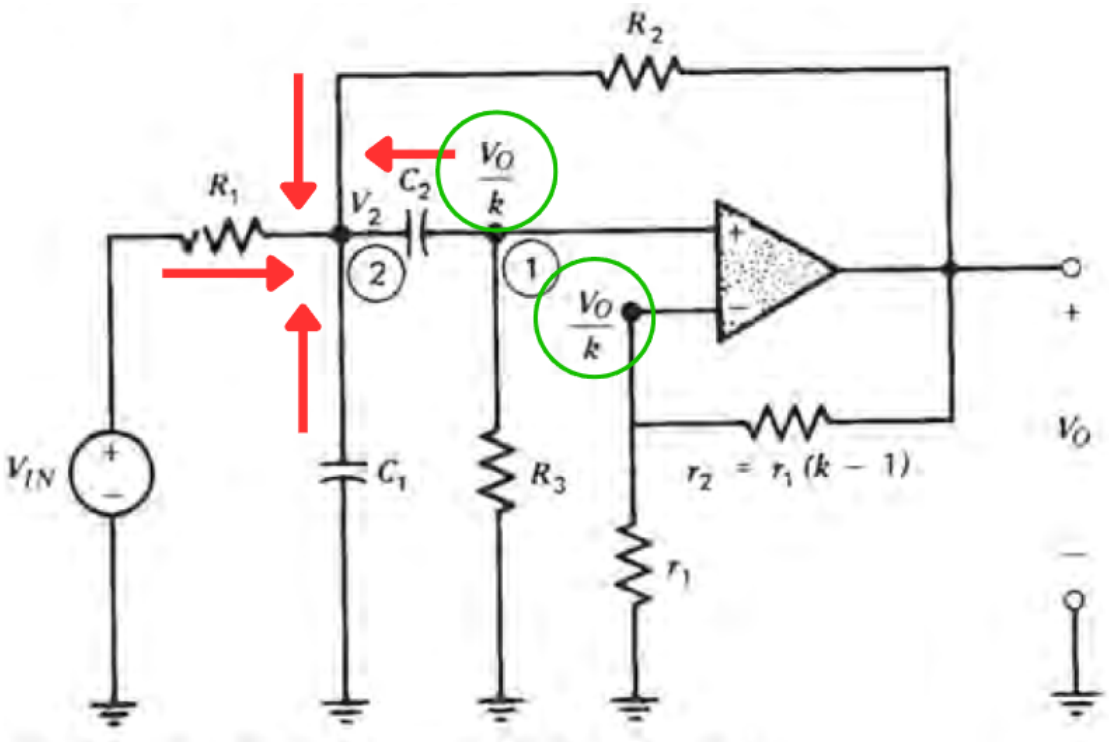

Figura 3: Topología de realimentación positiva (Sallen-Key BP).

```
import sympy as sp

# Definimos las variables simbolicas
V2, Vo, Vi, C1, C2, R1, R2, R3, s, k = sp.symbols('V2 Vo Vi C1 C2 R1 R2 R3 s k')

# Definimos las ecuaciones
eq1 = V2 * (C1 * s + C2 * s + 1/R2 + 1/R1) - Vo/R2 - Vi/R1 - (C2 * Vo * s)/k
eq2 = (Vo * (C2 * s + 1/R3)) / k - C2 * V2 * s

# Resolver el sistema de ecuaciones para V2 y Vo
solucion = sp.solve([eq1, eq2], (V2, Vo))

# Extraer la solucion para Vo
Vo_sol = solucion[Vo]

# Simplificar la expresion de Vo/Vi
funcion_transferencia = sp.simplify(Vo_sol / Vi)

# Manipular la funcion de transferencia para que tenga la forma deseada
# Dividimos numerador y denominador por el coeficiente de s^2 en el denominador
coeficiente_s2 = C1 * C2 * R1 * R2 * R3

# Simplificar nuevamente
funcion_transferencia = sp.simplify(funcion_transferencia/coeficiente_s2)

# Mostrar la funcion de transferencia
print("Funcion de transferencia Vo/Vi:")
sp.pretty_print(funcion_transferencia)
```

El resultado de impreso a la salida es:

```
k*s/(C1*R1*(C1*C2*R1*R2*R3*s**2 + C1*R1*R2*s + C2*R1*R2*s - C2*R1*R3*k*s + C2*R1*R3*s
```

Esta se puede reordenar para obtener una mejor aproximación a la función de transferencia deseada:

$$G(s) = \frac{k_1 s}{s^2 + a_1 s + b_1} \tag{1}$$

La función de transferencia tendrá la forma:

$$G(s) = \frac{k}{C_1 R_1} \cdot \frac{s}{s^2 + s\left(\frac{1}{C_2 R_3} + \frac{1}{C_1 R_3} - \frac{k}{C_1 R_2} + \frac{1}{C_1 R_2} + \frac{1}{C_1 R_1}\right) + \frac{1}{C_1 C_2 R_2 R_3} + \frac{1}{C_1 C_2 R_1 R_3}} \tag{2}$$

2.2.2. Realimentación Negativa (Sallen-Key)

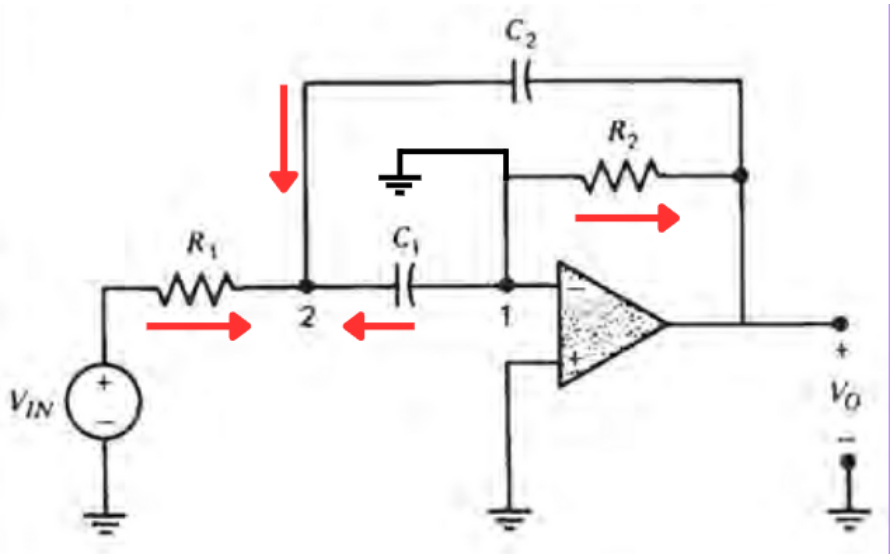

Figura 4: Topología de realimentación negativa (BP).

```python
import sympy as sp # type: ignore

# Definimos las variables simbolicas
V2, Vo, Vi, C1, C2, R1, R2, R3, s, k = sp.symbols('V2 Vo Vi C1 C2 R1 R2 R3 s k')

# Definimos las ecuaciones
eq1 = V2 * (C1 * s + C2 * s + 1/R1) - Vi/R1 - C2 * Vo * s
eq2 = -C1 * V2 * s -Vo/R2

# Resolver el sistema de ecuaciones para V2 y Vo
solucion = sp.solve([eq1, eq2], (V2, Vo))

# Extraer la solucion para Vo
Vo_sol = solucion[Vo]

# Simplificar la expresion de Vo/Vi
funcion_transferencia = sp.simplify(Vo_sol / Vi)

# Manipular la funcion de transferencia para que tenga la forma deseada
# Dividimos numerador y denominador por el coeficiente de s^2 en el denominador
coeficiente_s2 = C1 * R2

# Simplificar nuevamente
funcion_transferencia = sp.simplify(funcion_transferencia/coeficiente_s2)

print(funcion_transferencia)
# Mostrar la funcion de transferencia
print("funcion de transferencia Vo/Vi:")
```

```
sp.pretty_print(funcion_transferencia)
```

Resultando en la siguiente Función de Transferencia:

$$G = \frac{-s}{C_1C_2R_1R_2s^2 + C_1R_1s + C_2R_1s + 1}$$

La función de transferencia tiene la forma:

$$G = -\frac{\frac{1}{C_2R_1}s}{s^2 + s\left(\frac{1}{C_2R_2} + \frac{1}{C_1R_2}\right) + \frac{1}{C_1C_2R_1R_2}} = \frac{k_2s}{s^2 + a_2s + b_2}$$

2.2.3. Definición de Componentes

2.2.3.1. Realimentación Positiva

En el diseño de filtros activos, **Daryanani** en **Principles of Active Network Synthesis And Design** propone un método para simplificar el diseño al reducir el número de incógnitas. En este caso, se asumen condiciones como:

$$C_1 = C_2 = 1 \quad y \quad R_1 = R_2 = R_3 = R$$

Esto reduce el número de variables y permite resolver el sistema de ecuaciones de manera más sencilla. Estas suposiciones son comunes en el diseño de filtros y se basan en la normalización de componentes.

Las ecuaciones de diseño proporcionadas son:

Función de transferencia:

$$G = \frac{k_1s}{s^2 + a_1s + b_1} = \frac{5252{,}26s}{s^2 + 3184{,}42s + 66320656}$$

Relaciones entre parámetros:

$$\begin{aligned} k_1 &= \frac{k}{C_1R_1} \\ a_1 &= \frac{1}{C_2R_3} + \frac{1}{C_1R_3} - \frac{k}{C_1R_2} + \frac{1}{C_1R_2} + \frac{1}{C_1R_1} \\ b_1 &= \frac{1}{C_1C_2R_2R_3} + \frac{1}{C_1C_2R_1R_3} \end{aligned}$$

Condiciones de Daryanani:

$$C_1 = C_2 = 1 \quad y \quad R_1 = R_2 = R_3 = R$$

Ecuaciones simplificadas:

$$\begin{aligned} -\frac{k}{R} + \frac{4}{R} &= 3184{,}42 \\ \frac{2}{R^2} &= 66320656 \end{aligned}$$

Para poder obtener el valor correspondiente a la resistencia R y C, según la simplificación ofrecida por Daryanani, se aplicó el siguiente algoritmo:

```
import sympy as sp

# Definir las variables simbolicas
k, R ,C= sp.symbols('k R C')
C=100e-9 # defino a C en 100 nF

# Ecuaciones simplificadas
eq1 = -k/(C*R) + 4/(C*R) - 3184.42
eq2 = 2/(C**2*R**2) - 66320656

# Resolver el sistema de ecuaciones
R =sp.solve(eq2,R) # defino el valor de R
R=R[1]
eq1 = -k/(C*R) + 4/(C*R) - 3184.42 # redefino la ecuacion para el nuevo valor de R
```

```
K, =sp.solve(eq1,k) # como es cuadratica obtengo un valor positivo y otro negativo

# Mostrar la Solucion
print("Solucion del sistema de ecuaciones:")
print(f"k = {K}")
print(f"R = {R}")
```

Generando el siguiente resultado:

```
#Solucion del sistema de ecuaciones:
k = 3.44700534446283
R = 1736.56319058784
```

Pudimos aproximar a estos resultados implementando las siguientes resistencias comerciales:

$$
\begin{aligned}
R &= 1,5\ k\Omega + 220\ \Omega + 18\ \Omega \\
k \approx 3,45 &= 1 + \frac{r_2}{r_1} \\
r_1 &= 1\ k\Omega \quad y \quad r_2 = 2,44\ k\Omega \\
r_2 &= 2,2\ k\Omega + 720\ \Omega // 720\ \Omega // 720\ \Omega
\end{aligned}
$$

Analizando a cada resistencias, sin perder precisión, ya que las tolerancias de los componentes garantizan que el valor real esté dentro del rango aceptable. Esto facilitó la selección de componentes y el diseño del circuito, redefiniendo los valores:

$$
\begin{aligned}
R &= 1,8\ k\Omega \\
r_1 &= 1\ k\Omega \\
r_2 &= 2,2\ k\Omega + 220\ \Omega \\
C_1 &= 100\ nF \\
C_2 &= 100\ nF
\end{aligned}
$$

2.2.3.2. Realimentación Negativa