Programming Task

Task 1-1: Signal Generation (45 points)

You are required to implement the functions in `task_1_1.py`.

Please generate the following signals by implementing `generate_signal(id)`, where `id` is the index of the signal. The sampling rate `f_s` is given by `self.fs` (`self.fs >= 1000Hz`) (You do not need to care about `self.fs`). The time range is $-1 \leq t < 1s$

1. Generate a pure tone signal with frequency and phase offset.

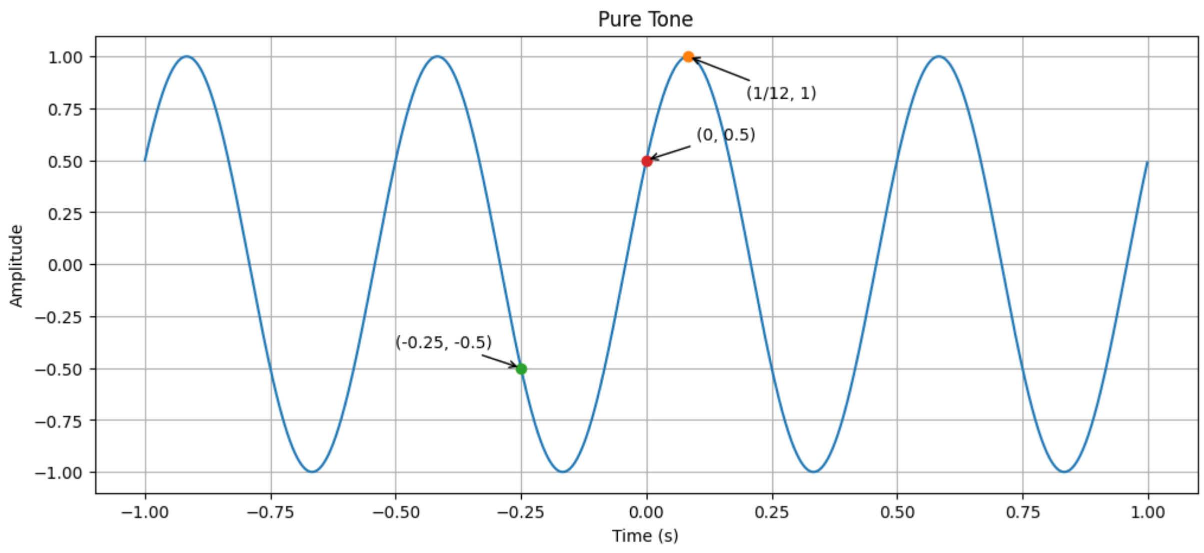

Figure 1: Pure Tone (Task 1-1-1)

2. Generate rectangle wave (pulse) and triangle wave.

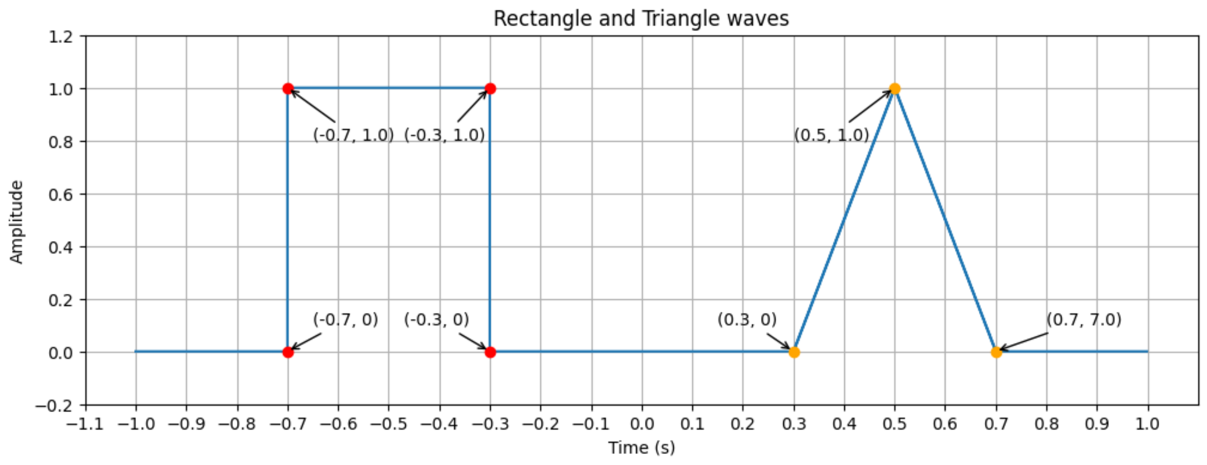

Figure 2: Rectangle and triangle wave (Task 1-1-2)

3. Generate the complex signal according to the real and imaginary parts. These two are both pure tone signals.

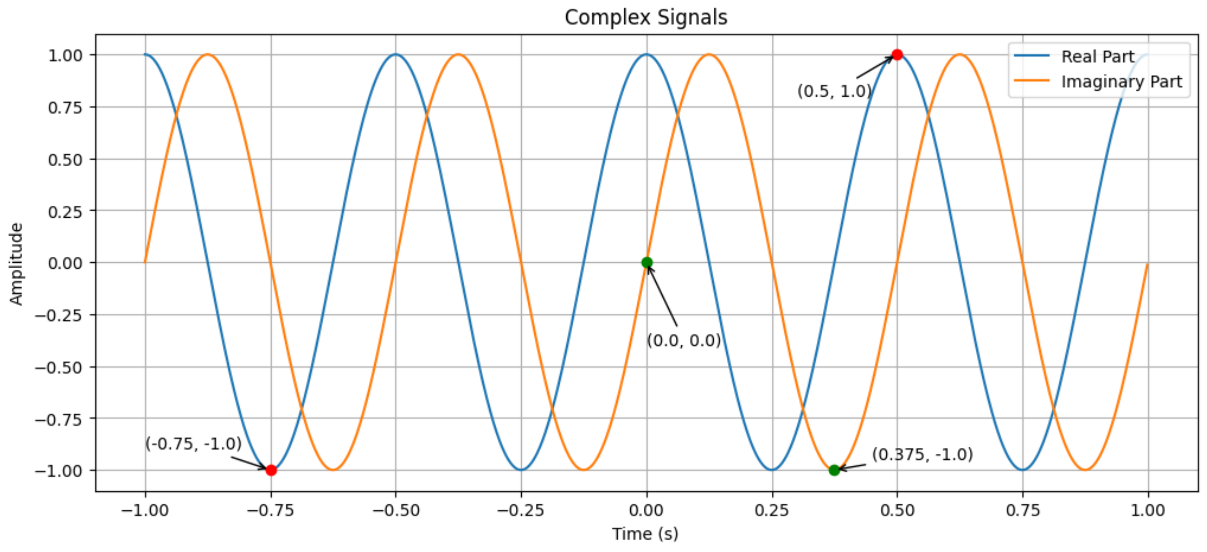

Figure 3: Complex signal (Task 1-1-3)

You should both return the time stamps (in s) and signals. For facilitating the testing of your work, please strictly follow the return format.

After implementing the functions, you can run the following code to check the code:

```
python check.py --task 1
```

The code will also automatically generate the figures for you to check the correctness of your implementation. You can find the figures in the `fig` folder.

Task 1-2: Chirp Signal (30 points)

You are required to implement the functions in `task_1_2.py`.

A chirp signal is a type of signal in which the frequency changes over time. In radar and sonar signal processing, as well as in other fields, chirp signals are very useful because of their properties.

Based on different $f(t)$, you are supposed to generate different types of chirps. Your task is to generate the chirp signal (`s_t`) along with timestamp (`t`) and frequency function `f_t`. The sampling rate is given by `self.fs`. Time starts from 0. You are supposed to use the cosine function to generate the chirps.

1. Linear Chirp:

$$f(t) = f_{\text{start}} + \left(\frac{f_{\text{end}} - f_{\text{start}}}{T}\right) \cdot t$$

Please implement

```
def generate_linear_chirp(amplitude, period, duration, start_freq,
    end_freq, init_phase)
```

, where

- `amplitude` is the maximum amplitude of the chirp signal,
- `period` is duration of one chirp cycle in seconds,
- `duration` means total duration of the signal in seconds,
- `start_freq`: Starting frequency of the chirp in Hz,
- `end_freq`: Ending frequency of the chirp in Hz,
- `init_phase`: Initial phase of the chirp in radians.

You should return

- `t`: Timestamp values in second
- `f_t`: Frequency values over `t`
- `s_t`: Generated signal values

2. Quadratic Chirp:

$$f(t) = f_{\text{start}} + (f_{\text{end}} - f_{\text{start}}) * \frac{t^2}{T^2}$$

   Please implement

```
def generate_quar_chirp(amplitude, period, duration, start_freq,
    end_freq, init_phase)
```

After implementing the functions, you can run the following code to check the code:

```
python check.py --task 2
```

The code will also automatically generate the figures for you to check the correctness of your implementation. You can find the figures in the `fig` folder.

Task 1-3: Amplitude Modulation (25 points)

You are required to implement the function in `task_1_3.py`.

Now we consider a real communication system. The basic role of communication system is to convey messages over the medium.

Amplitude modulation (AM) is a technique used in communication systems where the amplitude of a high-frequency carrier wave is varied in proportion to a lower-frequency message signal. In this task, you will be guided to leverage amplitude modulation to generate signals.

Here we define the message signal as

$$m(t) = \cos(2\pi f_m t).$$

The message signal refers to the original signal that contains information, which is typically a low-frequency signal (e.g. audio, voice or data). Yet the low-frequency signal cannot be transmitted over long distances due to the high attenuation and long wavelengths. Therefore, we need to find a "vehicle" to carry the message signals. In communication system, it is called the carrier signal, which is

$$c(t) = \cos(2\pi f_c t + \phi).$$

There is no information content on the carrier signal.

The key behind modulation is to superimpose the low-frequency message signal into a high-frequency carrier signal, which allows the message to be transmitted over long distances efficiently. Here we leverage Amplitude Modulation (AM).

In AM, the amplitude of the carrier signal is varied in linear proportion of the instantaneous amplitude of the message signal. We model it as

$$s(t) = A_c(1 + \mu \cdot m(t)) \cdot c(t)$$

where

- A_c is the amplitude of the carrier wave
- μ is the modulation index
- f_m is the frequency of the message signal
- f_c is the frequency of the carrier signal
- ϕ is the initial phase of the carrier wave (in rad)

In this task, you are asked to implement the function

```
def generate_am_signal(self, Ac: float, mu: float, fm: float, fc: float
    , phase: float):
```

And you should return four numpy arrays:

- `t`: Timestamp values in second
- `m_t`: Message signal values
- `c_t`: Carrier signal values
- `s_t`: Amplitude Modulated signal values

In this task, we set the duration of the signal as $0 \leq t < 2s$. The sampling rate is given by `self.fs`. Please strictly follow the return format for facilitating the testing of your work.

After implementing the functions, you can run the following code to check the code:

```
python check.py --task 3
```

The code will also automatically generate the figures for you to check the correctness of your implementation. You can find the figures in the `fig` folder. The message signal will be shown as the **envelope** of the AM signal.

How to submit

Please run

```
python check.py --uid <YOUR_UID>
```

before submitting. This script performs automated tests on the examples provided in the docstrings. Failing these tests indicates potential critical issues in your code. Strive to resolve these problems. After that, it will create a zip file named after your `uid`. Make sure you enter the right `uid`.

It's important to avoid changing the names of any files, including both the zip file and the program files contained within. Altering file names can lead to grading errors. Ensure that all file names remain as they are to facilitate accurate assessment of your work.

Your submission to **Moodle** should consist solely of the **generated `*.zip` file**. It is your responsibility to double check whether your submitted zip file includes your latest work.