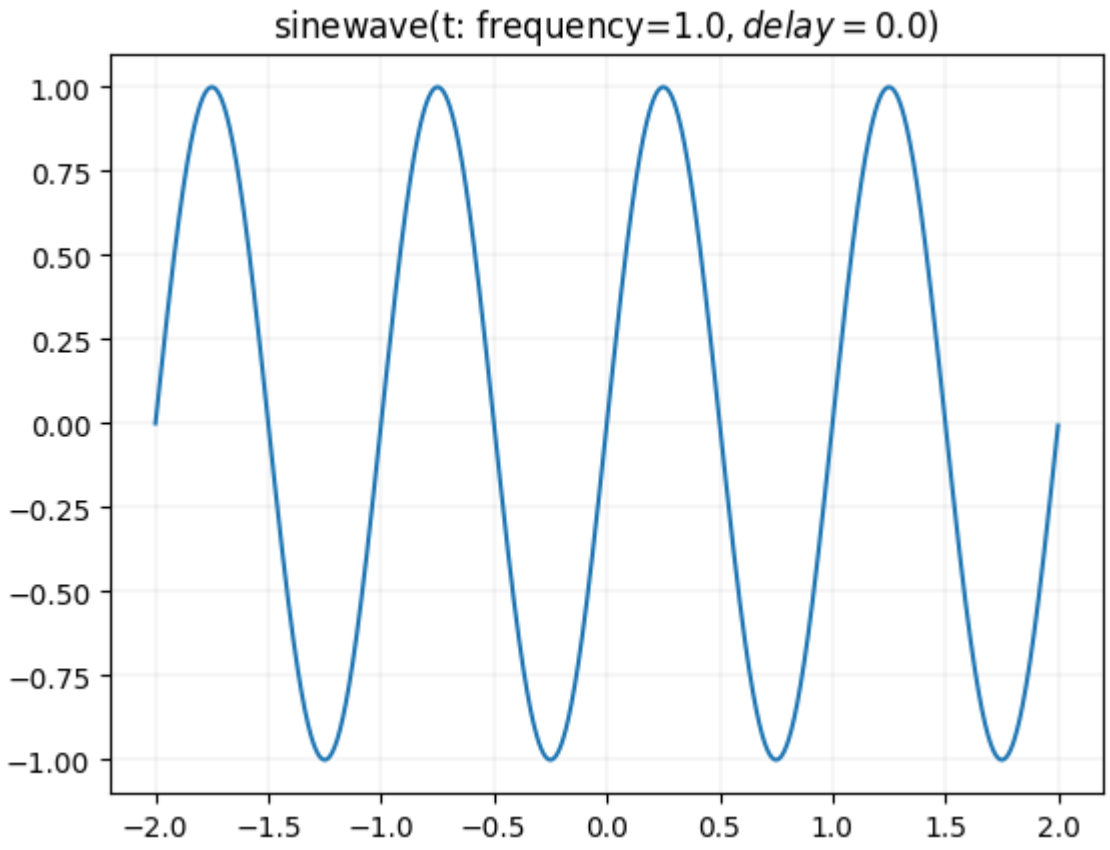
Assignment 1b: Functions and Computation

Hunter Welch 1/30/2023

1. Markdown and LaTeX

1a. Sinewave

In [9]:
```
import math
import matplotlib.pyplot as plt
import numpy as np

def sinewave(t, f = 1.0, d = 0.0):
    return np.sin(2 * math.pi * f * (t - d))


def plot_sinewave(t, f = 1.0, d = 0.0):
    y = sinewave(t, f, d)
    plt.plot(t, y)

    plt.title(f'sinewave(t: frequency=${f}, delay=${d})')
    plt.grid(color='gray', linestyle='-', linewidth=0.1)
    # plt.legend()
    plt.show()


t = np.arange(-2, 2, 0.001)
plot_sinewave(t)
```

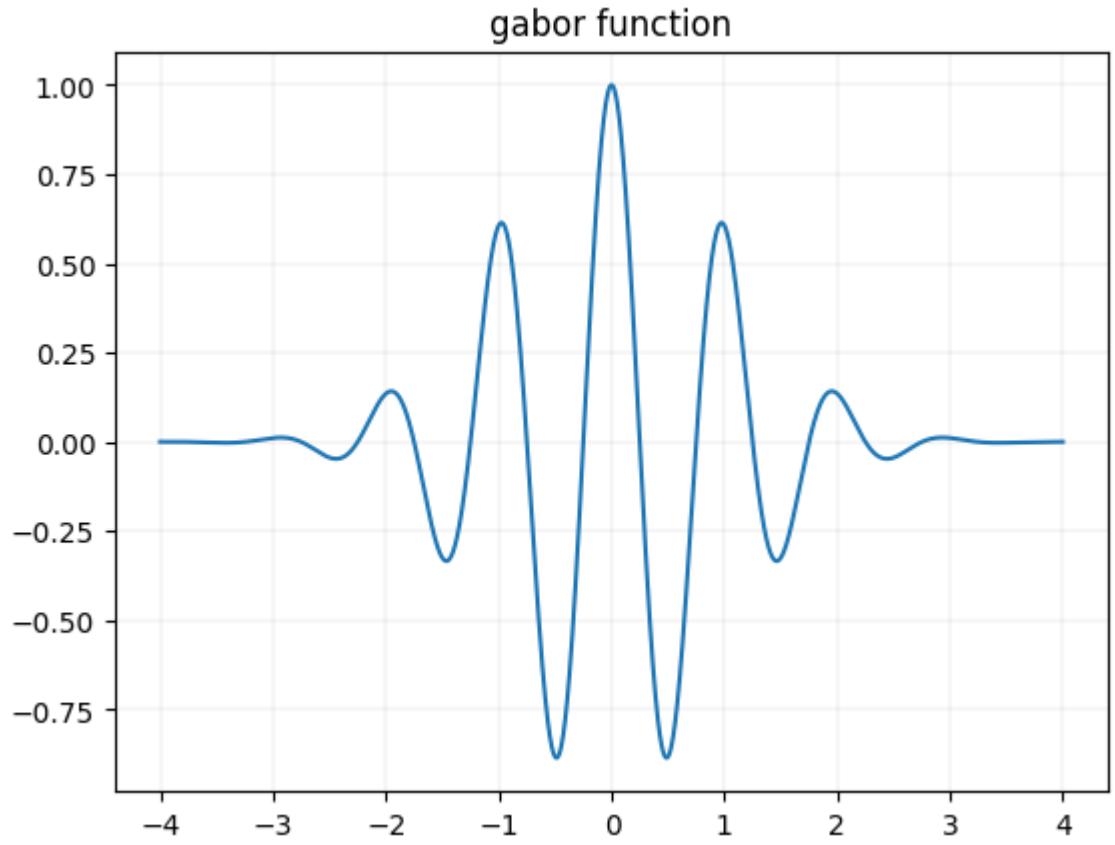


1b. Gabor

In [1]:

```python
import matplotlib.pyplot as plt
import numpy as np

def gabor(t, a = 1, f = 1, sigma = 1, phi = 0):
    vector = np.vectorize(np.float_)
    t = vector(t)
    return a * np.exp(- (t ** 2) / (2 * sigma ** 2)) * np.cos(2 * np.pi * f * t + p

def plot_gabor(t, a = 1, f = 1, sigma = 1, phase = 0):
    y = gabor(t, a, f, sigma, phase)
    plt.plot(t, y)

    plt.title(f'gabor function')
    plt.grid(color='gray', linestyle='-', linewidth=0.1)
    # plt.legend()
    plt.show()

def gabore(t, a = 1, f = 1, sigma = 1):
    phi = 0
    return gabor(t, a, f, sigma, phi)

def gaboro(t, a = 1, f = 1, sigma = 1):
    phi = np.pi / 2
    return gabor(t, a, f, sigma, phi)

def gabor_norm(f = 1, sigma = 1, phi = 0, f_s = 100):
    t = []
```

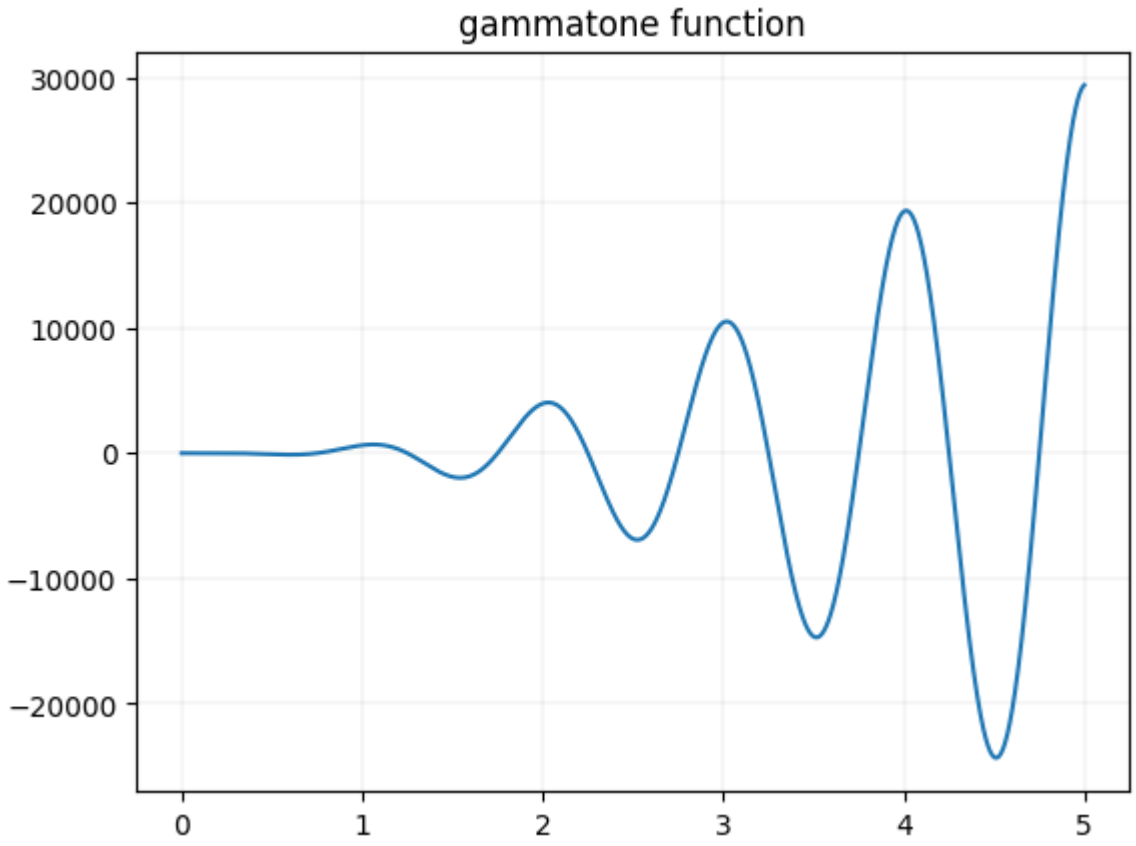
```
    x = np.arange(0, f_s, 1.)
    for i in x:
        t.append(i / f_s)
    t = np.array(t)
    a = 1
    gab = gabor(t, a, f, sigma, phi, f_s)
    return np.linalg.norm(gab)

def gabore_norm(f = 1, sigma = 1, f_s = 100):
    phi = 0
    gabor_norm(f, sigma, phi, f_s)

def gaboro_norm(f = 1, sigma = 1, f_s = 100):
    phi = np.pi / 2
    gabor_norm(f, sigma, phi, f_s)


t = np.arange(-4, 4, 0.001)
plot_gabor(t)
```



1c. Gammatone

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np


def erb(f):
    return 24.7 * ((4.37 * f) / 1000 + 1)
```

```
def get_b(f):
    return 1 / (1.019 * erb(f))

def gammatone_norm(t, n = 4, f = 1, phi = 0):
    b = get_b(f)
    gamma = t ** (n - 1) * np.exp(-2 * np.pi * b * t) * np.cos(2 * np.pi * f * t +
    return np.linalg.norm(gamma)

def gammatone(t, n = 4, f = 1, phi = 0):
    b = get_b(f)
    a = gammatone_norm(t, n, f, phi)
    y = []
    for t_i in t:
        if t_i == 0:
            y.append(0.)
        else:
            y.append(a * t_i ** (n - 1) * np.exp(-2 * np.pi * b * t_i) * np.cos(2 *
    return np.array(y)


def plot_gammatone(t, n = 4, f = 1, phi = 0):
    y = gammatone(t, n, f, phi)
    plt.plot(t, y)

    plt.title(f'gammatone function')
    plt.grid(color='gray', linestyle='-', linewidth=0.1)
    # plt.legend()
    plt.show()


t = np.arange(0, 5, 0.001)
plot_gammatone(t)
```



2. Simple Computation

2a. Local Maxima

In [2]:
```
def localmaxima(arr):
    local_maxes = []
    i = 1
    while i < len(arr) - 1:
        if arr[i - 1] < arr[i] and arr[i + 1] < arr[i]:
            local_maxes.append(i)
        i += 1
    return local_maxes

test_arr = [1, 2, 3, 4, 1, 2, 3, -1, -5, -7, 0, -1]
print(localmaxima(test_arr))
```

```
[3, 6, 10]
```

2b. Crossings

In [3]:
```
# a function to compute the indicies of where a function first equals or crosses a
def crossings(arr, threshold, direction = 'both'):
    crosses = []
    i = 1
    while i < len(arr):
        if arr[i] == threshold and arr[i - 1] != threshold:
            crosses.append(i)
        elif arr[i] > threshold and arr[i - 1] < threshold and (direction == 'both'
```

```
            crosses.append(i)
        elif arr[i] < threshold and arr[i - 1] > threshold and (direction == 'both'
            crosses.append(i)
        i += 1
    return crosses



test_arr = [1, 2, 3, 4, 1, 2, 3, -1, -5, -7, 0, -1, 1]
exp = [4, 7, 12]
print(crossings(test_arr, 1))
```

```
[4, 7, 12]
```

2c. Envelope

In [5]:

```
# function to downsample data and get the lower, upper, and block indicies of each

def envelope(y, nblocks = 10):
    ylower = []
    yupper = []
    blockindices = []

    size_block = int(len(y) / nblocks)

    remainders = len(y) % nblocks
    # if we have remainders the first few blocks need to take an extra element
    if remainders > 0:
        size_block += 1

    index = 0
    iteration_num = 0
    while index < len(y):
        # here we are setting the size of the block back to the original number aft
        if iteration_num == remainders and remainders != 0:
            size_block -= 1
        else: size_block
        if index + size_block < len(y):
            upper = index + size_block
        else:
            upper = len(y)
        arr = y[index: upper]
        ylower.append(min(arr))
        yupper.append(max(arr))
        blockindices.append(index)
        index = upper
        iteration_num += 1
    return ylower, yupper, blockindices

test_arr = [1, 2, 3, 4, 1, 2, 3, -1, -5, -7, 0, -1, 1, ]
yl, yu, bi = envelope(test_arr)
print("ylower: ", yl)
print("yupper: ", yu)
print("blockindicies: ", bi)
```

```
ylower:  [1, 3, 1, 3, -1, -5, -7, 0, -1, 1]
yupper:  [2, 4, 2, 3, -1, -5, -7, 0, -1, 1]
blockindicies:  [0, 2, 4, 6, 7, 8, 9, 10, 11, 12]
```