Exercise

Exercise 1

- Add new param of **Signal** to the constructor to include a periodic function
- Define the **Wave** object
  - *ts*: np.ndarray
  - *ys*: np.ndarray
- Refactor the **print_stats** method

Load *classes* module

Before continuing, load *classes.*

Download from here

```
%load classes.py
```

Exercise 2

- write a **DictSerializerMixin** class with a **serialize** method that prints amplitude, phase and frequency of a signal as dict;
- extend **SinWaveformBasedSignal** class to include a new method that calls the mixin *serialize* method.
  You can import **SinWaveformBasedSignal** from *classes* module.

Exceercise 3

Part 1: The HighPass filter (remove higher freq than cutoff)

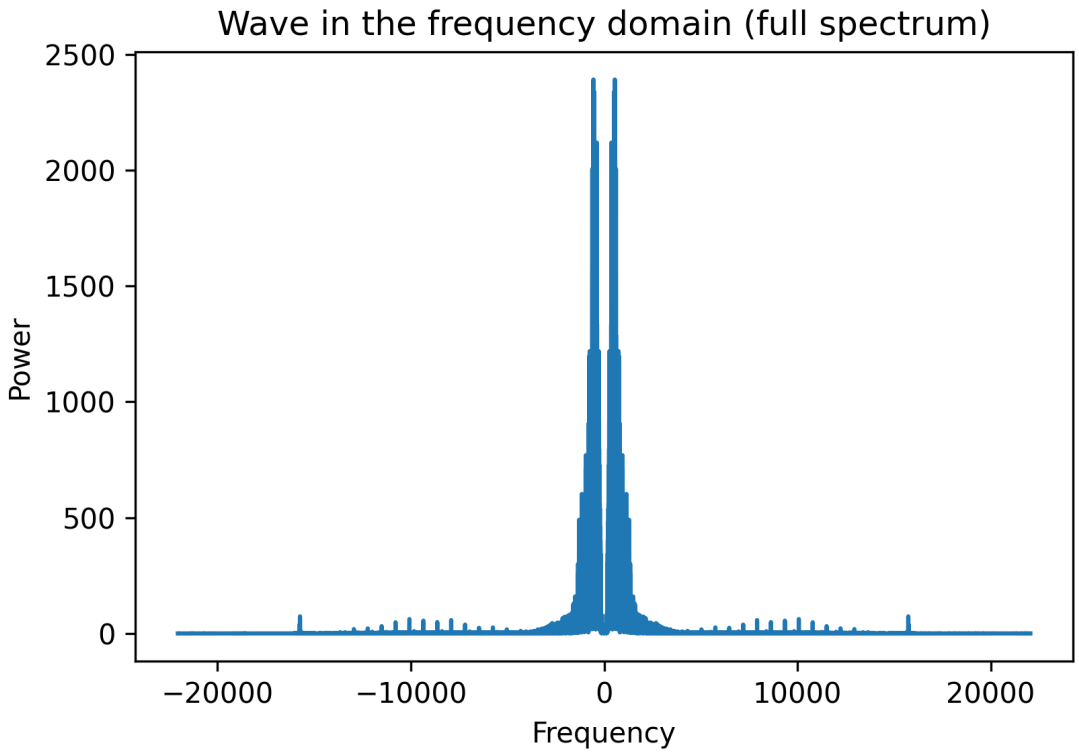
- Write an **HighPassFilter** that implements the **Filter** class.
  Import the latter from the utils.py library.

  The method needs to call the *high_pass* function with *hs*, *fs* as parameters.

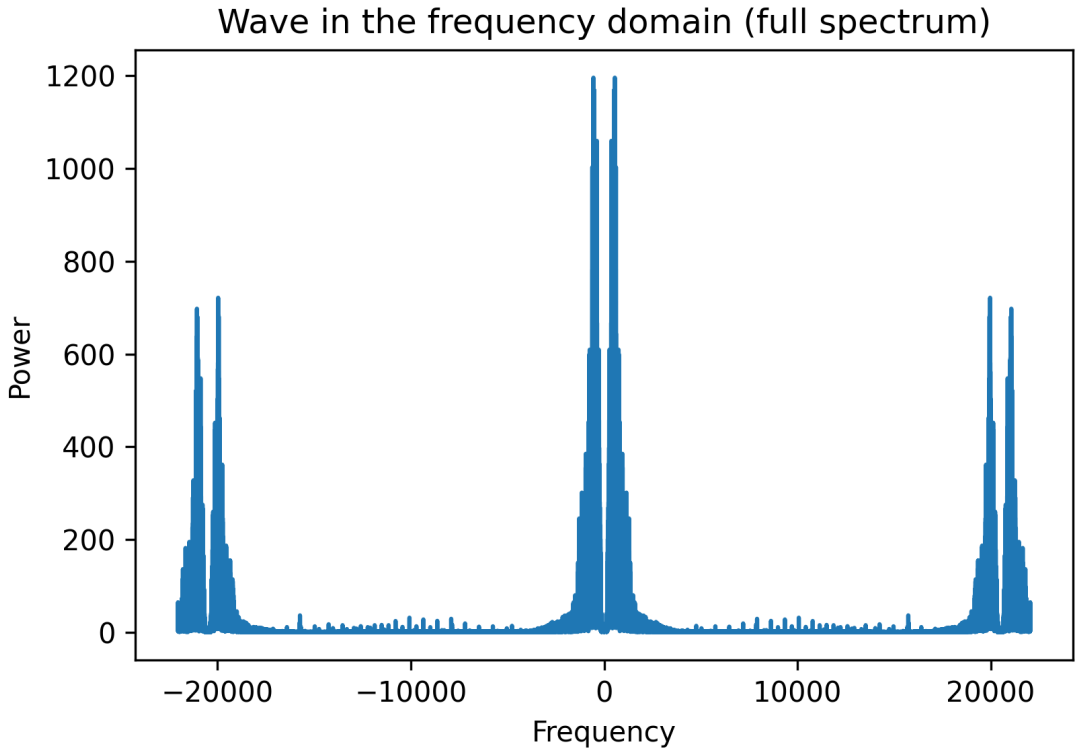
Part2: Filtering side-bands after AM modulation

- Write a A**mplitudeModulationWithLowPassFiltering** class that overrides the **demodulate** method in the **AMModulator** to appl
  Find the latter in the utils.py module.
- The new implementation should remove the side bands of modulated wave when the wave reaches its final destination. To do so, you need to demodulate as usual and then apply a low pass filtering.
- Talking in code, you need to define the new method as follow:
  - call the original **demodulate** method;
  - compute the spectrum use the **make_spectrum** method. This return an instance of *Spectrum* with three different attribute:
    - *hs*: powers (np.ndarray)
    - *fs*: frequency (np.ndarray)
    - *framerate*: sampling rate (int)

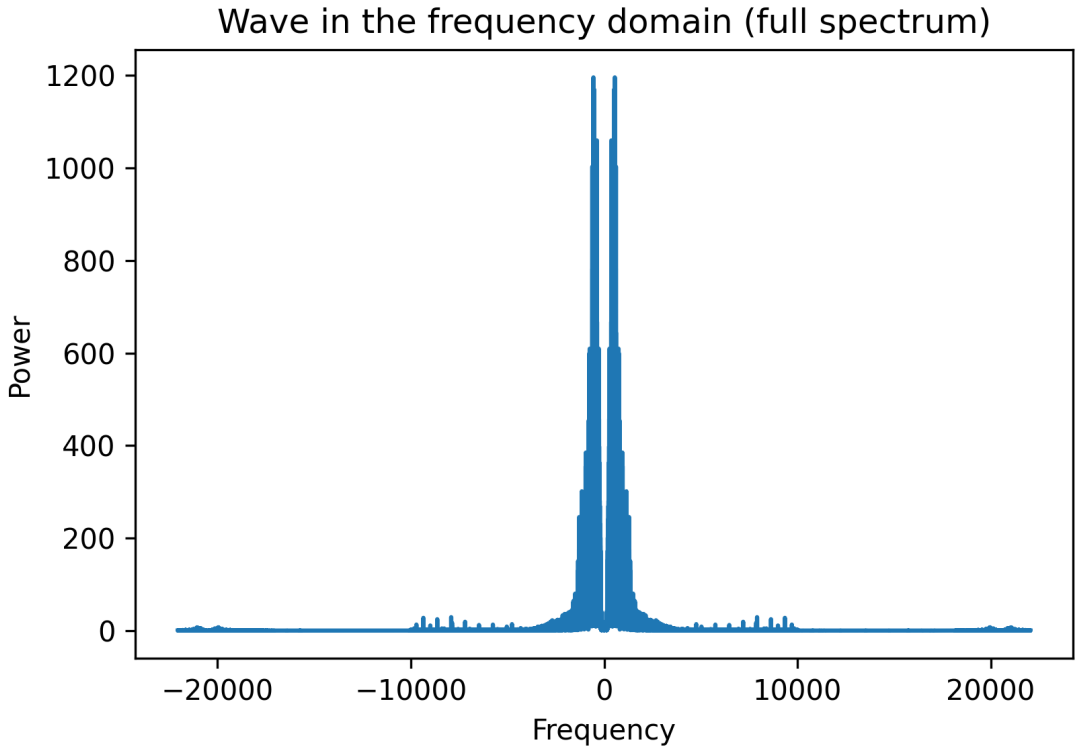
- Apply the filtering. Import **LowpassFilter,** call the **filter** method providing *hs, fs , cutoff* set to 10000. This method returns the filtered version of *hs*
- Call the **make_wave** over the spectrum class
- Plot the result calling the **plot_full_fft** method over wave: you should get the equal spectrum of the original wave



Origianl wave



Demodulated wave



Demodulated and filtered wave at 10KHz

Exercise 4

Part 1: Define an iterator for looping samples in reverse order

- define a **ReverseIterator** that implements the Iterator abstract base class

Part 2: Let the client choose which kind of iterator logic should use

- Define a **IteratorCreator** interface witha a *createIterator(self, condition='sequential')** method
- Implement the interface returning the correct iterator based on condition:
  - if *condition* ***equals to*** *sequential,* then return the **SequentialIterator,** otherwise the **ReverseIterator**
- Refactor the **WaveSampleCollection** class