Audió fájl visszhangosítása hatékonyan

Vörös Asztrik

2021. december 21.

Kivonat

Feladatom, hogy egy tetszőleges audió fájlt[1] visszhangossá tegyek, 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 audió fájl átkonvertálható ebbe a rendszerbe. Gyakorlatban az impulzusválaszt egy, a szobában készített tapssal fogom közelíteni[2].

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 Matlab segítségével a beolvasott hangfile-t a megfelelő mintavételi frekvenciával (soundsc, audioplayer).
4. Töltsön le egy tetszőleges terem akusztikai impulzusválaszát 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 konvolvált 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 belemenjek 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 *fft(data, n)* függvény (Fast Fourier Transformation) diszkrét Fourier transzformációt (DFT) hajt végre, úgy hogy a végeredményként adott vektor mérete n legyen[3] Az *ifft(data)* ennek az inverzét hajta végre, azaz az inverz Fourier transzformációt.
Mivel az fft algoritmus hatékonyabb futásidővel rendelkezik 2 hatvány méretű bemenetek esetén[4], ezért az utóbbi paramétert arra fogjuk felhasználni, hogy a hossznál nagyobb 2^k ($k \in \mathbb{Z}$) méretet adjunk meg (és célszerűen a legkisebb ilyet). Végül figyelnünk 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 (*conv(data$_1$, 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 konvolváltat 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 beli kódunkban a következő fog szerepelni:

[1] pavarotti_original.wav
[2] impresp.wav
[3] Matlab forum
[4] Matlab documentation: Input arguments: n
[5] Levezetés
[6] Matlab help

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

energy

A Parseval tételnek megfelelő energiáját számolja ki egy adatra.

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

rescaleByEnergy

Energiamegtartó skálázást tesz lehetővé. Mivel az energiaszámításnál az adatok négyzeteit adjuk össze, így a kettő energiaaránynak 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].

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

rescaleChunk

A szelet méretének négyzetével osztja vissza az adatot. *rescaleByEnergy* függvényt nem tudtam felhasználni real time működés esetén, így ott ezt a visszanormalizálást használom.[8]
rescaleByEnergy-nél említett okok miatt van rá szükség[9].

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

getAudioMono

Akár nem monó hanganyag betöltését teszi lehetővé.

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

playAudio

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

```
function playAudio(audio, sampleRate)
    player = audioplayer(audio, sampleRate);
    player.playblocking();
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.

[7] Matlab documentation: Input arguments: y: Data Type of y: double
[8] NumPy documentation: Normalization
[9] Matlab documentation: Input arguments: audioToDevice

```
function data = paddingZero(data, size)
    if length(data) > size
        disp("wrong param");
    end
    data = [data; zeros(size - length(data), 1)];
end
```

paddingZeroMultiple

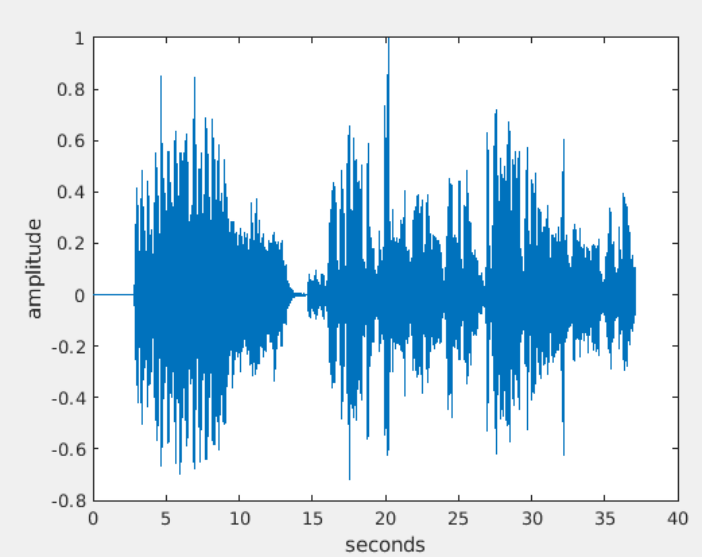
A vektort kiegészíti annyi nullával, hogy a hossza osztható legyen a megadott számmal.

```
function data = paddingZeroMultiple(data, n)
    data = [data; zeros(n - mod(length(data), n), 1)];
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.

```
clear all;

[inp, inpSampleRate] = getAudioMono("pavarotti_original.wav");

% plot
% create x axis based on inpSampleRate
duration = length(inp) / inpSampleRate;
delta = duration / length(inp);
plot(delta:delta:duration, inp);
xlabel('seconds');
ylabel('amplitude');

% playback
playAudio(inp, inpSampleRate);

function playAudio(audio, sampleRate)
    player = audioplayer(audio, sampleRate);
    player.playblocking();
end

function [data, dataSampleRate] = getAudioMono(src)
    [data, dataSampleRate] = audioread(src);
    data = data(:,1);
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 fájlba[10].

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

```
function addEffect(srcInp, srcImpresp, srcOutp, shouldConv)
    % read
    [inp, inpSampleRate] = getAudioMono(srcInp);
    [impresp, impRespSampleRate] = getAudioMono(srcImpresp);

    % resample
    inpResampled = resample(inp, impRespSampleRate, inpSampleRate);
    inpResampledSampleRate = impRespSampleRate;

    % convolve
    if shouldConv
        outp = conv(inpResampled, impresp);
    else
        % set size to the output of time domain convolution to avoid circular property of dft
        % set size to be a power of 2
        outpLength = length(inpResampled) + length(impresp) - 1;
        nfft = 2^nextpow2(outpLength);

        % convolution in frequency domain
        inpResampledFFT = fft(inpResampled, nfft);
        impRespFFT = fft(impresp, nfft);
        outp = ifft(inpResampledFFT .* impRespFFT);

        % cut the padding
        outp = outp(1:outpLength);
    end
    outpSampleRate = impRespSampleRate;

    % rescale
    % it's important to use inp not inpResampled as inpResampled's values can also go out of
        range
    outp = rescaleByEnergy(outp, energy(outp), energy(inp));

    % write
```

[10] pavarotti_conv.wav

```
    audiowrite(srcOutp, outp, outpSampleRate);
end
```

Érdekes kísérlet volt az időtartományban és a frekvenciatartományban végzett konvolúció közötti időkülönbség. Az impulzus válasz 4 másodperc hosszú volt 48kHZ mintavételezés frekvenciával, az audió 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 szeleteltem fel, mintha realtime érkezne az adat és úgy teszteltem az algoritmus működését[13]. Ezután az overlap algoritmust kiszerveztem egy másik függvénybe, hogy ezt a mikrofont használó megoldás is fel tudja használni.
A függvények során gyakran egészítek ki a *chunkSize* többbszörösére, mivel alapvetően chunk egységegben dolgozunk, ezzel megszűntetve lekezelendő speciális eseteket a kódolás során. A *simulateRealTime* függvény ezeken kívűl nem tartalmaz újdonságot.

```
simulateRealTime("pavarotti_original.wav", "echo.wav", "pavarotti_realtime_simulated.wav",
    2048);
```

```
function simulateRealTime(srcInp, srcImpresp, srcOutp, chunkSize)
    % read
    [inp, inpSampleRate] = getAudioMono(srcInp);
    [impresp, impRespSampleRate] = getAudioMono(srcImpresp);

    % resample
    inpResampled = resample(inp, impRespSampleRate, inpSampleRate);
    inpResampledSampleRate = impRespSampleRate;

    % set size to the output of time domain convolution to avoid circular property of dft
    % set size to be a power of 2
    nfft = 2^nextpow2(chunkSize + length(impresp) - 1);

    % cache FFT of impresp
    impRespFFT = fft(impresp, nfft);

    % outside visibility of outp, overlap
    outp = zeros(0,1);
    outpSampleRate = impRespSampleRate;
    overlap = zeros(0,1);

    % padding to chunkSize (easier to handle)
    inpResampled = paddingZeroMultiple(inpResampled, chunkSize);

    % split to chunks
    tic
    for idx = 1:chunkSize:length(inpResampled)
        from = idx;
        to = idx + chunkSize - 1;
        chunk = inpResampled(from:to);

        [chunkOutp, overlap] = addEffectToChunk(chunk, impRespFFT, nfft, chunkSize, overlap);

        outp = [outp; chunkOutp];
    end
    toc
```

[11] Matlab help: Algorithms
[12] Matlab help: Overlap add
[13] pavarotti_realtime_simulated.wav

```
    % append remaining overlap
    outp = [outp; overlap];

    % cut padding
    outpLength = length(inpResampled) + length(impresp) - 1;
    outp = outp(1:outpLength);

    % write
    audiowrite(srcOutp, outp, outpSampleRate);
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 akalmazva van a már leírt *rescaleChunk* és a szintén indokolt *paddingZeroMultiple*.

```
% overlap-add algorithm
function [chunkOutp, overlap] = addEffectToChunk(chunk, impulseFFT, nfft, chunkSize, overlap)
    if length(overlap) == 0
        % overlap should be at least chunkSize + 1
        % being multiple of chunkSize makes it easier to work with
        overlap = zeros(chunkSize * 2, 1);
    end

    % convolution in frequency domain
    chunkConved = ifft(fft(chunk, nfft) .* impulseFFT);

    % rescale for chunkOutp, overlapCurrent
    chunkConved = rescaleChunk(chunkConved, chunkSize);

    % set output based on convolution and overlap (from previous convolutions)
    chunkOutp = chunkConved(1:chunkSize) + overlap(1:chunkSize);

    % remove used overlap part
    overlap = overlap(chunkSize+1:end);

    % calculate new overlap based on unused part of chunkConved
    overlapCurrent = chunkConved(chunkSize+1:end);
    %overlapCurrent = rescaleChunk(overlapCurrent, chunkSize);
    % |overlapCurrent| > |overlap|
    overlap = paddingZero(overlap, length(overlapCurrent)) + overlapCurrent;

    % make it to a multiple of chunkSize for easier use
    overlap = paddingZeroMultiple(overlap, chunkSize);
end
```

Végül találva egy reprodukálható működést megírtam a mikrofon alapú kódot is. Itt eleinte csak az impulzusválaszt olvastam be, majd annak frekvenciatartománybeli 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.

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

```
function addEffectToMic(srcImpulse, chunkSize, micFs)
    % read
    [impulse, impulseSampleRate] = getAudioMono(srcImpulse);

    % set size to the output of time domain convolution to avoid circular property of dft
    % set size to be a power of 2
    nfft = 2^nextpow2(chunkSize + length(impulse) - 1);

    % cache FFT of impulse
    impulseFFT = fft(impulse, nfft);

    % 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, impulseSampleRate);
22      impulseSampleRate = adr.SampleRate;
23
24      while true
25          chunk = adr();
26          [chunkOutp, overlap] = addEffectToChunk(chunk, impulseFFT, 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.