Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e Computação

Disciplina: Sinais e Sistemas

Trabalho Computacional

Plotagem do resultado da convolução de sinais

Luís Filipe Silva Forti 14592348
Docente responsável: Prof. Marcos Rogério Fernandes

São Carlos
2º semestre / 2024

Sumário

Lista de Figuras

1. Introdução

Neste trabalho foi analizada como calcular a convolução entre duas funções quaisquer funcA e funcB e como animar o processo do cálculo. Foram analisadas algumas convoluções e sua propriedade de comutação.

2. Materiais e métodos

O projeto foi realizado em Jupyter Notebook por meio da linguagem Python. Os gráficos foram feitos com a biblioteca matplotlib.

Inicialmente, foi criada a função para processar a convolução. Ela requer duas funções e seus nomes, além de um vetor de valores de tempo onde deveria calcular. Assim ela pode criar os dois gráficos superiores, os quais mostram as funções dadas no intervalo.

Então, para calcular a convolução, foi feita uma análise da fórmula.

Figura 1: *Fórmula da convolução*

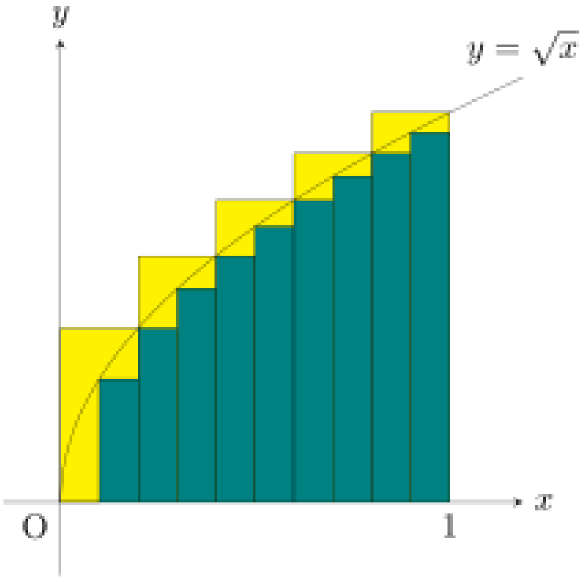
$$(f * g)(t) = h(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t-\tau)\, d\tau$$

Fonte: Wikipedia

Criou-se um loop para passar por cada instante de tempo do intervalo escolhido, onde esse seria o instante τ da convolução. Recalcula-se os valores de funcB, deslocando-a em relação ao t. Como τ está negativo para a funcB e deslocada, ela deve então estar espelhada em relação ao ponto t. Ao multiplicar os valores das duas funções, obtém-se os valores que serão utilizados para o cálculo da convolução total. Estes valores se tornaram o gráfico inferior direito das imagens, demonstrando os valores da convolução no instante τ.

A integral de uma função é definida como a área definida pela mesma no intervalo. Por esta definição, pode-se interpretar os valores calculados como a altura de retângulos e, se calculada a largura entre cada valor, pode-se encontrar a área dos mesmos.

Figura 2: *Aproximação da integral*



Fonte: Wikipedia

Se o vetor de tempo for uniformemente distribuído, então também serão os valores calculados. Assim, pode-se definir que a distância entre dois valores é dada pelo comprimento total de tempo dividido pela quantidade de valores. Assim pode-se calcular a integral da convolução no instante t.

$$\int_a^b funcA(\tau) \cdot funcB(t-\tau)d\tau \approx \sum_{\tau=a}^{b} funcA(\tau) \cdot funcB(t-\tau) \cdot \frac{tempo[tempo.size-1]-tempo[0]}{tempo.size}$$

Assim pode-se calcular a convolução total entre duas funções quaisquer, a qual foi desenhada como o gráfico inferior esquerdo.

Desta forma, obteve-se o seguinte *codigo*:

```

import numpy as np
import matplotlib.pylab as plt
#Funcoes necessarias para atualizar graficos no Jupyter Notebook
from IPython.display import display, clear_output

#Plota o grafico das funcoes e sua convolucao
def PlotConvolucao(funcA, nomeFuncA, funcB, nomeFuncB, tempo):
    #Cria o plot
    fig, ax = plt.subplots(2,2)
    fig.set_figheight(10)
    fig.set_figwidth(10)

    #Cria as variaveis para armazenar os valores das funcoes
    valA = np.zeros(shape=(tempo.size))
    valB = np.zeros(shape=(tempo.size))
    valConv = np.zeros(shape=(tempo.size))

    #Para cada t em tempo
    for i, t in enumerate(tempo):
            #Calcula e salva o valor das funcoes
            valA[i] = funcA(t)
            valB[i] = funcB(t)

    #Faz o plot das funcoes
```

```
    ax[0,0].plot(tempo, valA, "-b", label="f1 = " + nomeFuncA)
    ax[0,0].grid()

    ax[0,1].plot(tempo, valB, "-r", label="f2 = " + nomeFuncB)
    ax[0,1].grid()

    #Para cada instante tau
    for posTau, tau in enumerate(tempo):
        #Calcula a funcao b com o deslocamento de tau
        for i in range(0, tempo.size):
            #Tempo negativo para espelhar a funcao
            valB[i] = funcB(-(tempo[i] - tau))

        #Calcula os valores da convolucao entre as funcoes no instante
        conv = valA * valB

        #integral = area no intervalo definido = somatorio das areas
        #Como conv tem os valores no instante, que equivalem a altura, basta multiplicar pelo comprimento de cada trecho
        #(tempo[tempo.size - 1] - tempo[0]) = comprimento total
        #tempo.size = quantidade de trechos
        #(tempo[tempo.size - 1] - tempo[0])/tempo.size = comprimento de cada trecho
        #np.sum(conv) * (tempo[tempo.size - 1] - tempo[0])/tempo.size = area total no instante
        valConv[posTau] = np.sum(conv) * (tempo[tempo.size - 1] - tempo[0])/tempo.size

        #Limpa o plot da convolucao
        ax[1,0].cla()
        #Plota a funcao A
        ax[1,0].plot(tempo, valA, "-b")
        #Plota a funcao B no instante atual
        ax[1,0].plot(tempo, valB, "-r")
        #Plota os valores da convolucao calculados ate o momento
        ax[1,0].plot(tempo, valConv, "-g", label="f1 conv f2")
        #Linha tracejada vertical que marca o instante atual
        ax[1,0].axvline(x = tau, color = 'r', ls='--', label = 'tau')
        ax[1,0].grid()

        #Limpa o plot da convolucao no instante atual
        ax[1,1].cla()
        #Plota os valores da convolucao no momento atual
        ax[1,1].plot(tempo, conv, "-g", label="Convolucao em tau")
        ax[1,1].grid()

        #Ativa as legendas, todas no canto superior direito
        ax[0,0].legend(loc=1)
        ax[0,1].legend(loc=1)
        ax[1,0].legend(loc=1)
        ax[1,1].legend(loc=1)

        #Desenha os graficos
        display(fig)

        #Fala para limpar o grafico quando houver um novo para desenhar
        clear_output(wait = True)
        #Espera um pequeno intervalo de tempo, para a animacao ocorrer em, no minimio, 5 segundos
        plt.pause(5/tempo.size)

#Rect(t)
def Rect(t):
    if abs(t) > 0.5:
        return 0
    return 1

#sin(t)
def Seno(t):
```

```
    return np.sin(t)

#t/3 * Rect(t/3 - 1/2)
def MeioTriang(t):
    return t/3 * Rect(t/3 - 1/2)

#t/3 * sin(t) * Rect(t/3 - 1/2)
def MeioTriangSenoidal(t):
    return Seno(t) * MeioTriang(t)
```

3. Resultados

Para o primeiro teste, foram utilizadas duas funções retangulares de 1 de largura, as quais geraram um triângulo de altura 1 e largura 2 (Figura 3), assim como esperado (Figura 4).

Aqui notou-se algo: a operação é extremamente lenta. Após pesquisar, descobriu-se que a biblioteca matplotlib não é eficiente para apresentação em tempo real (Fonte: https://stackoverflow.com/questions/8955869/why-is-plotting-with-matplotlib-so-slow), no entanto, por falta de tempo e como a falha não afeta os resultados, não foi corrigida.

Figura 3: *Convolução entre duas funções retangulares*

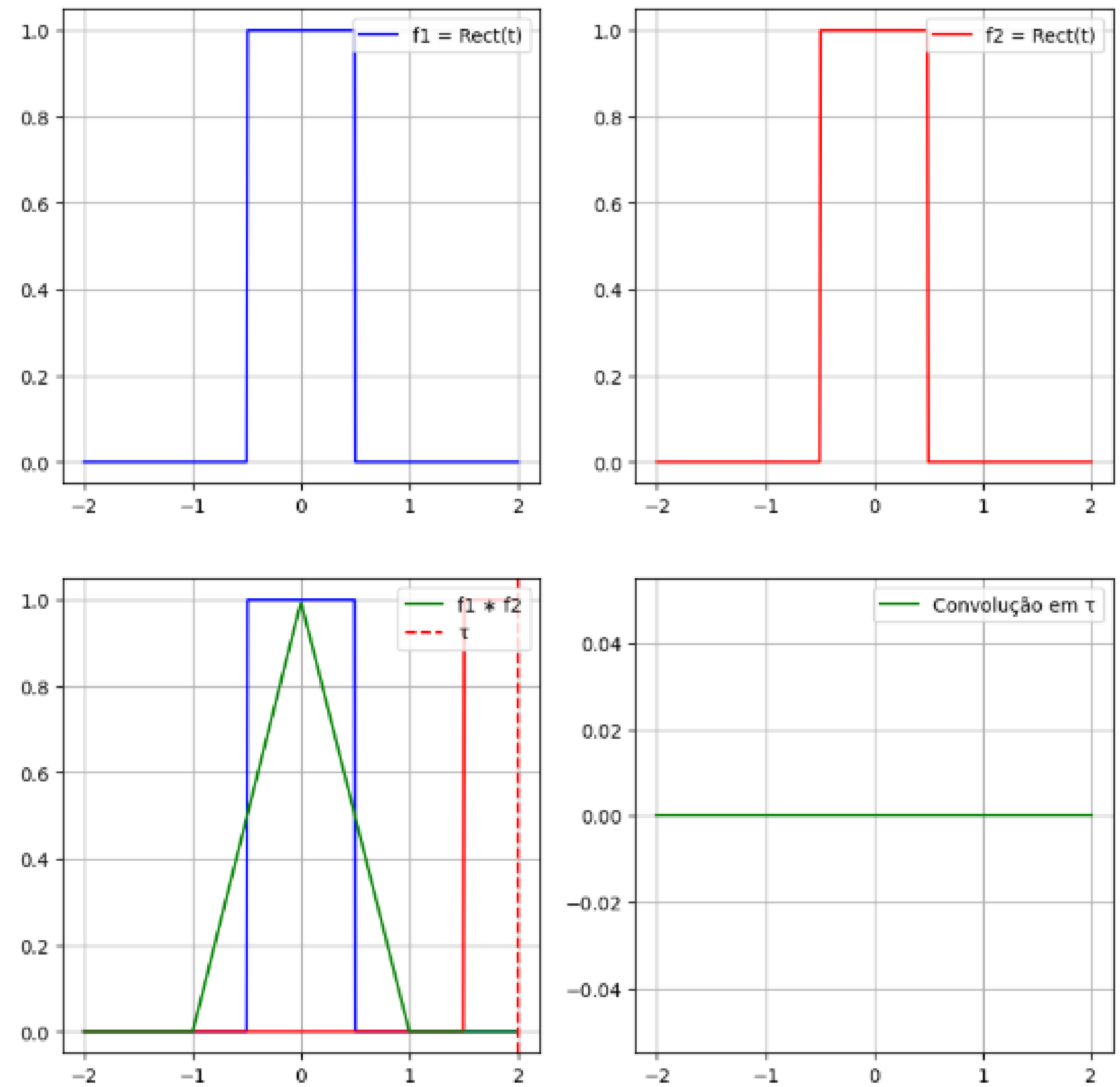
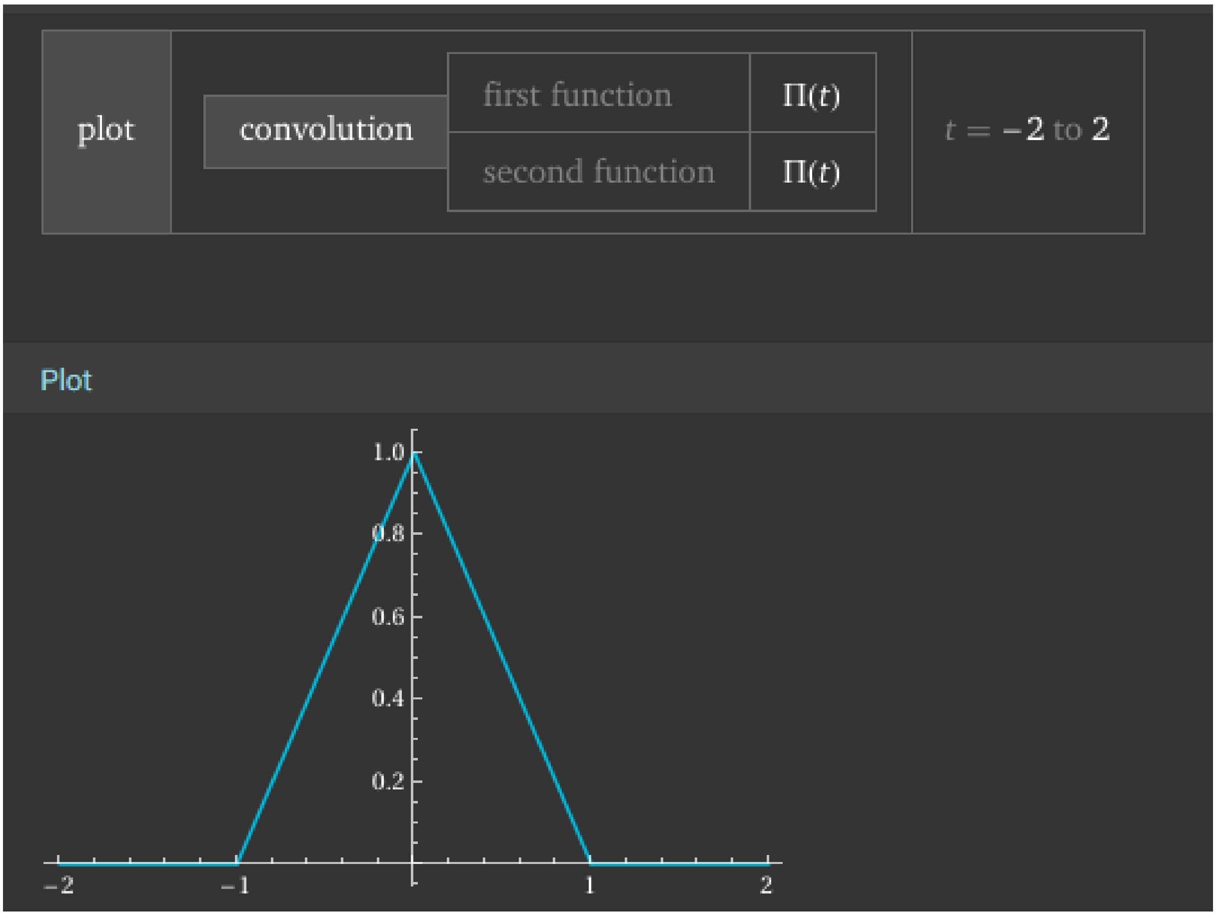


Figura 4: *Resultado esperado*



Fonte: *Wolfram*

Para o segundo teste, foram utilizadas uma função retangular de 1 de largura e uma função seno. O resultado foi uma função seno com amplitude ligeriamente menor (Figura 5), assim como o esperado (Figura 7). Para testar a propriedade da comutatividade da convolução, foram reutilizadas as funções.

Ao recalcular invertendo as funções, obteve-se a imagem 6. Percebe-se que o começo da convolução resultante difere do teste anterior. Isto pode ser explicado pelo fato de todas as contas serem feitas no intervalo definido, então nos primeiros instantes a função pode variar dependendo da função em deslocamento.

Figura 5: *Convolução entre função retangular com seno deslocando*

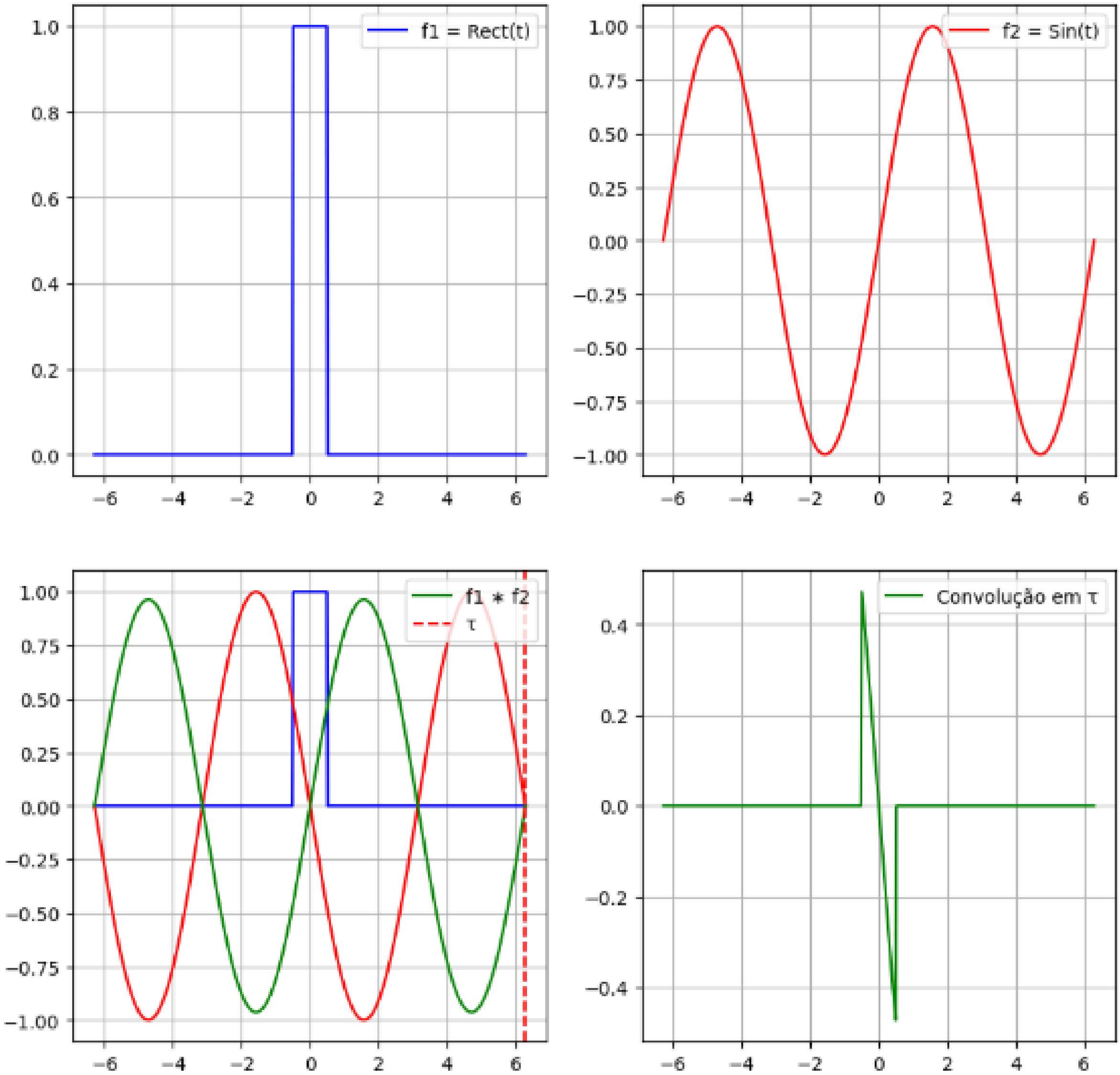


Figura 6: *Convolução entre seno com função retangular deslocando*

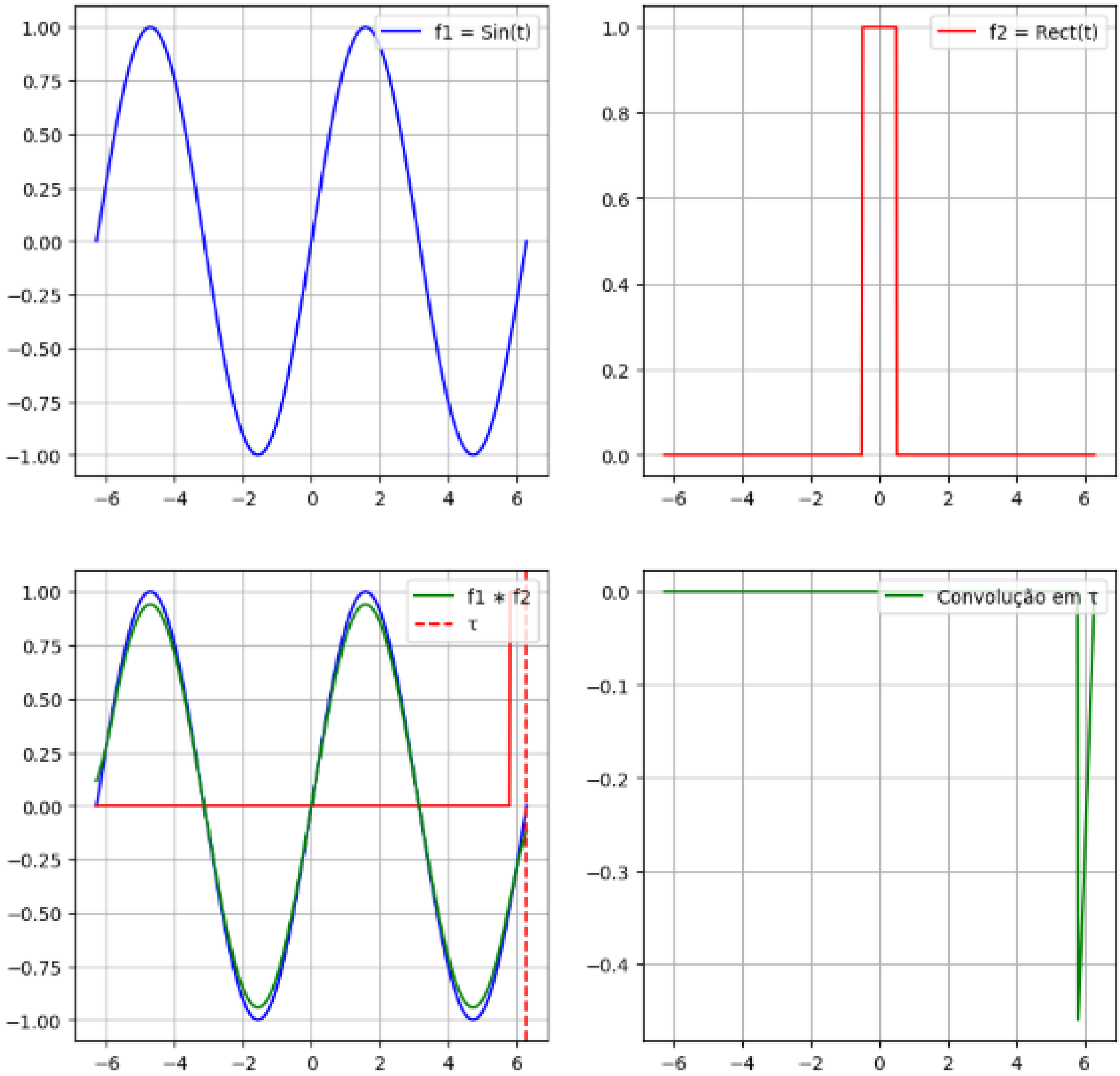
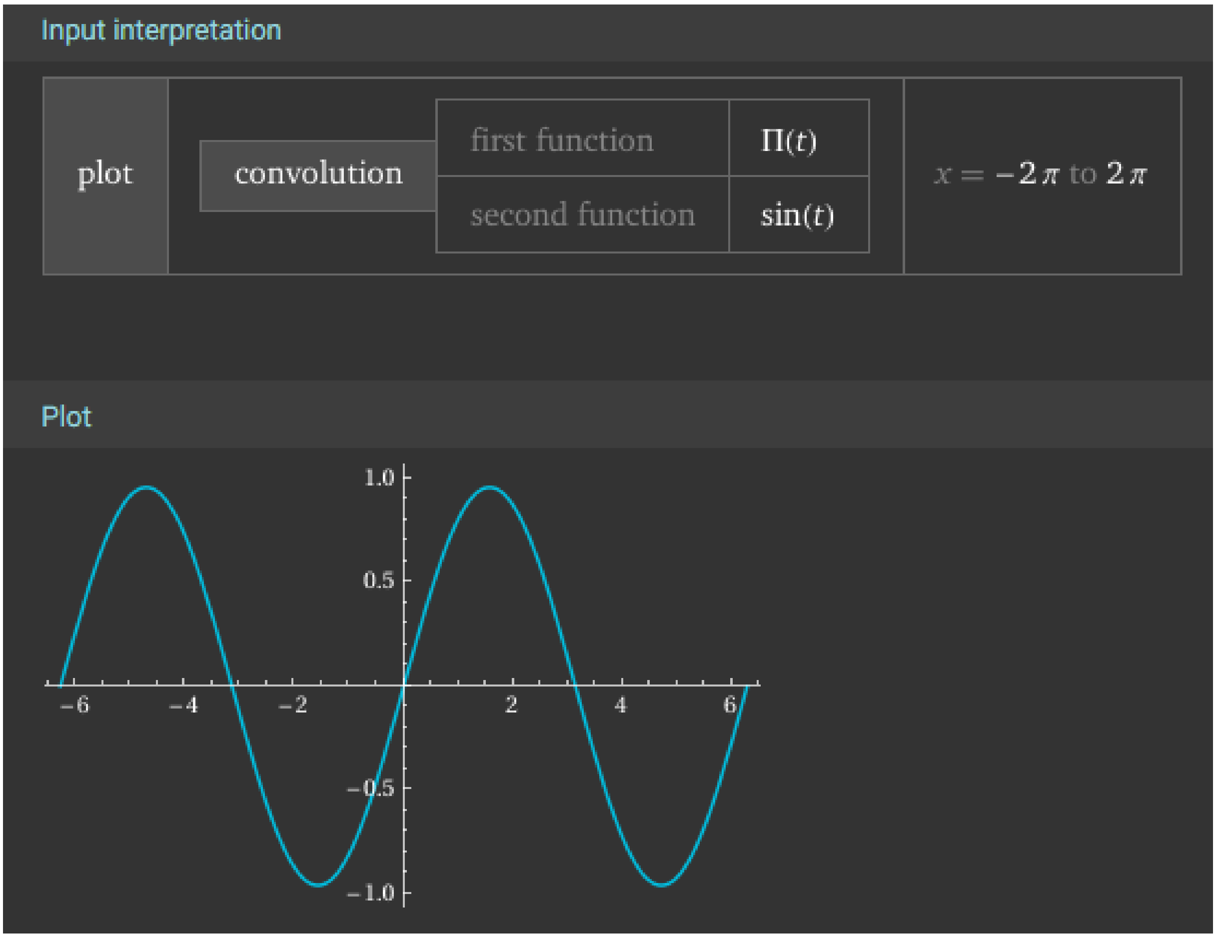


Figura 7: *Resultado esperado*



Fonte: Wolfram

Para testar os resultados com funções criadas pelo usuário, foi criada a função MeioTriang e passada para o cálculo, gerando os resultados abaixo.

Figura 8: *Convolução entre função customizada com função retangular deslocando*

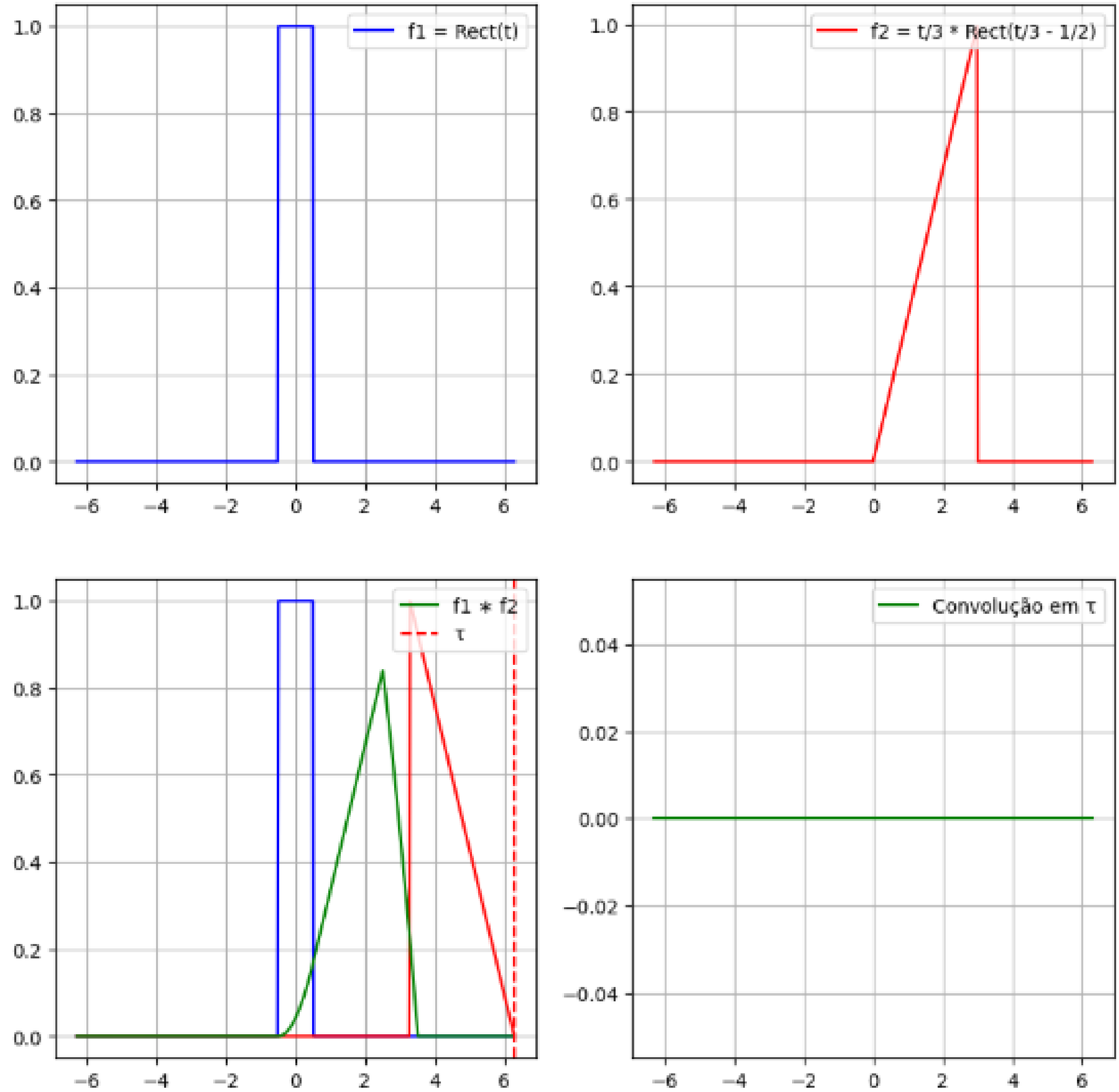
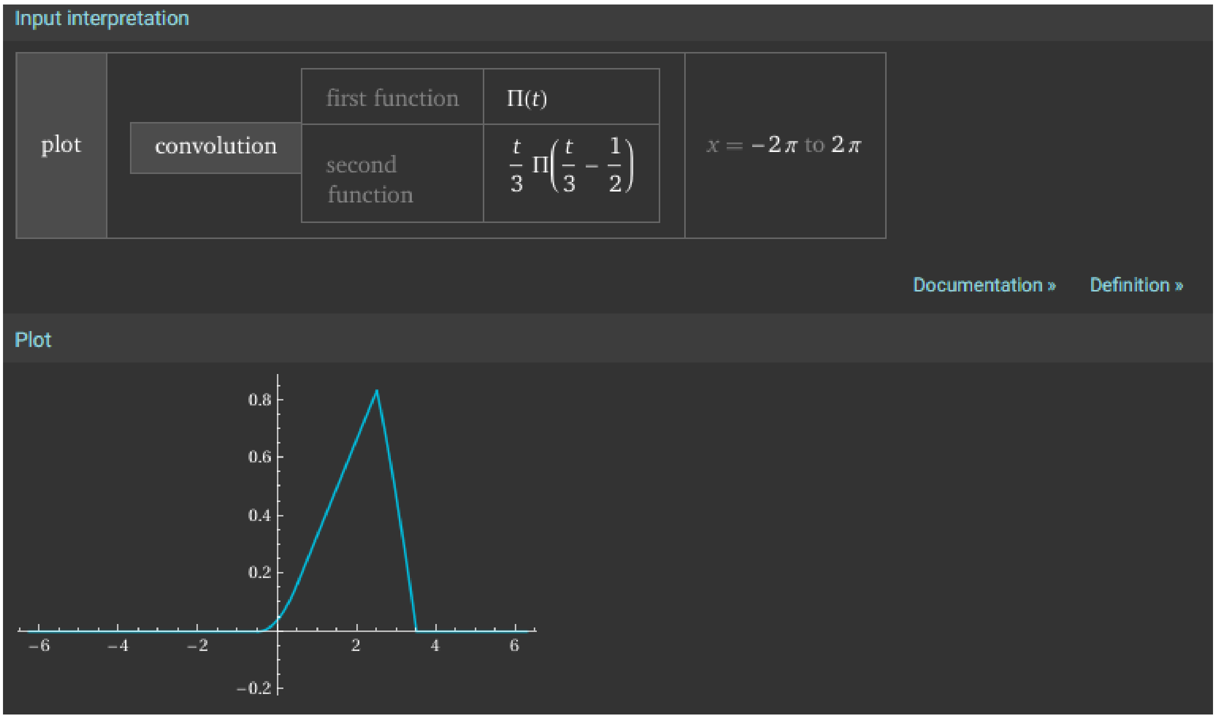


Figura 9: *Resultado esperado*



Fonte: *Wolfram*

Para os testes finais, foi criada uma outra função customizada pela multiplicação da função anterior com a seno e foi então calculada sua convolução com outra função seno, com a segunda deslocando (Figura 10) e com a primeira deslocando (Figura 11).

Este teste tem como objetivo forçar um novo limite ao código, testando uma função customizada com uma outra função contínua.

Como pode-se ver abaixo, os resultados diferem entre si novamente no começo, sendo o primeiro correto e o segundo incorreto, o mesmo erro notado no exemplo 2. Assim percebe-se que o cálculo da convolução não pode depender apenas do intervalo de tempo escolhido, mas deve calcular uma região infinita, dificultando sua realização computacional.

Figura 10: *Convolução entre função customizada com seno deslocando*

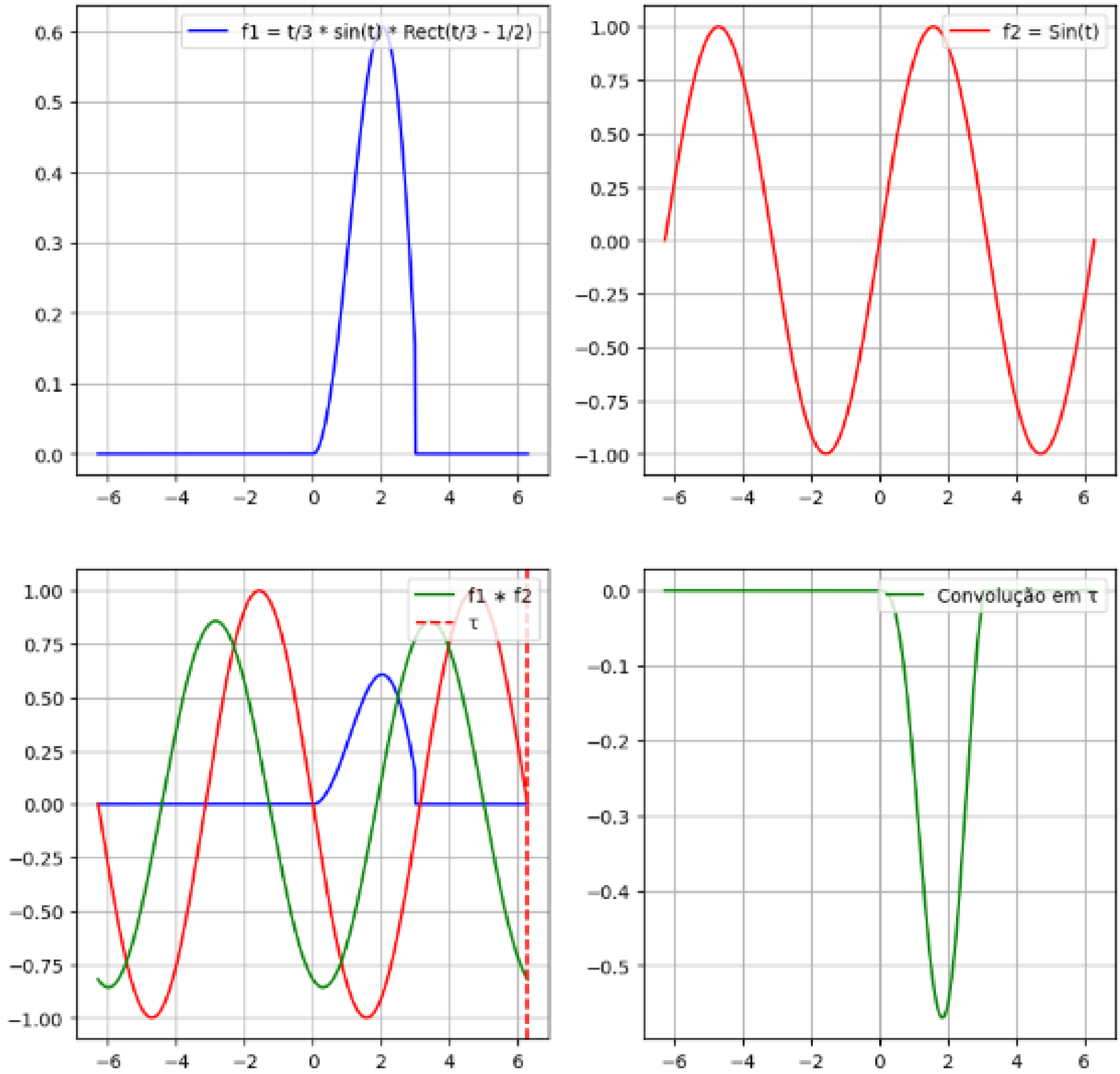


Figura 11: *Convolução entre seno com função customizada deslocando*

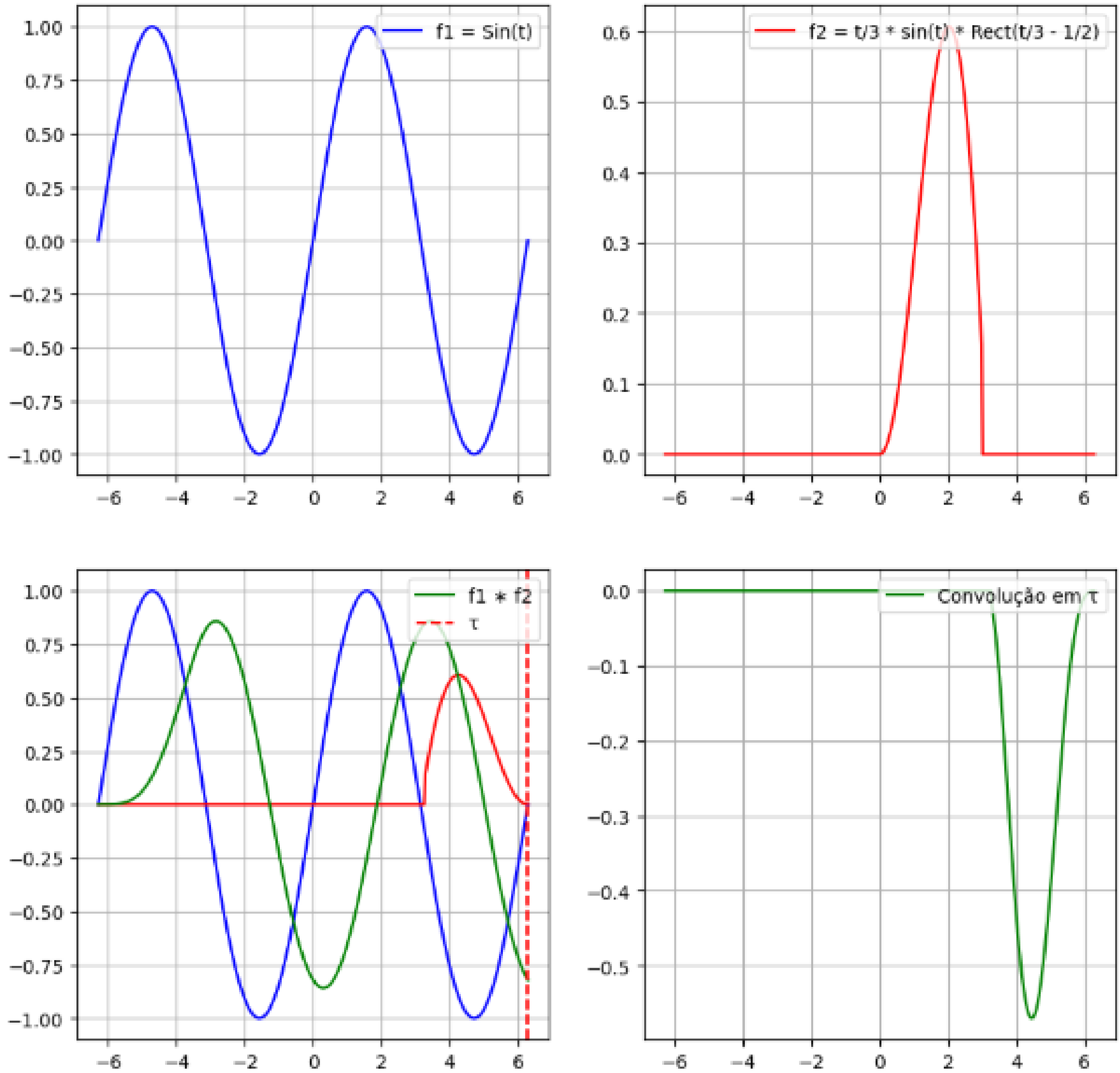
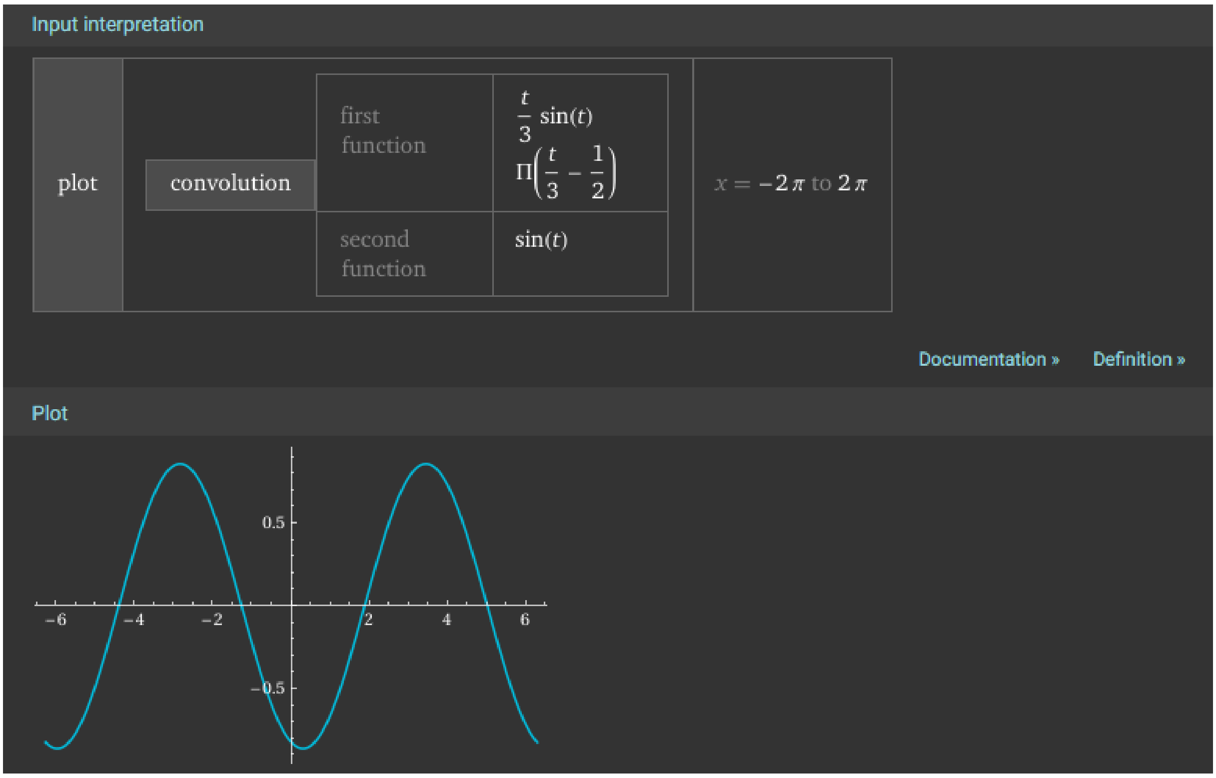


Figura 12: *Resultado esperado*



Fonte: Wolfram

4. Conclusão

Este trabalho realizou uma análise sobre a convolução, verificando sua definição e criando uma interpretação computacional para a mesma, permitindo seu cálculo por meio de operações mais simples. Foi então criado um código para realizar tais operações e apresentar para o usuário o processo que está sendo feito, assim permitindo sua interpretação gráfica. Por fim, foram encontradas duas falhas na solução: primeiro que a operação como um todo é extremamente lenta e, segundo, que ela falha em calcular os valores nos primeiros instantes.

Em relação à primeira falha, infelizmente não houve tempo para correção da mesma, mas ela não cancela a validade dos resultados, apenas demonstra que há muito espaço para melhoras. Em relação à segunda, sua correção requer uma análise num intervalo de tempo infinito, mostrando uma falha na interpretação feita.