Hangtechnikai effektek programozása

Fiala Péter

Gyakorlati útmutató

Tartalomjegyzék

1. Bevezetés

Ezen a gyakorlaton a legalapvetőbb hangtechnikai effektek egyszerűbb programjait ismerheted meg. A gyakorlat célja az, hogy a közölt Matlab programrészek segítségével mélyebb betekintést nyerhess az effektek működési mechanizmusába.

2. Hangjelek beolvasása és lejátszása

Ebben a feladatcsoportban egyszerű hangfeldolgozási feldatokkal találkozol, lényegében alap Matlab függvényekkel és programozási technikákkal ismerkedsz meg. Fontos, hogy a közölt megoldások minden sorát alaposan értsd, ezután a bonyolultabb feladatok sem fognak gondot okozni.

1. Feladat.

Olvass be egy wav fájlt, majd játszd vissza különböző mintavételi frekvenciákon.

```
[u, fs] = wavread('samples/Gaga.wav');  % read sound sample from wav
sound(u, fs); % playback
sound(u, fs*2); % double speed
sound(u, fs/2); % half speed
soundsc(u, fs); % normalised playback
```

2. Feladat.

Add össze a sztereó jel két csatornáját, távolítsd el a DC ofszetet, majd normalizáld az eredményt!

```
u = sum(u,2);         % sum in the 2nd direction (columns)
u = u - mean(u,1);    % mean in the first direction (rows)
u = u / max(abs(u)); % normalise
```

3. Feladat.

Készíts a monó jelből álsztereó jelet, mely a bal oldalhoz képest a jobb oldalt kissé késleltetve tartalmazza!

```
T = 1e-2;                 % delay between left/right [s]
n = fs*T;                 % delay in samples
N = length(u);            % number of samples
kleft = 1:N;              % linear index vector
kright = min(kleft+n,N);      % shifted index vector
y = [u(kleft) u(kright)];
```

Az álsztereó jel esetében érdemes a késleltetett oldalt kissé erősíteni, hogy nehogy azt higgyük, hogy balról érkezik a hang. Próbáld ki, hogy mekkora erősítés és késleltetés esetén kapsz kellemes hangzást!

4. Feladat.

Torzítsd a hangjelet úgy, hogy adott szinttel (drive) megerősíted, majd ± 1-es szinten vágsz. A hangerőt (level) utólag állíthatod be.

```
drive = 10;             % drive in dB
u = u * 10^(drive/20); % overdrive
u(u>1) = 1;             % logical indexing !!!
u(u<-1) = -1;
```

A vágásra másik megoldás:

```
u = min(max(u, -1), 1);
```

3. Zengetők

A lineáris diszkrét rendszer rendszeregyenlete

$$y_k + \sum_{i=1}^{n} a_i y_{k-i} = \sum_{j=0}^{m} b_j x_{k-j} \tag{1}$$

ahol x a bemenő jel, y a kimenő jel, a_i és b_j pedig a rendszeregyenlet együtthatói.

A Matlab `y = filter(a, b, x)` függvénye a fenti formában megadott lineáris rendszer szimulációját végzi, vagyis adott bemenetre kiszámolja a kimenő jelet. A filter függvény paraméterei:

- `a` az a_i együtthatók vektora, az első tag szükségszerűen 1.
- `b` a b_j együtthatók vektora
- `x` a bemenő jel mintái

3.1. IIR zengetők

Képzeljük el azt az egyszerű zengetőt, mely egy direkt jelet, majd annak T késleltetéssel megjelenő visszhangjait írja le. Könnyen látható, hogy a válasz felírása

$$y_k = x_k + A y_{k-n} \tag{2}$$

ahol $T = n\Delta T$, vagyis a válasz megegyezik a direkt jellel plusz az n mintával korábbi válaszjel A-szorosával. Ezen rendszer impulzusválasza egy n mintánként megjelenő exponenciálisan csökkenő impulzussorozat, melynek kvóciense A.

Ha több, különböző időkésleltetéssel és csillapítással jellemzett visszhang kombinált hatását szeretnénk modellezni, a megfelelő rendszeregyenlet

$$y_k = x_k + \sum_{l=1}^{r} A_l y_{k-n_l} \tag{3}$$

ahol $n_l \Delta T$ az l-edik visszhang késleltetése, A_l pedig a csillapítása.

5. Feladat.
Valósítsd meg az IIR zengetőt. Próbáld ki az alábbi paraméterekkel:

	1	2	3	4
dT [ms]	20	30	50	70
A [-]	0.1	0.2	0.01	0.1

```
dT = [20 30 50 70]*1e-3;
A  = [1e-1 2e-1 1e-2 2e-1];

b(1) = 1;
a(1) = 1;
a(round(fs*dT)) = -A; % direct indexing

y = filter(b, a, u);
```

4. Fázismoduláción alapuló effektek

4.1. A fázismoduláció

Az $u(t)$ jel fázismodulált változata $y(t) = u(t + f(t))$, ahol $f(t)$ a moduláló jel. Egyszerű esetben $f(t) = T\sin(\omega_0 t)$, vagyis az eredeti jelet harmonikusan változó késleltetéssel hallgatjuk.

Diszkrét változatban $y_k = u(k + K\sin(\omega_0 k/f_s))$, ahol $K = Tf_s$

6. Feladat.

Valósítsd meg a fázismodulációt! Ehhez mindössze az u jel új indexvektorát kell előállítanod. Figyelj arra, hogy az indexvektor csupa egész értéket tartalmazzon, illetve ne indexeld se túl, se alul az eredeti jelet!

```
N = length(u);
k = (1 : N)'; % orig. sample index vector
T = 1e-2;     % depth of delay line [s]
K = T * fs;   % depth in samples
f0 = 5;       % freq. of oscillation [Hz]

k_mod = K * sin(2*pi*f0/fs*k);  % modulating sample indices
k_out = round(k + k_mod);   % output indices
k_out = max(min(k_out, N), 1); % avoid over/underindexing

y = u(k_out); % reindexing
```

4.2. Kórus és társai

A legegyszerűbb kórus effektet úgy kapjuk meg, hogy az eredeti u(t) jelet hozzáadjuk annak fázismodulált $u(t + f(t))$ változatához. Eredményként egy időben változó frekvenciájú lyukszűrő-hatást érünk el.

7. Feladat.

Valósítsd meg a kórus effektet, és hallgasd meg különböző paraméterekkel! Figyelj arra, hogy milyen paramétertartományok jellemzőek a kórusra, és melyek a (visszacsatolás nélküli ős)flangerre.

4.3. Pitch-shifterek

Pitch-shifter alatt azt értjük, hogy egy hangmintát úgy játszunk le $1 + r$-szeres sebességgel, hogy nem változik a lejátszás időtartama. Ez is visszavezethető a fentebbi fázismodulációra. Legyen $y(t) = u(t + r \cdot t)$. Ez a lineáris moduláció egy frekvenciatolást valósít meg. Hogy ne fogyjanak el hamarabb a hangmintáink, a moduláló jelet adott T időközönként visszaállítjuk zérus értékre. Így végeredményben egy fűrészjellel moduláljuk az eredeti hangjelünk fázisát.

$$y(t) = u(t + r \cdot (t \mathrm{mod} T)) \tag{4}$$

illetve diszkrét időben egyszerűen

$$y(k) = u(k + r \cdot (k \mathrm{mod} K)) \tag{5}$$

ahol $K = T \cdot f_s$, A fenti fázismoduláció szükségszerűen T időközönként megjelenő, jól hallható pattanásokat eredményez az eredő hangmintában.

8. Feladat.

Valósítsd meg a fenti pitch shiftert, és hallgasd meg különböző r és T értékekre! Figyeld meg a T időközönként megjelenő pattanásokat!

```
N = length(u);  % length of sound sample
k = (1 : N)';   % original index vector

r = .25;    % shift ratio [-]
T = .2;     % saw width [s]
K = T*fs;   % saw width in samples

k_mod = r * mod(k, K);  % modulating saw signal
k_out = round(k + k_mod);      % resultant indices
k_out = max(min(k_out, N), 1);  % no overindexing

y = u(k_out);  % reindex
```

A pattanások megszüntethetők, ha két párhuzamos $y(t)$ jelet állítunk elő, melyek fűrészjelei $T/2$-vel késnek egymáshoz képest, majd a két jelet periodikusan átúsztatjuk egymásba. Az átúsztatáshoz olyan súlyozó jelpárt kell alkalmaznunk, mely összege állandó. Kézenfekvő megoldás az emelt koszinusz és annak fázistolt változata:

$$w_1(t) = \frac{1 - \cos(2\pi t/T)}{2} \tag{6}$$

$$w_2(t) = \frac{1 + \cos(2\pi t/T)}{2} \tag{7}$$

9. Feladat.

Valósítsd meg a pattogásmentes pitch shifter algoritmust! Hallgasd meg az eredményt!

```
N = length(u);
k = (1 : N)';   % original index vector

r = .25;    % shift ratio
T = .2;     % saw period [s]
K = .2 * fs;    % saw period [samples]

k_mod1 = r * mod(k, K); % modulating saw indices
k_mod2 = r * mod(k-K/2, K);

k_out1 = round(k + k_mod1); % resulting indices
k_out2 = round(k + k_mod2);

k_out1 = max(min(k_out1, N), 1); % no overindexing
k_out2 = max(min(k_out2, N), 1);

w1 = .5 * (1 - cos(2*pi*k/K));  % weight signals (sum = 1)
w2 = .5 * (1 + cos(2*pi*k/K));

y = u(k_out1).*w1 + u(k_out2).*w2;  % reindex and mix
```

10. Feladat.

A pitch shifter algoritmust ültesd át függvénybe, mely csak az $u(k)$ jelmintákat, az r arányt és a K fűrészszélességet kapja paraméterként. A függvény segítségével készíts harmoniser effektet, mely egy énekszólamra dúr harmóniát illeszt. A dúr hármas arányai $1 : 5/4 : 3/2$. Ha szomorú vagy, inkább illessz mollt! Annak frekvenciaarányai $1 : 6/5 : 3/2$.

11. Feladat.

A pitch shifter segítségével valósítsd meg azt a rokon effektet, mely egy hangmintát úgy tud lassítani-gyorsítani, hogy a hang magassága nem változik! Ez a feladat könnyen visszavezethető $1+r$-szeres hangmagasságmódosításra és $1 + r$-szeres mintavételifrekvencia-csökkentésre.

5. Dinamikaprocesszorok

A dinamikaprocesszor (kompresszor, expander, zajzár stb) a jel dinamikatartományát változtatja. A processzort a karakterisztikája írja le, mely a bemenő jelszinthez egy kimenő jelszintet rendel hozzá. Mind a bemenő, mind a kimenő jelszint alatt a hangjel aktuális teljesítményét értjük[1], és a szinteket a karakterisztikában dB-ben adjuk meg.

A dinamikaprocesszor algoritmusa a következő:

1. Meghatározzuk az $u(t)$ jel $u_{\mathrm{RMS}}(t)$ futó RMS szintjét.

[1] vagy esetleg csúcsértékét

2. A karakterisztikából kiolvasssuk az aktuális kimenő szintet és a szükséges beavatkozó (Gain) $g(t)$ erősítést

3. A bemenő $u(t)$ jelet a szükséges $g(t)$ erősítéssel módosítjuk, hogy megkapjuk az $y(t) = u(t) \cdot g(t)$ kimenő jelet.

Az ördög természetesen a részletekben alszik.

5.1. RMS-számítás exponenciális átlagolással

A bejövő jel futó RMS-szintjét az alábbi csúszóablakos átlagolással definiáljuk:

$$u_{\mathrm{RMS}}(t) = \sqrt{\frac{1}{T}\int_{t-T}^{t} u^2(\tau)\mathrm{d}\tau} \tag{8}$$

vagyis a jel T hosszú múltját vesszük figyelembe az energia meghatározásakor. A fenti definíció nehézkes azért, mert pontos számításához tárolnunk kell a jel aktuális T mély múltját. Ez kiküszöbölhető, ha a definíciót úgy módosítjuk, hogy a múltat nem négyszög, hanem azzal azonos területű, végtelen mély exponenciális csúszóablakkal súlyozzuk. Az így adódó futó RMS definíciója:

$$u_{\mathrm{RMS}}(t) \approx \sqrt{\frac{1}{T}\int_{-\infty}^{t} u^2(\tau)\mathrm{e}^{-(t-\tau)/T}\mathrm{d}\tau} \tag{9}$$

Térjünk át diszkrét időbe ΔT mintavételi időközzel. Az integrált szummával közelíthetjük:

$$u^2_{\mathrm{RMS},k} = \frac{1}{T}\sum_{i=-\infty}^{k} u_i^2 \mathrm{e}^{-(k-i)\Delta t/T}\Delta t \tag{10}$$

$$= \frac{1}{K}\sum_{i=-\infty}^{k} u_i^2 \alpha^{k-i} \tag{11}$$

ahol $K = T/\Delta t = Tf_s$ az ablakszélesség mintákban mérve, valamint

$$\alpha = \mathrm{e}^{-1/K} \tag{12}$$

Az exponenciális súlyozás előnye, hogy a futó RMS következő mintája könnyen számolható az aktuális RMS minta és a jel következő mintája ismeretében:

$$u^2_{\mathrm{RMS},k} = \frac{1}{K}\sum_{i=-\infty}^{k} u_i^2 \alpha^{k-i} \tag{13}$$

$$= \frac{1}{K}u_k^2 + \alpha u^2_{\mathrm{RMS},k-1} \tag{14}$$

Azonnal látszik, hogy egy elsőrendű IIR-szűrővel van dolgunk, melynek bemenete az u^2 jel, kimenete pedig az u^2_{RMS} jel. A szűrés matlabos megvalósítása:

```
K_rms = T_rms * fs;   % window width in samples
alpha_rms = exp(-1/K_rms);  % exp. ratio
u2rms = filter(1/K_rms, [1, -alpha_rms], u.^2); % filtering
```

A bemenő jel rms szintje alapján a kimenő rms-szintet a karakterisztikából olvashatjuk ki. A karakterisztikát tipikusan töréspontosan adjuk meg:

```
Comp = [    % compressor characteristics
    % in   out
    -1000 -1000
      -70   -70
      -50   -30
      -10   -25
        0   -20
    ];
```

A kimenő szintet lineáris interpolációval határozhatjuk meg:

```
indB = 10*log10(u2rms); % express rms in dB
outdB = interp1(Comp(:,1), Comp(:,2), indB); % output dB level
```

5.2. A beavatkozó jel és annak szűrése

Az aktuális bemenő és kimenő jelszintek alapján meghatározhatjuk a pillanatnyi $g(t)$ beavatkozó jelet (gain), mely egy nemnegatív erősítést (esetleg gyengítést) ad meg minden időpontban. A $g(t)$ jel közvetlen alkalmazását általában kerüljük, mert túl hirtelen változásokat eredményez a kimenet szintjében. Helyette a $g(t)$ beavatkozó jelet adott felfutó és lefutó időállandóval (attack, release) szűrjük. A mi alkalmazásunkban az egyszerűség kedvéért egyetlen simító időállandót alkalmazunk. A két időállandós módszerhez ciklusban kéne végigmennünk a mintán, ami Matlab programoknál igen lassú működést eredményezne. A szűrés az RMS szűrővel analóg módon történik, csak itt nem a jel négyzetét szűrjük:

```
T_Gain = 1e-3;   % time constant of gain filter
K_gain = T_Gain * fs;
gain_filt = filter(1/K_gain, [1, -exp(-1/K_gain)], gain);
```

Végezetül álljon itt a dinamikaprocesszor teljes kódja:

```
Comp = [    % compressor characteristics
    % in   out
    -1000 -1000
      -70   -70
      -50   -30
      -10   -25
        0   -20
    ];

```

```
% compute rms square with exponential averaging
T_rms = 5e-3;   % time constant of RMS
K_rms = T_rms * fs;   % window width in samples
alpha_rms = exp(-1/K_rms);  % exp. ratio
u2rms = filter(1/K_rms, [1, -alpha_rms], u.^2); % filtering

indB = 10*log10(u2rms); % express rms in dB
outdB = interp1(Comp(:,1), Comp(:,2), indB); % output dB level

gaindB = outdB - indB;  % gain in dB
gain = 10.^(gaindB/20);  % linear gain

T_Gain = 1e-3;   % time constant of gain filter
K_gain = T_Gain * fs;
gain_filt = filter(1/K_gain, [1, -exp(-1/K_gain)], gain);

y = gain_filt .* u; % multiply output with filtered gain
```

12. Feladat.

A karakterisztika megfelelő módosításával valósítsd meg a klasszikus threshold/ratio paraméterekkel definiált kompresszort.

13. Feladat.

A karakterisztika megfelelő módosításával valósíts meg zajzárat! Szuperponálj egy beszédhang-mintára fehérzajt, és próbáld ki a zajzárat működés közben.

14. Feladat.

Valósíts meg olyan dinamikaprocesszort, mely két jelen dolgozik, és az u jel szintjének függvényében a v jel szintjét módosítja. Pl. automatikusan halkuljon le az aláfestő zene, ha megszólal a beszélő.