

Audió fájl visszhangosítása hatékonyan

Vörös Asztrik

2021. december 21.

Kivonat

Feladatom, hogy egy tetszőleges audio fájl¹ visszhangossá tegyék, eleinte az egész fájl felhasználásával, majd szeletelve, ezzel real time működés feltételét biztosítva. Ehhez egy visszhangos teremben készített impulzusválaszra van szükség, hiszen ez leírja a szobát, mint rendszert, így bármilyen normál audio fájl átkonvertálható ebbe a rendszerbe. Gyakorlatban az impulzusválaszt egy, a szobában készített tapssal fogom közelíteni².

- 1 • Olvasson be egy zenei hangfájlt (.mp3, .wave, stb... kiterjesztésűt) Matlab-ba. Ehhez használható az "audioread", vagy "waveread" függvény valamelyike
- 2 • Jelenítse meg a beolvasott hangmintákat (a mintavételi frekvencia beolvasásával), egy idő/amplitúdó skálán ("plot" függvény). Stereo hangfájl esetén csak az egyik hangsávot használja.
- 3 • Játssza le a Matlab segítségével a beolvasott hangfájl-t a megfelelő mintavételi frekvenciával (soundsc, audioplayer).
- 4 • Töltsön le egy tetszőleges terem akusztikai impulzusválaszt az [Openairlib](#), vagy a [Gamax](#) honlapjáról.
- 5 • Hozza az zeneszámot és az impulzusválaszt közös mintavételi frekvenciára (resampling),
- 6 • majd konvolválja a jeleket.
- 7 • Játssza le az konvolvál jelet, hallható-e a terem hatása a zenére?
- 8 • A módosított zeneszámot mentse el ugyanabban a formátumban, mint amiben az eredeti zeneszám volt.
- 9 • Végezze el az előbbi lépéseket valós idejű hangfeldolgozással, visszhangosítsa a mikrofonba érkező jelet és játssza ki hangszórón. Ehhez mintát és segítséget talál a [mathworks](#) honlapján.

A feladatot egy darab zip file-ban kell feltölteni. A feltöltött állománynak tartalmaznia kell az eredeti hangmintát, a matlabkódot, valamint a módosított és elmentett hang állományt is. Ennek határideje a félév utolsó napja.

Matlab

A Matlab egy alapvetően mátrixokra épített programozási nyelv, mely rendelkezik azokkal a funkcionalitásokkal, melyek egy ilyen feladat elvégzéséhez praktikusak, ha rendelkezünk az adott csomagokkal. Szükségszerű, hogy a Matlab itt használt függvényeinek működésébe belemerjünk felületesen, hogy érthetőek legyenek a megfontolások. Szintén célszerű a segédfüggvények taglalását is a lényegi résztől különvenni, így ezt is itt teszem meg. A dokumentum során taglalt függvények egyben szintén megtalálhatóak csatolmányként.

fft, ifft, conv, nextpow2

Az $\text{fft}(\text{data}, n)$ függvény (Fast Fourier Transformation) diszkrét Fourier transzformációt (DFT) hajt végre, úgy hogy a végeredményként adott vektor mérete n legyen³ Az $\text{ifft}(\text{data})$ ennek az inverzét hajtja végre, azaz az inverz Fourier transzformációt.

Mivel az fft algoritmus hatékonyabb futásidővel rendelkezik 2 hatvány méretű bemenetek esetén⁴, ezért az utóbbi paramétert arra fogjuk felhasználni, hogy a hosszal nagyobb 2^k ($k \in \mathbb{Z}$) méretet adjunk meg (és célszerűen a legkisebb ilyen). Végül figyelniük kell, hogy inverz Fourier transzformációkor a kimenet végéből levágjunk, hiszen a sebesség miatt extra üres adatokkal egészítettük ki a bementet. Mivel az időtartományban végzett konvolúció ($\text{conv}(\text{data}_1, \text{data}_2)$) eredményének hossza $|\text{data}_1| + |\text{data}_2| - 1$ (mely ekvivalens a frekvenciatartomány elvégzett konvolúcióval visszaalakítás után), így az eredménynek a végéből a kiegészített mértéket kell levágnunk.

Másik kiemelendő dolog, hogy DFT esetén a frekvenciatartománybeli szorzattal nem kaphatjuk meg (visszalakítás után) az időbeli konvolválta a ciklikus tulajdonsága miatt. Azonban ha adott data_1 hosszát kiegészítjük $|\text{data}_2| - 1$ -gyel akkor már nem tud átfedés történni (megszűnik a ciklikus tulajdonság), így ekkor ekvivalens lesz az időbeli konvolúcióval^{5,6}.

Az előző kettő megállapítás után tudhatjuk, hogy a minimális hossz $|\text{data}_1| + |\text{data}_2| + 1$, amihez keressük az első nála nagyobb egész kettő hatványt. Ez utóbbinak a kitevőjét a nextpow2 adja meg számunkra. Tehát a matlab belső kódunkban a következő fog szerepelni:

¹pavarotti_original.wav

²impresp.wav

³Matlab forum

⁴Matlab documentation: Input arguments: n

⁵Levezetés

⁶Matlab help

```
1 nfft = 2^nextpow2(length(data1) + length(data2) - 1);
```

energy

A Parseval tételnek megfelelő energiáját számolja ki egy adatra.

```
1 function value = energy(data)
2     value = sum(data .* data);
3 end
```

rescaleByEnergy

Energiamegtartó skálázást tesz lehetővé. Mivel az energiaszámításnál az adatok négyzeteit adjuk össze, így a kettő energiaarányának a négyzetével skáláz.

Erre azért van szükség, mivel konvolúció során a bemenetet egy másik függvény értékeivel szorozzuk, így ki fogunk esni az *audiowrite* által wav fájlba írható vektor értékkészletéből $[-1, 1]^7$.

```
1 % only use energy for caching reasons in caller side
2 function data = rescaleByEnergy(data, energyCurrent, energyOriginal)
3     if energyOriginal ~= 0
4         ratio = energyCurrent / energyOriginal;
5         data = data .* (1/sqrt(ratio));
6     end
7 end
```

rescaleChunk

A szelet méretének négyzetével osztja vissza az adatot. *rescaleByEnergy* függvényt nem tudtam felhasználni real time működés esetén, így ott ezt a visszanzormalizálást használok.⁸

rescaleByEnergy-nél említett okok miatt van rá szükség⁹.

```
1 function data = rescaleChunk(data, chunkSize)
2     data = data ./ sqrt(chunkSize);
3 end
```

getAudioMono

Akár nem monó hanganyag betöltését teszi lehetővé.

```
1 function [data, dataSampleRate] = getAudioMono(src)
2     [data, dataSampleRate] = audioread(src);
3     data = data(:,1);
4 end
```

playAudio

Hanganyag lejátszását teszi lehetővé. Blokkoló utasítás, így megakadályozza, hogy a program végetérjen és a hanganyag félbeszakadjon.

```
1 function playAudio(audio, sampleRate)
2     player = audioplayer(audio, sampleRate);
3     player.playblocking();
4 end
```

paddingZero

A vektort kiegészíti annyi nullával, hogy a megadott hosszt elérje a vektor hossza. Ha a megadott hossz kisebb a vektor hosszánál, akkor értesít erről a kimeneten.

⁷Matlab documentation: Input arguments: y: Data Type of y: double

⁸NumPy documentation: Normalization

⁹Matlab documentation: Input arguments: audioToDevice

```

1 function data = paddingZero(data, size)
2     if length(data) > size
3         disp("wrong param");
4     end
5     data = [data; zeros(size - length(data), 1)];
6 end

```

paddingZeroMultiple

A vektort kiegészíti annyi nullával, hogy a hossza osztható legyen a megadott számmal.

```

1 function data = paddingZeroMultiple(data, n)
2     data = [data; zeros(n - mod(length(data), n), 1)];
3 end

```

1-3. feladat

A feladat egy wav fájl, kiplotolása és lejátszása volt.

Beolvasáshoz a már korábban taglalt *getAudioMono*-t használtam, mely megadja az adathoz tartozó mintavételi frekvenciát is (*inpSampleRate*).

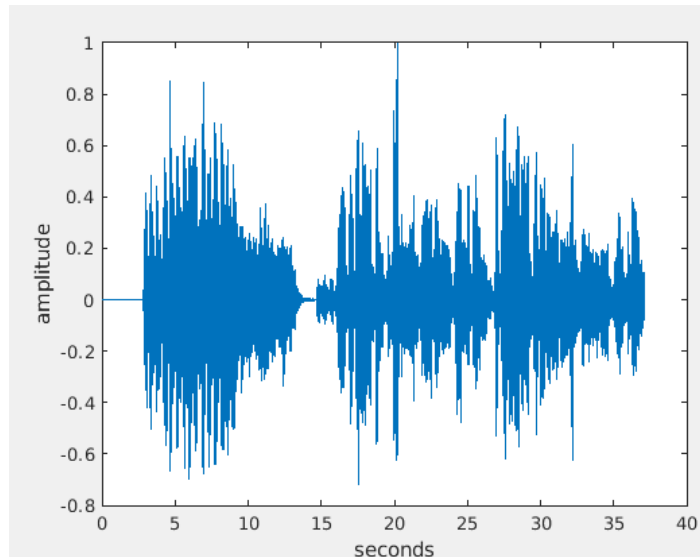
A kirajzoláshoz ki kellett számolnom a teljes fájl hosszát és 2 adat között eltelő időt, a felhasznált adatok alapján. Mivel a frekvencia azt adja meg, hogy másodpercenként hány mintavételezés készült, így ha a teljes adat hosszát ezzel elosztjuk, akkor megkapjuk hány másodperces volt eredetileg. A 2 adat között eltelő időt például úgy is megkaphatjuk, hogy a hosszt elosztjuk az adatok számával, mivel azok között eltelő idő konstans. Ezután már csak a *plot(x, y)* függvénynek meg kell adnia megfelelő formátumban az előbb kiszámoltakat.

A hanganyag lejátszásához az *audioplayer(data, sampleRate)*-et használtam, melyet nem aszinkron kell lejátszani (*.playblocking()*), különben a program azonnal véget érne.

```

1 clear all;
2
3 [inp, inpSampleRate] = getAudioMono("pavarotti_original.wav");
4
5 % plot
6 % create x axis based on inpSampleRate
7 duration = length(inp) / inpSampleRate;
8 delta = duration / length(inp);
9 plot(delta:delta:duration, inp);
10 xlabel('seconds');
11 ylabel('amplitude');
12
13 % playback
14 playAudio(inp, inpSampleRate);
15
16 function playAudio(audio, sampleRate)
17     player = audioplayer(audio, sampleRate);
18     player.playblocking();
19 end
20
21 function [data, dataSampleRate] = getAudioMono(src)
22     [data, dataSampleRate] = audioread(src);
23     data = data(:,1);
24 end

```



4-8. feladat: Teljes adatra végzett konvolúció

Miután a fájlokat beolvastam, a bemenetet és az impulzus választ közös frekvenciára hozzam a `resample(data, p, q)` függvény segítségével, ahol az egyes adatokat $\frac{p}{q}$ -val szorozza be a függvény. A konvolúciót el lehet végezni idő- és frekvenciatartományban, melyet a `shouldConv`-val lehet állítani. Ezután újraskáláztam az értékeket, hogy a megfelelő értékkészletben legyenek. Ezeknek a működését a Matlab szekcióban lehet megtalálni. Végül `audiowrite(src, data, sampleRate)` függvénnyel kiírtam az eredményt egy wav fájlba¹⁰.

```
1 addEffect("pavarotti_original.wav", "echo.wav", "pavarotti_conv.wav", false);

function addEffect(srcInp, srcImpresp, srcOutp, shouldConv)
2     % read
3     [inp, inpSampleRate] = getAudioMono(srcInp);
4     [impresp, imprespSampleRate] = getAudioMono(srcImpresp);
5
6     % resample
7     inpResampled = resample(inp, imprespSampleRate, inpSampleRate);
8     inpResampledSampleRate = imprespSampleRate;
9
10    % convolve
11    if shouldConv
12        outp = conv(inpResampled, impresp);
13    else
14        % set size to the output of time domain convolution to avoid circular property of dft
15        % set size to be a power of 2
16        outpLength = length(inpResampled) + length(impresp) - 1;
17        nfft = 2^nextpow2(outpLength);
18
19        % convolution in frequency domain
20        inpResampledFFT = fft(inpResampled, nfft);
21        imprespFFT = fft(impresp, nfft);
22        outp = ifft(inpResampledFFT .* imprespFFT);
23
24        % cut the padding
25        outp = outp(1:outpLength);
26    end
27    outpSampleRate = imprespSampleRate;
28
29    % rescale
30    % it's important to use inp not inpResampled as inpResampled's values can also go out of
    range
31    outp = rescaleByEnergy(outp, energy(outp), energy(inp));
32
33    % write
```

¹⁰pavarotti_conv.wav

```

34     audiowrite(srcOutp, outp, outpSampleRate);
35 end

```

Érdekes kísérlet volt az időtartományban és a frekvenciatartományban végzett konvolúció közötti időkülönbség. Az impulzus válasz 4 másodperc hosszú volt 48kHz mintavételezés frekvenciával, az audio meg 3 perc 38 másodperc hosszú. Míg időtartományban számomra közelítőleg 6 perc volt, addig frekvenciatartományban közelítőleg 5 másodperc elvégezni a konvolúciót.

Overlap-add algoritmus

Overlap algoritmust arra tudjuk használni, hogy ne csak az egész fájl ismeretében végezhessük el a konvolúciót, hanem már annak szeletein is. Ezzel van lehetőségünk real time végezni az impulzus válasszal a konvolúciót.

Az algoritmus lényege, hogy az egy szeletre elvégzett konvolúció eredményéből csak a szelet mérethez tartozó kezdő részt tekintjük a kimenetnek, a túllógást majd a további szeletek konvolúciós eredményéhez adjuk hozzá.^{11 12}

9. feladat: Real time konvolúció

Ennek a feladatnak a megoldásához az overlap add algoritmust használtam. Sajnos a webes matlab nem képes a mikrofon kezelésére és az offline linuxos verzió nem támogatja megbízhatóan az audiokezelést. Emiatt először az előző feladatokban használt zenét szelgettem fel, mintha realtime érkezne az adat és úgy teszteltem az algoritmus működését¹³. Ezután az overlap algoritmust kiszerveztem egy másik függvénybe, hogy ezt a mikrofont használó megoldás is fel tudja használni.

A függvények során gyakran egészíték ki a *chunkSize* többszörösére, mivel alapvetően chunk egységekben dolgozunk, ezzel megszüntetve lekezelendő speciális eseteket a kódolás során. A *simulateRealTime* függvény ezeken kívül nem tartalmaz újdonságot.

```

1 simulateRealTime("pavarotti_original.wav", "echo.wav", "pavarotti_realtime_simulated.wav",
2                 2048);
3
4 function simulateRealTime(srcInp, srcImpresp, srcOutp, chunkSize)
5     % read
6     [inp, inpSampleRate] = getAudioMono(srcInp);
7     [impresp, imprespSampleRate] = getAudioMono(srcImpresp);
8
9     % resample
10    inpResampled = resample(inp, imprespSampleRate, inpSampleRate);
11    inpResampledSampleRate = imprespSampleRate;
12
13    % set size to the output of time domain convolution to avoid circular property of dft
14    % set size to be a power of 2
15    nfft = 2^nextpow2(chunkSize + length(impresp) - 1);
16
17    % cache FFT of impresp
18    imprespFFT = fft(impresp, nfft);
19
20    % outside visibility of outp, overlap
21    outp = zeros(0,1);
22    outpSampleRate = imprespSampleRate;
23    overlap = zeros(0,1);
24
25    % padding to chunkSize (easier to handle)
26    inpResampled = paddingZeroMultiple(inpResampled, chunkSize);
27
28    % split to chunks
29    tic
30    for idx = 1:chunkSize:length(inpResampled)
31        from = idx;
32        to = idx + chunkSize - 1;
33        chunk = inpResampled(from:to);
34
35        [chunkOutp, overlap] = addEffectToChunk(chunk, imprespFFT, nfft, chunkSize, overlap);
36
37        outp = [outp; chunkOutp];
38    end
39    toc

```

¹¹Matlab help: Algorithms

¹²Matlab help: Overlap add

¹³pavarotti_realtime_simulated.wav

```

37
38 % append remaining overlap
39 outp = [outp; overlap];
40
41 % cut padding
42 outpLength = length(inpResampled) + length(impresp) - 1;
43 outp = outp(1:outpLength);
44
45 % write
46 audiowrite(srcOutp, outp, outpSampleRate);
47 end

```

Az *addEffectToChunk* függvény alapvetően az overlap add algoritmust hajtja végre. Kiemelendő, hogy első iterációban az *overlap* hossza 0 lenne, ezért ki kell egészítenünk minimum 1 *chunkSize*-ra. Azonban ha pont 1 *chunkSize* lenne, akkor a felhasznált overlap levágásakor szintén kezelendő szélső eset lenne, így érdemesebb annak a kétszeresére választani a méretet. Ezenkívül alkalmazva van a már leírt *rescaleChunk* és a szintén indokolt *paddingZeroMultiple*.

```

1 % overlap-add algorithm
2 function [chunkOutp, overlap] = addEffectToChunk(chunk, imprespFFT, nfft, chunkSize, overlap)
3     if length(overlap) == 0
4         % overlap should be at least chunkSize + 1
5         % being multiple of chunkSize makes it easier to work with
6         overlap = zeros(chunkSize * 2, 1);
7     end
8
9     % convolution in frequency domain
10    chunkConved = ifft(fft(chunk, nfft) .* imprespFFT);
11
12    % rescale for chunkOutp, overlapCurrent
13    chunkConved = rescaleChunk(chunkConved, chunkSize);
14
15    % set output based on convolution and overlap (from previous convolutions)
16    chunkOutp = chunkConved(1:chunkSize) + overlap(1:chunkSize);
17
18    % rescale after convolution
19    %chunkOutp = rescaleChunk(chunkOutp, chunkSize);
20
21    % remove used overlap part
22    overlap = overlap(chunkSize+1:end);
23
24    % calculate new overlap based on unused part of chunkConved
25    overlapCurrent = chunkConved(chunkSize+1:end);
26    %overlapCurrent = rescaleChunk(overlapCurrent, chunkSize);
27    % |overlapCurrent| > |overlap|
28    overlap = paddingZero(overlap, length(overlapCurrent)) + overlapCurrent;
29
30    % make it to a multiple of chunkSize for easier use
31    overlap = paddingZeroMultiple(overlap, chunkSize);
32 end

```

Végül találva egy reprodukálható működést megírtam a mikrofon alapú kódot is. Itt eleinte csak az impulzusválaszt olvastam be, majd annak frekvenciartománybeli értékeit is kiszámoltam, hogy ne kelljen minden bejövő szeletnél újraszámolni. Ezután a beépített Matlab függvényekkel lekértem a mikrofont és a hangszórót leíró eszközöket. Végül a végtelenségig kérek a mikrofontól szeletet, amit átdolgozok az *addEffectToChunk* függvénnyel, majd a hangszórón lejátszom az eredményt.

```

1 addEffectToMic("echo.wav", 2048, 44100);

```

```

1 function addEffectToMic(srcImpresp, chunkSize, micFs)
2     % read
3     [impresp, imprespSampleRate] = getAudioMono(srcImpresp);
4
5     % set size to the output of time domain convolution to avoid circular property of dft
6     % set size to be a power of 2
7     nfft = 2^nextpow2(chunkSize + length(impresp) - 1);
8
9     % cache FFT of impresp

```

```

10    imprespFFT = fft(impresp, nfft);
11
12    % outside visibility of outp, overlap
13    overlap = zeros(0,1);
14
15    % read from microphone
16    adr = audioDeviceReader('SamplesPerFrame', chunkSize, 'NumChannels', 1, "Device", "USB2.0
    Camera: Audio (hw:0,0)", "SampleRate", 32000);
17    setup(adr); % Call setup to reduce the computational load of initialization in an audio
    stream loop.
18    adw = audioDeviceWriter('SampleRate', adr.SampleRate, 'SupportVariableSizeInput', true, '
    BufferSize', chunkSize);
19
20    %resample
21    impresp = resample(impresp, adr.SampleRate, imprespSampleRate);
22    imprespSampleRate = adr.SampleRate;
23
24    while true
25        chunk = adr();
26        [chunkOutp, overlap] = addEffectToChunk(chunk, imprespFFT, nfft, chunkSize, overlap);
27        adw(chunkOutp);
28    end
29 end

```

Tesztelésem során arra jutottam, hogy adott szeletméret esetén a következők tapasztalhatók az én konfigurációmon: 512 esetén a hang rossz minőségű, 1024 esetén késés figyelhető meg, 2048 egy jó választásnak tűnik, 4096-tól újra elkezd nőni a késés.