Nome: Victor Yaegashi Setti

RA: 206362

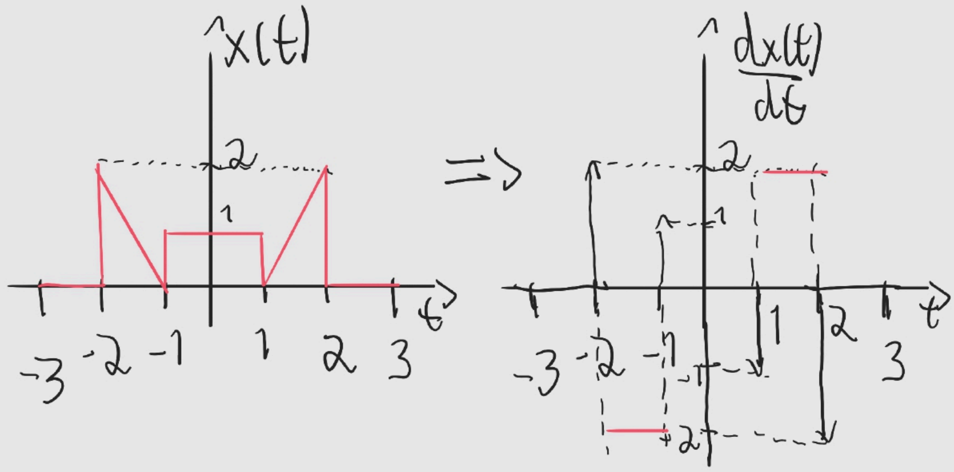
Importando as bibliotecas que serão usadas e definindo pi = π

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import pandas as pd
from IPython.display import Image, display

pi = np.pi
```

Item A: Calculando manualmente os coeficientes da série de Fourier para a onda x(t)

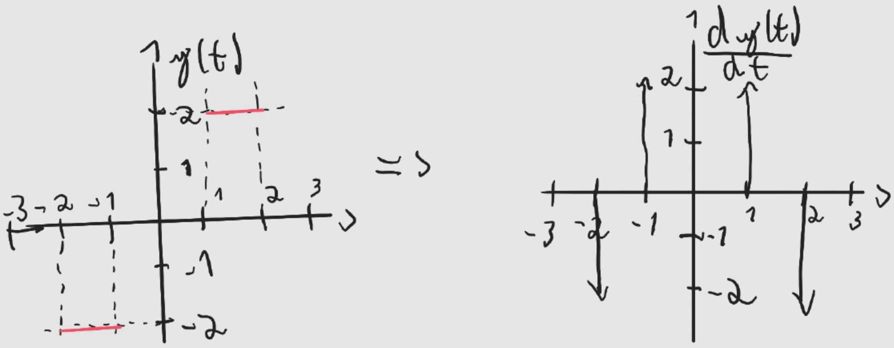
In [10]:
```
nome_imagens = ["itemA1.jpg", "itemA2.jpg", "itemA3.jpg"]

for nome_imagem in nome_imagens:
    display(Image(filename=nome_imagem, width=500))
```

a)



$$a_k = x_k$$

$$(jk\omega_0)x_k = \left(\frac{1}{6}\cdot 2e^{2jk\omega_0}\right) + \left(\frac{1}{6}\cdot 1e^{jk\omega_0}\right) + \left(\frac{1}{6}(-1)e^{-jk\omega_0}\right) + \left(\frac{1}{6}(-2)e^{-2jk\omega_0}\right) + y_k$$



$$(jk\omega_0)\, y_k = \left(\frac{1}{6}(-2)e^{2jk\omega_0}\right) + \left(\frac{1}{6}\cdot 2e^{jk\omega_0}\right) + \left(\frac{1}{6}\cdot 2e^{-jk\omega_0}\right) + \left(\frac{1}{6}(-2)e^{-2jk\omega_0}\right)$$

$$\Rightarrow a_k = \frac{1}{jk\omega_0}\left[\frac{1}{3}e^{2jk\omega_0} + \frac{1}{6}e^{jk\omega_0} - \frac{1}{6}e^{-jk\omega_0} + \frac{1}{3}e^{-2jk\omega_0}\right] - \frac{1}{k^2\omega_0^2}\left[-\frac{1}{3}e^{2jk\omega_0} + \frac{1}{3}e^{jk\omega_0} + \frac{1}{3}e^{-jk\omega_0} - \frac{1}{3}e^{-2jk\omega_0}\right]$$

$$\Rightarrow a_k = \frac{1}{jk\omega_0}\left[\frac{1}{3}\left(\cancel{\cos}(2k\omega_0) + j\sin(2k\omega_0)\right) + \frac{1}{6}\left(\cancel{\cos}(k\omega_0) + j\sin(k\omega_0)\right) - \frac{1}{6}\left(\cancel{\cos}(k\omega_0) - j\sin(k\omega_0)\right) - \frac{1}{3}\left(\cancel{\cos}(2k\omega_0) - j\sin(2k\omega_0)\right)\right]$$

$$- \frac{1}{k^2\omega_0^2}\left[-\frac{1}{3}\left(\cos(2k\omega_0) + \cancel{j\sin}(2k\omega_0)\right) + \frac{1}{3}\left(\cos(k\omega_0) + \cancel{j\sin}(k\omega_0)\right) + \frac{1}{3}\left(\cos(k\omega_0) - \cancel{j\sin}(k\omega_0)\right) - \frac{1}{3}\left(\cos(2k\omega_0) - \cancel{j\sin}(2k\omega_0)\right)\right]$$

$$\Rightarrow a_k = \frac{1}{jk\omega_0}\left[\frac{2}{3}j\sin(2k\omega_0) + \frac{1}{3}j\sin(k\omega_0)\right] - \frac{1}{k^2\omega_0^2}\left[-\frac{2}{3}\cos(2k\omega_0) + \frac{2}{3}\cos(k\omega_0)\right]$$

$$\Rightarrow a_k = \frac{2}{3k\omega_0}\sin(2k\omega_0) + \frac{1}{3k\omega_0}\sin(k\omega_0) + \frac{2}{3k^2\omega_0^2}\cos(2k\omega_0) - \frac{2}{3k^2\omega_0^2}\cos(k\omega_0)$$

Como $\omega_0 = \frac{2\pi}{6} = \frac{\pi}{3} \Rightarrow \omega_0^2 = \frac{\pi^2}{9}$

$$\boxed{\Rightarrow a_k = \frac{2}{k\pi}\sin\left(\frac{2k\pi}{3}\right) + \frac{1}{k\pi}\sin\left(\frac{k\pi}{3}\right) + \frac{6}{k^2\pi^2}\cos\left(\frac{2k\pi}{3}\right) - \frac{6}{k^2\pi^2}\cos\left(\frac{k\pi}{3}\right)}$$

$$a_0 = \frac{1}{6}\int_{-3}^{3} x(t)\,dt = \frac{1}{6}\left[\int_{-2}^{-1}(-2-2t)\,dt + \int_{-1}^{1} 1\,dt + \int_{1}^{2}(-2+2t)\,dt\right]$$

$$= \frac{1}{6}\left[\left[-2t-t^2\right]_{-2}^{-1} + \left[t\right]_{-1}^{1} + \left[-2t+t^2\right]_{1}^{2}\right]$$

$$= \frac{1}{6}\left[(2-1-\cancel{4}+\cancel{4}) + (1+1) + (-\cancel{4}+\cancel{4}+2-1)\right] = \frac{4}{6} = \boxed{\frac{2}{3}}$$

Item B: Gerando uma aproximação para x(t) utilizando a série de Fourier e plotando os gráficos

Fazendo uma função para calcular a função x(t) no período de -3 a 3

In [11]:
```
def funcao_original():
    vetor = []

    valores_t = np.linspace(-3, 3, 1000)
    for t in valores_t:
        if (-2 <= t and t < -1):
            valor = -2 - 2*t
```

```
        elif (-1 <= t and t < 1):
            valor = 1
        elif (1 <= t and t < 2):
            valor = -2 + 2*t
        else:
            valor = 0
        vetor.append(valor)
    return vetor
```

Fazendo uma função para calcular os coeficientes da série de Fourier truncada com os harmônicos de índice -N a N

In [12]:
```
def coeficientes_serie_fourier(n: int):
    coeficientes = []
    for k in range(-n, n+1):
        if k == 0:
            ak = 2/3
        else:
            ak = (2/(k*pi))*np.sin((2*k*pi)/3) + (1/(k*pi))*np.sin((k*pi)
        coeficientes.append(ak)
    return coeficientes
```

Fazendo uma função para calcular a série de Fourrier truncada num tempo t

In [13]:
```
def serie_fourier(n: int, t: float):
    valor = 0
    coeficientes = coeficientes_serie_fourier(n)
    for k in range(-n, n+1):
        exponencial = np.exp(1j*k*(pi/3)*t)
        valor += coeficientes[k+n] * exponencial
    return valor
```

Fazendo uma função para calcular uma função aproximada de x(t) usando a série de Fourier truncada

In [14]:
```
def funcao_aproximada(n: int):
    vetor = []

    valores_t = np.linspace(-3, 3, 1000)
    for t in valores_t:
        valor = serie_fourier(n, t)
        vetor.append(np.real(valor))
    return vetor
```

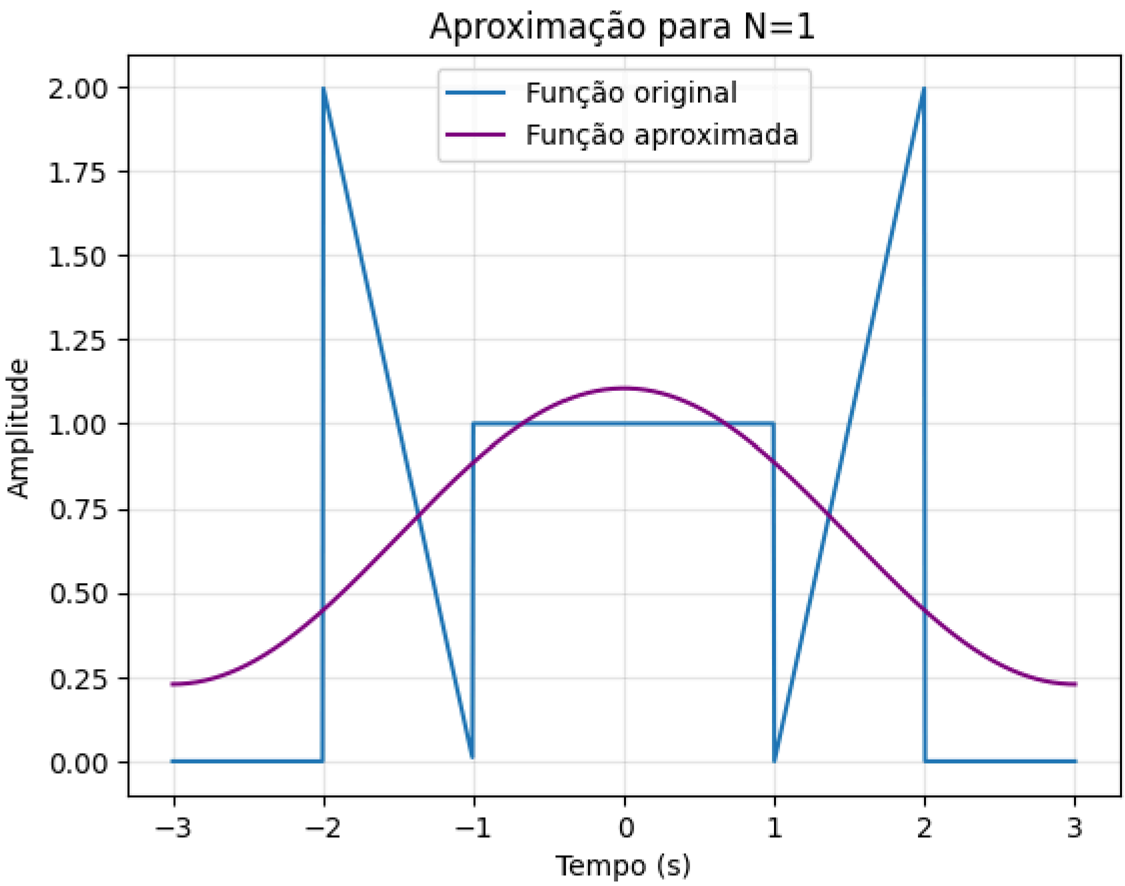
Fazendo uma função para plotar a comparação das ondas para qualquer N inteiro inserido

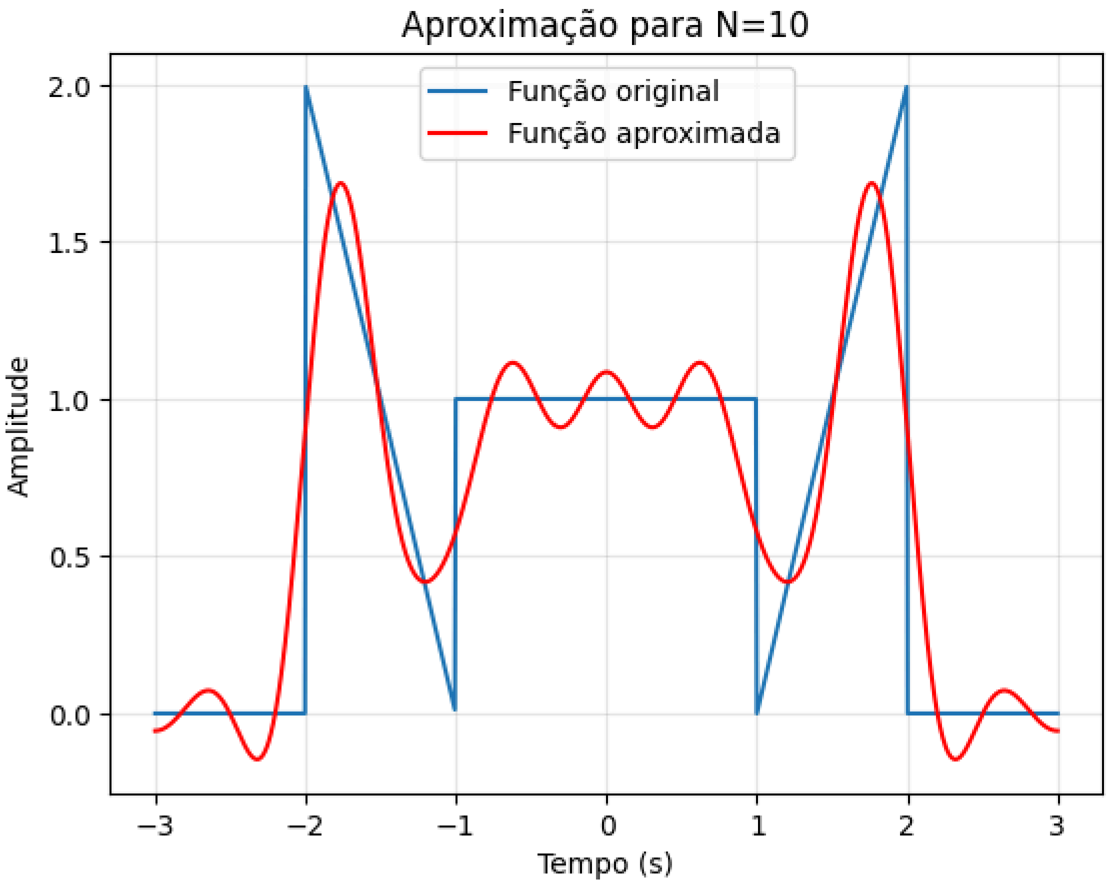
In [15]:
```
def plot_comparacao(n: int, color: str):
    original = funcao_original()
    aproximada = funcao_aproximada(n)

    valores_t = np.linspace(-3, 3, 1000)
    plt.plot(valores_t, original, label="Função original")
    plt.plot(valores_t, aproximada, label="Função aproximada", color=colo
    plt.xlabel("Tempo (s)")
    plt.ylabel("Amplitude")
    plt.title(f"Aproximação para N={n}")
```

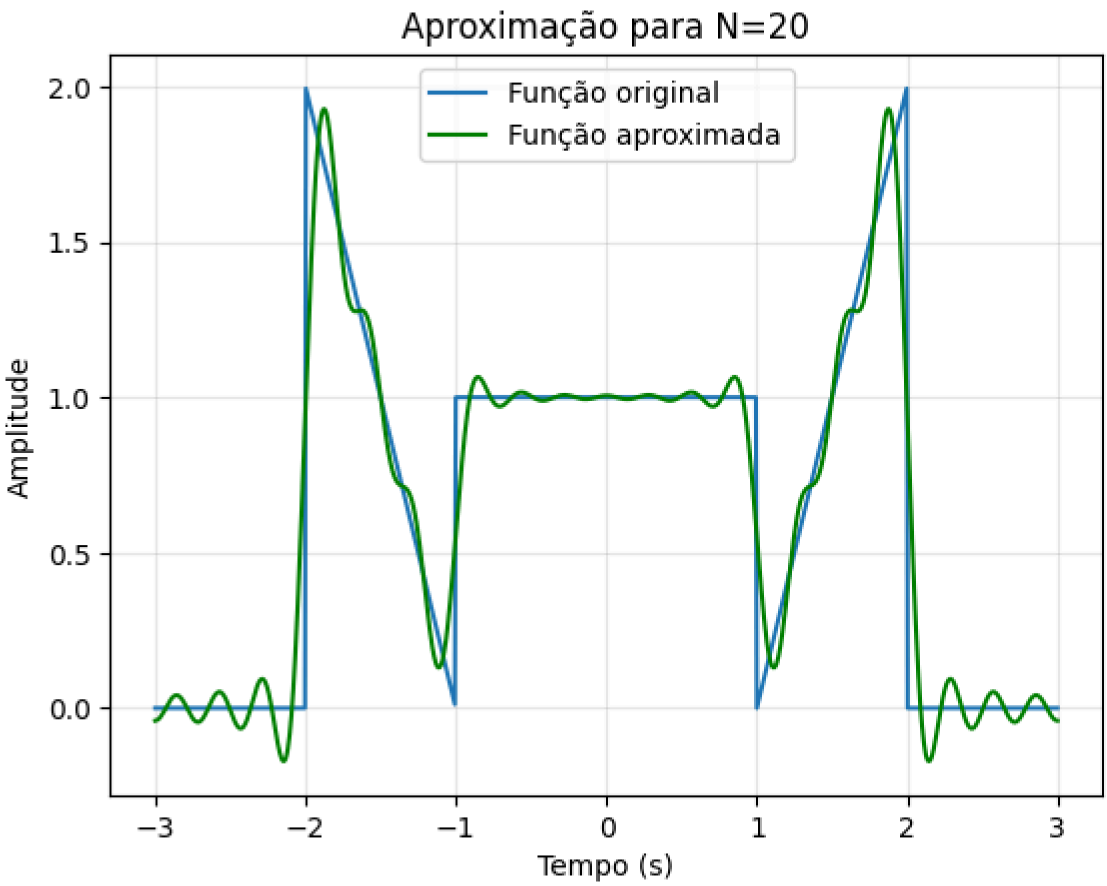
```
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
```

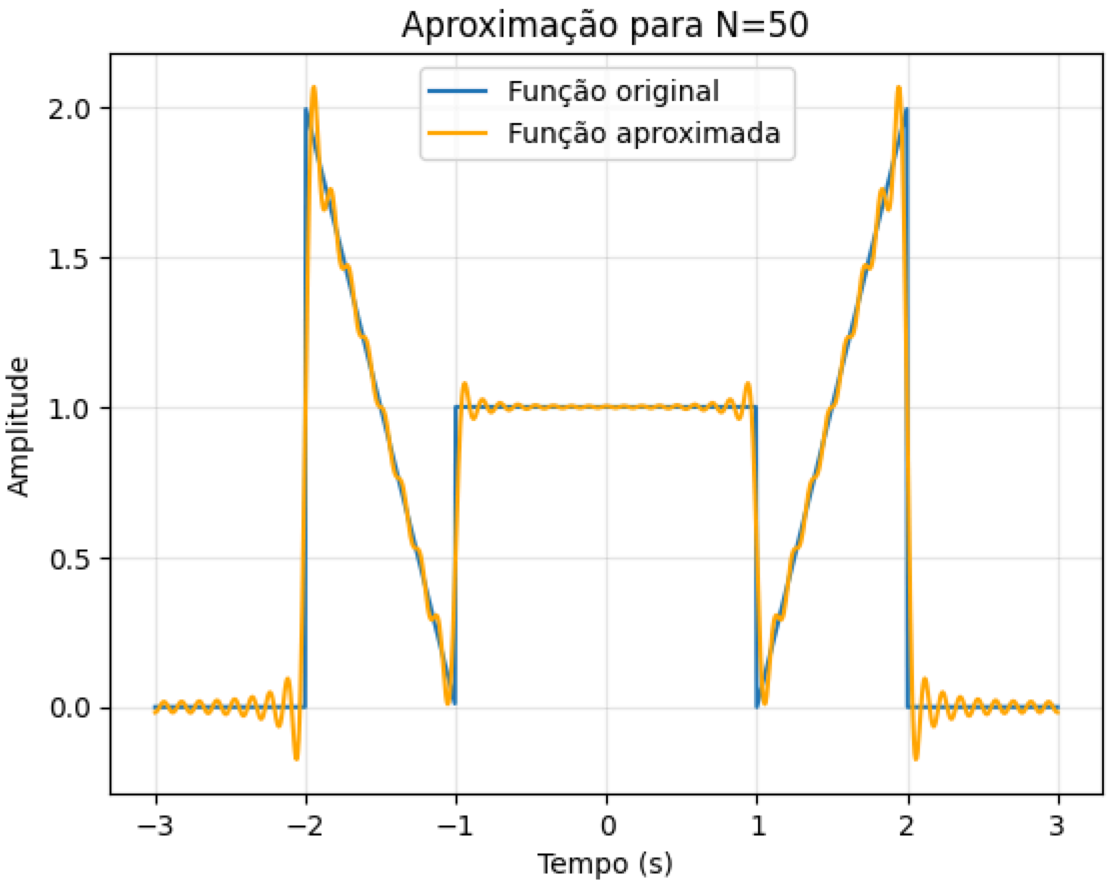
Plotando os gráficos

In [16]:
```
plot_comparacao(1, "purple")
plot_comparacao(10, "red")
plot_comparacao(20, "green")
plot_comparacao(50, "orange")
```




Aproximação para N=10
Função original
Função aproximada
Amplitude
Tempo (s)
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3


Aproximação para N=20
Função original
Função aproximada
Amplitude
Tempo (s)
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3



Item C: Calculando a potência média do erro

Fazendo uma função que retorna o valor da função x(t) original, dado um t específico

In [17]:
```
def x_original(t: float):
    if (-2 <= t and t < -1):
        valor = -2 - 2*t
    elif (-1 <= t and t < 1):
        valor = 1
    elif (1 <= t and t < 2):
        valor = -2 + 2*t
    else:
        valor = 0
    return valor
```

Fazendo uma função que retorna o valor da função x'(t) aproximada, dado um t específico. Apenas chama a função serie_fourier(n, t) pois já está implementada anteriormente.

In [18]:
```
def x_aproximado(n: int, t: float):
    return np.real(serie_fourier(n, t))
```

Fazendo uma função que retorna o valor da potência média do erro, dado um N

In [19]:
```
def potencia_erro_aproximacao(n: int):
    def integrante(t):
```

```
        return (x_original(t) - x_aproximado(n, t))**2

    intervalos = [(-3, -2), (-2, -1), (-1, 1), (1, 2), (2, 3)]

    resultado_total = 0
    for a, b in intervalos:
        resultado, erro = sp.integrate.quad(integrante, a, b)
        resultado_total += resultado

    potencia_erro = (1/6) * resultado_total
    return potencia_erro
```

Printando as potências dos erros para diferentes N

In [20]:
```
print(f"Potência média do erro (N=01): {potencia_erro_aproximacao(1)}")
print(f"Potência média do erro (N=10): {potencia_erro_aproximacao(10)}")
print(f"Potência média do erro (N=20): {potencia_erro_aproximacao(20)}")
print(f"Potência média do erro (N=50): {potencia_erro_aproximacao(50)}")
```

```
Potência média do erro (N=01): 0.23735329719741194
Potência média do erro (N=10): 0.05466854453254392
Potência média do erro (N=20): 0.02413836816738945
Potência média do erro (N=50): 0.009934094228806103
```

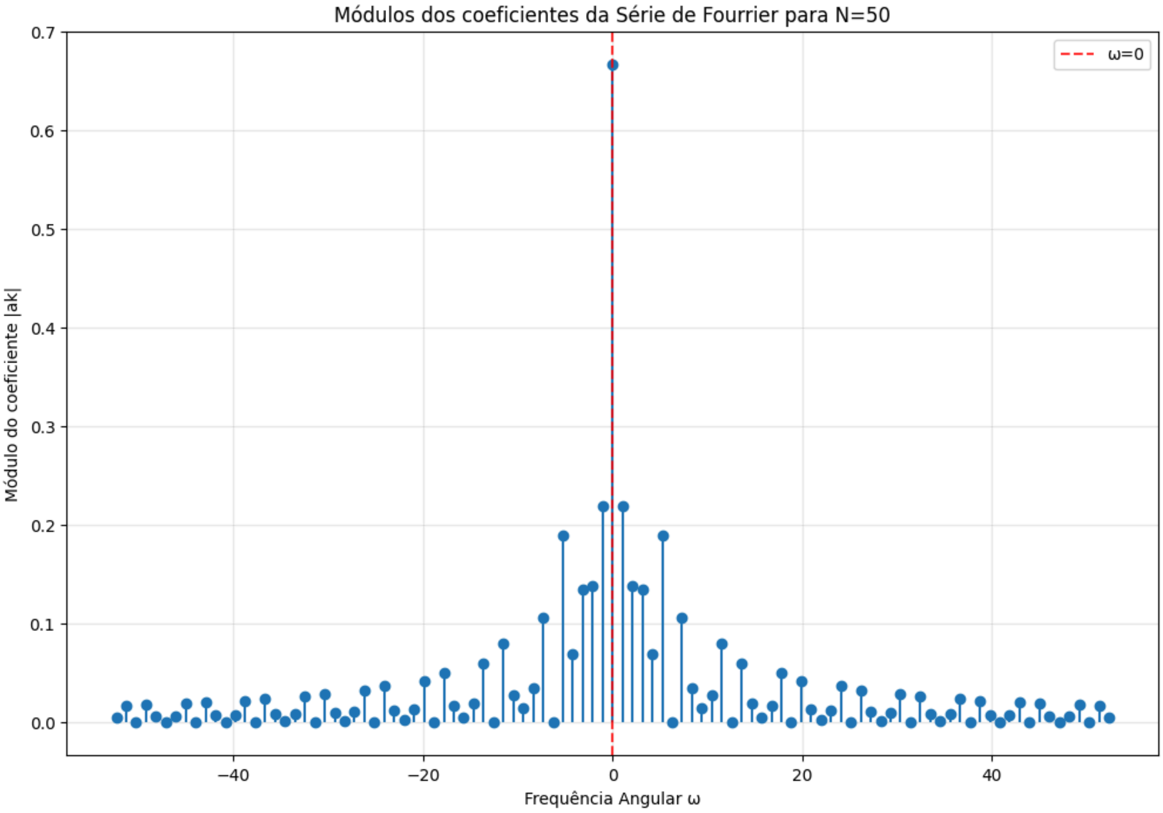
Item D: Exibindo o módulo dos coeficientes da série em função de ω

Fazendo uma função que plota o gráfico do módulod dos coeficientes da série de Fourier em função de ω

In [21]:
```
def plot_modulo_coeficientes(n: int):
    coeficientes = coeficientes_serie_fourier(n)

    modulo_coeficientes = []
    for coeficiente in coeficientes:
        modulo_coeficientes.append(abs(coeficiente))

    valores_k = np.arange(-n, n+1)
    valores_omega = valores_k * pi/3

    plt.figure(figsize=(12, 8))
    plt.stem(valores_omega, modulo_coeficientes, basefmt=" ")
    plt.xlabel("Frequência Angular ω")
    plt.ylabel("Módulo do coeficiente |ak|")
    plt.title(f"Módulos dos coeficientes da Série de Fourrier para N={n}"
    plt.grid(True, alpha=0.3)

    plt.axvline(x=0, color="red", linestyle="--", alpha=0.8, label="ω=0")
    plt.legend()

    plt.show()
```

Plotando o gráfico do módulo dos coeficientes

In [22]:
```
plot_modulo_coeficientes(50)
```



Discussão da simetria:

Pela análise do gráfico acima, nota-se uma simetria de refexão no eixo ω = 0 (pontilhado em vermelho).
Essa simetria é prevista uma vez que a função x(t) é uma função real e, portanto, satisfaz a relação: $a_{-k} = a_k^*$

Assim: $|a_{-k}| = |a_k^*| = |a_k|$

Item E: Plotando o módulo e a fase da resposta em frequência H(jω) em função de ω e discutindo sua ação de filtro

Fazendo uma função que retorna o valor de H(jω) seguindo a formula proposta e as informações de R e C dadas na figura

In [23]:
```
def resposta_frequencia(valores_omega):
    r = 100e3
    c = 1e-6
    omega_c = 1/(r*c)

    vetor = []

    for omega in valores_omega:
        valor = omega / (omega - (1j * omega_c))
        vetor.append(valor)

    return vetor
```

Fazendo uma função que plota o módulo da resposta em frequência H(jω)

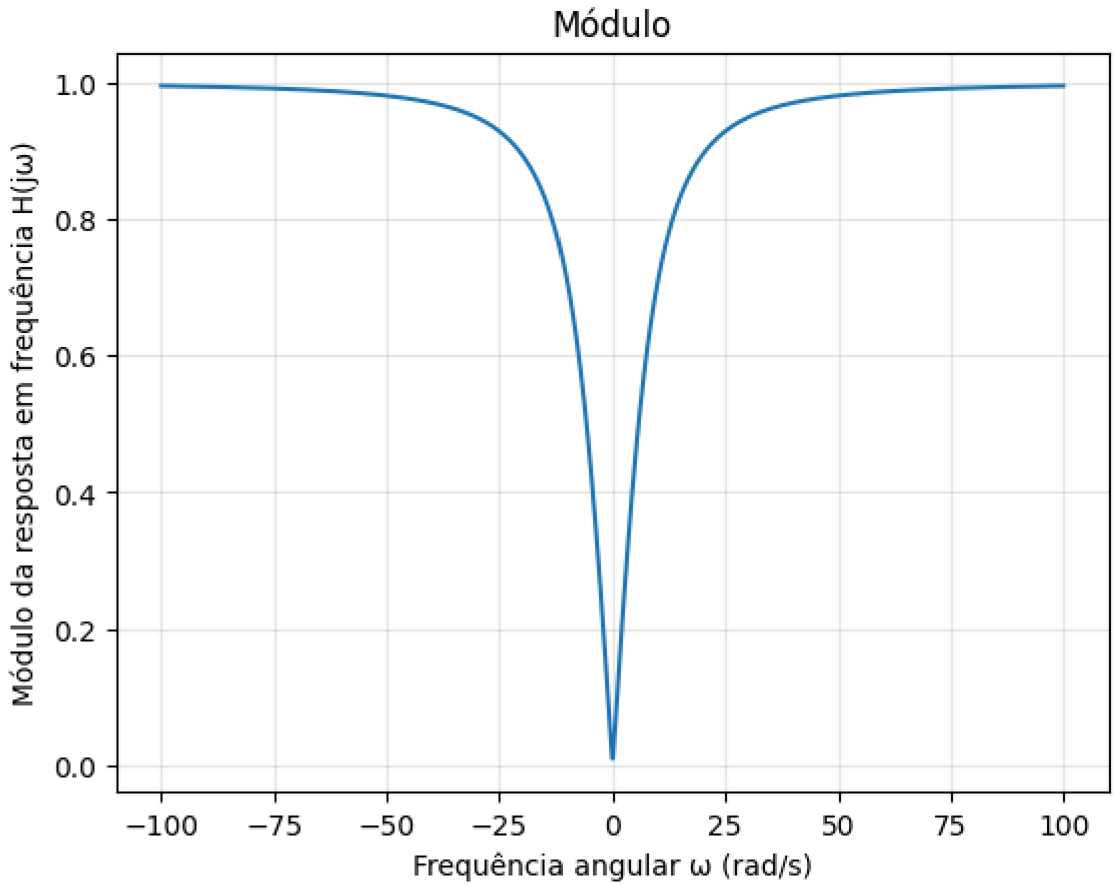
In [24]:
```
def plot_modulo_resposta():
    valores_omega = np.linspace(-100, 100, 1000)

    valores_h = resposta_frequencia(valores_omega)

    valores_mod_h = np.abs(valores_h)

    plt.plot(valores_omega, valores_mod_h)
    plt.xlabel("Frequência angular ω (rad/s)")
    plt.ylabel("Módulo da resposta em frequência H(jω)")
    plt.title(f"Módulo")
    plt.grid(True, alpha=0.3)
    plt.show()
```

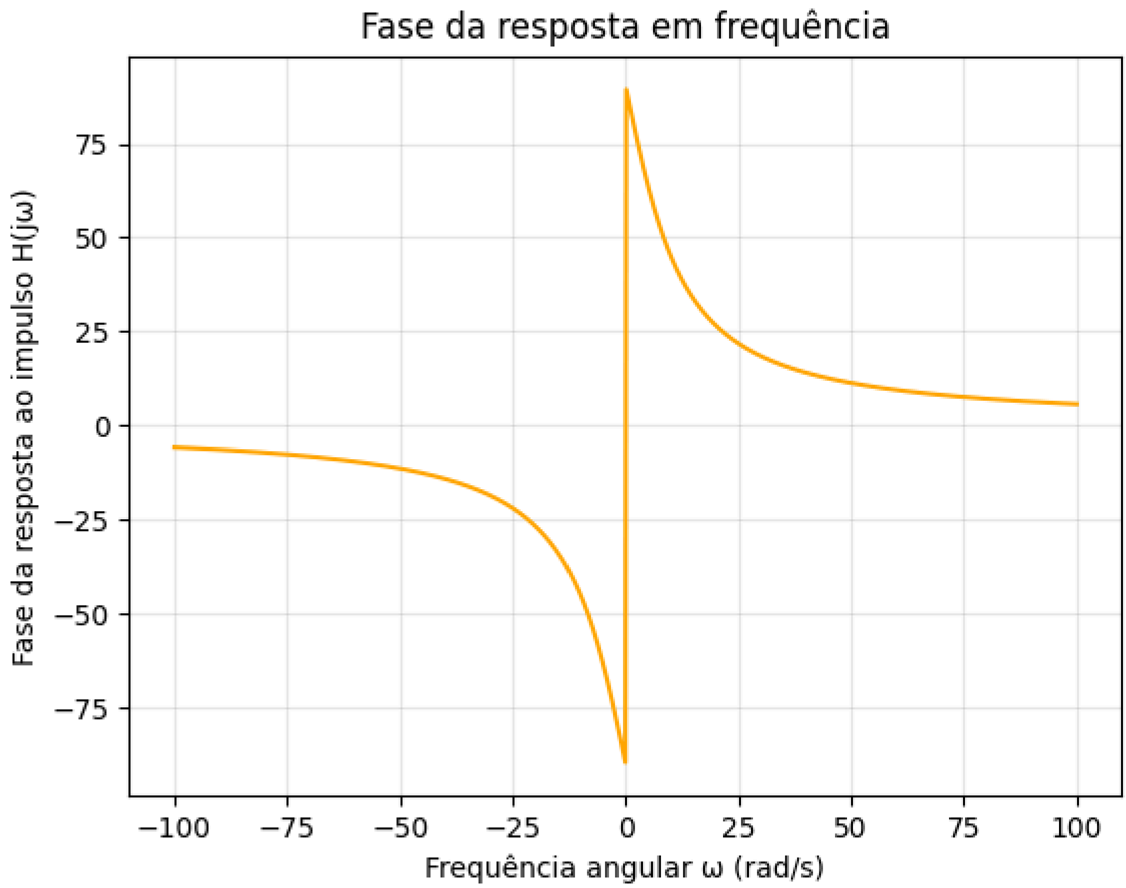
Fazendo uma função que plota a fase da resposta em frequência H(jω)

In [25]:
```
def plot_fase_resposta():
    valores_omega = np.linspace(-100, 100, 1000)

    valores_h = resposta_frequencia(valores_omega)

    valores_fase_h = np.angle(valores_h) * 180 / pi

    plt.plot(valores_omega, valores_fase_h, color="orange")
    plt.xlabel("Frequência angular ω (rad/s)")
    plt.ylabel("Fase da resposta ao impulso H(jω)")
    plt.title(f"Fase da resposta em frequência")
    plt.grid(True, alpha=0.3)
    plt.show()
```

Plotando os gráficos do módulo e da fase da resposta ao impulso H(jω)

In [26]:
```
plot_modulo_resposta()
plot_fase_resposta()
```





Discussão da ação de filtro:

Analisando o módulo, uma vez que os valores próximos da origem se aproximam de zero, pode-se deduzir que o filtro é um passa altas e, além disso, como essa aproximação se dá de modo suave e não abrupta, sabe-se também que é um filtro não ideal.

Analisando a fase, como ela não varia de maneira linear com relação à frequência, sabe-se que a forma do sinal será modificada.

Item F: Plotando a saída y(t) resultante da passagem da onda x(t) (aproximada em N=50) pelo filtro H(jω)

Fazendo uma função que calcula os diferentes valores de ω com base no $\omega_o = \pi/3$

In [27]:
```
def calcular_valores_omega(n: int):
    vetor = []
    for k in range(-n, n+1):
        valor = k * (pi/3)
        vetor.append(valor)
    return vetor
```

Fazendo uma função que calcula o valor, em um tempo específico t, resultante da passagem da onda x(t) (aproximada em N) no filtro H(jω)

In [28]:
```
def saida_filtro(n: int, t: float):
    valor = 0

    coeficientes = coeficientes_serie_fourier(n)

    valores_omega = calcular_valores_omega(n)
    respostas = resposta_frequencia(valores_omega)

    for k in range(-n, n+1):
        exponencial = np.exp(1j*k*(pi/3)*t)
        valor += coeficientes[k+n] * respostas[k+n] * exponencial
    return np.real(valor)
```

Fazendo uma função que calcula a saída y(t) aproximada em N

In [29]:
```
def funcao_saida_filtro(n: int):
    vetor = []

    valores_t = np.linspace(-3, 3, 1000)
    for t in valores_t:
        valor = saida_filtro(n, t)
        vetor.append(valor)
    return vetor
```

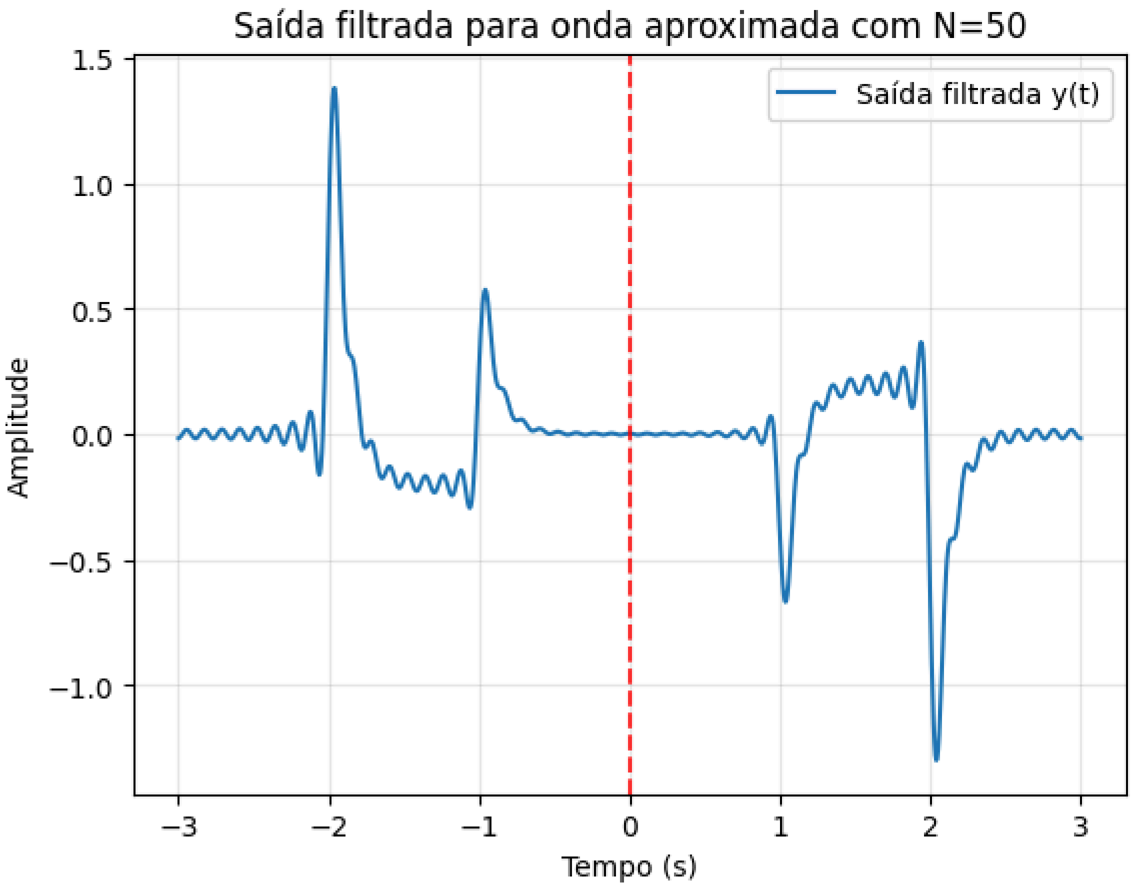
Fazendo uma função que plota a saída y(t) em um gráfico

In [30]:
```
def plot_saida_filtro(n: int):
    saida = funcao_saida_filtro(n)

    valores_t = np.linspace(-3, 3, 1000)
```

```
    plt.plot(valores_t, saida, label="Saída filtrada y(t)")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Amplitude")
    plt.title(f"Saída filtrada para onda aproximada com N={n}")
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.axvline(x=0, color="red", linestyle="--", alpha=0.8, label="t=0")

    plt.show()
```

Plotando a saída y(t)

In [31]:
```
plot_saida_filtro(50)
```



Comentários sobre a forma da onda y(t):

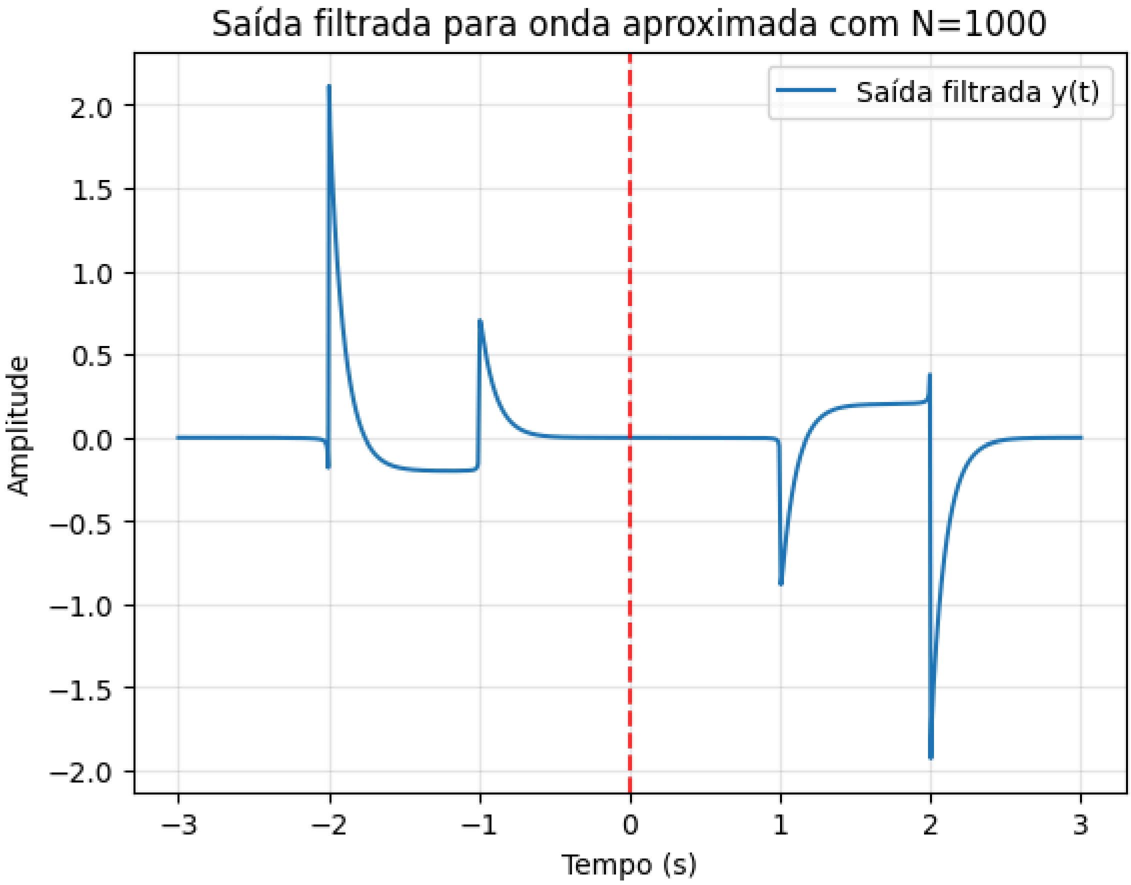
Nota-se que o filtro realizou uma inversão da paridade da onda, uma vez que x(t) possui, notavelmente, paridade par, mas y(t), conforme visível no gráfico acima, apresenta paridade ímpar.

Nota-se também que as amplitudes máximas e mínimas estão um pouco menores.

Item G: Explicando as diferenças entre o gráfico gerado pela aproximação e o gráfico exato

Plotando uma saída y(t) resultante da passagem da onda x(t) (aproximada em N=1000) pelo filtro H(jω)

In [32]:
```
plot_saida_filtro(1000)
```



É notável que a com um N bem maior que 50, como no exemplo acima, a saída gerada se aproxima cada vez mais à saída exata representada na Figura 3, apresentando mesma amplitude e curvas presentes nas mesmas posições.

Deste modo, caso N aproxime-se do infinito, a saída gerada será quase que perfeitamente idêntica à saída exata, com exeção apenas do efeito de Gibbs que se manterá presente nos pontos de inflexão.