Lab1

February 20, 2021

[167]:
```python
import matplotlib.pyplot as plt
import numpy as np
from numpy import cos, sin, arctan2, sqrt
```

[168]:
```python
from IPython.core.pylabtools import figsize
figsize(9, 3)

from IPython.core.display import HTML
HTML("""
<style>
.output_png {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
</style>
""")
```

[168]:
```
<IPython.core.display.HTML object>
```

0.0.1 Representação de funções periódicas via séries de Fourier

Seja a função $f(t)$ periódica com período P, isto é $f(t) = f(t+nP)$ para $n = 1, 2, 3, ...$, e absolutamente integrável num intervalo de comprimento P. Considerando que a função $f(t)$ admite representação em termos de uma série trigonométrica de Fourier, $f(t)$ pode ser escrita como

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{n\pi t}{T}\right) + b_n \sin\left(\frac{n\pi t}{T}\right)\right]$$

em que a_n e b_n são os coeficientes de Fourier da expansão de $f(t)$, calculados de acordo com as expressões

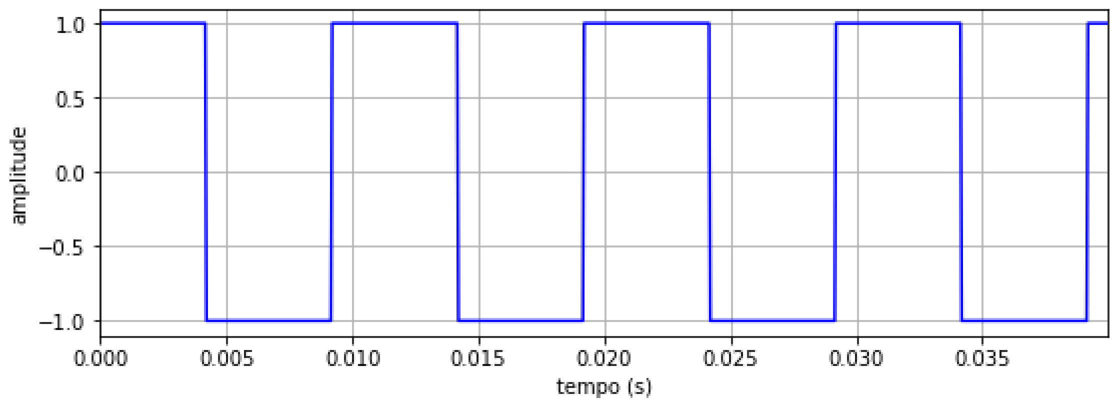
$$a_n = \frac{2}{P}\int_P f(t)\cos\left(2\pi t\frac{n}{P}\right)dt b_n = \frac{2}{P}\int_P f(t)\sin\left(2\pi t\frac{n}{P}\right)dt.$$

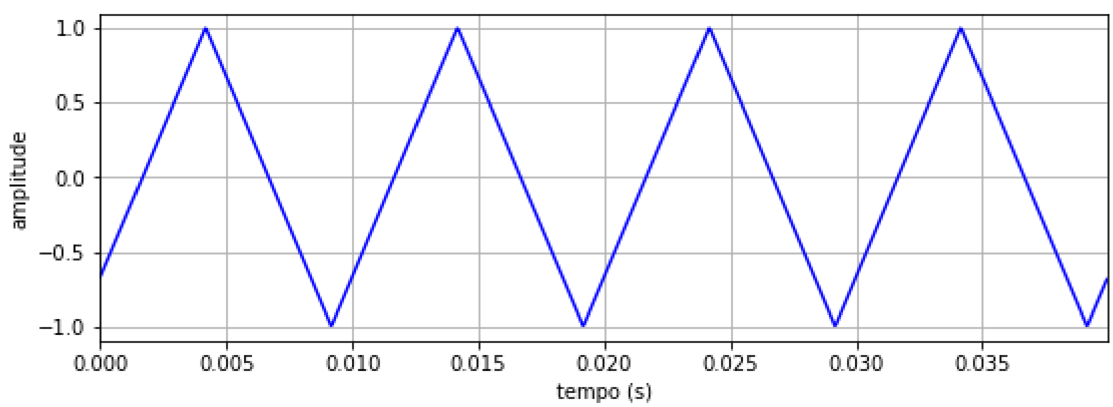
Em particular, temos que

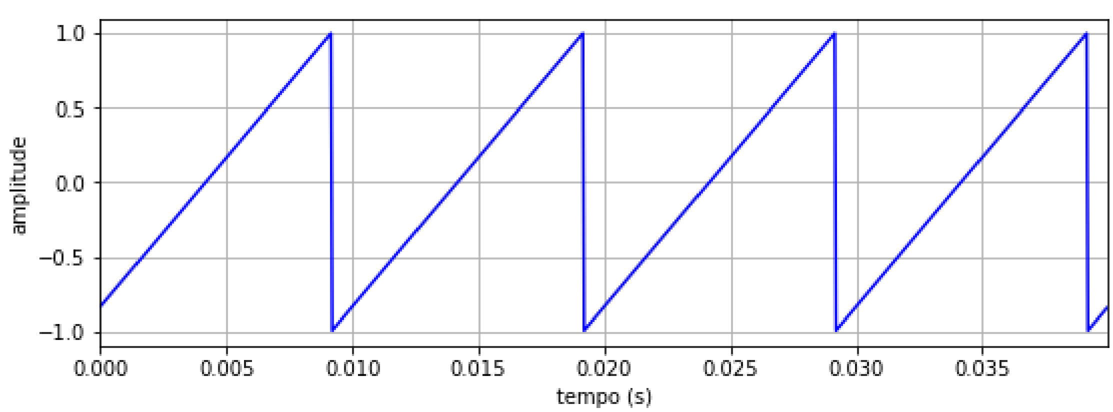
$$
\begin{aligned}
a_0 &= \frac{2}{P}\int_P f(t)\cos(0)dt = \frac{2}{P}\int_P f(t)dt \qquad (1)\\
b_0 &= \frac{2}{P}\int_P f(t)\sin(0)dt = 0. \qquad (2)
\end{aligned}
$$

[169]:

```
from scipy.signal import square, sawtooth

f0 = 100     # Frequência fundamental
fa = 200*f0   # Frequência de amostragem

t    = np.arange(0, 40e-3, 1/fa)
    = np.pi
    = /6

quad   = square(2**f0*t + )                 # onda quadrada com frequência␣
 ↪fundamental f0 e phase inicial
triang = sawtooth(2**f0*t + , width = 0.5) # onda triangular com frequência␣
 ↪fundamental f0 e phase inicial
dente  = sawtooth(2**f0*t + )               # onda dente de serra com frequência␣
 ↪fundamental f0 e phase inicial

plt.figure()
plt.plot(t, quad,'b')
plt.ylim(quad.min(0)-0.1, quad.max(0)+0.1)
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.grid()

plt.figure()
plt.plot(t, triang,'b')
plt.ylim(triang.min(0)-0.1, triang.max(0)+0.1)
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.grid()

plt.figure()
plt.plot(t, dente,'b')
plt.ylim(dente.min(0)-0.1, dente.max(0)+0.1)
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.grid()
```


1.0
0.5
0.0
−0.5
−1.0
amplitude
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
tempo (s)


1.0
0.5
0.0
−0.5
−1.0
amplitude
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
tempo (s)


1.0
0.5
0.0
−0.5
−1.0
amplitude
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
tempo (s)

0.0.2 Cálculo dos coeficientes de Fourier via integração numérica

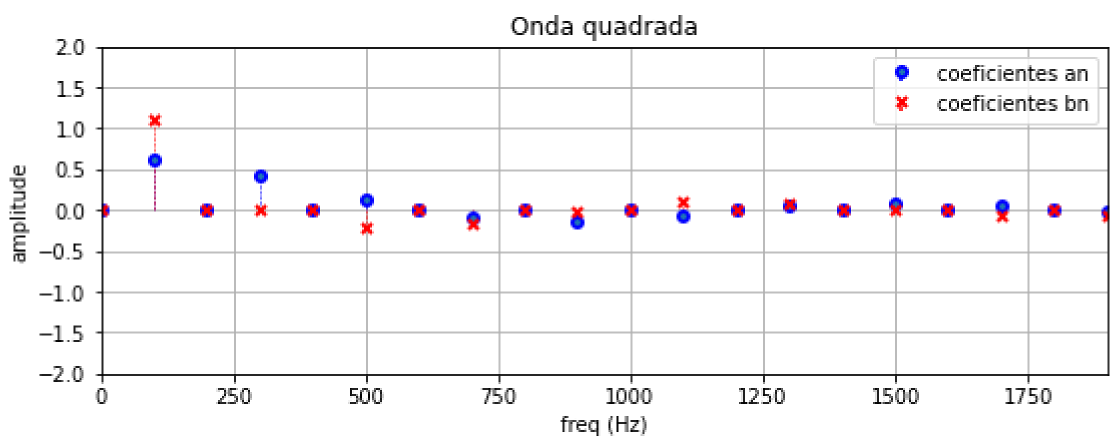
Primeiramente, vamos definir uma função para calcular numericamente os coeficientes a_n e b_n da série de Fourier para uma função periódica $f(t)$ qualquer.

[170]:
```
def fourierCoeff(t,f,P,n):
    """
     t  : vetor de instantes de tempo contendo pelo menos um período completo da␣
 ↪função [segundos]
     f  : vetor de valores de f(t) calculados para cada instante em t
     P  : período fundamental de f [segundos]
     n  : ordem do coeficiente de Fourier desejado [número inteiro]
     an : coeficiente an
     bn : coeficiente bn

    """
    dt = t[1]-t[0]      # período de amostragem [passo de integração]
    N  = np.ceil(P/dt) # número de amostras correspondente a um período completo␣
 ↪da onda
    an = 0
    bn = 0

    # integração numérica dos coeficientes an e bn
    for ind in range(0, int(N)):
        an = an + f[ind]*cos(2*np.pi*t[ind]*n/P)*dt
        bn = bn + f[ind]*sin(2*np.pi*t[ind]*n/P)*dt

    an = an*2/P
    bn = bn*2/P

    return an, bn
```

0.0.3 Aproximação via série de Fourier para a onda quadrada

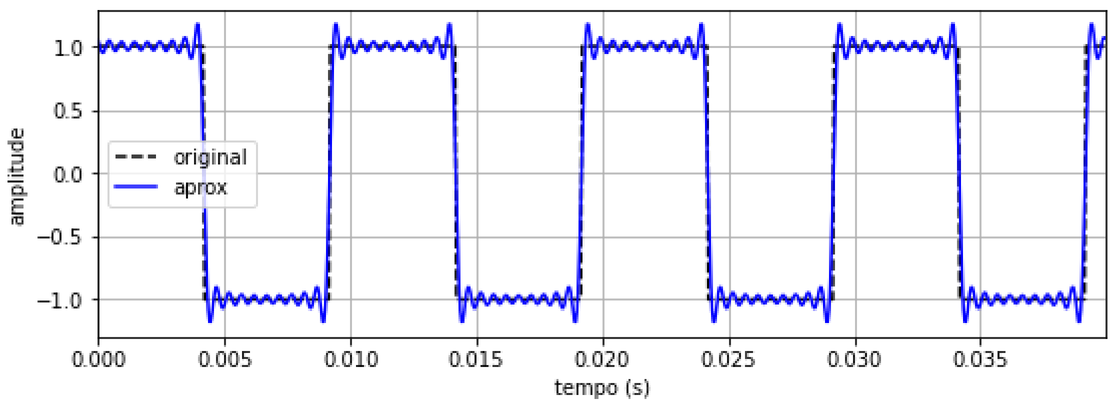
[171]:
```
# Coeficientes de Fourier da onda quadrada

ncoeffs = 20  # número de componentes harmônicos (incluindo componente dc, n=0)

quad_an = np.zeros((ncoeffs, 1));
quad_bn = np.zeros((ncoeffs, 1));
xf      = np.zeros((ncoeffs, 1));

for n in range(0,ncoeffs): # calcula coeficientes de Fourier para n=0 até␣
 ↪n=ncoeffs-1
    quad_an[n], quad_bn[n] = fourierCoeff(t, quad , 1/f0, n)
    xf[n] = n*f0
```

```python
plt.figure()
(markers, stemlines, baseline) = plt.stem(xf, quad_an,'b', use_line_collection=True,label = 'coeficientes an')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='o', markersize=5, markeredgecolor="b", markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="b", linewidth=0.5 )

(markers, stemlines, baseline) = plt.stem(xf, quad_bn,'r', use_line_collection=True,label = 'coeficientes bn')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='x', markersize=5, markeredgecolor="r", markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="r", linewidth=0.5 )

plt.title('Onda quadrada')
plt.legend()
plt.xlim(0,xf.max(0))
plt.ylim(-2,2)
plt.xlabel('freq (Hz)')
plt.ylabel('amplitude')
plt.grid()
```



[172]:
```python
# Aproximação da onda quadrada via somatório de harmônicas da série de Fourier

ncoeff = 20 # número de componentes harmônicos (incluindo componente dc, n=0)

quad_aprox = 0
for n in range(0,ncoeff):
    if n != 0:
        quad_aprox = quad_aprox + quad_an[xf==n*f0]*cos(2**n*f0*t)\
```

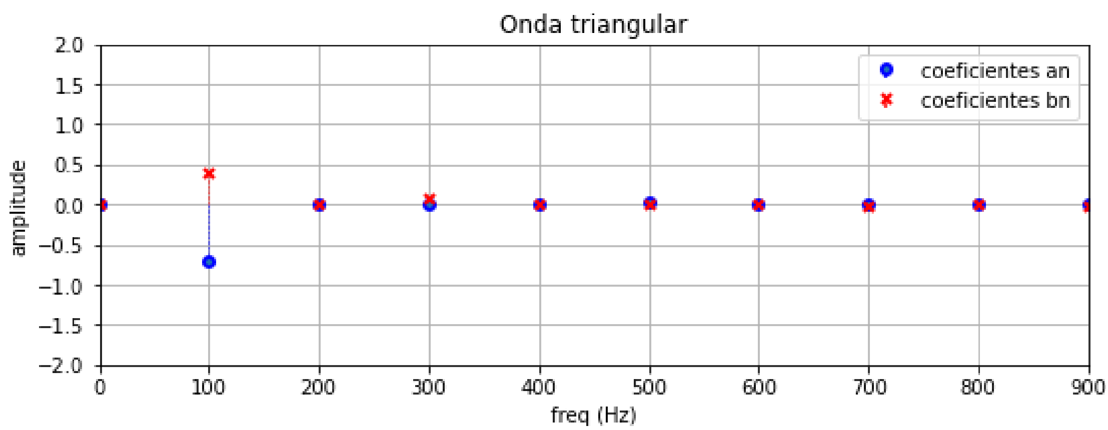
```
                                + quad_bn[xf==n*f0]*sin(2**n*f0*t) # soma␣
 ↪n-ésimo componente harmônico
    else:
        quad_aprox = quad_an[xf==n*f0]/2

plt.plot(t, quad,'k--',label = 'original')
plt.plot(t,quad_aprox,'b',label = 'aprox')
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.legend()
plt.grid()
```



0.0.4 Aproximação via série de Fourier para a onda triangular

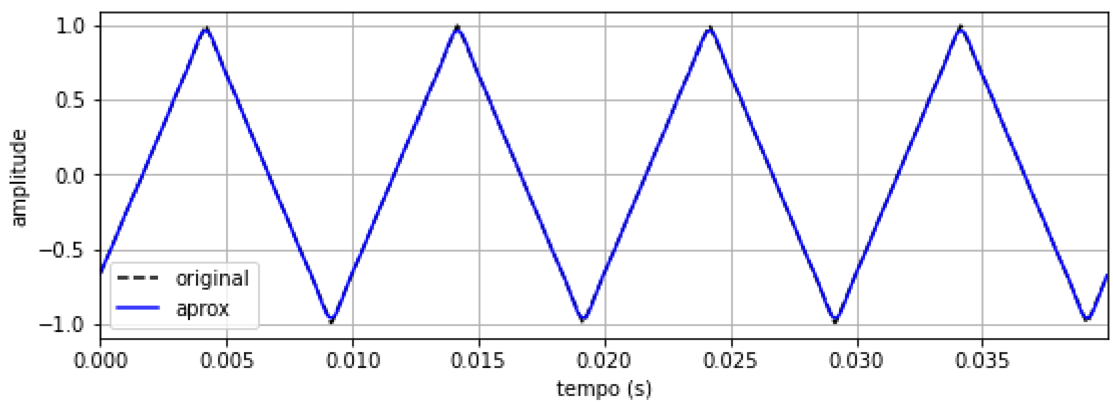
[173]:
```
# Coeficientes de Fourier da onda triangular

ncoeffs = 10 # número de componentes harmônicos (incluindo componente dc, n=0)

triang_an = np.zeros((ncoeffs, 1));
triang_bn = np.zeros((ncoeffs, 1));
xf        = np.zeros((ncoeffs, 1));

for n in range(0,ncoeffs): # calcula coeficientes de Fourier para n=0 até␣
 ↪n=ncoeffs-1
    triang_an[n], triang_bn[n] = fourierCoeff(t, triang , 1/f0, n)
    xf[n] = n*f0

plt.figure()
(markers, stemlines, baseline) = plt.stem(xf, triang_an,'b',␣
 ↪use_line_collection=True,label = 'coeficientes an')
plt.setp(baseline, visible=False)
```

```
plt.setp(markers, marker='o', markersize=5, markeredgecolor="b",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="b", linewidth=0.5 )

(markers, stemlines, baseline) = plt.stem(xf, triang_bn,'r',␣
 ↪use_line_collection=True,label = 'coeficientes bn')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='x', markersize=5, markeredgecolor="r",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="r", linewidth=0.5 )

plt.title('Onda triangular')
plt.legend()
plt.xlim(0,xf.max(0))
plt.ylim(-2,2)
plt.xlabel('freq (Hz)')
plt.ylabel('amplitude')
plt.grid()
```



[174]:
```
# Aproximação da onda triangular via somatório de harmônicas da série de Fourier

ncoeff = 10 # número de componentes harmônicos (incluindo componente dc, n=0)

triang_aprox = 0
for n in range(0,ncoeff):
    if n != 0:
        triang_aprox = triang_aprox + triang_an[xf==n*f0]*cos(2**n*f0*t)\
                                    + triang_bn[xf==n*f0]*sin(2**n*f0*t) # soma␣
 ↪n-ésimo componente harmônico
    else:
        triang_aprox = triang_an[xf==n*f0]/2
```

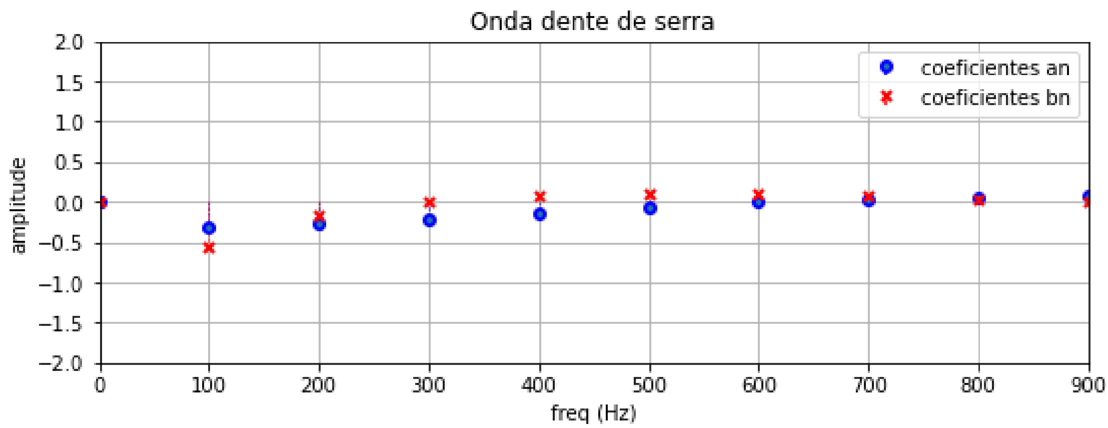
```
plt.plot(t, triang,'k--',label = 'original')
plt.plot(t,triang_aprox,'b',label = 'aprox')
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.legend()
plt.grid()
```



0.0.5 Aproximação via série de Fourier para a onda dente de serra

[175]:
```
# Coeficientes de Fourier da onda dente de serra

ncoeffs = 10 # número de componentes harmônicos (incluindo componente dc, n=0)

dente_an = np.zeros((ncoeffs, 1));
dente_bn = np.zeros((ncoeffs, 1));
xf       = np.zeros((ncoeffs, 1));

for n in range(0,ncoeffs): # calcula coeficientes de Fourier para n=0 até␣
 ↪n=ncoeffs-1
    dente_an[n], dente_bn[n] = fourierCoeff(t, dente, 1/f0, n)
    xf[n] = n*f0

plt.figure()
(markers, stemlines, baseline) = plt.stem(xf, dente_an,'b',␣
 ↪use_line_collection=True,label = 'coeficientes an')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='o', markersize=5, markeredgecolor="b",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="b", linewidth=0.5 )
```

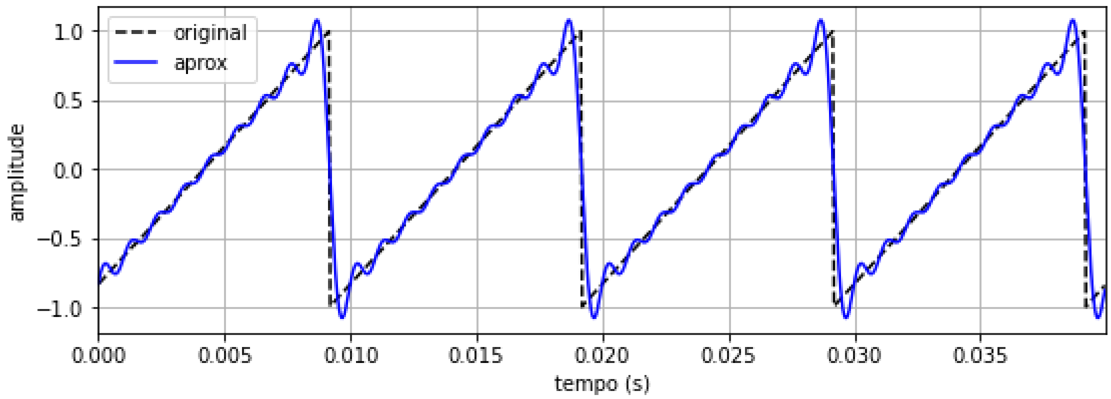
```
(markers, stemlines, baseline) = plt.stem(xf, dente_bn,'r',␣
 ↪use_line_collection=True,label = 'coeficientes bn')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='x', markersize=5, markeredgecolor="r",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="r", linewidth=0.5 )

plt.title('Onda dente de serra')
plt.legend()
plt.xlim(0,xf.max(0))
plt.ylim(-2,2)
plt.xlabel('freq (Hz)')
plt.ylabel('amplitude')
plt.grid()
```



[176]:

```
# Aproximação da onda dente de serra via somatório de harmônicas da série de␣
 ↪Fourier

ncoeff = 10 # número de componentes harmônicos (incluindo componente dc, n=0)

dente_aprox = 0
for n in range(0,ncoeff):
    if n != 0:
        dente_aprox = dente_aprox + dente_an[xf==n*f0]*cos(2**n*f0*t)\
                                  + dente_bn[xf==n*f0]*sin(2**n*f0*t) # soma␣
 ↪n-ésimo componente harmônico
    else:
        dente_aprox = dente_an[xf==n*f0]/2

plt.plot(t, dente,'k--',label = 'original')
plt.plot(t, dente_aprox,'b',label = 'aprox')
```

```
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.legend()
plt.grid()
```



0.0.6 Reescrevendo a série de Fourier na sua forma harmônica

Podemos expressar a série de Fourier de maneira simplificada utilizando a sua forma harmônica, utilizando apenas a função cosseno para representar os componentes harmônicos da função.

$$f(t) = A_0 + \sum_{n=1}^{\infty} A_n \cdot \cos\left(2\pi t \frac{n}{P} - \theta_n\right)$$

em que

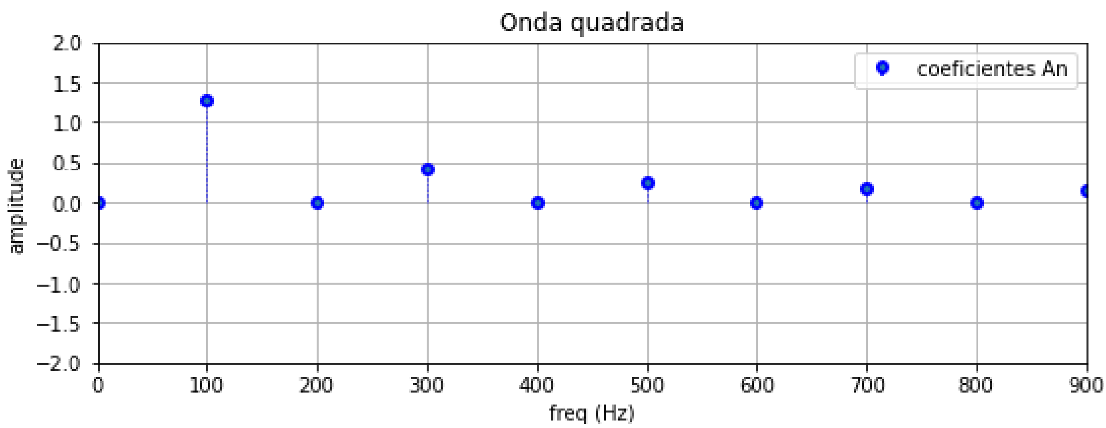
$$A_0 = \frac{a_0}{2},\ A_n = \sqrt{a_n^2 + b_n^2} \text{ para } n \geq 1,\ \theta_n = \arctan\left(\frac{b_n}{a_n}\right)$$

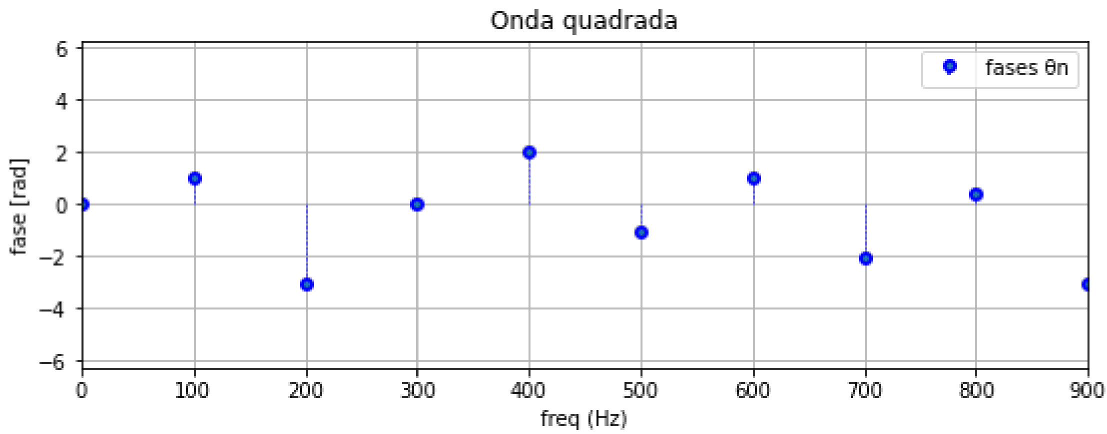
Esta representação é interessante porque nos permite determinar diretamente a energia associada a cada componente de frequência do sinal.

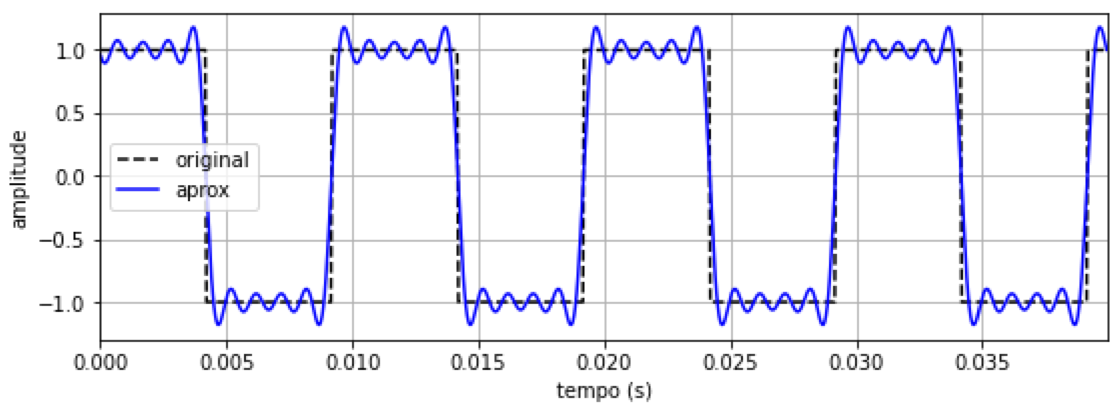
[180]:
```
# Aproximação da onda quadrada via somatório de harmônicas da série de Fourier

ncoeffs = 10 # número de componentes harmônicos (incluindo componente dc, n=0)

quad_an = np.zeros((ncoeffs, 1))
quad_bn = np.zeros((ncoeffs, 1))
xf      = np.zeros((ncoeffs, 1))
An      = np.zeros((ncoeffs, 1))
n      = np.zeros((ncoeffs, 1))
```

```
for n in range(0,ncoeffs): # calcula coeficientes de Fourier para n=0 até␣
 ↪n=ncoeffs-1
    quad_an[n], quad_bn[n] = fourierCoeff(t, quad , 1/f0, n)
    xf[n] = n*f0

for n in range(0,ncoeffs):
    if n != 0:
        An[n] = sqrt(quad_an[n]**2 + quad_bn[n]**2)
        n[n] = arctan2(quad_bn[n], quad_an[n])      # calcula arctan(bn/an)     ␣
 ↪
    else:
        An[n] = quad_an[n]/2


plt.figure()
(markers, stemlines, baseline) = plt.stem(xf, An,'b',␣
 ↪use_line_collection=True,label = 'coeficientes An')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='o', markersize=5, markeredgecolor="b",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="b", linewidth=0.5 )

plt.title('Onda quadrada')
plt.legend()
plt.xlim(0,xf.max(0))
plt.ylim(-2,2)
plt.xlabel('freq (Hz)')
plt.ylabel('amplitude')
plt.grid()

plt.figure()
(markers, stemlines, baseline) = plt.stem(xf, n,'b',␣
 ↪use_line_collection=True,label = 'fases n')
plt.setp(baseline, visible=False)
plt.setp(markers, marker='o', markersize=5, markeredgecolor="b",␣
 ↪markeredgewidth=2)
plt.setp(stemlines, linestyle="--", color="b", linewidth=0.5 )

plt.title('Onda quadrada')
plt.legend()
plt.xlim(0,xf.max(0))
plt.ylim(-2* ,2*)
plt.xlabel('freq (Hz)')
plt.ylabel('fase [rad]')
plt.grid()
```

```
quad_aprox = 0

for n in range(0,ncoeffs):
    if n != 0:
        quad_aprox = quad_aprox + An[n]*cos(2**n*f0*t - n[n]) # soma n-ésimo␣
 ↪componente harmônico

    else:
        quad_aprox = An[n]


plt.figure()
plt.plot(t, quad,'k--',label = 'original')
plt.plot(t,quad_aprox,'b',label = 'aprox')
plt.xlim(0,t.max(0))
plt.xlabel('tempo (s)')
plt.ylabel('amplitude')
plt.legend()
plt.grid()
```







0.1 Distorção

O conceito de distorção aparece no contexto das disciplinas relacionadas à engenharia elétrica, particulamente nas áreas de processamento de sinais e comunicações. Diz-se que um sinal sofreu distorção quando o mesmo teve a sua forma original modificada, excetuando-se os casos em que tal modificação corresponde a uma mudança de escala de amplitude, ou a um atraso no tempo, que obviamente não alteram a forma de nenhum sinal. Distorções podem ser classificadas em dois tipos básicos:

Distorções lineares: causadas por operações lineares sofridas pelo sinal.

Distorções não-lineares: causadas por operações não-lineares sofridas pelo sinal.

0.1.1 Quando um sistema linear e invariante no tempo (LIT) não causará distorção?

As duas operações lineares que não distorcem um sinal estão resumidas em

$$y(t) = Gx(t-\tau). \tag{1}$$

Podemos considerar $y(t)$ (1) como sendo a saída de um sistema LIT, cuja resposta ao impulso é $h(t) = G\delta(t-\tau)$, quando o sinal $x(t)$ é aplicado na entrada. Analisando (1) no domínio da frequência, temos

$$Y(f) = Ge^{-j2\pi f\tau}X(f) \tag{3}$$

$$\tag{4}$$

$$Y(f) = H(f)X(f), \text{ com } H(f) = Ge^{-j2\pi f\tau}. \tag{2}$$

Note que a resposta em amplitude de $H(f)$ é constante $(|H(f)| = G)$ e a resposta em fase $(\angle H(f) = -2\pi f\tau)$ é uma função linear de f. Portanto, a menos que o sistema LTI possua resposta em frequência na forma expressa em (0.1.1), $y(t)$ será uma versão distorcida de $x(t)$.

Verificação para o caso de sinais periódicos Considerando que o sinal $x(t)$ é periódico e admite representação em termos de uma série de Fourier, temos

$$x(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos\left(2\pi n f_0 t - \theta_n\right)$$

logo, de (1) temos que

$$y(t) = GA_0 + \sum_{n=1}^{\infty} GA_n \cos\left[2\pi n f_0 (t-\tau) - \theta_n\right]$$
$$y(t) = GA_0 + \sum_{n=1}^{\infty} GA_n \cos\left[2\pi n f_0 t - 2\pi n f_0 \tau - \theta_n\right]$$
$$y(t) = GA_0 + \sum_{n=1}^{\infty} GA_n \cos\left[2\pi n f_0 t - \theta_n + \Delta\theta\right]$$

Desse modo, perceba que para que o sinal $x(t)$ não sofra distorção, todos os seus componentes de frequência devem sofrer o mesmo ganho G (ou atenuação, se $|G| < 1$) e o desvio de fase $\Delta\theta$ gerado pelo atraso τ deve ser linear com relação à frequência $\Delta\theta = (-2\pi f_0 \tau)n$, o que está de acordo com (0.1.1)

```
[ ]:
```