Audio Visualizer

Autor: Mamut Nuray, 331AB

An: 3

Cuprins:

1. Introducere

1.1. Scopul lucrarii:

Sunetul este un concept abstract pentru majoritatea, nimeni nu se gandeste atunci cand rosteste ceva cum se desfasoara acesta actiune sau cum se poate receptiona acest mesaj trimis, totul se intampla in mod natural, fara a considera ce este in spatele acestui proces. Provocarea apare atunci cand se incearca reproducerea prin mijloace nenaturale, de exemplu captarea undelor vocale si reproducerea la perfectie a secventei rostite. Inregistrarile sonore dateaza din a doua jumatate a secolului al XIX-lea, si nu sunt ceva revolutionar in zilele noastre, cand progresul tehnologic creste exponential din an la an, dar cu toate astea nu multa lume intelege ce este sunetul, cum este reprodus sau ce auzim mai exact.

Audio visualizer-ele (sau vizualizatoare audio) se pot gasi peste tot, de la clipuri video pentru melodii, incarcate pe internet, pana la mesaje vocale trimise online, este evident ca acestea nu sunt nimic nou, dar sunt ceva destul de interesant cand sunt analizate. Cum complexitatea sonora poate sa fie reprezentata doar de cateva caracteristici cheie, care sunt expuse cu grija, utilizand efectele vizuale, creeand un intreg care captiveaza privitorul. Acestea sunt asa accesibile publicului larg, gasindu-se in atatea varietati la doar o simpla cautare pe Google, fiind limpede ca exista o fascinatie, care este datorata armoniei dintre aspectul vizual si a celui auditiv, completandu-se perfect unul pe celelat.

Un audio visulizer este construit pe baza undelor sonore, fiind o reprezentare concreta a sunetului, fie ca e vorba de o melodie, fie de zgomotul inconjurator. Acesta poate oferi o intelegere minimala asupra ideii de captare si reproducere a sunetului intr-un mod prietenos pentru cineva fara nicio cunostinta in domeniul aferent. Este ceva de necontestat ca vizualizarea unui fenomen este mult mai eficienta, in comparatie cu studiul clasic, mai ales in contextul unui subiect abstract.

Prin urmare, scopul lucrarii este sa faca sunetul ceva palpabil, demistificand conceptele din spatele definitiilor complicate, in asa fel in cat oricine ar putea sa inteleaga ideea de unda sonora, cum se leaga de frecventa si cum semnalele sinusoidale nu sunt asa inspaimantatoare, fiind doar blocurile primare ale sunetelor pe care le putem auzi.

1.2 Prezentare aplicatie:

Aplicatia prezentata in aceasta lucrare se foloseste de input-ul de la microfon pentru a capta informatia audio necesara, care urmeaza sa fie procesata in timp real si reprezentata grafic, intr-un format 3D, fluxul audio este continuu (nu se opreste pana nu se inchide aplicatia). Pentru implementarea aplicatiei s-a folosit python, impreuna cu functionalitatile de OpenGL prezente in interfata grafica utilizator PyQtGraph pentru reprezentarea grafica 3D.

2. Concepte de baza si suport tehnic necesar

2.1. Caracteristicile Sunetului

“Sound is one type of longitudinal-mechanical waves that we can sense with our ears, when sound waves travel into our ears and reaches the eardrum, it oscillate at the same frequency of the sound wave which, these oscillation is sent to the brain as electric signals which make us hear. Sound is produced when the source vibrates. Such as a string of a violin or the membrane of drum, the vibrating source vibrates the air molecules next to it causing a pressure variation in the air, we must differentiate between the movement of the molecules itself (which moves back and forth about its equilibrium position, a characteristic of longitudinal waves.) and the compression wave which is the sound. The sound frequency equals the frequency of the vibrating source producing it, A single-frequency sound wave traveling through air will cause a sinusoidal pressure variation. “[2]

Dupa cum se expune mai sus, sunetul este rezultatul vibratiilor, prin oscilatiile moleculelor de aer. Aceasta oscilatie a moleculelor duce la o variatie a presunii aerului, creeand unda sonora vizibila in fig1.

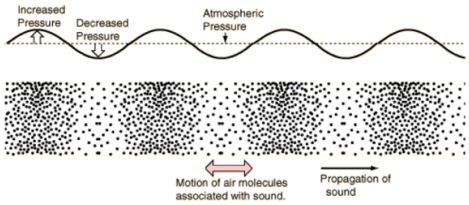

fig. 1[8]

Semnal poate fi caracterizat de ampitudine (cat de inalt este semnalul), perioada (lungimea unui ciclu) si frecventa (este invers proportionala cu perioada), ca in fig2. Cand unda sonora are frecventa constanta se observa ca semnalul este chiar o sinusoida pura.

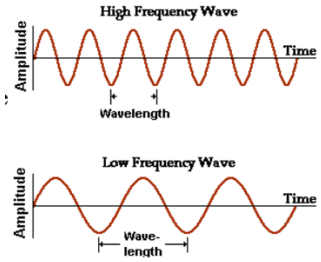

fig. 2[5]

Analiza Fourier

“The mathematician Fourier proved that any continuous function could be produced as an infinite sum of sine and cosine waves. His result has far-reaching implications for the reproduction and synthesis of sound. A pure sine wave can be converted into sound by a

loudspeaker and will be perceived to be a steady, pure tone of a single pitch. The sounds from orchestral instruments usually consists of a fundamental and a complement of harmonics, which can be considered to be a superposition of sine waves of a fundamental frequency f and integer multiples of that frequency." [6]

In natura nu exista perfectiune, iar sunetul nu face exceptie, dar dupa cum a fost demonstrat de catre Fourier, orice semnal poate fi descompus in mai multe sinusoide de diferite frecvente pentru a fi recreat ca o suma de mai multe semnale individuale

- Seria Fourier
  "Seriile Fourier sunt o unealtă matematică folosită pentru a analiza funcțiile periodice descompunându-le într-o sumă ponderată de funcții sinusoidale componente care sunt uneori denumite armonice Fourier normale, sau pe scurt armonice." [18] fig. 3

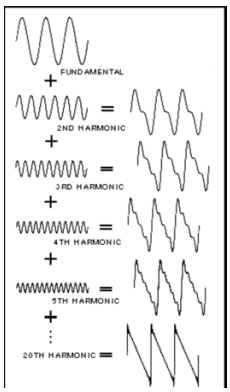

fig. 3

- Transformata Fourier
  "Although Fourier series can represent periodic waveforms as the sum of harmonically-related sinusoids, Fourier series can't represent *non-periodic* waveforms. However, the Fourier transform is able to represent *non-periodic* waveforms as well. It achieves this by applying a limiting process to lengthen the period of any waveform to infinity and then treating that as a periodic waveform."[17]

2.2. Suport tehnic

Principalele carateristici a proiectului sunt :

- Secventa audio trebuie sa fie redata de la microfon
- Procesarea datelor in timp real
- Afisare in timp real caracteristici input audio intr-o interfata grafica

Aplicatia este structurate in 2 parti:

1. Colectarea datelor din secventa audio si analizarea acestora

In programul [] utilizeaza biblioteca PyAudio[12] pentru inregistrarea datelor utilizand metoda open() pentru pornirea fluxului de date de la microfon, iar dupa urmand colectarea acestora intr-un buffer cu metoda read(), intrucat aceasta returneaza un obiect de tip bytes trebuie convertit in formatul int16 (cu valori de la -32768 pana la 32767). Urmatorul pas este utilizarea transformatei fourier rapida pentru a schimba domeniul de timp in domeniul frecventei, analiza undei sonore fiind in functie de frecventa in acest caz.

O alta modalitate de a realiza aceasta operatiunea poate fi gasita in programul dat ca exemplu pentru bibloteca souddevice[16] , inconvenientul fiind complexitatea codului

Bibliteca librosa[4] prezinta metrode ajutatoare in analiza sunetului, nefiind nevoie de conversie pentru a putea afisa forma semnalului, dar aceasta nu prezinta posibilitatea de a capta input-ul de la microfon, fiind foarte dezavantajoasa in implementarea aplicatie curente

2. Vizualizarea datelor

matplotlib[6]:

Reprezinta cea mai simpla metoda de vizualizare, trebuie doar sa se fixeze limitele axelor, exista metode similare utilizand PyQtGraph[1], Librosa[4], PyGame[3]

O alta metoda intalnita este de a grupa frecvrentele impreuna cu amplitudinea respectiva (o lista cu indexul frecventa si valoarea sa fie amplitudinea), acestea se impart in sectiuni si se face media ampitudinilor in functie de sectiune (pentru a reduce numarul de frecvente). Se creeaza puncte/patrate care reprezinta cate o sectiune cu amplitudinea valorii mediei sectiunii corespunzatoare.[19]

In programul [7] nu se face vizualizarea datelor, dar este implementata o animatie 3D in PyQtGraph.OpenGL. Animatia finala este un teren care se misca cu ajutorul adaugarii de zgomt unui obiect de tipul gl.GLMeshItem()

3. Prezentare tehnica a etapei de realizare/implementare

3.1. Biblioteci utilizate

3.1.1. PyAudio

Biblioteca PyAudio se afla la baza proiectului, oferind capabilitatea de a inregistra si a reda secvente audio. PyAudio creeaza legaturi intre python si API-ul AudioPort, cu baza in C++. Aceasta ofera o simplitate codului, fiind usor de utilizat in program, comparativ cu alte biblioteci de care au functionalitatea de a inregistra audio. Aplicatia dezvoltata se va folosi doar de caracteristica de inregistrare pentru a receptiona undele audio primite de a microfon.

Metodele folosite in aplicatie:

- pyaudio.PyAudio() -> creeaza un obiect de tipul PyAudio
- pyaudio.PyAudio().open(format, channels, rate, input, output, frames_per_buffer) -> porneste inregistrarea si creeaza un obiect de tip Stream, captand fluxul de date care provine de la sursa de input (in cazul de fata, microfon)
- pyaudio.Stream.read(frames_per_buffer) -> citeste datele dintr-un cadru

3.1.2. PyQtGraph

Este o inerfata grafica utilizator nativa pentru python cu baza in PyQt/PySide si NumPy, avantajul acesteia fiind viteza de procesare si afisare a datelor, carcteristici necesare cand se lucreaza cu date in timp real, utilizandu-se pentru aplicatii stiintifice si de inginerie.

Metodele folosite in aplicatie:

- QtGui.QApplication(sys.argv)
- QtGui.QApplication.instance().exec_()
- QtCore.QTimer()

PyQtGraph.OpenGL: PyQtGraph se foloseste de OpenGL pentru reprezentari 3D, aceasta functionalitate este inca in starea de development, prezentand doar cateva capabilitati regasite in OpenGL.

Metodele folosite in aplicatie:

- gl.GLViewWidget(): creeaza o fereastra in perspectiva 3D, cu controale pentru marire/micsorare, rotire
- gl.GLViewWidget().setWindowTitle(), .setGeometry(x, y, w, h), .setCameraPosition(distance, elevation), .show(): seteaza titlul, geometria, pozitia camerei si afiseaza ferastra creata
- gl.GLViewWidget().addItem(obj): adauga un element ferestrei create
- gl.GLMeshItem(faces, vertexes, faceColors, drawEdges, smooth): creeaza o plasa poligonala de tip triunghiular
- gl.GLMeshItem().setMeshData(faces, vertexes, faceColors, drawEdges, smooth): (re)atribuie caracteisticile precizate ale obiectului
- gl.GLMeshItem.setGLOptions(): seteaza distributa de culare plasei create

3.1.3. OpenSimplex[14]

OpenSimplex noise is an n-dimensional gradient noise function that was developed in order to overcome the patent-related issues surrounding Simplex noise, while continuing to also avoid the visually-significant directional artifacts characteristic of Perlin noise.

Metodele folosite in aplicatie:

- OpenSimplex(): creeaza un obiect
- self.noise.noise2(): adauga zgomot

3.1.4. Struct [11]

Este capabil sa faca conversia dintre valorile din Python si structurile din C, reprezentate ca Python String

Metoda folosita in aplicatie:

- struct.unpack(): conversie byte in string

3.1.5. **NumPy** [13]

Metodele folosite in aplicatie:

- np.arange()
- np.array()
- reshape()

3.1.6. **sys** [15]

Modulul ofera utilizatorului accesul la variabile folosite de interpretor sau accesul la functii care interactioneaza puternic cu interpretorul. Este mereu disponibil

Metoda folosita in aplicatie:

- sys.argv: “The list of command line arguments passed to a Python script. argv[0] is the script name (it is operating system dependent whether this is a full pathname or not). If the command was executed using the -c command line option to the interpreter, argv[0] is set to the string '-c'. If no script name was passed to the Python interpreter, argv[0] is the empty string.”

3.2. Vizualizarea in PyQtGraph.OpenGL

- ideea de baza:
    1. Se creeaza un obiect plasa poligonala de tip triunghiular
    2. Se seteaza coordonatela punctelor (x, y, z) astfel incat punctele sa formeze o matrice patratica din perspectiva 2D (“de sus”, fara sa fie luata in considerare inaltimea z), acestea o sa fie pastrate in vectorul varfuri, de tipul (Nv, 3)
    3. In vectorul fete este de tipul (Nf, 3) si primeste 3 valori (care creeaza un triunghi) pentru un element, aceste valori reprezita indecsii vectorului varfuri, deci indecsii matricei patratice create mai sus; fiecare 4 puncte care formeaza un patrat de lungime minima vor fi vazut ca 2 triunghiuri, in alte cuvinte 2 elemente din vectorul fete (de exemplu a[0][0], a[0][1], a[1][0], a[1][1] o sa formeze doua triunghiuri: (a[0][1], a[0][0],a[1][0]) si (a[0][1], a[1][1],a[1][0]))
- implementare:
    1. pentru coordonatele stocate in vectorul varfuri se utilizeaza 2 vectori identici xp, yp de lungime 40 cu vaori de la [-20, 20] initializati inainte

2. se folosesc 2 for-uri pentru a creea matricea de care am amintit mai sus pentru coordonatele x, y, iar pentru z se folosi o matrice care are datele extrase din unda sonora
3. pentru vectorul fete similar doar ca se pune accent pe indeccsii vectorului varfuri

Aceasta idee a fost gasita in [7]

3.3. Arhitectura aplicatiei

Clasa AudioVisualizer:

- __init__:
  1. Cand se creeaza un obiect de tip AudioVisualizer() se initializeaza obiectul de tip gl.GLViewWidget() (se seteaza un titlu, geometria, pozitia camerei) si se afiseaza fereastra creeata
  2. Se porneste inregistrarea, utilizand biblioteca PyAudio
  3. Urmeaza sa fie construite valorile pentru vectorii varfuri, fete si culori in functia plasa()
     a. La prima apelare a functiei se initalizeaza un vectorul data pentru a colecta datele undei sonore, de dimensiunea corespunzatoare numarului de esantioane dintr-un cadru, fara a colecta datele audio, acesta este transformat intr-o matrice patratica dimensiune xp
     b. se creeaza vectorul de varfuri
     c. se creeaza vectorul de fete
     d. Pentru fiecare fata se adauga o culoare (in cazul acesta fetele au aceeasi culoare)
  4. Se creeaza obiectul de tip gl.GLMeshItem() cu ajutorul valorilor construite in functia plasa() si se adauga la obiectul fereastra pentru a fi vizibil
- start:
  1. porneste procesul
- update:
  1. citeste datele undei audio in vectorul data
  2. apeleaza functia plasa()
     a. Intrucat vectorul data a fost deja initializat, se vor analiza datele din acesta, in rest procesul fiind identic cu apelarea initala a functiei plasa() prezentata mai sus
  3. se apeleaza metoda setMeshData() care va updata valorile pentru vertexes, faces, faceColors pentru a putea fi vizibila schimbarea undei sonore
- animatie:
  1. se porneste timer-ul setat la 30 de secunde
  2. se cheama metoda start()
  3. cand se termina un ciclu se cheam metoda update()
  4. si se repeta pana la inchiderea ferestrei

4. Mod de utilizare aplicatie

4.1. Testare functionalitati

4.1.1. Verificare date

Initial se doreste sa se testeze modalitatea de colectare si procesare date, in acest sens s-a folosit [6] ca punct de plecare si intelegere a procesului in sine

In program s-a adaugat afisarea graficului in domeniul frecventei cu ajutorul transormatei fourier rapida fig.4.

```
while 1:
    data = stream.read(CHUNK)
    dataInt = struct.unpack(str(CHUNK) + 'h', data)
    line.set_ydata(dataInt)
    line_fft.set_ydata(np.abs(np.fft.fft(dataInt))*2/(11000*CHUNK))
    fig.canvas.draw()
    fig.canvas.flush_events()
```

fig. 4

Pentru teste s-a folosit un generator de tonuri pentru a observa functionalitatea corecta. Testele se fac pentru frecventele: 50Hz (fig. 5), 500Hz (fig. 6) si 5000 Hz (fig. 7)

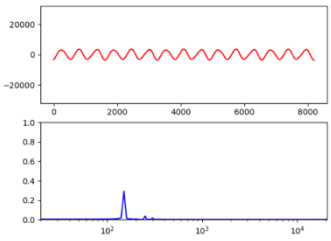

fig. 5

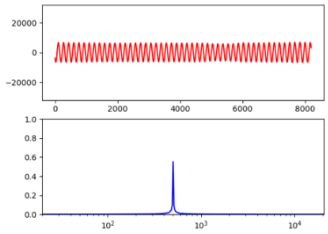

fig. 6

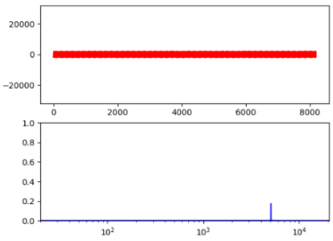

fig. 7

Cu ajutorul acestor teste se poate observa foarte clar relatia dintre frecventa si forma semnalului rezultat intarind teoria explicata anterior.

Se observa clar ca aplicatia receptioneaza si transmite unda sonora in modul dorit, singura problema este data de micile perturbatii care afecteaza semnalul care intra in microfon (vizibile in fig. 5), dar acestea sunt de asteptat in contextul transmiterii datelor prin intermediul microfonului intr-o incapere care nu este izolata fonic.

4.1.2 Vizualizarea

Pentru modelul grafic s-a plecat de la programul [7] ca baza a vizualizarii.

S-au implementat cateva modificari la aparenta obiectului gl.GLMeshItem(), si la perspectiva ferestrei pentru a ajunge la forma finala dorita fig. 8:

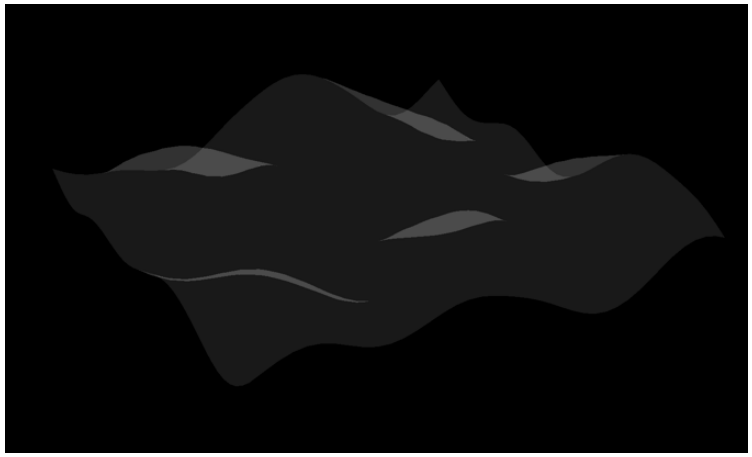

fig. 8

4.1.3 Combinarea datelor cu vizualizarea

Pentru a vizualiza datele este necesara legarea acestora de modelul realizat mai sus, observandu-se urmatoarele schimbari importante ale codului:

- adaugarea bibliotecii PyAudio, si utilizarea acesteia pentru a primi input de la microfon fig. 9

```
self.p = pa.PyAudio()
self.stream = self.p.open(
    format = pa.paInt16,
    channels = 1,
    rate = self.RATE,
    input=True,
    frames_per_buffer=self.CHUNK,
)
```

fig. 9

- modificarea constructiei vectorului varfuri, inmultind parametrul pentru axa z cu elementul corespunzator din matricea de valori ale undei sonore, astfel incat inaltimea obiectului se modifica o data cu datele de la microfon
- crearea metodei plasa() care modifica vectorii varfuri, fete, culori
- adaugarea in metoda update() citirea datelor si trimitera acestora pentru a modifica valorile vectorilor varfuri, fete, culori
- pentru analiza datelor undei sonore s-a ales vizualizarea schimbarii amplitudinii

Aplicatia finala reprezinta un audio visualizer care se foloseste de catre microfon pentru a capta undele audio, si a reprezenta grafic amplitudinea semnalului de intrare intr-un format 3D la care este adaugat un zgomot, procesul fiind unul continuu.

4.1.4. Teste finale:

Se vor realiza teste la aceleasi frecvente ca cele anterioare (50Hz fig.10, 500Hz fig. 11, 5000Hz fig. 12), pentru a verifica functionalitatea aplicatiei finale:

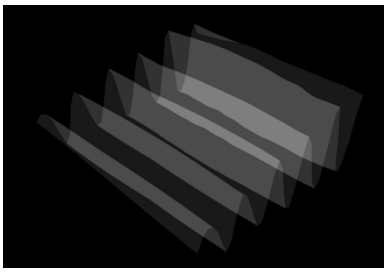

fig. 10

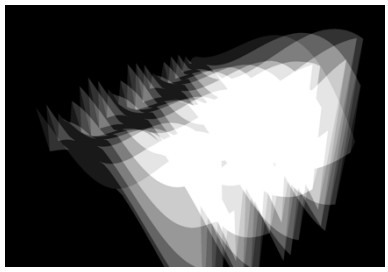

fig. 11

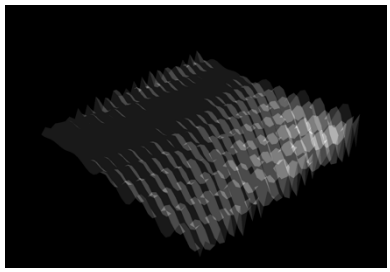

fig. 12

Chiar daca aplicatia nu reprezinta caracteristica de frecventa, aceasta inca se poate observa, avand in vedere relatia dintre forma undei sonore, frecventa si perioada (cu cat frecventa este mai mare perioada devine mai mica, cu alte cuvinte semnalul devine mai rapid). Cand se compara rezultatele cu cele anterioare se poate concluziona ca acestea sunt similare, singurele diferente fiind reprezentarea 3D si adugarea zgomotului.

4.2 Interactiunea cu utilizatorul

Pentru a folosi aplicatia este necesar un enviroment python si instalarea bibliotecilor prezentate mai sus. In plus, existenta unui microfon.

Se ruleaza codul si apare fereastra aplicatiei care prezinta vizualizarea grafica a semnalului de la microfon. Fereastra se poate mari sau micsora, dupa preferinta, prezentand capabilitati de zoom-in/zoom-out, iar perspectiva se poate schimba, prin apasare si miscare a mouse-ului. Aceasta se inchide doar cand se apasa butonul de iesire al ferestrei.

5. Concluzii

Intr-o lume plina pe tehnologii inovatoare, dorinta de cunoasterea este extrem de importanta deoarce fara aceasta nu ar fi existat aceasta evolutie, umanitate stagnand intr-o realitate monotona, plina de concepte straine.

Undele sonore sunt peste tot si au existat mereu, ele stau la baza umanitatii in sine, iar traiul intr-o viata fara acestea devine un gand inspaimantator pe care nimeni nu si l-ar dori. De aceea intelegerea sunetului devine ceva intrigant, lamurirea neclaritatilor si a idelilor abstracte duc catre cunoastere, iar cunoastera catre inovatie si creare.

Proiectul deschide o usa pentru aceia care nu au avut aceasta problematica in vedere sau poate au fost prea coplesiti de teoria care sta in spatele subiectului, dar, cum se poate observa din aplicatie, sunetul nu este nimic inspaimantator, ci doar o vibratie nergulata care poate fi descompusa in mai multe semnale.

6. Referinte bibliografice

1. Schwarden (2016). https://swharden.com/blog/2016-07-31-real-time-audio-monitor-with-pyqt/
2. Ahmed Abokhalil. (2020) On the nature of sound
3. (2021) https://www.henryschmale.org/2021/01/07/pygame-linein-audio-viz.html
4. Analyticsindiamag, https://analyticsindiamag.com/step-by-step-guide-to-audio-visualization-in-python/
5. Computer Science Toronto, https://www.cs.toronto.edu/~gpenn/csc401/soundASR.pdf
6. Fazals, https://fazals.ddns.net/spectrum-analyser-part-1/
7. GitHub,https://github.com/markjay4k/Audio-Spectrum-Analyzer-in-Python/blob/master/terrain.py
8. Harvard, https://scholar.harvard.edu/files/schwartz/files/lecture6-waves.pdf
9. Hyperphisics, http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/fourier.html
10. Rochester, http://astro.pas.rochester.edu/~aquillen/phy103/Lectures/D_Fourier.pdf
11. Journaldev, https://www.journaldev.com/17401/python-struct-pack-unpack
12. MIT, https://people.csail.mit.edu/hubert/pyaudio/docs/
13. NumPy. Web-site https://numpy.org/
14. Pypi, https://pypi.org/project/opensimplex/
15. Python, https://docs.python.org/3/library/sys.html
16. Python, https://python-sounddevice.readthedocs.io/en/0.3.14/examples.html#plot-microphone-signal-s-in-real-time
17. YouTube, https://www.youtube.com/watch?v=G4W1hLLNcic&t=349s