T2 Citirea, scrierea, redarea și afișarea semnalelor vocale

T2.1 Vorbirea. Caracteristici fundamentale 52

T2.2 Fundamente ale achiziției și stocării semnalului vocal 54

T2.3 Citirea semnalului vocal din fișier 56

T2.4 Redarea semnalului vocal 61

T2.5 Vizualizarea semnalului vocal 62

T2.6 Scrierea eșantioanelor audio în fișier . 65

T2.7 Concluzii 66

După scurta introducere în Python, primul tutorial de prelucrare a semnalului vocal se va axa pe principalele module și funcții utilizate pentru procesarea fișierelor audio. Totodată, vor fi introduse și noțiuni de bază referitoare la producerea vorbirii și modelele matematice/programatice de producere a vorbirii, precum și noțiuni referitoare la achiziția și stocarea digitală a semnalului vocal.

T2.1. Vorbirea. Caracteristici fundamentale

Comunicarea reprezintă una dintre caracteristicile cele mai importante ale evoluției speciei umane. Prin comunicare se realizează de fapt transferul de informație de la sursă la destinație cu ajutorul limbii sau a unui limbaj comun. Acest transfer poate fi efectuat într-o formă scrisă sau verbală. Dacă forma scrisă este de cele mai multe ori o codare brută a mesajului, folosind un set finit de simboluri grafice, forma verbală cuprinde un univers complex, potențial infinit, de factori și ele- mente care interacționează pentru a facilita transmiterea mesajului către destinatar. Alături de fonemele ce alcătuiesc mesajul, caracteristicile fiziologice ale vorbitorului, accentul, intonația și ritmul vorbirii, precum și elementele non-verbale (mimică, gestică, poziția corpului, etc.) determină modul în care destinatarul decodează și interpretează mesajul transmis. Astfel că același mesaj codat în mod identic în formă scrisă, nu va fi niciodată reprodus identic în mod verbal, nici chiar de către același vorbitor la intervale consecutive, scurte de timp. Această variabilitate de codare a mesajului în formă verbală reprezintă una dintre cele mai importante provocări din domeniul mod- elării inteligenței și a fiziologiei umane. Această provocare se referă atât la metodele de sinteză a vorbirii, precum și la cele de recunoaștere a acesteia.

Din punct de vedere fiziologic, sistemul vocal uman este compus din ansamblul organelor fona- toare: plămâni, esofag, laringe, glotis (corzi vocale), faringe, cavitatea orală, cavitatea nazală, limbă, văl palatin, maxilar și buze. În Fig. T1.1 este prezentat acest ansamblu cu excepția plămânilor. Aceștia din urmă reprezintă sursa vorbirii prin expirația aerului, iar restul organelor contribuie la modularea acestui flux de aer și generarea sunetului perceput în exteriorul corpului fie de către unul sau mai mulți observatori, fie de un dispozitiv de achiziție a sunetului, cum ar fi un microfon.

Pornind de la sistemul anatomic de producere a vorbirii, modelarea acestui proces complex necesită anumite simplificări și aproximări, multitudinea proceselor implicate și complexitatea lor neputând fi modelată corect în totalitate. Astfel că există mai multe modele acceptate ca fiind valide pentru modelarea vorbirii din punct de vedere matematic, iar în continuare sunt prezentate cele mai importante dintre acestea:

- modelul tuburilor acustice;
- modelul filtrelor trece-bandă;
- modelul sinusoidal;
- modelul liniar-separabil sau sursă-filtru;
- modelul undelor glotale.

Prin modelarea producerii vorbirii, aplicațiile de codare, recunoaștere sau sinteză a semnalului vocal pot fi mult mai ușor implementate. Rezultatele lor apropiindu-se în ultimii ani de calitatea vocii naturale.

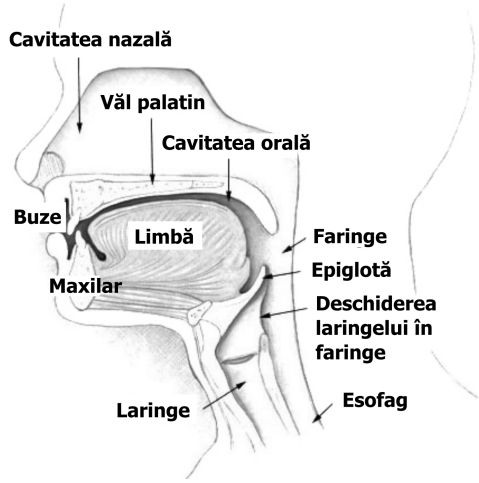

Fig.T2.1. Fiziologia sistemului vocal uman

T2.2. Fundamente ale achiziției și stocării semnalului vocal în format digital

Semnalul vocal poate fi achiziționat folosind un microfon. Cu ajutorul diafragmei, microfonul realizează conversia mișcării particulelor de aer generate de fonație, în curent electric (semnal).

Semnalul astfel rezultat poate fi stocat fie în format analogic, fie digital. Deși în domeniul analogic, informația stocată este în mod teoretic fără pierderi de informație, acest format face mai dificilă analiza și postprocesarea semnalului înregistrat. Astfel că, cel mai comun mod de stocare în ziua de azi, este cel digital.

Formatul de stocare digital sau conversia analog-digitală a semnalului implică un număr de aproximări și pierderi de informație. Dintre acestea, cele mai importante sunt:

- **frecvența de eșantionare (Fs)** (aproximarea în domeniul timp): reprezintă numărul de eșantioane de semnal dintr-o secundă ce vor fi stocate. Unitatea de măsură penrtu aceasta este Hertz-ul [Hz], iar cele mai utilizate valori ale frecvenței de eșantionare ale unui semnal vocal sunt: 8kHz, 16kHz și 48kHz. O regulă esențială pentru selectarea

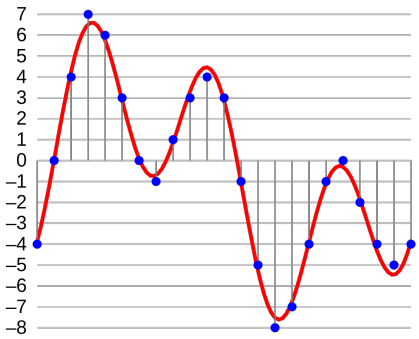

Fig.T2.2. Conversia digitală a semnalului - forma de undă analogică este redată cu roșu, iar reprezentarea digitală cu albastru. Barele verticale reprezintă aproximarea în timp, dată de perioada de eșantionare. Punctele albastre reprezintă aproximarea în amplitudine.

valorii frecvenței de eșantionare este bazată pe teorema lui Nyquist, care spune că **frecvența de eșantionare a unui semnal trebuie să fie egală cu cel puțin dublul frecvenței utile maxime din semnalul de intrare**. În cazul vorbirii, frecvența maximă este de aproximativ 10kHz, astfel că o frecvență de eșantionare de 22kHz ar fi cea corectă și suficientă. Însă, majoritatea informației utile din semnalul vocal este situată în banda de 300-3400Hz. Astfel că, în versiunile inițiale ale sistemelor de achiziție de semnal vocal (sau telefonie), frecvența de eșantionare era setată la 8kHz.

- **rezoluția de bit (sau numărul de biți/eșantion - en. bits per sample - bps)** (aproximare în domeniul amplitudinii): este numărul de biți pe care este stocată informația din fiecare eșantion de semnal și este direct proporțional cu rezoluția de amplitudine a eșantioanelor. Valori comune utilizate în prelucrarea digitală a semnalului vocal sunt: 8bps, 16bps sau 32bps. Rezoluția de bit ne dă acuratețea aproximării amplitudinii semnalului.

În ceea ce privește stocarea datelor audio, cel mai utilizat format digital este **WAV**: `https://en.wikipedia.org/wiki/WAV` . Acesta nu codează semnalul digital în niciun fel, ci stochează eșantioanele ca atare. Alte formate uzuale pentru semnalul audio sunt: MP3, OGG sau FLAC. Acestea realizează o anumită codare/compresie a semnalului și pot mări gradul de distorsiune al semnalului original.

Odată ce semnalul vocal este în format digital, acesta poate fi analizat și procesat folosind diverse unelte și metode. Cea mai simplă analiză este reprezentată de vizualizarea formei de undă în domeniul timp. Prin simpla vizualizare a valorilor eșantioanelor semnalului vocal achiziționat, o serie de caracteristici specifice pot fi extrase. Cele mai evidente sunt: periodicitatea, amplitudinea și numărul de treceri prin zero ale semnalului.

O altă metodă fundamentală de analiză a semnalului vocal este cea de analiză în frecvență sau analiza spectrală. Spectrul este obținut cu ajutorul transformatei Fourier și va fi discutat într-un capitol viitor.

Următoarele secțiuni ale acestui tutorial se vor axa pe accesarea datelor audio din fișier și vizualizarea acestora.

T2.3. Citirea semnalului vocal din fișier

Citirea din fișier a eșantioanelor unui semnal digitizat necesită cunoașterea formatului de stocare al datelor din fișier. Astfel că sunt necesare o serie de funcții specializate ce pot interpreta acest format prin identificarea antetului specific, a eventualei codări și a datelor sau valorilor eșantioanelor în sine. Rezultatul acestor funcții este de obicei o structură specifică ce conține datele într-un format interpretabil de limbajul de programare utilizat. În cele mai multe cazuri, eșantioanele semnalului sunt returnate sub forma unei matrici de valori întregi sau reale. Pe lângă eșantioane, este important să obținem și caracteristicile acestor, precum frecvența de eșantionare la care s-a făcur digitizarea, rezoluția de bit și numărul de canale audio (mono sau stereo).

În Python, unul dintre modulele ce conține funcții specializate pentru prelucrarea fișierelor audio de tip `*.wav` este modulul `wave`.[1] Astfel că, dacă dorim să utilizăm aceste funcții, va trebui să importăm modulul `wave` în codul nostru:

[1]:
```
import wave
```

OBS T2.1 În loc de sintaxa anterioară, putem importa toate funcțiile modulului fără a mai fi necesară prefixarea acestora:

```
from wave import *
```

Însă, dacă utilizăm acest format s-ar putea să suprascriem funcții de bază ale Python, cu funcții definite doar în modulul `wave`. De exemplu, funcția `open` ce poate opera asupra oricărui fișier, va fi înlocuită cu cea disponibilă în modulul `wave`.

```
f = open('aa.txt')
```

Astfel că este recomandat să utilizăm un import simplu al modulului `wave` sau să folosim un alias:

[1] `https://docs.python.org/3/library/wave.html`

```
import wave as w
```

OBS T2.2 În funcție de sistemul de operare pe care rulăm Jupyter, calea către fișiere trebuie specificată în mod diferit. De exemplu, în Windows, separatorul implicit de nivel de stocare este backslash \, însă în Python, precum și în alte limbaje de programare backslash-ul reprezintă începutul unei secvențe escape (ex. `\n` - rând nou). Astfel că, pentru a utiliza acest separator în căile către fișiere, el trebuie dublat: `'cale\\catre\\fisier.wav'`. Pentru simplitate, se poate utiliza, însă, forwardslash chiar și în sistemul de operare Windows: `cale/catre/fisier.wav`

OBS T2.3 Atunci când dorim să accesăm fișiere ce nu sunt în directorul curent, este important să facem diferența dintre calea absolută (en. *absolute path*) și calea relativă (en. *relative path*). Calea absolută pornește din rădăcina arborelui director al sistemului de operare (ex. în Windows `C:\Users\Student\fisier.wav`) și nu este indicat să fie folosită datorită incompatibilităților ce apar la migrarea codului pe o altă mașină. Calea relativă pornește de la directorul curent și parcurge arborele director pas cu pas. De exemplu, dacă avem nevoie să accesăm un fișier ce se află într-un director părinte al celui curent, vom scrie: `..\director_parinte\fisier.wav`

Acum că avem acces la funcțiile de citire a fișierelor `*.wav` și știm cum să accesăm fișierele locale, putem să mergem mai departe și să citim primul nostru fișier audio. Înregistrări cu semnal audio sunt disponibile în directorul `speech_files/`

[2]:
```
input_wav_file = './speech_files/adr_rnd1_001.wav'
wav_struct = wave.open(input_wav_file, 'r')
```

Dacă ne uităm la definiția funcției `wave.open()`[2], putem să observăm faptul că returnează un obiect de tip `Wave_read`, sau un handler al datelor stocate în acel fișier. Prin simpla apelare a funcției `open()`, datele stocate nu sunt efectiv citite. Rezultatul funcției doar indică programului o anumită zonă din memorie de unde acestea pot fi extrase ulterior.

Ne reamintim faptul că atunci când stocăm un semnal audio chiar și în format necodat, precum cel `wav`, este foarte important să stocăm și alte informații legate de conținutul fișierului, cum ar fi **frecvența de eșantionare** și **rezoluția de bit** sau numărul de canale audio. Astfel că, obiectul `Wave_read` are un set de funcții predefinite ce pot returna aceste

[2] `https://docs.python.org/3/library/wave.html`

informații.

> **OBS T2.4** Numărul de canale pe care este stocat un semnal audio nu a fost discutat în partea introductivă, deoarece pentru voce nu se utilizează formatul stereo. Dacă, însă, acest format este utilizat, se poate presupune faptul că ambele canale conțin aceeași informație, iar unul dintre ele poate fi ignorat.

[3]:
```
# Returnează frecvența de eșantionare
sampling_frequency = wav_struct.getframerate()
print ("The sampling frequency of the file is: %d [Hz]" \
         %sampling_frequency)

# Returnează numărul de biți pe care e stocat un eșantion
bit_depth = wav_struct.getsampwidth()
print ("The sample width of the file is: %d \
         [bytes/sample] or %d [bits/sample] \
        %(bit_depth, bit_depth*8))

# Returnează numărul de canale
no_channels = wav_struct.getnchannels()
print ("The number of channels in the file is %d or %s" \
       %(no_channels, 'mono' if no_channels==1  \
        else 'stereo'))

# Retunează numărul de eșantioane
nframes = wav_struct.getnframes()
print ("The number of samples in the file is: %d" %nframes)

# Returnează tipul compresiei. Pentru fișiere wav,
# aceasta este 'None'
compression_type = wav_struct.getcomptype()
print ("The compression type of the file is: %s "  \
        %compression_type)

# Sau putem obține toată informația de mai sus
# într-o singură instrucțiune
(nchannels, sampwidth, framerate, nframes, comptype, \
         compname) = wav_struct.getparams()
```

[3]:
```
The sampling frequency of the file is: 48000 [Hz]
The sample width: 2 [bytes/sample] or 16 [bits/sample]
The number of channels in the file is 1 or mono
The number of samples in the file is: 174628
The compression type of the file is: NONE
```

Însă tot nu am citit valorile eșantioanelor. Pentru aceasta, vom utiliza funcția `wave.readframes(n)`, unde `n` specifică numărul de eșantioane pe care dorim să le citim. Dacă dorim să citim toate eșantioanele, putem seta `n` la valoarea `-1`. Funcția `readframes()` returnează un vector de obiecte de tip byte, ce pot fi interpretate și ca obiecte string.

Pentru a converti aceste obiecte în valori numerice, se poate utiliza funcția `numpy.frombuffer()` în care specificăm formatul `int16` corespunzător numerelor întregi stocate pe 16 biți, echivalent cu rezoluția de bit a fișierului nostru (precum și a majorității fișierelor de tip `wav`).

OBS T2.5 NumPy este modulul fundamental pentru calculule numerice în Python: `http://www.numpy.org/`. Pentru a-l utiliza trebuie să-l importăm în cod. O convenție generală de utilizare a sa implică și folosirea alias-ului `np` pentru el:

[4]:
```
import numpy as np
# Citim biții din fișierul de intrare
wav_bytes = wav_struct.readframes(-1)
# Afișam tipul datelor citite
print ("wav_bytes is of type %s" %type(wav_bytes))
# Convertim datele în format int16
wav_data = np.frombuffer(wav_bytes, dtype='int16')
# Afișăm tipul datelor convertite
print ("wav_data is of type %s" %type(wav_data))
```

[4]:
```
wav_bytes is of type <type 'str'>
wav_data is of type <type 'numpy.ndarray'>
```

OBS T2.6 În cazul în care doriți să rulați codul de mai sus de mai multe ori, după ce s-a realizat citirea, cursorul de fișier este poziționat la sfârșitul acestuia. Astfel că, pentru a reciti datele trebuie utilizată funcția `rewind()`: `wav_struct.rewind()`. Cu excepția cazului în care ați închis deja fluxul de fișier și se va genera o eroare.

> **OBS T2.7** Trebuie să avem grijă să **închidem întotdeauna** fluxul de fișier după ce au fost citite datele. Altfel s-ar puta să apară ulterior erori de citire/scriere pe disc sau pierderi de date.

[5]:
```
# Închidem fluxul de fișier
wav_struct.close()
```

După ce am rulat pașii anteriori, eșantioanele de semnal și caracteristicile lor sunt stocate în variabilele noastre. Putem acum să le vizualizăm sau să le manipulăm.