Assignment 2
Sound Simulator

1. Submission Guidelines

Deadline:	3 PM on Friday 15 December 2023
Submission procedure:	Submit only one file labelled `simulator.py` through blackboard (via TurnItIn)
Version requirement:	Your code must run using **Python 3.11.4 on a PC**
Allowable import modules:	`math` only. No other modules may be imported into your `simulator.py` file.

2. Overview

Create 5 classes in your `simulator.py` file that can be used with the `import` system. We will type `import simulator` or a similar statement at the top of our own script (eg, `script.py`) and then use your classes to perform tasks. For different arguments used to initialise your class definition, or different arguments passed into the methods of your class objects, we will assess either the returned values of each method or the variable attributes of your class.

All code examples described here assume that `import simulator` or a similar statement has been executed prior to each function call. While we are providing example code, we will mark your code using a far wider range of test code, which are not being provided to you. It is important that you test your code under a wider range of conditions.

3. Classes `Transducer`, `Receiver`, and `Emitter`

We have provided you with a template for three classes: `Transducer` (parent class), and two derived classes `Receiver` and `Emitter`. The parent class, `Transducer`, contains the position in euclidean coordinates (x, y) of the transducers in units of metres, a time array in units of seconds, and a signal that represents an amplitude value for each point of time in the time array. Each of the derived transducers (`Receiver` and `Emitter`) inherit the variable attributes initialised in the parent class (`Transducer`). Copy the following template code into your `simulator.py` file. You are asked to fill in the `generate_signal()` method.

Template code:

```
class Transducer():

   def __init__(self, x, y, t_array):
      self.x = x
      self.y = y
      self.t_array = t_array
      self.signal = len(self.t_array)*[0]


class Receiver(Transducer):

   def __init__(self, x, y, t_array):
      super().__init__(x, y, t_array)


class Emitter(Transducer):

   def __init__(self, x, y, t_array):
      super().__init__(x, y, t_array)

   def generate_signal(self, f_c, n_cycles, amplitude):

      # include your code here
      pass
```

Write the following method in the `Emitter` class:

`signal = generate_signal(self, f_c, n_cycles, amplitude)`

Arguments: The *f_c* argument should be of type `float` and represents the centre frequency of the sinusoid in units Hz.

The *n_cycles* argument should be of type `int` and represents the number of cycles.

The *amplitude* argument should be of type `float` and represents the amplitude of the sinusoid. There are no units.

Return values: The returned `signal` should be a 1-dimensional `list` containing values of type `float`. It should contain the sinusoidal signal generated. It should be returned as a value and also stored in `self.signal`.

Implementation: Generate a sinusoidal signal with a frequency, number of cycles, and amplitude specified in the arguments. Store the sinusoidal signal in `self.signal` and return `self.signal` as well. The sinusoidal signal should begin at the first time defined within the `self.t_array` list. Outside the region containing the sinusoid, the signal amplitude should be 0.

4. Class `SoundSimulator`

The `SoundSimulator` class simulates sound from emitters travelling to receivers. It takes in lists of emitters and receivers and runs an acoustic simulation. The emitters emit a signal that is defined in their `signal` variable attribute and for time steps and durations defined by the time array (`t_array`). The receiver listens to the sound emitted from the emitter with the same time steps and durations defined by the time array (`t_array`). Based on the spatial distance between the emitters and receivers, we would expect some time for the signal to travel from each of the emitters to each of the receivers. The time it takes to travel is defined by the speed of sound (`sos`) and the distance traversed. Since there may be more than one emitter, add up all the emitted signals that arrive at a receiver for each time point.

4.1. Attributes

Objects of type `SoundSimulator` should have the following variable attributes:

`emitters` A 1-dimensional array of type `list`. Each element contains an object of type `Emitter`.

Each emitter in this list will emit sound.

`receivers` A 1-dimensional array of type `list`. Each element contains an object of type `Receiver`.

Each receiver in this list will receive sound from each of the emitters.

`t_array` A 1-dimensional array of type `list`. Each element contains a value of type `float`.

This represents time. Each element value should represent time in units of seconds.

`sos` A value of type `float`.

This represents the speed of sound in units of metres per second. The sound from the emitters will travel to the receivers at this speed.

4.2. Methods

`__init__(self, emitters, receivers, t_array, sos)`

Return values: None

Arguments: The *emitters* argument should be of type `list` and contain objects of type `Emitter`. This represents a list of emitters that will be simulated. The default value should be an empty list.

The *receivers* argument should be of type `list` and contain objects of type `Receiver`. This represents a list of receivers that will be simulated. The default value should be an empty list.

The *t_array* argument should be of type `list` and contain objects of type `float`. The default value should be an empty list.

The *sos* argument should be of type `float` and represents the speed of sound in units metres per second. The default value should be 1500.0.

`receivers = run(self)`

Arguments: None

Return values: Return an object of type `list` where each element is an object of type `Receiver`.

Implementation: Using the variable attributes attained after the initialisation process, run the simulation.

For each `Receiver` object in `self.receiver`, calculate the sound that is captured from each of the emitters. You will only need to record sound during the time period defined by `self.t_array`.

Sound from each emitter will have to travel over a period of time before it reaches each receiver. This delay can be determined by the speed of sound (`self.sos`).

The sound amplitude will reduce by 1/distance.

Sound from two or more emitters experience superposition. Add the two or more signals up while taking into account the time it took those signals to travel from the emitter to the receiver.

For each object of type `Receiver`, update its `signal` variable attribute to reflect the sound that it captured from one or more emitters. Do not alter the order of the receivers on the list. Make sure to not only update `self.receiver`, but also return an equivalent list of receivers.

5. `BeamFormer` Class

The `BeamFormer` class attempts to reconstruct the acoustic sources that may have been generated from a defined region of interest. This is done by using the receivers' signals and locations through a delay and sum algorithm. The acoustic source strength, $\mathbf{q}(\mathbf{r},t)$, is defined as follows:

$$q(\boldsymbol{r},t) = \frac{1}{N}\sum_{i=1}^{N} |\boldsymbol{r}_i - \boldsymbol{r}| p_i\left(\boldsymbol{r}_i, t + \Delta t_{i,relative}\right)$$

$\mathbf{r}$ represents the location where we want to reconstruct the acoustic source and $\mathbf{t}$ is time in seconds. N is the number of receivers with $\mathbf{r}_i$ being the location of the receiver. $\mathbf{p}$ is the signal captures at the receiver. $\mathbf{c}$ is the speed of sound in units metres per second. $|\boldsymbol{r}_i - \boldsymbol{r}|$ is the distance between a receiver and the reconstruction location. The space from where sound will be reconstructed will be defined by a two-dimensional region, which is specified by two 1-dimensional arrays specifying the x-axis (`x_array`) and the y-axis (`y_array`). $\mathbf{i}$ refers to the receiver index. The time delay Δt_i between $\mathbf{r}$ and $\mathbf{r}_i$ is defined as follows…

$$\Delta t_i = \frac{|\boldsymbol{r}_i - \boldsymbol{r}|}{c}$$

$$\Delta t_{i,relative} = \Delta t_i - \Delta t_{i,min}$$

Where $\Delta t_{i,min}$ is the shortest time delay between $\mathbf{r}$ and closest receiver. $\Delta t_{i,relative}$ is the relative time delay as defined above.

5.1. Attributes

Objects of type `Beamformer` should have the following variable attributes:

`receivers` A 1-dimensional array of type `list`. Each element contains an object of type `Receiver`.

Each receiver in this list will receive sound from each of the emitters.

`x_array` A 1-dimensional array of type `list`. Each element contains a value of type `float`.

This represents the space along the x-axis that you will reconstruct the acoustic source from. The unit for each value is metres.

`y_array` A 1-dimensional array of type `list`. Each element contains a value of type `float`.

This represents the space along the y-axis that you will reconstruct the acoustic source from. The unit for each value is metres.

`t_array` A 1-dimensional array of type `list`. Each element contains a value of type `float`.

This represents time. Each element value should represent time in units of seconds.

`sos` A value of type `float`.

This represents the speed of sound in units of metres per second. The sound from the emitters will travel to the receivers at this speed.

`field` A 3-dimensional array of type `list`. The first dimension contains an array of type `list`. The second dimension contains an array of type `list`, whose elements are of type `float`.

This represents the acoustic source strength, **q(r,t)**, that is reconstructed on a two-dimensional array whose spatial ranges are defined by `x_array` and `y_array`. The temporal length of the acoustic source strength signal should be the same length as `t_array`. The first dimension represents distance along the y-axis, the second dimension represents distance along the x-axis, the third dimension represents time.

5.2. Methods

`__init__(self, receivers, x_array, y_array, t_array, sos)`

Arguments: The *receivers* argument should be of type `list` and contain objects of type `Receiver`. This represents a list of receivers that will be simulated. The default value should be an empty list.

The *x_array* argument should be of type `list` and contain objects of type `float`. The default value should be an empty list.

The *y_array* argument should be of type `list` and contain objects of type `float`. The default value should be an empty list.

The *t_array* argument should be of type `list` and contain objects of type `float`. The default value should be an empty list.

The *sos* argument should be of type `float` and represents the speed of sound in units metres per second. The default value should be 1500.0.

Return values: None

Implementation: Create the `field` attribute using the lengths of `y_array`, `x_array`, and `t_array`. Upon initialisation, the `field` array should contain 0.0 values.

`generate_field(self)`

Arguments: None

Return values: None

Implementation: Using the variable attributes attained after the initialisation process, run the simulation to update the `field` attribute as defined by the equation for **q(r,t)**.

When calculating the source strength, the delay and summing algorithm may result in summing values outside the bounds of the allocated durations. For out of bound values, assume that the signal amplitude is 0.

Implementation of `generate_field()` is the most difficult part of this assignment and also worth the most marks.

6. Coding rules

Do not declare any variables or functions in the global space of `simulator.py`. The global space definitions must have the 5 classes requested only. A `main()` function could be used if you so desire, but is not required. Only the `math` module can be imported in `simulator.py`.

7. Marking criteria

We will mark your submitted `simulator.py` code according to the following categories:

- 90 marks: Implementation and efficiency
  - 25 marks: base marks awarded for demonstration of basic coding skills
  - 05 marks: code compiles
  - 15 marks: `Transducer`, `Emitter`, and `Receiver` class implementation
  - 15 marks: `SoundSimulator` implementation and efficiency
    - 10 marks for implementation
    - 5 marks for efficiency (only awarded with full 10 implementation marks awarded)
  - 30 marks: `BeamFormer` implementation and efficiency
    - 15 marks for implementation
    - 15 marks for efficiency (only awarded with full 15 implementation marks awarded)
- 10 marks: Coding style and commenting

We will import your `simulator` module near the top of our script to test your code. We will run several test conditions against each of your classes and methods. We will investigate what was assigned in your attribute variables, so the type, length, etc. of your attribute variables must be accurately defined. We will test far more test cases than the examples we have included in this assignment sheet.

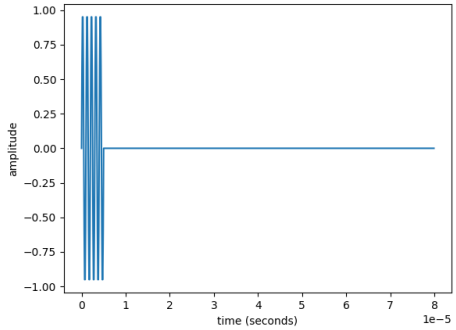
7. Sample Code

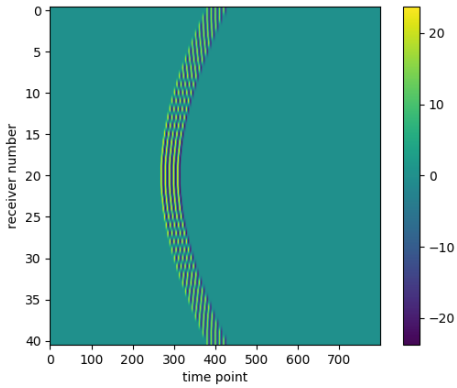
The following code…

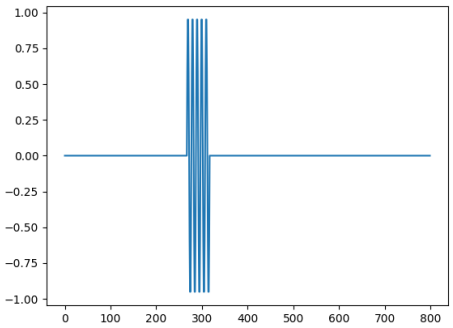
```
# script.py

from matplotlib import pyplot    # you may use whatever plotting library of software you want
                           # you will need to install this module before using it
                           # https://matplotlib.org/stable/index.html

from simulator import *


def main():

   # ------------------------------------------------------------------
   # setup the 'clock' with which the simulation will be run

   t_delta = 0.1e-6                        # time step size [seconds]
   t_N = 800                               # number of time points

   t_array = [t_i*t_delta for t_i in range(t_N)]  # the master clock, array of time [seconds]

   sos = 1500.0                            # speed of sound [metres/second]


   # ------------------------------------------------------------------
   # setup receivers and emitters

   # arrange a linear order of receivers from -0.04 m to 0.04 m
   receiver_positions = [[receiver_x / 1000.0, 0.04] for receiver_x in range(-40, 41, 2)]

   emitters = []
   receivers = []

   emitters.append(Emitter(0, 0, t_array))
   signal = emitters[0].generate_signal(1e6, 5, 1)

   fig, axs = pyplot.subplots(1)
   axs.plot(t_array, signal)
   axs.set_xlabel('time (seconds)')
   axs.set_ylabel('amplitude')
   # pyplot.show()

   for pos in receiver_positions:
      receivers.append(Receiver(pos[0], pos[1], t_array))

   # ------------------------------------------------------------------
   # run the simulation

   simulation = SoundSimulator(emitters, receivers, t_array, sos)
   receivers = simulation.run()

   img = []
   for i in range(len(receivers)):
      img.append(receivers[i].signal)

   fig, axs = pyplot.subplots(1)
   imgplot = axs.imshow(img)
   axs.set_aspect(20)
   axs.set_xlabel('time point')
   axs.set_ylabel('receiver number')
   fig.colorbar(imgplot)
   pyplot.show()

   # ------------------------------------------------------------------
   # set the field of view from which we will visualise sound

   x_range = [-0.02, 0.02]                  # x-axis range [metres]
   x_N = 51                              # number of points along x axis
   x_delta = abs(x_range[0]-x_range[1])/(x_N-1) # step size along x axis [metres]

   y_range = [-0.02, 0.02]                  # y-axis range [metres]
   y_N = 51                              # number of points along y axis
   y_delta = abs(y_range[0]-y_range[1])/(y_N-1) # step size along y axis [metres]

   x_array = []
```

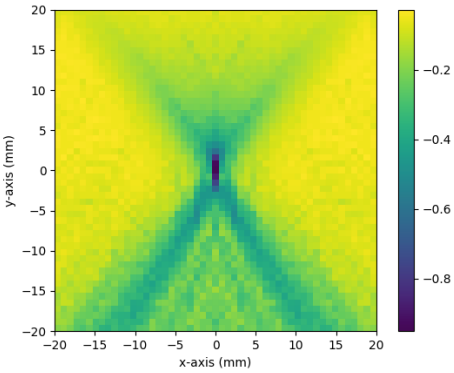
```
    for i in range(x_N):
        x_array.append(x_range[0]+i*x_delta)

    y_array = []
    for i in range(y_N):
        y_array.append(y_range[0]+i*y_delta)


    # ----------------------------------------------------------------
    # reconstruct a field, where sound may be located

    beam_former = BeamFormer(receivers, x_array, y_array, t_array, sos)
    beam_former.generate_field()

    fig, axs = pyplot.subplots(1)
    axs.plot(beam_former.field[25][25])

    img = []
    for j in range(len(y_array)):
        row = []
        for i in range(len(x_array)):
            row.append(min(beam_former.field[j][i]))
        img.append(row)

    fig, axs = pyplot.subplots(1)
    imgplot = axs.imshow(img, origin='lower', extent=[x_array[0]*1000, x_array[-1]*1000,
y_array[0]*1000, y_array[-1]*1000])
    axs.set_xlabel('x-axis (mm)')
    axs.set_ylabel('y-axis (mm)')
    fig.colorbar(imgplot)
    pyplot.show()

if __name__ == '__main__':
    main()
```

Should produce the following figures:
note: we will mark class attributes, and NOT your figures.






1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400
500
600
700
800


20
15
10
5
0
−5
−10
−15
−20
y-axis (mm)
−20
−15
−10
−5
0
5
10
15
20
x-axis (mm)
−0.2
−0.4
−0.6
−0.8