Nedoločenost angleškega jezika

Uvod

Claude E. Shannon je leta 1948 v prelomnem članku *A Mathematical Theory of Communication* [1] postavil temelje moderne informacijske teorije. V njem je vpeljal nedoločenost (entropijo) kot mero informacije. V članku *Prediction and Entropy of printed English* [2], iz leta 1951, je omenjeno mero uporabil pri analizi angleških besedil. Hotel je ugotoviti, koliko informacije, v povprečju, nosi posamezna črka. Problema se je lotil analitično in eksperimentalno. Rezultati so pokazali, da črka angleške abecede nosi v povprečju približno 1,5 bita informacije [3].

Podoben poskus želimo ponoviti tudi mi. Ugotoviti želimo približek povprečne informacije, ki jo nosijo črke angleške abecede. Izhajamo iz nedoločenosti naključne spremenljivke X, ki predstavlja črko angleške abecede. Ob predpostavki, da so zaporedne črke neodvisne in poznamo njihove verjetnosti, velja:

$$H(X) = -\sum_{i}^{I} p(x_i) \log_2 p(x_i) \quad . \tag{1}$$

Če z naključnima spremenljivkama X_p in X_n označimo dve zaporedni črki v besedilu in zgornja predpostavka drži, lahko za nedoločenost para črk zapišemo

$$H(X_p, X_n) = H(X_p) + H(X_n) \quad . \tag{2}$$

Ker so črke v besedilu med seboj odvisne velja $H(X_p, X_n) < H(X_p) + H(X_n)$. Povprečno informacijo $H(X_n|X_p) < H(X_n)$, ki jo dobimo, ko izvemo črko X_n, lahko izračunamo kot

$$H(X_n|X_p) = H(X_p, X_n) - H(X_p) \quad . \tag{3}$$

Ta količina nam predstavlja povprečno informacijo na črko, ki jo dobimo, če predhodno črko že poznamo. Pri obravnavi nam X_p lahko predstavlja poljuben niz črk (pare, trojice, ...)

$$\mathbf{X_p} = (X_1, X_2, ..., X_{n-1}) \quad . \tag{4}$$

Ker X_n predstavlja naslednjo črko v istem besedilu, lahko zapis poenostavimo

$$\mathbf{X_{pn}} = (X_1, X_2, ..., X_p, X_n) \quad , \quad \text{sledi} \tag{5}$$

$$H(X_n|\mathbf{X_p}) = H(\mathbf{X_{pn}}) - H(\mathbf{X_p}) \tag{6}$$

$$= -\sum_{i}^{I} p(\mathbf{x_{pn_i}}) \log_2 p(\mathbf{x_{pn_i}}) + \sum_{j}^{J} p(\mathbf{x_{p_j}}) \log_2 p(\mathbf{x_{p_j}}) \quad . \tag{7}$$

- I predstavlja število vseh možnih nizov dolžine n,
- J predstavlja število vseh možnih nizov dolžine $n-1$,
- $p(\mathbf{x_{pn_i}})$ je verjetnost pojavitve niza $\mathbf{x_{pn_i}}$ dolžine n,
- $p(\mathbf{x_{p_i}})$ je verjetnost pojavitve niza $\mathbf{x_{p_i}}$ dolžine $n-1$.

V limiti, ko se n približuje neskončnosti, se $H(X_n|\mathbf{X_p})$ približuje pravi vrednosti (povprečni informaciji na znak).

Naloga

Napišite funkcijo z imenom `naloga1` v programskem jeziku Matlab, ki izračuna približek povprečne informacije na znak $H(X_n|\mathbf{X_p})$ za dano število poznanih predhodnih črk. Vhodna argumenta funkcije sta stolpični vektor (niz) z besedilom in število poznanih predhodnih črk (celoštevilska vrednost na intervalu [0,3]). Besedilo lahko vsebuje vse črke angleške abecede (male in velike), števila in poljubna ločila ter presledke, tabulatorje, ipd. Iz vhodnih podatkov odstranite vse znake, ki niso črke angleške abecede. Črke nato spremenite v velike tiskane.

Prototip funkcije:

```
function H = naloga1(besedilo,p)
% besedilo - stolpicni vektor znakov (char)
% p  - stevilo poznanih predhodnih znakov; 0, 1, 2 ali 3.
%    p = 0 pomeni, da racunamo povprecno informacijo na znak
%        abecede brez poznanih predhodnih znakov: H(X1)
%    p = 1 pomeni, da racunamo povprecno informacijo na znak
%        abecede pri enem poznanem predhodnemu znaku: H(X2|X1)
%    p = 2: H(X3|X1,X2)
%    p = 3: H(X4|X1,X2,X3)
%
% H - skalar; povprecna informacija na znak abecede
%     z upostevanjem stevila poznanih predhodnih znakov p

H = nan;
```

Testni primeri
Na učilnici se nahajajo trije testni primeri besedil, za katere imate podane tudi povprečno informacijo na znak za predpisano število poznanih predhodnih znakov. Podatki so podani v obliki datotek `.mat`, ki jih naložite v Matlab s pomočjo ukaza `load ime_datoteke.mat`. Po izvedenem ukazu se bodo v okolju Matlab pojavile spremenljivke z besedili in rezultati. Priloženo imate tudi funkcijo `test_naloga1`, ki jo lahko uporabite za preverjanje pravilnosti rezultatov, ki jih vrača vaša funkcija. Pri testiranju vaših funkcij upoštevajte naslednje omejitve:

- rezultat je pravilen, če se od danega razlikuje za manj kot 10^{-3},
- izvajanje funkcije je časovno omejeno na 30 sekund.

Namigi

Predprocesiranje:
Vhod: Danes je lep dan! Naredil bom 1. domaco nalogo.
Izhod: DANESJELEPDANNAREDILBOMDOMACONALOGO

Računanje povprečne informacije na znak:
Najprej je potrebno izračunati verjetnosti posameznih nizov črk glede na to, kateri približek računamo oz. koliko predhodnih znakov poznamo. Ko imamo izračunane verjetnosti, uporabimo formulo za entropijo. Ali jo lahko izračunamo tudi brez uporabe zanke 'for'?

Uporabne funkcije: `char`, `double`, `upper`, `isletter`, `unique`, `histcounts`.

Literatura

[1] C. E. Shannon: A mathematical theory of communication. Bell system technical journal, zv. 27, 1948.

[2] C. E. Shannon: Prediction and entropy of printed English. Bell Systems Technical Journal, zv. 30, str. 50–64, 1951.

[3] D.G. Luenberger: Information Science, Princeton University, str. 43-44, 2006.

Stiskanje podatkov

Uvod

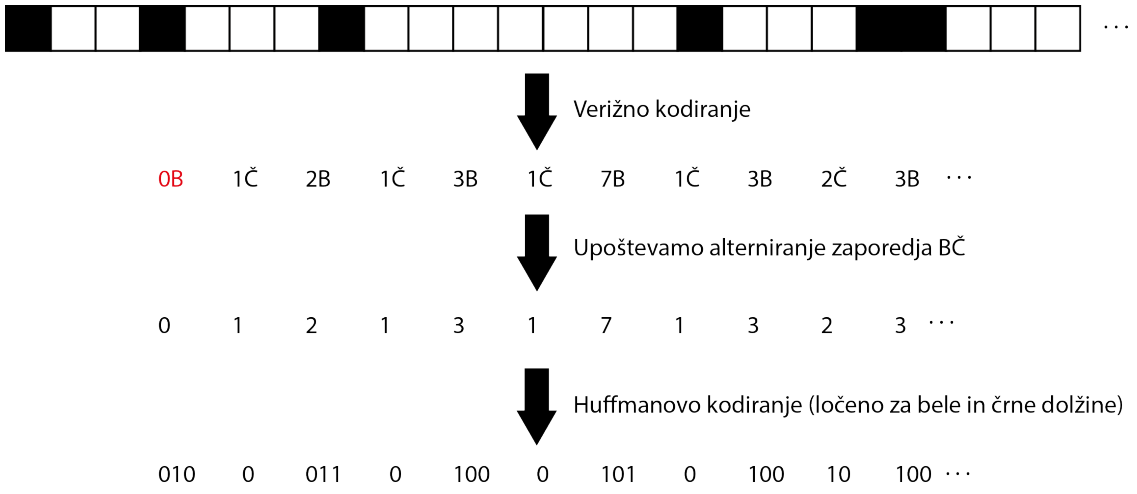
Tokrat se bomo posvetili problemu brezizgubnega stiskanja podatkov [1]. Poskusili bomo implementirati nekoliko prirejeno obliko postopka za stiskanje podatkov, ki ga uporabljajo telefaksi[1] pri pošiljanju slik in dokumentov preko telefonskega omrežja [2]. Postopek temelji na kombinaciji verižnega (ang. run-length encoding) in Huffmanovega kodiranja ter je zelo učinkovit pri kodiranju črno-belih slik.

Postopek stiskanja slik po standardu ITU-T T.4

V osnovi sliko, ki se pošilja po telefaksu, sestavlja poljubno število vrstic. V vsaki vrstici je 1728 črno-belih slikovnih točk. Kodiranje izvajamo nad vsako vrstico posebej. Najprej izvedemo verižno kodiranje, ki zaporedja belih ali črnih slikovnih točk nadomesti z njihovimi dolžinami. Standard predvideva, da se vsaka vrstica vedno začne z belo slikovno točko; če temu ni tako, na začetek vrstice umetno dodamo zaporedje belih slikovnih točk dolžine nič.

Sledi Huffmanovo kodiranje, ki se izvaja nad dolžinami zaporedij, ločeno za črne in za bele slikovne točke. Standard že vnaprej predpisuje Huffmanove kodne zamenjave za vse možne dolžine. Te kodne zamenjave so bile sestavljene na podlagi statistične analize večjega števila dokumentov. V naši nalogi teh predpisanih kodnih zamenjav ne bomo uporabili, ampak jih bomo izračunali na podlagi frekvenc dolžin zaporedij, ki se pojavijo v celotni sliki, ki jo želimo kodirati. Zgraditi moramo dve Huffmanovi drevesi: eno za zaporedja črnih dolžin in eno za zaporedja belih dolžin. Iz obeh dreves nato izluščimo dolžine kodnih zamenjav in uporabimo postopek za grajenje kanoničnih Huffmanovih kodnih zamenjav[2]. S temi kodnimi zamenjavami nato nadomestimo dolžine zaporedij črnih in belih slikovnih točk v vsaki vrstici posebej. Postopek kodiranja je nakazan na sliki 1.

[1] `https://en.wikipedia.org/wiki/Fax`
[2] `http://en.wikipedia.org/wiki/Canonical_Huffman_code`



Slika 1: Postopek kodiranja dela vrstice slike.

Huffmanov kod

Algoritem za kodiranje kot rezultat vrne optimalne dolžine kodnih zamenjav glede na podane verjetnosti znakov. V našem primeru znake predstavljajo dolžine zaporedij črnih ali belih slikovnih točk. Psevdo-koda:

```
1. Ustvari seznam osnovnih znakov L(1..n) s pripadajočimi verjetnostmi.
2. Ponavljaj dokler ne pridemo do sestavljenega znaka z verjetnostjo 1:
    - v seznamu L poišči znaka z najmanjšima verjetnostma[3];
    - ustvari nov sestavljen znak;
    - njegova verjetnost naj bo vsota verjetnosti obeh znakov;
    - najdena znaka označi kot uporabljena;
    - nov sestavljen znak dodaj v seznam L;
    - zabeleži si, iz katerih znakov je sestavljen.
3. Za vse sestavljene znake v L (od zadnjega proti prvemu):
    - znak razstavi, tj. poišči oba znaka, ki ga sestavljata;
    - obema sestavnima deloma povečaj dolžino kodne zamenjave za 1.
```

Dobili smo dolžine kodnih zamenjav za vse znake, ki nastopajo v vhodnem sporočilu. Sedaj želimo za vsak znak poiskati kodno zamenjavo. V tej nalogi je potrebno kodne zamenjave zapisati v kanonični obliki. Kanonična oblika omogoča zapis kodne tabele v zelo kompaktni obliki. Vse, kar potrebujemo za dekodiranje, je tabela znakov s pripadajočimi dolžinami kodnih zamenjav. Psevdo-koda za kanonični kod:

[3]V primeru, da je več znakov z enakimi verjetnostmi, upoštevajte njihovo vrednost - znak najprej izbiramo glede na število elementov, ki jih vsebuje, nato glede na vrednost (manjše ima prednost).

```
1. Uredi znake po dolžinah kodnih zamenjav (KZ)
   in nato še po njihovi vrednosti (naraščajoče).
2. KZ = zeros(1, dolžina KZ);
3. Dokler so znaki na vhodu:
    - izstavi znak in KZ;
    - KZ = (KZ + 1) << ((dolžina naslednje KZ) - (dolžina trenutne KZ));
```

Operator `<<` pomeni bitni pomik v levo. Primer: 10_2 `<<` $2 = 1000_2$

Naloga

Napišite funkcijo z imenom `naloga2` v programskem jeziku Matlab. Funkcija mora implementirati kodiranje črno-belih slik na osnovi zgoraj opisanega postopka. Vhodni argument funkcije `vhod` je matrika ničel (črne slikovne točke) in enic (bele slikovne točke), ki predstavlja sliko, ki jo želimo poslati po telefaksu. Matrika, ki je tipa `logical`, vsebuje poljubno število vrstic, vsaka vrstica pa vsebuje 1728 elementov, ki ustrezajo zajetim slikovnim točkam. Izhodni argumenti funkcije so štirje:

- zakodirana slika `izhod`, ki jo podate v obliki vrstičnega vektorja tipa `double`, kjer so vse vrstice zlepljene skupaj;
- kompresijsko razmerje `R`, ki se izračuna kot $\frac{|\text{izhod}|}{|vhod|}$, kjer operacija $|\cdot|$ vrne število elementov podanega vektorja oziroma matrike;
- tabela dolžin kodnih zamenjav belih slikovnih točk `kodBela`;
- tabela dolžin kodnih zamenjav črnih slikovnih točk `kodCrna`.

Tabeli `kodBela` in `kodCrna` naj bosta predstavljeni kot matriki. Imata naj toliko vrstic, kot je število različnih dolžin zaporedij in dva stolpca: v prvem se nahaja dolžina zaporedja, v drugem pa dolžina kodne zamenjave.

Prototip funkcije:

```
function [izhod, R, kodBela, kodCrna] = naloga2(vhod)
% Kodiranje slike "vhod" po prilagojenem standardu ITU-T T.4.
%
% vhod      - matrika, ki predstavlja sliko [n X 1728]
% izhod     - binarni vrsticni vektor
% R         - kompresijsko razmerje
% kodBela   - tabela dolzin kodnih zamenjav belih slikovnih tock
% kodCrna   - tabela dolzin kodnih zamenjav crnih slikovnih tock

izhod = NaN;
R = NaN;
kodBela = [];
kodCrna = [];
```

Testni primeri

Na spletni učilnici se nahaja arhiv `naloga2.zip`, ki vsebuje tri testne slike, za katere imate podane vhode in izhode. Primeri so podani v obliki datotek .mat, ki jih naložite v Matlab s pomočjo ukaza `load ime_datoteke.mat`. Priloženo imate tudi funkcijo `test_naloga2`, ki jo lahko uporabite za preverjanje pravilnosti rezultatov, ki jih vrača vaša funkcija (primer klica: `test_naloga2('primeri',1);`). Pri testiranju vaše funkcije upoštevajte naslednje omejitve:

- rezultat je pravilen:
  - če se od danega razlikuje za manj kot 10^{-6} v primeru `R`;
  - vektor `izhod` in kodni tabeli `kodBela` in `kodCrna` naj bi se popolnoma ujemali z rešitvami. To pa ne bo čisto res, če ste dvoumnosti v implementaciji Huffmanovega kodiranja reševali drugače kot mi. V primeru, da ni ujemanja, bomo med preverjanjem nad `izhod` pognali dekodirnik (`FaxDecode.p`). Če se bo dekodirana slika ujemala z `vhod`, je vaš rezultat za `izhod`, `kodBela` in `kodCrna` pravilen.
- točkovanje:
  - 0,5 točke za pravilen `R`,
  - 0,5 točke za pravilne `izhod`, `kodBela` in `kodCrna`.
- izvajanje vaše funkcije je časovno omejeno na 60 sekund.

Namigi

Uporabne funkcije:
`sortrows()`, `dec2bin()`, `find()`, `unique()`, `histc()`, `min()`, `diff()`.

Literatura

[1] D.G. Luenberger: Information Science, Princeton University, pogl. 4, 2006.

[2] Protocols for Telematic Services-Standardization of Group 3 facsimile apparatus for document transmission, ITU-T T.4, CCITT.

Varno kodiranje

Električne ali magnetne motnje v računalniškem sistemu lahko povzročijo, da posamezni biti v glavnem pomnilniku (RAM) spontano spremenijo vrednost, kar lahko privede do nepravilnega delovanja računalniškega sistema. Z večanjem gostote zapisa na pomnilniških čipih in nižanjem napajalnih napetostih postaja ta težava vedno bolj pereča. Zagotavljanje visoke odpornosti pomnilniških vezij na motnje iz okolja in preprečevanje napak pri delovanju je še posebej pomembno v sistemih kot so podatkovni strežniki, industrijski krmilniki, sateliti in vesoljske sonde. Integriteta podatkov se v teh sistemih običajno zagotavlja z uporabo naprednih materialov pri gradnji, ki ščitijo podatke pred elektromagnetnim sevanjem in uporabo kodov za odkrivanje in odpravljanje napak.

Hammingov kod

Spekter kodov, ki se uporabljajo pri odpravljanju napak na podatkovnih medijih in v komunikaciji, je zelo širok. Zelo priljubljeni so na primer Reed-Solomonovi in konvolucijski kodi, vendar je izbor optimalnega koda za določeno aplikacijo močno odvisen od okolja, v katerem bo naprava delovala, računskih zmogljivosti strojne opreme, cene in drugih dejavnikov. Danes se na primer zaradi zagotavljanja hitrosti delovanja v pomnilnikih ECC DRAM (ang. Error Checking and Correcting Dynamic Random Access Memory) še vedno pogosto uporablja Hammingov kod H(127, 120), ki je zaradi strukture pomnilnika prilagojen tako, da z 8 varnostnimi biti ščiti 64 podatkovnih bitov. Obstajajo tudi naprednejše tehnologije za odkrivanje in odpravljanje napak, kot je IBM Chipkill, HP Chip spare in Intel SDDC.

Hammingov kod ste podrobno spoznali že na predavanjih (glej tudi [2]). Pri tokratni domači nalogi bomo uporabili družino sistematičnih Hammingovih kodov $H(n, k)$.

CRC

CRC (ang. Cyclic Redundancy Check) je družina cikličnih kodov za odkrivanje napak, ki se uporablja predvsem v digitalnih komunikacijskih napravah za detekcijo napak pri prenosu. Zelo razširjeni so postali zato, ker jih je mogoče enostavno implementirati v strojni opremi in se še posebej dobro obnesejo pri odkrivanju napak, povzročenih zaradi šuma v komunikacijskih kanalih. Za popravljanje napak niso najbolj primerni, zato v primeru napake oddajnik sporočilo običajno pošlje ponovno. Ciklični kod CRC-32 najdemo na primer v standardih Ethernet, SATA in MPEG-2, ciklični kod CRC-16 pa pri komunikaciji Bluetooth in v pomnilniških karticah SD, CRC-8-CCITT pa v vgrajenih sistemih.

V tej nalogi boste implementirali kod CRC na osnovi standarda CRC-16. Parametri standarda so naslednji:

- Polinom: 0x1021 $\rightarrow g(p) = p^{16} + p^{12} + p^5 + 1$,
- Začetna vrednost registra: 0x00,
- Prezrcali podatkovni bajt: Ne,

- XOR nad CRC: Ne,
- Prezrcali CRC: Ne,
- Primer vrednosti CRC za niz ASCII znakov "123456789": 0x31C30. Vsak znak vhodnega niza je v tem primeru kodiran z 8 biti, CRC pa je prikazan šestnajstiško.

Delovanje vašega programa lahko preverite tudi preko spletne strani `http://crccalc.com/`. V tem primeru glejte rezultate v vrstici `CRC-16/XMODEM`.

Naloga

V tokratni nalogi boste spoznali uporabo linearnih bločnih kodov za simetrični binarni komunikacijski kanal [1]. Scenarij gre takole:

1. Generirali smo binarni niz (sporočilo) z.
2. Preden sporočilo z pošljemo v komunikacijski kanal, ga obogatimo z m varnostnimi biti. V kanalu namreč lahko pride do napak. Za varovanje smo uporabili enega izmed sistematičnih Hammingovih kodov $H(n, k)$, kjer je $n = 2^m - 1$ in $k = n - m$. Tako zavarovano sporočilo z sedaj imenujemo x.
3. Zavarovano sporočilo x pošljemo po kanalu. Med prenašanjem se lahko njegova vsebina pokvari, kar pomeni, da se določenemu številu bitov obrne vrednost.
4. Na izhodu iz kanala sporočilo prevzameš ti. Tvoja naloga je, da popraviš morebitne napake v njem. Pri tem seveda uporabiš ustrezen kod. Sporočilu, ki pride iz kanala, pravimo y. Ko odkodiraš y, dobiš $\hat{z}$, ki je v najboljšem primeru enak poslanemu sporočilu z.
5. Sporočilo $\hat{z}$ vrneš kot rezultat v vrstičnem binarnem vektorju `izhod`.
6. Nad y izračunaš CRC vrednost po standardu CRC-16 in jo vrneš kot rezultat v šestnajstiškem zapisu v spremenljivki `crc`.

Napišite funkcijo z imenom `naloga3` v programskem jeziku Octave. Funkcija mora implementirati dekodiranje sporočil y, ki ste jih prejeli iz zašumljenega kanala. Funkcija `naloga3` kot vhodne argumente sprejme vrstični vektor `vhod`, `n` – dolžina kodne zamenjave in `k` – število podatkovnih bitov v kodni zamenjavi. Argumenta `n` in `k` definirata, kateri Hammingov kod je potrebno uporabiti. Vektor `vhod` predstavlja sporočilo y na izhodu iz kanala, njegovi elementi so ničle in enice (vektor tipa `double`). Izhodni argument funkcije je odkodirano sporočilo `izhod`, ki je sestavljeno iz podatkovnih bitov $\hat{z}$. Spremenljivka `izhod` mora biti vrstični vektor tipa `double`. Izhodni argument `crc` predstavlja CRC vrednost izračunano po standardu CRC-16 nad vektorjem `vhod`. Zapisan naj bo kot število v šestnajstiški obliki (niz šestnajstiških števk).

Prototip funkcije:

```
function [izhod, crc] = naloga3(vhod, n, k)
% Izvedemo dekodiranje binarnega niza vhod, ki je bilo
% zakodirano s Hammingovim kodom H(n,k)
% in poslano po zasumljenem kanalu.
% Nad vhodom izracunamo vrednost crc po standardu CRC-16.
%
% vhod  - binarni vektor y (vrstica tipa double)
% n     - stevilo bitov v kodni zamenjavi
% k     - stevilo podatkovnih bitov v kodni zamenjavi
% crc   - crc vrednost izracunana po CRC-16
%         nad vhodnim vektorjem (sestnajstisko)
% izhod - vektor podatkovnih bitov, dekodiranih iz vhoda
```

Testni primeri

Na učilnici se nahaja arhiv `TIS-naloga3.zip`, ki vsebuje tri testne primere sporočil. Primeri so podani v obliki datotek .mat, ki jih naložite v Octave s pomočjo ukaza `load ime_datoteke.mat`. Priloženo imate tudi funkcijo `test_naloga3`, ki jo lahko uporabite za preverjanje pravilnosti rezultatov, ki jih vrača vaša funkcija. Primer klica funkcije za preverjanje: `test_naloga3('primeri',1);`). Pri testiranju vaše funkcije upoštevajte naslednje omejitve:

- pol točke dobite, če se `izhod` ujema z rešitvijo,
- pol točke dobite, če se `crc` ujema z rešitvijo,
- izvajanje funkcije je časovno omejeno na 120 sekund.

Literatura

[1] D.G. Luenberger: Information Science, Princeton University, pogl. 6, 2006.

[2] H. S. Warren: Hacker's Delight, Second Edition, Addison-Wesley, Boston, 2012.

Signali in vzorčenje: razpoznavalnik akordov

Na koncu semestra se gotovo prileže malo glasbe. Osnovni gradnik glasbe je ton [1]. Vsak ton ima svoje ime in svojo frekvenco, npr. $A_1 = 440$ Hz. Ko poslušamo glasbo, največkrat poslušamo sozvočje večih tonov - če so ti toni trije ali jih je več, temu sozvočju pravimo akord [2]. V tokratni nalogi se bomo poglobili v spektralno analizo zvočnega signala in skušali razpoznati akorde, ki so morebiti prisotni v danih posnetkih.

Zaradi enostavnosti se bomo pri razpoznavanju omejili na akorde iz treh tonov, pri čemer vzamemo samo dve klavirski oktavi tonov: od C_1 do B_2 (oktava je interval 12 poltonov) [3]. Ton, ki je za oktavo višji od drugega tona, ima dvokratnik njegove frekvence. Primer: ton A_2 ima frekvenco 880 Hz, kar je dvakrat več kot A_1, torej $2 * 440$ Hz. V spodnji tabeli so navedeni vsi toni in njihove frekvence, ki lahko sestavljajo iskane akorde.

C_1	CIS_1	D_1	DIS_1	E_1	F_1	FIS_1	G_1	GIS_1	A_1	B_1	H_1
261,63	277,18	293,66	311,13	329,63	349,23	369,99	392	415,30	440	466,16	493,88
C_2	CIS_2	D_2	DIS_2	E_2	F_2	FIS_2	G_2	GIS_2	A_2	B_2	H_2
523,25	554,37	587,33	622,25	659,25	698,46	739,99	783,99	830,61	880	932,33	987,77

Naslednja tabela vsebuje imena in sestavne tone vseh akordov, ki jih želimo iskati v zvočnem signalu.

Cdur	C_1	E_1	G_1
Cmol	C_1	DIS_1	G_1
Ddur	D_1	FIS_1	A_1
Dmol	D_1	F_1	A_1
Edur	E_1	GIS_1	H_1
Emol	E_1	G_1	H_1
Fdur	F_1	A_1	C_2
Fmol	F_1	GIS_1	C_2
Gdur	G_1	H_1	D_2
Gmol	G_1	B_1	D_2
Adur	A_1	CIS_2	E_2
Amol	A_1	C_2	E_2
Hdur	H_1	DIS_2	FIS_2
Hmol	H_1	D_2	FIS_2

Opomba: pri molovskih akordih je navada, da se srednji ton zapiše kot znižan in ne kot zvišan ton. Primer za C-mol: ton DIS_1 je za pol tona zvišan ton D_1 in je v glasbeni teoriji enak znižanemu E_1, ki se mu reče ES_1. Ker gre za tone z istimi frekvencami se s tem ne bomo obremenjevali.

Naloga

Vaša naloga je, da v Matlabu napišete funkcijo `naloga4()`, ki bo razpoznavala akord v zvočnem posnetku. Funkcija prejme zvočni signal `vhod` in frekvenco vzorčenja `Fs`. Razpoznavo

izvedete tako, da signal transformirate s pomočjo hitre Fourierove transformacije – uporabite lahko vgrajeno funkcijo `fft()`, ki signal iz časovnega prostora prestavi v frekvenčni prostor. Iz transformiranega signala je namreč mogoče razbrati, kateri toni (sinusni signali) nastopajo v posnetku in iz tega ugotoviti, ali sestavljajo katerega izmed iskanih akordov. Izhod funkcije `naloga4()` je niz z imenom akorda, npr. `'Cdur'`. Če v zvočnem signalu `vhod` ni prisotnega nobenega akorda iz zgornje tabele, potem naj funkcija vrne prazen niz (`''`). V vsakem posnetku je prisoten največ en akord, ki traja od začetka do konca signala.

Pri razpoznavi akordov vam bo prav prišel izračun močnostnega spektra zvočnega posnetka. Upoštevajte, da je vhodni signal realen in je zato Fourierov transform simetričen glede na Nyquistovo frekvenco.

Zvočni posnetki lahko vsebujejo tudi "motilne" frekvence, ki pa niso enake frekvencam podanih tonov v zgornji tabeli (tudi niso enake frekvencam tonov v nižjih in višjih oktavah). Te je potrebno pri razpoznavanju ignorirati. Prav tako se lahko pojavijo tudi posnetki, ki ne vsebujejo vseh potrebnih tonov za določen akord ali pa sploh ne vsebujejo nobenega tona - v teh primerih je izhod vašega programa prazen niz.

Poseben primer predstavljajo posnetki, kjer so prisotni višjeharmonski ali alikvotni toni [4, 5] – to so celoštevilski večkratniki osnovnih tonov. Zaradi poenostavitve predpostavljajte samo frekvence, ki so dvakrat višje, kot so osnovni toni. Primer: akord C-dur sestavljajo toni C_1, E_1 in G_1. Če so poleg njih prisotni tudi kateri od njihovih dvokratnikov, torej C_2, E_2 ali G_2, jih razumemo kot višje harmonike in jih lahko ignoriramo – torej razpoznamo C-dur.

Na spletni učilnici oddajte datoteko `naloga4.m`, v kateri je implementirana zahtevana funkcija.

Prototip funkcije:

```
function izhod = naloga4(vhod,Fs)
% Funkcija naloga4 skusa poiskati akord v zvocnem zapisu.
%
% vhod  - vhodni zvocni zapis (vrsticni vektor tipa double)
% Fs    - frekvenca vzorcenja
% izhod - ime akorda, ki se skriva v zvocnem zapisu (niz);
%         ce frekvence v zvocnem zapisu ne ustrezajo nobenemu
%         od navedenih akordov, vrnemo prazen niz [].
```

Testni primeri

Na učilnici se nahaja arhiv `TIS-naloga4.zip`, ki vsebuje tri testne primere. Primeri so podani v obliki datotek `.mat`, ki jih naložite v Matlab s pomočjo ukaza `load ime_datoteke.mat`. Priloženo imate tudi funkcijo `test_naloga4`, ki jo lahko uporabite za preverjanje pravilnosti rezultatov, ki jih vrača vaša funkcija (primer klica: `test_naloga4('primeri',1);`).

Pri posameznem testnem primeru dobite točko, če se `izhod` popolnoma ujema z rešitvijo, ki je bodisi ime akorda bodisi prazen niz. Izvajanje funkcije je časovno omejeno na 20 sekund.

Rešitve, kjer funkcija `naloga4()` vedno vrača vnaprej določen rezultat, ki ni dobljen algoritmično (t.i. "hard-coded" rešitve), bodo ocenjene z 0 točkami.

Literatura

[1] Wikipedia: `https://sl.wikipedia.org/wiki/Ton`.

[2] Wikipedia: `https://sl.wikipedia.org/wiki/Akord`.

[3] Wikipedia: `https://en.wikipedia.org/wiki/Piano_key_frequencies`.

[4] Wikipedia: `https://sl.wikipedia.org/wiki/Alikvotni_toni`.

[5] Wikipedia: `https://en.wikipedia.org/wiki/Harmonic`.