ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra teorie obvodů

Komprese EEG signálu

Diplomová práce

Studijní program: Komunikace, multimedia a elektronika
Studijní obor: Bezdrátové komunikace

Vedoucí práce: Ing. Jakub Šťastný, Ph.D.

Václav Dajčar

Praha 2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Václav Dajčar

Název projektu: Komprese EEG signálu

Zásady pro vypracování:

1. Seznamte se s problematikou zpracování a vlastnostmi EEG a základními algoritmy pro kompresi obecných signálů a kompresi EEG signálu.
2. Srovnejte různé algoritmy. Zvolte vhodný přístup ke kompresi EEG signálu, navrhněte vhodný formát souboru pro ukládání komprimovaných dat. Implementujte funkce pro kompresi analyzovanými algoritmy a porovnejte dosažené kompresní poměry pro testovací EEG databázi.
3. Kompresní algoritmus integrujte do existujícího toolboxu pro zpracování EEG signálů. Přínos komprese vyhodnoťte pomocí klasifikačního experimentu s pohybovými EEG daty.

Vedoucí práce: Ing. Jakub Šťastný, Ph.D.

Prohlášení

Tímto prohlašuji, že jsem svoji diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, programové prostředky) uvedené v přiloženém seznamu.

V Praze dne 1. 1. 2014 Václav Dajčar

Poděkování

Děkuji především vedoucímu své práce, Ing. Jakubu Šťastnému, Ph.D., za četné, ochotné a odborné konzultace a pomoc, které přispěly k jejímu řešení. Dále bych chtěl ještě poděkovat všem, kteří mě hnali kupředu.

Anotace

Tato diplomová práce se zaobírá bezeztrátovou kompresí EEG dat. Komprimovaná data zabírají méně úložného prostoru a mohou tak být případně rychleji přenesena. Práce představuje vhodné komprimační techniky teoreticky a dále popisuje jejich praktické využití v rámci Toolboxu MEET. Využit je Huffmanův algoritmus a aritmetické kódování. EEG data se před samotnou komprimací vhodně upraví, aby zdrojová abeceda nabývala přiměřené mohutnosti. Výpočetním prostředím je software Matlab. Dosažené výsledky jsou porovnány a zhodnoceny z hlediska komprimačních poměrů a doby běhu jednotlivých algoritmů.

Klíčová slova: Bezeztrátová komprese, EEG data, Huffmanův algoritmus, aritmetické kódování, prediktor.

Annotation

This thesis focuses on a lossless compression of EEG data. Compressed data requires less storage space and thus can be transferred more quickly. This paper presents appropriate compression techniques theoretically and describes their practical usage within the Toolbox MEET. In the practical implementation the Huffman algorithm and arithmetic coding is deployed. EEG data are appropriately adjusted before the compression in order to acquire adequate source of alphabet cardinality. As a numerical computing environment the Matlab software is used. The obtained results are then compared and evaluated in terms of compression ratio and runtime performance of the algorithms.

Keywords: lossless data compression, EEG data, Huffman algorithm, arithmetic coding, predictor.

Obsah

Seznam použitých symbolů

A	průměrná délka kódových slov
c_i	délka i-tého kódového slova
C	kompresní poměr
D_{orig}	délka původních dat
D_{komp}	délka komprimovaných dat
e_n	rozdíl mezi skutečnou a predikovanou hodnotou
$E(.)$	operátor střední hodnoty
F_{ij}	prvky četnostní matice
H	entropie
p_i	pravděpodobnost výskytu i-tého symbolu
P_C, P_N	kumulativní pravděpodobnosti
R_x	korelační matice
V, σ^2	variance
x_n	n-tý vstupní vzorek
X_n	diskrétní náhodná proměnná
$\sum$	operátor sumace
$\prod$	operátor součinu

Seznam použitých zkratek

ANN –	Artificial Neural Network
BCI	brain computer interface
CHT	Collapsed Huffman Tree
DCT	Discrete Cosine Transform
EEG	elektroencefalogram
FPGA	Field Programmable Gate Array
IEEE	The Institute of Electrical and Electronics Engineers
KLT	Karhunen-Loeve transformace
LPC	Linear Predictive Coding
LZ77	Lempel-Ziv 77
MEET	Modular EEg processing Toolbox
PCM	Pulse Code Modulation
RLE	Run Length Encoding

Seznam obrázků

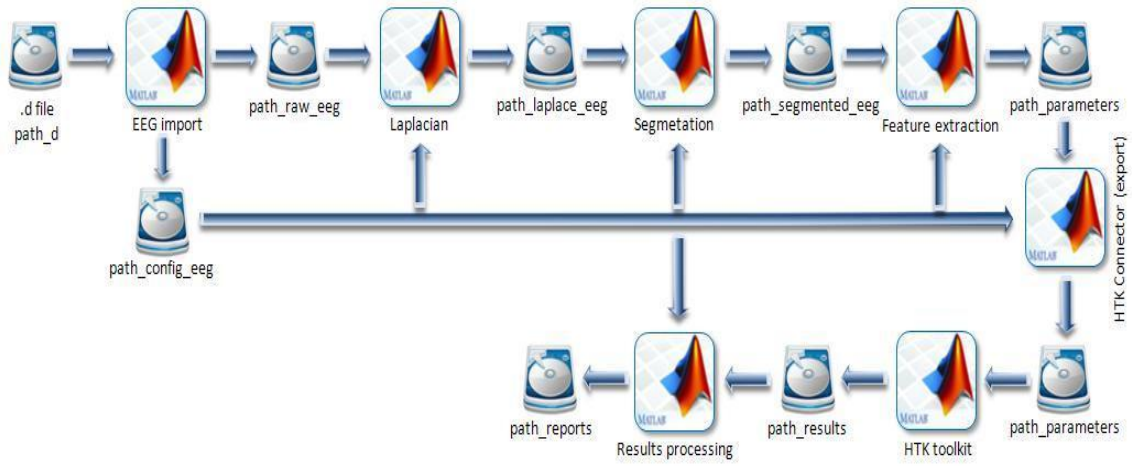
Seznam tabulek

1 Úvod

1.1 Motivace

Snaha o přenos a uchovánání čím dál tím většího obému dat, aniž by musela být zvětšována šířka pásma či úložný prostor, vedla již od 50. let 20. století, kdy svou optimální metodu kódování přenášených zpráv uveřejnil David A. Huffman [1], ke stále propracovanějším způsobům vypouštění dat nadbytečných. Technika vedoucí ke zmenšení využitého elektronického úložného prostoru daty či zkrácení času přenosu, u přenosových systémů pak ke snížení nároků na paměť, se nazývá komprese. Matematickým aparátem je informační teorie, efektivně ukládat či přenášet je pak možné komprimovaný text, zvuk, obraz a signálová data.

Komprimace signálových dat, konkrétně záznamu mozkové aktivity, tzv. elektroencefalografických dat, zkráceně EEG, je předmětem této práce. EEG data jsou zpracovávána a vyhodnocována v MEET (Modular EEg processing Toolbox) sloužícím jako rozhraní mezi mozkem a počítačem (BCI – Brain Computer Interface), který je prací v rámci FPGA laboratoře Katedry teorie obvodů na ČVUT [12]. Čím dál tím větší množství zpracovaných dat klade značné nároky na úložný prostor (řádově stovky gigabytů), a i když je cena za jeden gigabyte nízká, může být úložný prostor při užití vhodné kompresní techniky efektivně využit, nehledě na rychlejší načítání dat zmenšeného objemu. Na obrázku 1 je soustava jednotlivých částí MEET, komprese bude aplikována v modulech, které pracují s EEG signálem.



Obrázek 1: Cesta zpracovávaného EEG signálu v MEET [12]

1.2 Cíle práce

Hlavním cílem této práce je návrh vhodného kompresního algoritmu pro EEG signál. Algoritmus zajišťující bezeztrátovou kompresi bude využit pro potřeby Toolboxu FPGA laboratoře katedry obvodů. Softwarovým prostředím, v němž budou porovnávané algoritmy implementovány, je výpočetní prostředí Matlab. Jednotlivé kroky k dosažení hlavního cíle jsou:

1. Studie existujících kompresních algoritmů a jejich porovnání s ohledem na možnosti komprese EEG signálu
2. Analýza možností, které poskytuje samotné prostředí Matlab
3. Implementace vybraných algoritmů a srovnání jejich kompresních poměrů

1.3 Struktura práce

První část obsahuje motivaci pro uskutečnění této práce a její cíle.

Druhá část popisuje posouzení různých kompresních algoritmů z hlediska doby běhu, očekávaného kompresního poměru, komplexity implementace a paměťové náročnosti. Zahrnuje rovněž různé přístupy ke kompresi signálových dat.

Část třetí představuje kompresní techniky vhodné přímo pro EEG data, dále obsahuje přehled kompresních poměrů jednotlivých algoritmů.

Ve čtvrté kapitole se nachází rozbor způsobu ukládání dat výpočetním prostředím Matlab. Zodpovězeny jsou základní otázky, jako kolik bytů připadá na jedno číslo, jsou-li možné různé formáty ukládaných komprimovaných dat a je-li možné pomocí formátu vektoru ovlivnit způsob ukládání.

Kapitola pátá popisuje srovnání implementovaných algoritmů pro kompresi a dekompresi v prostředí Matlab. Obsahuje tabulku s dosaženými kompresními poměry algoritmů pro testovací EEG data. Zdůvodněn bude také výběr výsledného algoritmu, který bude v Toolboxu implementován.

Část šestá obsahuje zprávu o samotném zahrnutí vybraného algoritmu do Toolboxu a odhad úspory místa na disku díky kompresi EEG dat.

Poslední sedmá kapitola shrnuje práci a dosažené výsledky.

2 Kompresní algoritmy

Pro EEG signál bylo v [2] probráno hned několik bezeztrátových technik splňujících základní požadavek, kterým je perfektní rekonstrukce původní informace. V klinické praxi je pak tato podmínka nadřazena i lepšímu kompresnímu výkonu, kterého lze dosáhnout téměř bezeztrátovou kompresí [3].

Komprese lze dosáhnout přiřazením kratší posloupnosti bitů, tzv. kódového slova, pro nejčastěji se vyskytující symboly, vzorky dat a nezbytně delší bitové sekvence méně často se objevujícím vzorkům. Horní hranici datové komprese stanovuje Shannonův teorém vyjádřený entropií, která je založena na pravděpodobnostní distribuci informačního zdroje:

$$H = -\sum_{i=1}^{M} p_i \log_2(p_i) \qquad (2.1),$$

přičemž p_i je pravděpodobnost i-tého symbolu ze zdrojové abecedy $A=\{a_1,\ldots,a_M\}$. Kompresní poměr je pak dán vztahem:

$$C_{lim} = 1 - \frac{H}{b} \qquad (2.2),$$

kde b je původní pevně stanovený počet bitů na symbol.

Kompresní poměr se velmi často vyjadřuje procentuálně jako:

$$C = 100 \cdot \frac{D_{orig} - D_{komp}}{D_{orig}} \qquad (2.3),$$

D_{orig} a D_{komp} jsou délky původních a komprimovaných dat.

2.1 RLE (Run Length Encoding) kódování

Přístup této bezeztrátové metody kódování je následovný [4]: Objeví-li se položka *d* v *n* po sobě jdoucích položkách datového proudu, lze *n* výskytů nahradit jedinou dvojicí *nd*. Po přečtení prvního znaku se čítač nastaví na hodnotu 1 a znak se uloží. Následující znaky jsou porovnány s již uloženým znakem, a jsou-li shodné, hodnota čítače se navýší. Je-li načten jiný znak, operace závisí na hodnotě čítače. V případě nízké hodnoty se uložený znak zapíše do komprimovaného souboru a nově načtený znak se uloží. Jinak se zapíše únikový znak @, který je nezbytný pro dekompresi a který předchází dvojici *nd*, následovaný hodnotou čítače a uloženým znakem.V případě,

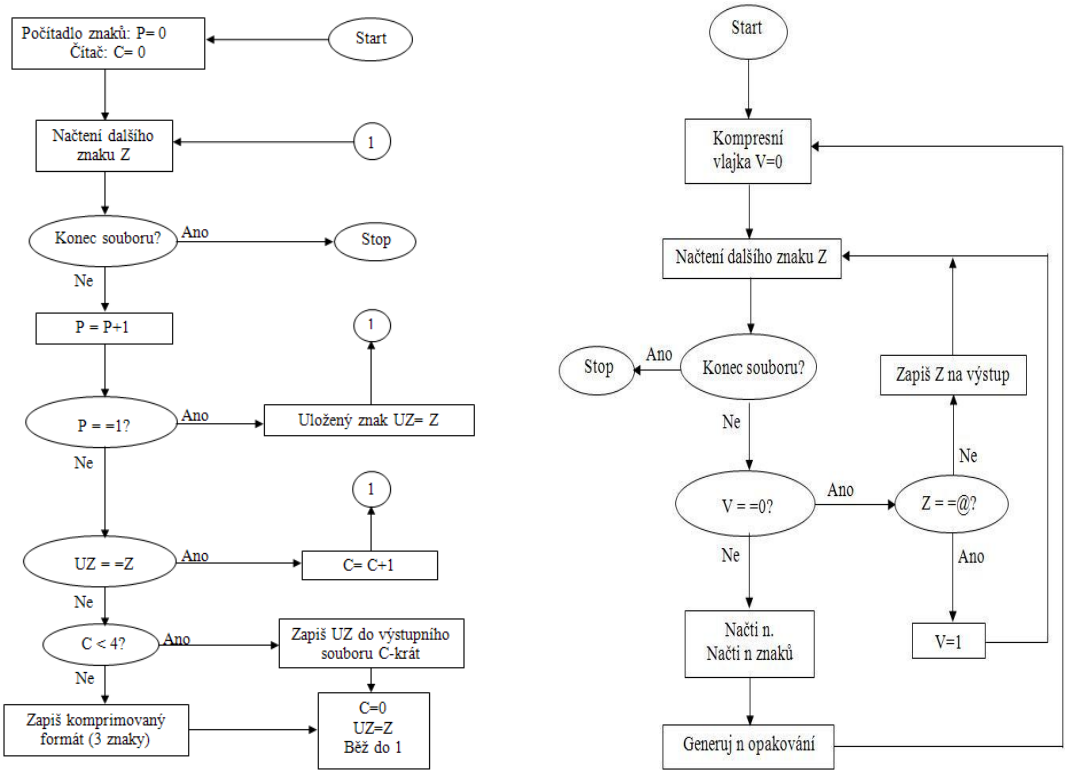
že by vstupní datový proud obsahoval všechny možné znaky zdrojové abecedy, lez využít metody MNP5, popsané v [4].

Dekomprese je přímočará. Je-li načten únikový znak @, čítač *n* spolu se samotným znakem *d* se načtou, znak *d* se pak n-krát zapíše do výsledného dekomprimovaného souboru.

Z výše uvedeného vyplývá, že ke kompresi dojde pouze tehdy, opakuje-li se položka *d* více jak čtyřikrát. Určitou představu o kompresních poměrech RLE získáme z následující úvahy, kdy uvažujeme datový soubor, řetězec o N znacích. Dále předpokládáme, že řetezec obsahuje M opakování průměrné délky L. Jednotlivá z M opakování jsou nahrazena třemi znaky (únikovým, čítačem, daty), takže velikost komprimovaného řetězce je N-MxL+3M. Kompresní poměr je dán vztahem:

$$C_{RLE} = \frac{N}{N - M(L-3)} \qquad (2.4),$$

Diagram kompresního a dekompresního algoritmu je znázorněn na obrázku 2.



Obrázek 2: Diagram kompresního (vlevo) a dekompresního (vpravo) RLE algoritmu [4]

2.2 Diferenciální PCM (Pulse Code Modulation)

Diferenciální pulsně kódová modulace je technika diferenciální bezeztrátové komprese [4], která se užívá především při kompresi obrazu. Spočívá v porovnávání každého symbolu p se symbolem referenčním, který je jedním z jeho předešle zakódovaných bezprostředních sousedů, a zakódování p do dvou částí: předpony a přípony. Předpona je počtem nejvýznamnějších bitů p stejných s bity referenčního symbolu a nabývá celočíselné hodnoty 0 až 8. Příponou jsou zbývající nejméně významné bity. Za účelem dalšího vylepšení komprese se předpona může dále kódovat, např. Huffmanovým kódováním buď s devíti pevně stanovenými kódovými slovy nebo s adaptivními kódovými slovy. Přípona se do výstupního souboru zapisuje nekódovaná, jelikož se předpokládá, že většina přípon nabývá malých hodnot. Referenční symbol je možné volit jedním ze tří způsobů:

- Symbol bezprostředně po levé straně právě kódovaného symbolu,
- Symbol nad právě kódovaným znakem,
- Znak na levém okraji s výjimkou předpony kratší než předurčený práh, kdy by referenčním byl symbol z předchozího bodu.

Metoda diferenciálního kódování počítá rozdíly $d_i=a_i-a_{i-1}$ mezi po sobě jdoucími datovými položkami a_i a kóduje, ať již ztrátově či bezeztrátově, diferenci d_i. První datová položka a_0 se nekóduje.

2.3 Huffmanovo kódování

Tato technika dosahuje komprese kratšími kódovými slovy pro nejčastěji se vyskytující symboly, méně často se objevující symboly pak mají delší kódová slova. Huffmanovy kódy jsou prefixové kódy minimální délky [6]. Kódování se nazývá prefixové, jestliže je prosté, kdy různým zdrojovým
znakům odpovídají vždy různá kódová slova, a žádné kódové slovo není prefixem jiného kódového slova.

Algoritmus Huffmanova kódování začíná sestupným seřazením pravděpodobností výskytu zdrojových symbolů [4], [5], [6]. Poté je tvořen strom se symbolem v každém listu od spodu nahoru. V každém kroku sestavování stromu se vyberou dva symboly s nejnižšími pravděpodobnostmi, přesunou se na vrchol částečného stromu v podobě libovolného symbolu, který representuje symboly původní. Seznam symbolů se postupně snižuje, nakonec jsou všechny symboly redukovány do jediného znaku, který představuje celou zdrojovou abecedu, tím je strom dokončen. Zpětným procházením stromu se získají jednotlivá kódová slova. Na obrázku 3 jsou znázorněny dva příklady

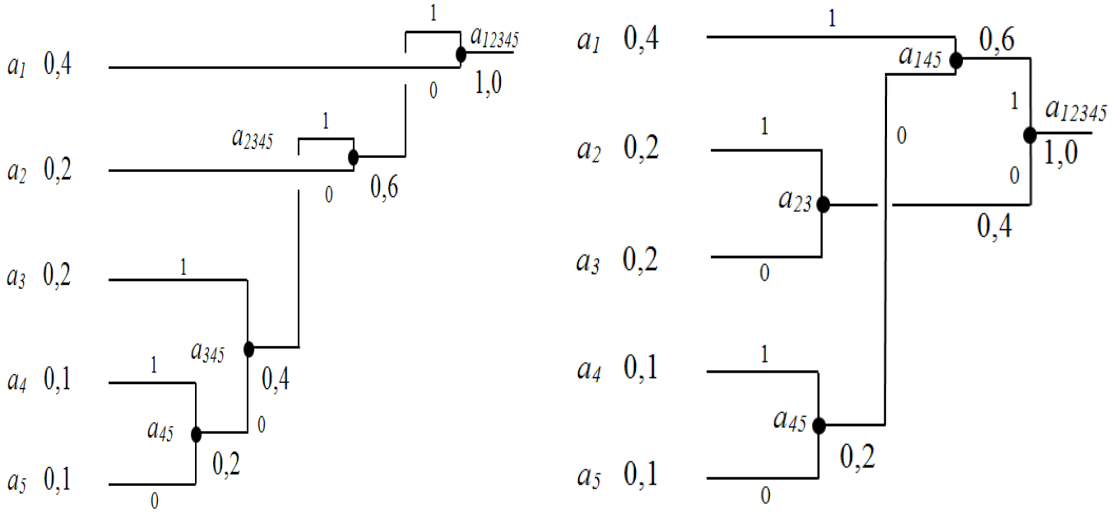
Huffmanova algoritmu kódujícího tutéž zdrojovou abecedu. Přiřazování bitů v jednotlivých větvích je libovolné a vede k rozdílným kódům, které však mají shodnou průměrnou délku A jednotlivých kódových slov délky c_i a pravděpodobnosti p_i výskytu symbolu a_i:

$$A = \sum_{1}^{n} c_i p_i \qquad (2.5)$$

Výběr nejlepšího kódu pak určuje nejnižší hodnota variance:

$$V = \sigma^2 = E(a_i - A)^2$$
$$= \frac{1}{n}\sum_{1}^{n}(a_i - A)^2 \qquad (2.6)$$

Variance je však významná tehdy, je-li nutné komprimovaný proud dat posílat komunikačním kanálem, při pouhém zápisu dat do souboru variance nehraje žádnou roli.



Obrázek 3: Dva příklady Huffmanova kódování zdrojových znaků a_i [4]

2.4 Aritmetické kódování

Dle Shannona je nejlepším, ve smyslu minimální průměrné délky, možným takový kompresní kód, pro který platí: k zakódování symbolu je potřeba $-lg\ p$ bitů, kde p je pravděpodobnost výskytu symbolu. Využitím aritmetického kódování za předpokladu přesného modelu pravděpodobnosti výskytu každého symbolu lze téměř dosáhnout optimálního výsledku. Nejdůležitější výhoda této metody kódování tkví v její

flexibilitě, může být totiž použita ve spojení s jakýmkoliv modelem, který může být i adaptivní, jež umí poskytnout sekvenci pravděpodobností. Propracované modely vstupních dat tedy vedou k velkým kompresním ziskům. Zmíněná flexibilita má však i stinnou stránku v podobě značných časových a prostorových nároků na datové struktury modelu, což vyúsťuje v pomalost kódování. Jak Huffmanovo tak i Ziv-Lempel kódování jsou rychlejší, protože pravděpodobnostní model je reprezentován přímo v datových strukturách použitých pro kódování. Další nevýhodou je, že tato metoda obecně nevytváří prefixový kód, což vylučuje souběžné kódování více procesory. Menším nedostatkem je pak potřeba označení konce souboru a slabá odolnost vůči chybám, obzvláště v součinnosti s adaptivními modely, což však lze prakticky vyřešit užitím vhodného kódování na opravu chyb [7].

Algoritmus aritmetického kódování má následující průběh [7]:

- Na začátku se zvolí „současný interval“ [D, H) nastavený na [0, 1).
- Pro každý symbol v souboru se provedou následující dvě operace:
    - Současný interval se rozdělí do podintervalů pro jednotlivé možné symboly abecedy. Velikost podintervalu je úměrná určené pravděpodobnosti, že symbol bude dalším v souboru, podle modelu vstupních dat.
    - Zvolí se podinterval odpovídající symbolu, který se v souboru objeví jako další, a tento se stane novým současným intervalem.
- Výstupem je dostatečný počet bitů tak, aby se výsledný interval odlišil od všech možných výsledných intervalů.

V druhém kroku je potřeba spočítat pouze podinterval odpovídající symbolu a_i, který se skutečně objeví. K tomu je nutné určit kumulativní pravděpodobnosti:

$$P_C = \sum_{k=1}^{i-1} p_k \qquad (2.7),$$

$$P_N = \sum_{k=1}^{i} p_k \qquad (2.8).$$

Nový podinterval je pak $[D + P_C (H - D), D + P_N (H - D))$.

3 Techniky komprese EEG dat

První hlubší výzkum v oblasti komprese EEG dat byl proveden v[2]. Kompresní algoritmy byly testovány na datech z 20 kanálového EEG při vzorkovací frekvenci 128 Hz na kanál a 8 bitovou přesností. Použité metody sahaly od Huffmanova kódování přes prediktivní techniky až po transformační metody. Aritmetické kódování pak bylo použito v [10]. Využití neuronových sítí jakožto prediktorů ve dvoustupňovém bezeztrátovém schématu popisuje [11].

3.1 Huffmanovo kódování EEG dat

Huffmanovo kódování vytváří nejlepší prefixový kód [1], jehož kompresní poměr je velmi blízko dosažitelné hranici (2.2).

Klasické schéma Huffmanova kódování však nemůže být využito, aby byl získán kód s maximální pevnou délkou slova, protože maximální hloubku Huffmanova stromu nelze kontrolovat [2]. Proto byla v [2] dále navržena technika založená na CHT (collapsed Huffman tree), kde je každý list spojen s proměnnou délkou řetězce bitů kódujícího symbol na daném listu, pak existuje přesně jeden list spojený s množinou symbolů místo pouhého jediného symbolu. Myšlenkou je nechat listy stromu kolabovat tak, že dlouhé cesty od kořene jsou zkráceny, čímž se omezí maximální délka bitového řetězce kódového slova.

Principem CHT je rozdělení abecedy zdrojových symbolů $A=\{a_1,\ldots,a_M\}$, přičemž množina $I=\{1,\ldots,M\}$ je množina indexů, do dvou disjunktních množin A_1 a A_2 s indexy v I_1, I_2. Předpokladem dále sestupné uspořádání pravděpodobností symbolů, tzn. je-li p_i pravděpodobnost a_i, pak:

$$p_1 > p_2 > \cdots > p_M \qquad (3.1).$$

CHT kompresní algoritmus pak považuje A_2 jako samostatný symbol s k zakódování, který má sdruženou pravděpodobnost:

$$p_s = \sum_{i \in I_2} p_i \qquad (3.2).$$

Kompresní poměr získaný touto metodou je dán vztahem:

$$C_{lim} = 1 - \frac{H}{b} - p_s \qquad (3.3),$$

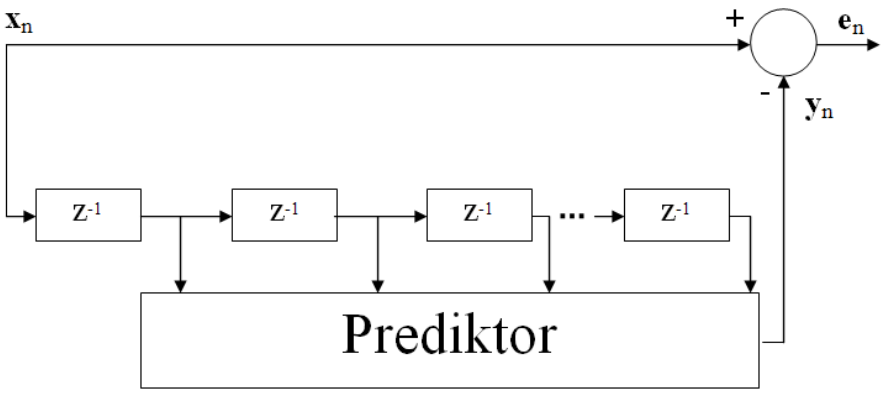
kde H je entropie symbolů $A_1 \cup \{s\} = \{a_{i_1}, \dots, a_{i_m}, s\}$.

3.2 Prediktivní kompresní techniky

Předpověď vzorku signálu založená na některých minulých hodnotách je vytvářena jednotkou zvanou prediktor (viz obrázek 4). Prediktivní schéma provádí vratnou transformaci rozšířeně užitou ke zmenšení entropie signálu: čím menší je entropie transformovaného signálu, tím větší je kompresní poměr. Při predikci dochází k chybě e_n vyjadřující rozdíl mezi skutečnou a predikovanou hodnotou. Tato chyba je následně kódována [2].

$$e_n = x_n - f(x_{n-1}, \dots, x_{n-N}) = x_n - \hat{x}_n \qquad (3.4),$$

kde x_n je *n*-tý vstupní vzorek (zkráceně za *x(nT)* se vzorkovací frekvencí *1/T*). Je-li pak predikce přesná po většinu času, chyby v predikci jsou blízké nule, a pro jejich zakódování mohou být použity krátké řetězce bitů, zatímco pro ostatní hodnoty e_n se užijí dlouhé bitové řetězce. V [2] a [3] bylo využito hned několik různých přístupů, jak vzorky signálu predikovat.



Obrázek 4: Transformace signálu založená na prediktivním schématu [2]

3.2.1 Markovský prediktor

Za předpokladu, že signál má Markovské vlastnosti a je vytvářen zdrojem modelovaným jako Markovský řetězec, může být implementován prediktor ze schématu z obrázku 4. Ve [2] byl k modelování signálu využit Markovský řetězec prvního řádu, který potřebuje určení všech podmíněných pravděpodobností typu $P[X_n = a_i \mid X_{n-1} = a_j]$, kde X_n je diskrétní náhodná proměnná nabývající hodnot z konečné abecedy $A=\{a_1,\ldots,a_M\}$. Při dostatečně velkém trénovacím souboru dat mohou být tyto pravděpodobnosti nahrazeny četnostmi. Četnostní matice, jejíž prvky F_{ij} počítají výskyt $X_n = a_i$ předcházený $X_{n-1} = a_j$, odpovídá rozdělení vzájemné pravděpodobnosti X_n a X_{n-1} a je použita k určení podmíněné pravděpodobnosti:

$$\prod_{ij} = P[X_n = a_i \mid X_{n-1} = a_j] \cong \frac{F_{ij}}{\sum_k F_{kj}} \qquad (3.5).$$

Hodnota X_n může být určena ze vztahu $E[X_n \mid X_{n-1}]$, odhad následné hodnoty symbolu a_j minimalizující střední kvadratickou chybu má tvar:

$$succ(a_j) = \sum_k k \prod_{kj} \quad \forall\, a_j \in A. \qquad (3.6).$$

3.2.2 Predikce digitálním filtrováním

Dalším možným řešením je digitální lineárně prediktivní (LP) filter popsaný diferenční rovnicí:

$$e_n = x_n - b_1 x_{n-1} - b_2 x_{n-2} - \cdots - b_N x_{n-N} \qquad (3.7).$$

Prediktor pak může být popsán rovnicí obsahující stejnou množinu koeficientů $\{b_1,\ldots,b_M\}$:

$$y_n = b_1 x_{n-1} + b_2 x_{n-2} + \cdots + b_N x_{n-N} \qquad (3.8).$$

Chybový signál e_n vyznačující se nízkou hodnotou entropie je možné obdržet minimalizací celkové kvadratické chyby:

$$E = \sum_n e_n^2 \qquad (3.9).$$

Při následující úmluvě v označení:

$$\bar{b^t} = [b_1 \dots b_N]$$
$$\bar{x}^t = [x_{n-1} \dots x_{n-N}] \qquad (3.10)$$

je lineární prediktor s minimální kvadratickou chybou tvaru $\hat{x} = \bar{b}^t \cdot \bar{x}^t$ dán řešením $\bar{b}^t$ rovnice:

$$\bar{b^t} \cdot \bar{R}_x = E[x_n \bar{x}^t] \qquad (3.11),$$

kde $\bar{R}_x$ je korelační matice procesu x_n. V praxi však skutečná autokorelační funkce není známa a na místo ní je použit odhad získaný z množiny vzorků. V [2] bylo zjištěno, že vysoké hodnoty *N* mají pouze nepatrný vliv na celkové zlepšení, a plně dostačujícím byl filtr druhého řádu.

LP se zakládá na předpokladu, že vstupní data jsou provedením časově diskrétního stacionárního náhodného procesu nabývajícího reálných hodnot, jehož matematický popis je znám. Předpoklad stacionarity však může být v rozumné míře použit na krátký úsek signálu. Často užitým přístupem je nalezení nejlepšího prediktoru na krátký úsek signálu užitím vztahů (3.7) - (3.11). Technika, která kóduje původní signál s koeficienty spočítanými na posloupnosti úseků, se nazývá LPC (Linear Predictive Coding). Přesné obnovení původního signálu je možné při zachování následujících údajů pro každý úsek: LP koeficienty $\{b_1,\dots,b_M\}$, prvních N vzorků a, pro každý následující bod, chyby predikce [2].

Jelikož koeficienty z (3.8) mohou být považovány za funkce závisle na čase, schopné se přizpůsobit chování signálu, lze je současnému stavu přiblížit buď LMS (Least Mean Square) adaptivním lineárním prediktorem popsaným rovnicí:

$$b_i(n) = b_i(n-1) + \beta x_{n-1} e_n \qquad (3.12),$$

kde β váhuje míru adaptace a e_n je chyba predikce v čase *n*, nebo SGN (sign) adaptivním prediktorem:

$$b_i(n) = b_i(n-1) + \Delta x_{n-1} sgn(e_n) \qquad (3.13),$$

kde Δ váhuje míru adaptace a $sgn(e_n)$ nábývá hodnot +1 nebo -1 podle znaménka chyby predikce. V [2] byly experimentálně zjištěny vhodné poslední dvě uvedené hodnoty.

3.2.3 Predikce umělou neuronovou sítí

Prediktorem může být i umělá neuronová síť (ANN – Artificial Neural Network), která využívá určitý počet posledních vzorků k předpovědi budo ucí události ve snaze sledovat signál. V [2] byla použita architektura vícevrstvého vnímání. ANN byla trénována s odolnou propagací s *N* vstupy, *H* skrytými uzly a jedním výstupem (*N – H – 1*). Symboly určené k zakódování jsou pak dány:

$$e_n = x_n - g(x_{n-1}, \dots, x_{n-N}) \qquad (3.14),$$

kde $g(x_{n-1}, \dots, x_{n-N})$ představuje výstup ANN. Další rozbor využití ANN v roli prediktoru byl proveden v [11], kde byl využit toolbox neuronových sítí softwaru Matlab a trénovací algoritmus Levenberg-Marquant, nejlepší výsledky dosahovala architektura SLFN (Single-layer Feedforward Network).

3.2.4 Autoregresivní model

Oblíbeným přístupem k modelování EEG dat je nakládat s nimi jako s časovými řadami popsanými autoregresivním (AR) procesem [3]:

$$x_n = \sum_{i=1}^{P} a_i x_{n-i} + e_n = \hat{x}_n + e_n \qquad (3.15),$$

kde *P* je řád AR procesu, jehož optimální hodnota byla v [3] určena na *P*=6..

Pro určení parametrů AR modelu byla v [3] využita Levinson-Durbinova metoda, chyby predikce jsou pak dány vztahem:

$$e_n^q = x_n - q\left(\sum_{i=1}^{P} a_i x_{n-i}\right) = x_n - \hat{x}_n^q \qquad (3.16),$$

přičemž *q(.)* je kvantizační funkce zaokrouhlující proměnnou na nejbližší celé číslo.

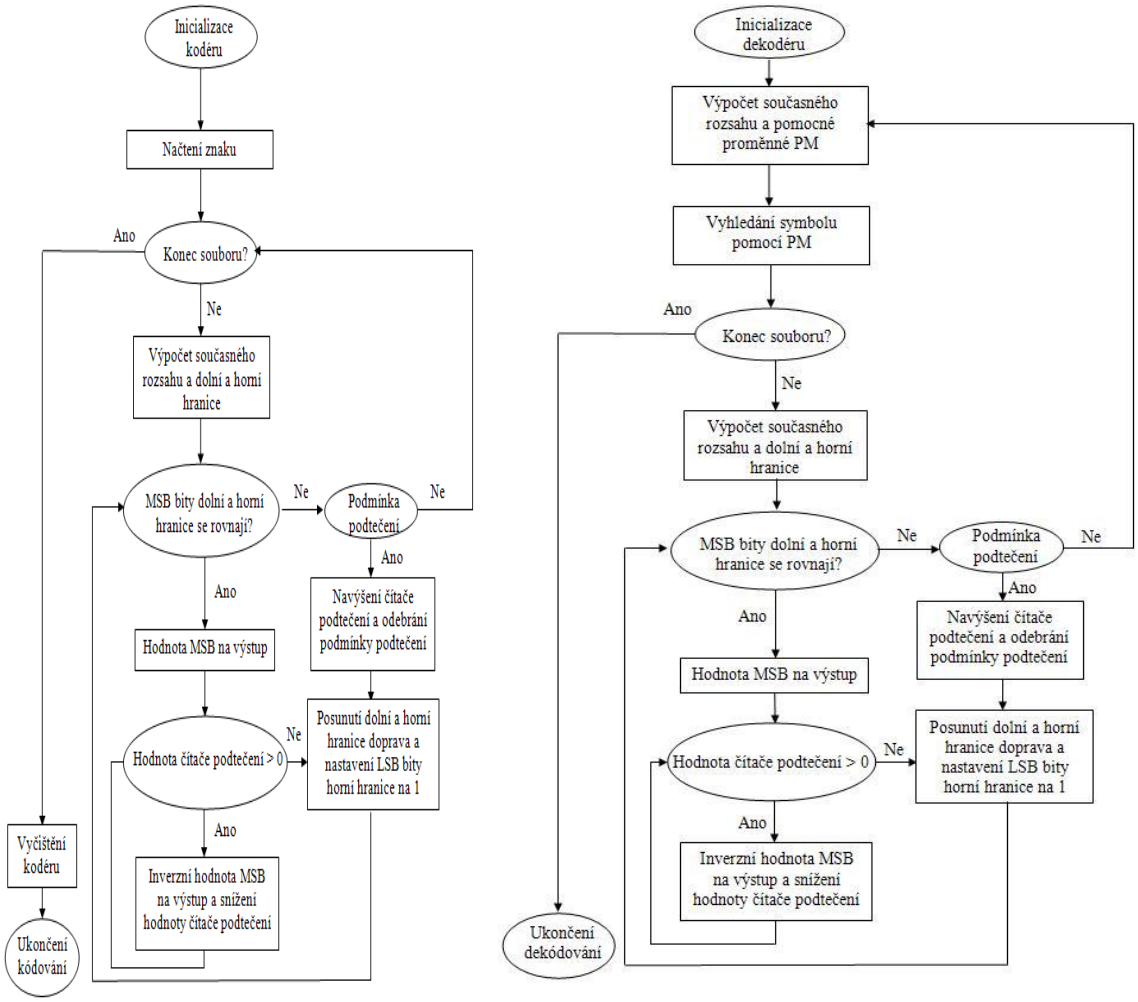
3.3 Transformační komprese

Posloupnost *N* vzorků signálu může být nahlížena jako bod *X* v *N*-rozměrném prostoru. Efektivnější reprezantace *X* může být dosaženo užitím ortogonální transformace *Y=TX*, kde *Y* představuje transformovaný vektor a *T* značí transformační matici [2]. Cílem je výběr podmnožiny z *Y* obsahující *M* položek, kde *M* je znatelně menší než *N* (zbývajících *N-M* položek je odstraněno), což vede ke kompresi. Huffmanovým kódováním rozdílu mezi původním signálem a signálem obnoveným z *M* komponent se tato technika stává bezeztrátovou, protože přesná rekonstrukce signálu může být získána z *M* položek a rozdílů. Nejlepší, avšak s nevýhodou v podobě značné

náročnosti na výpočetní čas, je Karhunen-Loeve transformace (KLT) [2]. Na druhém místě, i když se jedná o rychlejší algoritmus, je pak diskrétní kosinová transforamce (DCT). Oba algoritmy jsou podrobně popsány např. v [4].

3.4 Aritmetické kódování EEG

V [10] bylo využito aritmetické kódování [7] ke kompresi EEG dat. Kódovací i dekódovací algoritmus je na obrázku 5. Na vstupu kodéru byl použit diferenciátor a na výstupu dekodéru integrátor, což vedlo ke zvýšení komprese na celkových 49,60%.



Obrázek 5: Algoritmus aritmetického kodéru (vlevo)a dekodéru (vpravo)

3.5 Shrnutí speciálních metod komprese EEG dat

Dosažené výsledky v [2] jsou uvedeny v tabulce 1. Slovníkové metody jako LZ77 nejsou pro kompresi EEG dat vhodné z důvodu jejich stochastické povahy, neopakují se v nich tak frekventovaně přesně shodné vzory[3]. Velmi slibné výsledky dosáhly

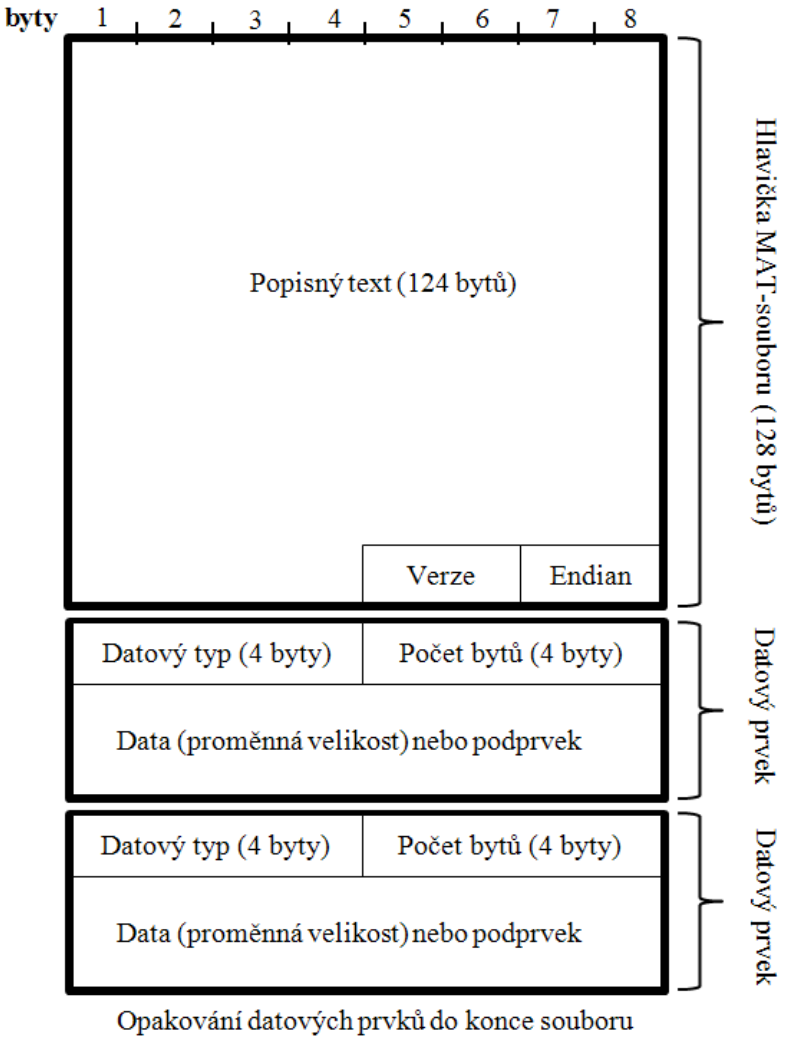
všechny prediktivní i transformační metody, ještě větších kompresních poměrů pak dosahují techniky téměř bezeztrátové komprese, viz např. [3] a [13].

Metoda komprese	Kompresní poměr v %	Vhodné pro EEG
GZIP	38,8	✗
Huf (sekce 3.1)	39,5	✓
Markov (sekce 3.2.1)	57,8	✓
Filter (sekce 3.2.2)	57,0	✓
ANN (sekce 3.2.3)	55,0	✓
LPC (sekce 3.2.2)	59,7	✓
Adapt-LMS (sekce 3.2.2)	58,1	✓
Adapt-SGN (sekce 3.2.2)	58,1	✓
DCT (sekce 3.3)	47,3	✓
VQ (sekce 3.3)	59,0	✓

Tabulka 1: Přehled kompresních poměrů [2]

4 Ukládání a načítání dat v Matlabu®

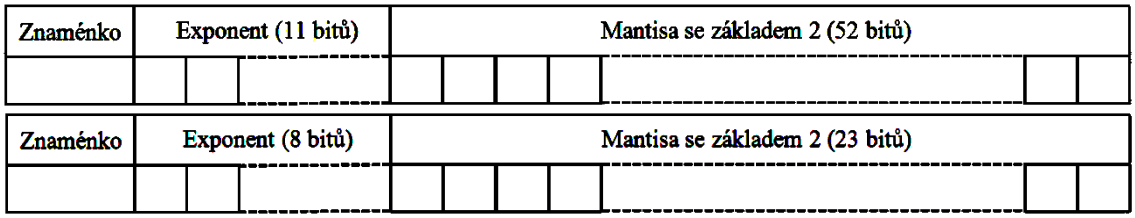
Výpočetní prostřední Matlab® využívá pro ukládání dat také vlastního formátu, který má příponu *mat*, tzv. MAT-soubory, v nichž jsou data uložena v binární podobě. Podrobný popis je znázorněn na obrázku 6. Po 128 bytové hlavičce následuje jeden či více datových prvků. Každý datový prvek má svůj 8 bytový štítek, tzv. *tag*, který popisuje druh dat a jaký počet bytů data zabírají. Samotná data pak mohou být celočíselná a vyjádřená v 8 až 64 bitech, s plovoucí řádovou čárkou anebo typu pole [14].



Obrázek 6: Formát MAT-souboru softwaru Matlab® verze 5 [14]

Matlab® reprezentuje čísla s plovoucí řádovou čárkou buď ve formátu *double-precision* nebo *single-precision* (viz obrázek 7). Proměnné, nestačí-li celočíselné vyjádření 8 až 64 bitovým integerem byť pouze u jediné hodnoty proměnné, jsou defaultně ukládány v *double-precision* formátu. Formát *double-precision* je v Matlabu konstruován dle IEEE® standardu 754, kdy jakákoliv hodnota uložená jako *double*

vyžaduje 64 bitů, pro druhý zmíněný formát je to 32 bitů. Znaménkový bit určuje, zda je číslo kladné či záporné, maximální hodnota exponentu je 2^{10-1}, tedy 1023, a celkově se pohybuje v rozmezí od -1023 do 1023. Číslice v mantise představují binární zlomek, kde každá následující číslice reprezentuje následující mocninu dvou ve jmenovateli zlomku (např. číslo $15{,}1875_{10}$ lze vyjádřit jako $1111{,}0011_2$, kdy 0,1875 jsou 3/16, nebo 1/8 + 1/16, což je 0*1/2 + 0*1/4 + 1*1/8 + 1*1/16).



Obrázek 7: Reprezentace čísla s plovoucí řádovou čárkou ve formátu *double-precision* (nahoře) a *single precision* (dole) [15]

5 Komprese EEG signálu

Testovací soubor dat, který je zde předmětem komprese, byl pořízen s vzorkovací frekvencí 256 Hz pro účely práce [16]. Zaznamenaná 45-kanálová EEG data se v prvním kroku zpracování v MEET převádí z d-souboru (nativní formát EEG přístroje), kde jsou hodnoty uloženy jako 16-bitový integer, do MAT-souboru, do nějž jsou hodnoty po provedení aritmetické operace dělení ve skriptu *dRead.m* ukládány skriptem *d2mat.m* ve formátu *doble-precision*. Pro účely zpracování signálu v MEET je však plně dostačující formát *single-precision* s poloviční bitovými nároky na vyjádření hodnot. Rozdíl mezi hodnotami určený dle vztahu (4.1), kde eeg_s je vektor hodnot ve formátu *single-precision* a eeg_d vektor hodnot ve formátu *double-precision*, byl, s přihlédnutím k drobnému problému s přesností uvedeném v předchozím odstavci, nulový.

$$\Delta = \frac{\sum(eeg_s - eeg_d)^2}{\sum(eeg_d)^2} \qquad (4.1)$$

Samotné ukládání do MAT-souboru lze provést pomocí příkazu *save*, který však ve volitelných argumentech nenabízí volbu formátu ukládaných proměnných, převod do formátu *single-precision* je tedy nutné provést ještě před užitím příkazu *save*. Další možností je pak příkaz *fwrite*, který volbu přesnosti podporuje, nutné je však použití příkazu *fopen* pro otevření souboru před samotným zápisem dat. K načtení dat slouží příkaz *load*, resp. *fread*.

5.1 Změna formátu EEG signálu

V tabulce 2 jsou srovnány velikosti dat, která byla filtrována Laplaceovským filtrem (LT_8NL), po uložení do MAT-souboru. Očekávání, že při *single-precision* formátování, které se v Matlabu provádí příkazem *single*, bude mít výsledný soubor poloviční velikost, se potvrdilo. Matlab samotný (použitá vezre R2010b), nezadá-li se příkazu *save* příslušný přepínač určující verzi MAT-souboru, pak defaultně data ukládá komprimovaná, kdy využívá kompresního algoritmu gzip. Z tabulky 2 je patrný nízký kompresní poměr této metody komprese, necelá 4%, jednoduchým přetypováním dat na *single-precision* již však lze značné úspory docílit.

V MEET byl doposud využíván příkaz *save* bez jakékoliv výše zmíněné úpravy. Pro prvotní úsporu dat byl ve skriptu *save_eeg*, který nahradí příkaz *save* všude tam, kde se pracuje s EEG signálem, použit příkaz *single* za účelem přetypování formátu, bylo tak dosaženo 50% úspory.

Typ formátování	Velikost souboru [kB]	Přepínač Matlabu	Úspora vůči Double-precision [%]
Double-precision	2 769 507	-v6	0
Komprese gzip a double-precision	2 663 849	-	3,82
Single-precision	1 384 884	-v6	49,99
Komprese gzip a Single-precision	1 276 418	-	53,91

Tabulka 2: Porovnání velikostí uloženého souboru při single precision a double-precision v Matlabu verze R2010b

5.2 Algoritmy komprese

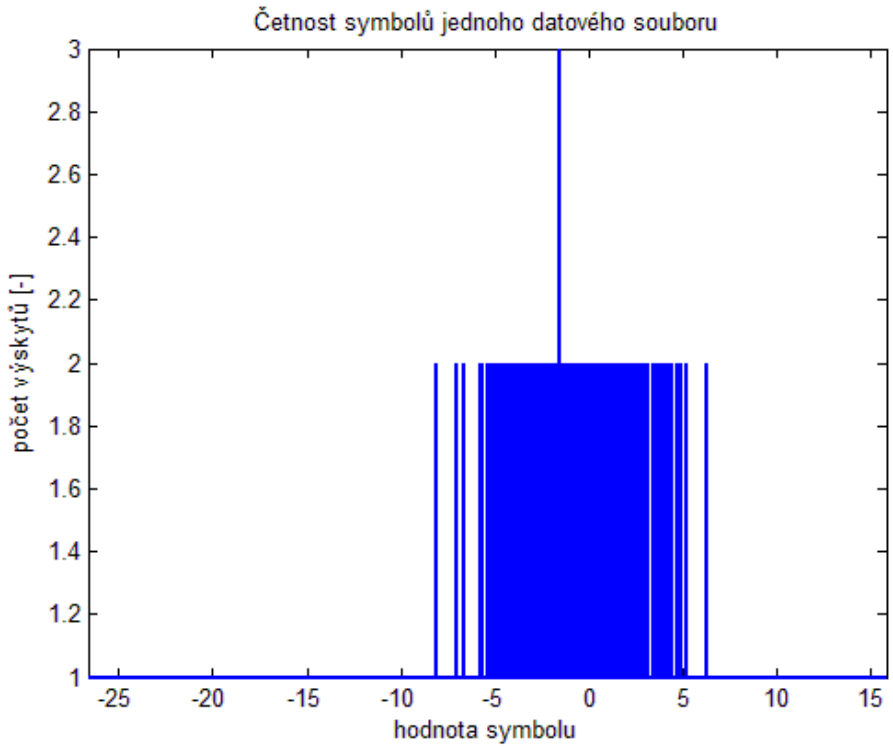
Pro samotnou kompresi byl zvolen Huffmanův algoritmus a aritmetické kódování. Tyto jsou přímo součástí výpočetního softwaru Matlab, skript *huffmanenco.m*, resp. *arithenco.m*.

Vstupními hodnotami, které se prvnímu uvedenému skriptu předávají, jsou slovník a samotná data určená ke kompresi. Slovník, jehož sestavení je popsáno v části 2.3 a který jednotlivým symbolům zdrojové abecedy přiřazuje jedinečný bitový řetězec, lze vytvořit pomocí zabudovaného skriptu *huffmandict.m*. Tento navíc spočítá i průměrnou délku kódového slova. Slovníkový skript vyžaduje jako vstupní parametry abecedu zdrojových symbolů a četnost jejich výskytu. Pro stanovení obou hodnot, vektorů, byla užita vestavěná funkce *tabulate*, která vrací četnostní tabulku o třech sloupcích, kde první sloupec obsahuje vzestupně seřazené unikátní hodnoty, druhý pak četnost jejich výskytu a třetí vyjadřuje procentuální zastoupení.

Skript aritmetického kódování vyžaduje zadání samotných dat a četnost výskytu symbolů, slovník není potřeba.

Použití skriptu *huffmanenco.m*, resp. *arithenco.m*, ke kompresi dat v podobě, v jaké jsou po Laplaceovské filtraci, není možné hlavně z důvodu výskytu mnoha jedinečných hodnot. Situace je názorně vidět na obrázku 8, kdy v souboru čítajícím celkem 251762 hodnot, jich je 251295 různých, jedinečných. Mohutnost zdrojové abecedy je tedy příliš veliká. Naskýtají se alespoň dvě možnosti, jak k problému přistoupit:

- kvantování hodnot,
- vyjádření čísla s plovoucí řádovou čárkou v binární podobě



Obrázek 8: Četnost symbolů v souboru čítajícím 251762 položek, z toho 251295 jich je různých

Při určování kompresního poměru byla jako původní brána data ve formátu *single-precision* bez komprese gzip (viz tabulka 2). Po kompresi byla data ulože příkazem *save* s přepínačem –v6, který vypne jinak defaultní kompresi gzip. Tato se totiž ve výsledné velikosti uložených dat nijak neprojeví, důvodem jsou neopakující se vzory v již komprimovaných datech.

5.2.1 Kvantování hodnot a Huffmanovo kódování

Nakvantováním čísel s plovoucí řádovou čárkou, která ve formátu *single-precision* zabírají každé 32 bítů, lze získat celočíselné kladné hodnoty. Tyto se následně převedou na již kratší binární řetězec, jež se rozloží na jednotlivé byty. Převodem bytů zpět do decimálního vyjádření se získá zdrojová abeceda v řádu stovek symbolů, což je oproti původním statisícům znatelné zlepšení, o tři řády. Pole obsahující binární posloupnost, která je výstupem Huffmanova algoritmu, se dá opět převést na celá čísla reprezentovaná každé 8 bity. Celý postup pak lze v Matlabu vyjádřit dle kódu uvedeném v příloze A (popis kódu uvozen znakem procent a vyveden v zelené barvě).

Tímto postupem bylo dosaženo dle vztahu (2.3) následujícího kompresního poměru: C_{k_H}=29,97%. Rozdíl mezi hodnotami původními a dekomprimovanými vyjádřený vztahem (4.1) je zanedbatelný (12.4725e-013).

5.2.2 Vyjádření čísla s plovoucí řádovou čárkou v binární podobě a Huffmanovo kódování

Vyjádřením čísla s plovoucí řádovou čárkou se zde rozumí jeho reprezentace v *single-precision* formátu dle standartu IEEE® 754, jehož podrobný popis byl uveden v kapitole 4 této práce. 32 bitový řetězec lze rozčlenit na 4 byty, které se následně mohou převést do decimální podoby. Získá se tím opět zdrojová abeceda přiměřené mohutnosti. Další kroky jsou pak shodné jako u kvantování. Samotný algoritmus je pak uveden v příloze B (popis kódu uvozen znakem procent a vyveden v zelené barvě).

Tímto postupem bylo dosaženo dle vztahu (2.3) následujícího kompresního poměru: C_{f2b_H}=4,28%. Rozdíl mezi hodnotami původními a dekomprimovanými vyjádřený vztahem (4.1) je nulový.

5.2.3 Kvantování hodnot a aritmetické kódování

Nakvantované hodnoty, tak jak byly spočteny v odstavci 5.2.1, byly dále komprimovány i aritmetickým kódováním. Celý kód (viz příloha C) je téměř totožný až na vytvoření slovníku, ke kterému zde nedochází, a volání příslušého skriptu, tj. *arithenco.m.*

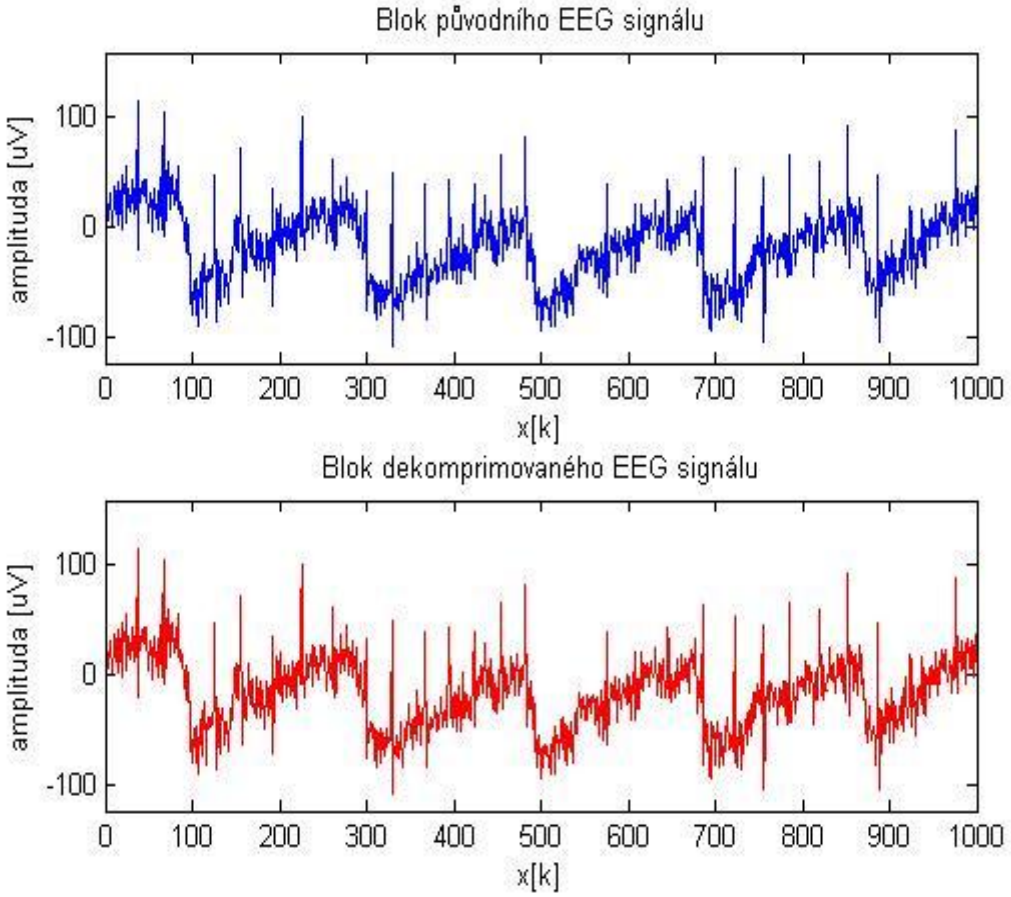
Tímto postupem bylo dosaženo dle vztahu (2.3) následujícího kompresního poměru: C_{k_A}=31,16%. Rozdíl mezi hodnotami původními a dekomprimovanými vyjádřený vztahem (4.1) je zanedbatelný (1.2222e-012).

5.3 Algoritmy dekomprese

Účelem dekomprese je získání původních, nekomprimovaných dat, tato by měla být v případě bezeztrátové komprese naprosto nezměněná oproti datům originálním. Porovnáním původních a dekomprimovaných dat lze tedy celkově ověřit správnou funkčnost komprese. Numericky byl rozdíl mezi původními a dekomprimovanými daty uveden u jednotlivých kompresních postupů, vizuálně lze část průběhu obou vektorů dat zhodnotit na obrázku č. 9.

Jelikož je každý z uvedených kompresních postupů složen z několika kroků, bylo při vytváření dekompresních algoritmů postupováno po jednotlivých krocích poskládaných v opačném pořadí. Po každém kroku dekomprese byl porovnán vektor dat s jeho příslušným protějškem vzniklým při kompresi. Data byla porovnávána pomocí vestavěné funkce *isequal,* která vrátí hodnotu logická 1 v případě, že se všechny položky vektoru shodují, a hodnotu logická 0 v případě opačném. Tímto způsobem byly

odhaleny a opraveny nedostatky v kompresním kódu. Jednotlivé dekompresní algoritmy jsou uvedeny v příloze D, E a F.



Obrázek 9: Část průběhu původního a dekomprimovaného signálu

5.4 Porovnání kompresních a dekompresních algoritmů

Pro porovnání algorimů byly důležité především dva parametry: kompresní poměr a doba běhu. V tabulce č. 3 jsou oba parametry shrnuty, testovací soubor pro určení doby běhu měl velikost 1.007kB, kompresní poměr byl vztažen k datům ve formátu *single-precision* bez komprese gzip (viz tabulka 2).

Pro měření doby běhu byl použit nástroj *profiler*, který je v Matlabu vestavěn. *Profiler* umožňuje sledovat dobu běhu každé funkce, včetně těch vnořených, je dokonce možné zjistit, která operace ve funkci zabere nejvíce času. Optimalizována tak byla např. často používaná funkce *bin2dec*, která převádí binární řetězec na decimální hodnotu, doba běhu po optimalizaci se snížila přibližně šestkrát.

Z dosažených hodnot lze poměrně jednoznačně určit vhodnou kompresní techniku, jedná se o aritmetické kódování, kterým se komprimují kvantované hodnoty. Ač je

kompresní poměr této techniky téměř shodný s Huffmanovým kódováním, přičemž doba běhu komprese je delší, čas potřebný pro dekompresi je přibližně 4,5 kratší. Tento rozdíl tkví v nutnosti vyhledání a porovnávní bitových řetězců s řetězci slovníku.

Úspory dat bylo dosaženo především díky kvantizaci hodnot, mohutnost zdrojové abecedy byla totiž 256, což je pro kompresní algoritmus poměrně neefektivní. Počet kvantizačních bitů zvolený s ohledem na 16 bitová celočíselná vstupní data, která se následně při Laplaceovské filtraci průměrují, byl nastaven na 22.

Nutné je však podotknout, že doba běhu komprese, obzvláště pak dekomprese, je značnou slabinou, v MEET jsou totiž zpracovávána data o objemu řádově stovek gigabytů.

Postup uvedený v sekci 5.2.2 se pak jeví jako naprosto neefektivní jak po stránce úspory místa tak i neúnosně dlouhé doby běhu.

	Algoritmus	Kompresní poměr [%]	Doba běhu ○[s]
Komprese	Huff._kvant. (sekce 5.2.1)	29,97	113
	Huff_float (sekce 5.2.2)	4,28	224
	Aritm_kvant (sekce 5.2.3)	31,16	154
Dekomprese	Huff._kvant. (sekce 5.3)	-	2090
	Huff_float (sekce 5.3)	-	2145
	Aritm_kvant (sekce 5.3)	-	453

Tabulka 3: Přehled kompresních poměrů a doby běhu

6 Začlenění algoritmů do Toolboxu MEET

Jak již bylo předesláno v částti 5.1 této práce, budou do Toolboxu MEET zabudovány dvě nové funkce. První z nich, funkce *save_eeg*, nahradí funkci *save*. Tato funkce obsahuje kompresní algoritmus zvolený v části 5.4, tzn. aritmetické kódování aplikované na kvantované hodnoty. Stejně tak bude začleněna funkce *load_eeg* nahrazující funkci *load.*

Obě nové funkce pracují s MAT-soubory, samy totiž používají standardní příkaz *save* a *load*. Příkaz *save* je pak použit s přepínačem –v6, který zamezuje v Matlabu defaultní kompresi gzip.

V tuto chvíli může být za největší přínos považováno jednoduché přetypování EEG dat do formátu *single-precision*, které oproti původnímu formátu *double-precision* šetří 50% úložného prostoru, a není časově náročné (viz kapitola 5).

Zvolený kompresní algoritmus sice přináší dalších 30% úspory, dosažení tohoto výsledku však značně časově náročné je.

7 Závěr

Tato práce se zaměřila na možnost komprese EEG dat, která jsou zpracovávána Toolboxem MEET. Teoretický rozbor dostupných kompresních technik naznačil, který kompresní algoritmus by mohl být vhodný. Technik určených přímo pro EEG data je známo větší množství (viz [2], [8] a [11]), ne každá se však pro účely této práce hodila (např. DCT uvedená v kapitole 3.3).

V potaz byly vzaty i dostupné možnosti samotného výpočetního softwaru Matlab, jehož zabudované skripty pro Huffmanovo a aritmetické kódování byly nakonec použity. Díky jednoduchému přetypování číselných hodnot na *single-precison* (32 bitů na jednu číselnou hodnotu). byla dosažena 50 % úspora úložného prostoru, původně byla totiž data ukládána v *double-precision* formátu (64 bitů na jednu číselnou hodnotu). Tento postup byl popsán v kapitole 5.

Před aplikací kompresních algoritmů musela však být data upravena z důvodu velké hodnoty mohutnosti zdrojové abecedy (řádově 10^5), vyzkoušeny byly dvě možnosti: vyjádření čísla s plovoucí řádovou čárkou v 32 bitovém řetězci, který se následně rozčlenil na 4 byty, a kvantování. Kvantované hodnoty se pak daly také vyjádřit bitovým řetězcem, avšak kratším než 32 bitovým, což mělo za důsledek větší kompresní poměr. Rozložením dlouhého bitového řetězce na jednotlivé byty se tedy dala získat zdrojová abeceda o kardinalitě 256, tato hodnota však není ještě zdaleka optimální.

Pří výběru vhodného kompresního, resp. dekompresního algoritmu bylo přihlédnuto ke dvěma ukazatelům, jednak ke kompresnímu poměru, druhým rozhodujícím faktorem pak byla doba běhu algoritmu. Dosažené výsledky jsou zaneseny v tabulce 3 (část 5.4). Na jejich základě byl zvolen výsledný algoritmus, jmenovitě se jedná o aritmetické kódování.

Výsledný kompresní poměr je přibližně 30 %, stinnou stránkou je však doba běhu, kdy na soubor o velikosti 1MB bylo potřeba 154 sekund pro jeho kompresi, a dalších 453 vteřin pro rekonstrukci původního signálu. Pomocí *profileru*, nástroje pro měření doby běhu funkcí, bylo zjištěno, které výrazy jsou za časovou náročnost zodpovědné. Nebylo však možné tyto operace deaktivovat obdobně jak to bylo provedeno u funkce *bin2dec*, jedná se totiž o klíčové výpočty (např. počítání dolní a horní hranice intervalu, viz 2.4).

Zadání diplomové práce bylo sice splněno, použití kompresního postupu v současném uspořádání však nelze doporučit z důvodu časové náročnosti komprese,

především pak dekomprese. Řešením této situace by mohlo být začlenění vhodného prediktoru (viz 3.2) před samotný kompresní algoritmus.

8 Literatura

[1] HUFFMAN, David A. A Method for the Construction of Minumum Redundacny Codes, Proceedings of the IRE, Sept. 1952, Volume: 40, Issue: 9

[2] ANTONIOL, Giuliano; TONELLA, Paolo. EEG Data Compression Techniques - IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, VOL. 44, NO. 2, FEBRUARY 1997

[3] MEMON, Nasir; KONG, Xuan; CINKLER, Judit. Context-based lossless and near-lossless compression of EEG signals. IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 3, NO. 3, SEPTEMBER 1999

[4] SALOMON, David. Data Compression. Springer-Verlag London Limited 2007. ISBN-10 1-84628-602-6

[5] ADÁMEK, Jiří. Kódovani. SNTL Praha 1989

[6] MATOUŠEK, Radomil. Metody kódování. VUT v Brně 2006

[7] HOWARD, Paul G.; VITTER, Jeffrey Scott. Practical Implementations of Arithmetic Coding. Technical Report No. 92-18, April 1992

[8] NAVE, Gil; COHEN, Amon. ECG Compression Using Long-Term Prediction. IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING. VOL. 40, NO.9. SEPTEMBER 1993

[9] MAGOTRA, Neeraj; MANDAYAM, Giridhar; SUN, Mingui; McCOY, Wes. Lossless compression of electroencephalographic (EEG) data. Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on 12-15 May 1996

[10] O´SHEA, D.; McSWEENEY, R.; SPAGNOL, C.; POPOVICI, E. Efficient Implementation of Arithmetic Compression for EEG. Dostupný z WWW: <usb.issc.ie/download/file/288.pdf>

[11] SRIRAAM, N.; ESWARAN, C. Performance Evaluation of Lossless Two-stage Compression Schemes for EEG Signal. International Journal of Information and Communication Engineering 12 2005. Dostupný z WWW: <citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.4983&rep=rep1&type=pdf>

[12] ŠŤASTNÝ, J. MEET: MODULAR EEG PROCESSING TOOLBOX. Dostupný z WWW: <http://amber.feld.cvut.cz/fpga//publications/meet_abstract.pdf>

[13] SRIRAAM, N.; ESWARAN, C. Performance Evaluation of Neural Network and Linear Predictors for Near-Lossless Compression of EEG Signals. IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 12, NO. 1, JANUARY 2008. Dostupný z WWW: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4358888>

[14] The MATH WORKS Inc. MATLAB - The Language of Technical Computing - MAT-File Format Version 5. Dostupné na WWW:< https://maxwell.ict.griffith.edu.au/spl/matlab-page/matfile_format.pdf>

[15] WALILISCH, Pascal; LUSIGNAN, Michael; BENAYOUN, Marc; BAKER, Tanya I.; DICKEY, Adam S.; HATSOPOULOS, Nicholas G. Matlab® - Matlab® for Neuroscientists, An Introduction to Scientific Computing in Matlab. Elsevier Inc. 2009. ISBN: 978-0-12-374551-4. Dostpné z www: < http://www.matematica.ciens.ucv.ve/files/Manuales/Manuales/Matlab%20-%20Matlab%20for%20Neuroscientists.pdf>

[16] ŠŤASTNÝ, Jakub; SOVKA, Pavel. High-Resolution Movement EEG Classification - Computational Intelligence and Neuroscience, Volume 2007, Article ID 54925. Dostpné z www: < http://amber.feld.cvut.cz/fpga/publications/S1687526507549254.pdf>

Příloha A – Huffmanovo kódování a kvantování hodnot - komprese

```
% určení délky vstupního vektoru s EEG daty
lg=length(eeg_block);
% určení minimální a maximální hodnoty
mn=min(eeg_block);
mx=max(eeg_block);
% počet bitů
N=22;
% kvantizační krok
delta=(mx-mn)/2^N;
% naplnění výsledného pole nulami z důvodu přidělení paměti
q_k=zeros(1,lg);
% cyklus pro výpočet jednotlivých kvantovaných hodnot
for i=1:lg
q_k(i)=fix((eeg_block(i)-mn)/delta);
end;
% určení délky největší kvantované hodnoty převedené na binírní řetězec
q_k_mx_l=length(dec2bin(max(q_k)));
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_n=uint8(zeros(1,3*lg));
ni=1;
% cylus pro převod nakvantované hodnoty na binární řetězec, rozdělení na jednotlivé
byty a jejich          % následné převedení do decimální podoby
for i=1:length(q_k)
   nr=dec2bin(q_k(i),q_k_mx_l);
   eeg_n(ni)=bin2dec(nr(1:8));
   ni=ni+1;
   eeg_n(ni)=bin2dec(nr(9:16));
   ni=ni+1;
   eeg_n(ni)=bin2dec(nr(17:end));
   ni=ni+1;
end
% počítání četnosti výskytu symbolů
```

```
st=tabulate(eeg_n);
sf(1,:)=st(:,1)
sf(2,:)=st(:,2)
% vytvoření slovníku
[dict,avglen] = huffmandict(sf(1,:),sf(2,:)/sum(sf(2,:)));
% volání skriptu Huffmanova kódování
comp = huffmanenco(eeg_n,dict);
% určení počtu běhů cyklu
runs=ceil(length(comp)/8);
% naplnění výsledného pole nulami z důvodu přidělení paměti
encoded_signal_block=zeros(1,runs);
fb=1;
lb=fb+7;
% určení počtu posledních bitů s využitím modulo aritmetiky
if mod(length(comp),8)~=0
   rest=mod(length(comp),8);
else
   rest=8;
end
% převod 0 a 1 z pole comp na decimální číslo
for i=1:runs
   if i~=runs
   t=comp(fb:lb);
   encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   else
      lb=fb+rest-1;
      t=comp(fb:lb);
      encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   end
   fb=lb+1;
   lb=fb+7;
end
% přetypování na 8 bitové celé číslo
eeg=uint8(encoded_signal_block);
```

```
%  uložení dat a pomocných proměnných
save(path_to_save,'eeg','dict','-v6');
```

Příloha B – Huffmanovo kódování a 32 bitové vyjádření čísla s plovoucí řádovou čárkou - komprese

```
%  zjištění velikosti obou dimenzí vstupního vektoru dat
[ro,co]=size(eeg);
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_new=zeros(1,4*co);
% pomocná proměnná
bitstr='';
input_eeg=eeg;
k=1;
% cyklus pro získání 32 bitů představujících single-precision formát
for i=1:length(input_eeg)
   cnt=0;
% určení znaménkového bitu, 1 pro záporné hodnoty, nula pro kladné
   if input_eeg(i)<0
      sgn='1';
   else
      sgn='0';
   end
% nyní již je možné pracovat se všemi vstupními hodnotami jako s kladnými
% čísly
   input_eeg(i)=abs(input_eeg(i));
   cn=input_eeg(i);
% určení exponentu
   if input_eeg(i)>=2
      xpnt=floor(log2(input_eeg(i)));
      exponent=dec2bin(127+xpnt);
      tmp_mnts=input_eeg(i)/2^xpnt-1;
   elseif input_eeg(i)>=1 && input_eeg(i)<2
      exponent='01111111';
      tmp_mnts=input_eeg(i)-1;
   else
      while cn < 1
```

```
        cn=2*cn;
        cnt=cnt+1;
        end
        exponent=dec2bin(127-cnt,8);
        tmp_mnts=cn-1;
    end
    mnts='';
% naplnění mantisy
    for m=1:23
        tmp_mnts=2*tmp_mnts;
        if tmp_mnts>=1
            mnts(m)='1';
            tmp_mnts=tmp_mnts-1;
        else
            mnts(m)='0';
            tmp_mnts=tmp_mnts;
        end
    end
% sestavení výsledného binárního řetězce
    flt2bin=strcat(sgn,exponent,mnts);
% převod jednotlivých bytů do decimální podoby
    eeg_new(k)=bin2dec(flt2bin(1:8));
    k=k+1;
    eeg_new(k)=bin2dec(flt2bin(9:16));
    k=k+1;
    eeg_new(k)=bin2dec(flt2bin(17:24));
    k=k+1;
    eeg_new(k)=bin2dec(flt2bin(25:32));
    k=k+1;
end
% počítání četnosti výskytu symbolů
st=tabulate(eeg_new);
sf(1,:)=st(:,1);
sf(2,:)=st(:,2);
```

```
% přetypování na typ double
sf=double(sf);
% vytvoření
[dict,avglen] = huffmandict(sf(1,:),sf(2,:)/sum(sf(2,:)));
% volání skriptu Huffmanova kódování
comp = huffmanenco(eeg_new,dict);
% určení počtu běhů cyklu
runs=ceil(length(comp)/8);
% naplnění výsledného pole nulami z důvodu přidělení paměti
encoded_signal_block=zeros(1,runs);
fb=1;
lb=fb+7;
% určení počtu posledních bitů s využitím modulo aritmetiky
if mod(length(comp),8)~=0
   rest=mod(length(comp),8);
else
   rest=8;
end
% převod 0 a 1 z pole comp na decimální číslo
for i=1:runs
   if i~=runs
   t=comp(fb:lb);
   encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   else
      lb=fb+rest-1;
      t=comp(fb:lb);
      encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   end
   fb=lb+1;
   lb=fb+7;
end
% přetypování na 8 bitové celé číslo
eeg=uint8(encoded_signal_block);
%  uložení dat a pomocných proměnných
```

```
save(path_to_save,'eeg','dict','-v6');
```

Příloha C – Aritmetické kódování a kvantování hodnot - komprese

```
% určení délky vstupního vektoru s EEG daty
lg=length(eeg_block);
% určení minimální a maximální hodnoty
mn=min(eeg_block);
mx=max(eeg_block);
% počet bitů
N=22;
% kvantizační krok
delta=(mx-mn)/2^N;
% naplnění výsledného pole nulami z důvodu přidělení paměti
q_k=zeros(1,lg);
% cyklus pro výpočet jednotlivých kvantovaných hodnot
for i=1:lg
q_k(i)=fix((eeg_block(i)-mn)/delta);
end;
% určení délky největší kvantované hodnoty převedené na binírní řetězec
q_k_mx_l=length(dec2bin(max(q_k)));
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_n=uint8(zeros(1,3*lg));
ni=1;
% cylus pro převod nakvantované hodnoty na binární řetězec, rozdělení na jednotlivé
byty a jejich          % následné převedení do decimální podoby
for i=1:length(q_k)
   nr=dec2bin(q_k(i),q_k_mx_l);
   eeg_n(ni)=bin2dec(nr(1:8));
   ni=ni+1;
   eeg_n(ni)=bin2dec(nr(9:16));
   ni=ni+1;
   eeg_n(ni)=bin2dec(nr(17:end));
   ni=ni+1;
end
% počítání četnosti výskytu symbolů
```

```
st=tabulate(eeg_n);
sf(1,:)=st(:,1)
sf(2,:)=st(:,2)
seq = eeg_n+1;
% volání skriptu aritmetického kódování
counts =(sf(2,:));
comp = arithenco(seq,counts);
% určení počtu běhů cyklu
runs=ceil(length(comp)/8);
% naplnění výsledného pole nulami z důvodu přidělení paměti
encoded_signal_block=zeros(1,runs);
fb=1;
lb=fb+7;
% určení počtu posledních bitů s využitím modulo aritmetiky
if mod(length(comp),8)~=0
   rest=mod(length(comp),8);
else
   rest=8;
end
% převod 0 a 1 z pole comp na decimální číslo
for i=1:runs
   if i~=runs
   t=comp(fb:lb);
   encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   else
      lb=fb+rest-1;
      t=comp(fb:lb);
      encoded_signal_block(i)= sum(t.*2.^(numel(t)-1:-1:0));
   end
   fb=lb+1;
   lb=fb+7;
end
% přetypování na 8 bitové celé číslo
eeg=uint8(encoded_signal_block);
```

```
%  uložení dat a pomocných proměnných
save(path_to_save,'eeg','counts','-v6');
```
Příloha D

```
[r c]=size(dict)
dict_new=zeros(r,c+1);
for i=1:r
    dict_new(i,1)=dict{i,1};
    dict_new(i,2)=length(dict{i,2});
    t=dict{i,2};
    dict_new(i,3)=sum(t.*2.^(numel(t)-1:-1:0));
end
```

Příloha D – Huffmanovo kódování a kvantování hodnot – dekomprese

```
% určení délky posledního bitového řetězce
len_last=length(zeros(1,rest));
% určení délky EEG dat
len_eeg=length(eeg);
% naplnění výsledného pole nulami z důvodu přidělení paměti
dec_eeg=zeros(1,(8*length(eeg(1:end-1)))+len_last);
fb=1;
lb=fb+7;
% cyklus pro převod decimálních hodnot do pole 0 a 1
for i=1:len_eeg
   if i~=len_eeg
      tmp=dec2bin(eeg(i),8);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   else
      lb=fb+len_last-1;
      tmp=dec2bin(eeg(i),len_last);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   end
   fb=lb+1;
   lb=fb+7;
end
% volání dekompresního skriptu
eeg_block=huffmandeco(dec_eeg,dict);
% určení délky binárního řetězce posledního bytu
if mod(q_k_mx_l,8)~=0
   rest=mod(q_k_mx_l,8);
else
   rest=8;
end
lgt=length(eeg_block)/3;
```

```
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_dec_new=zeros(1,lgt);
ni=1;
% cyklus pro sestavení binárního řetězce ze 3 decimálních hodnot
for i=1:length(eeg_dec_new)
   tmp1=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp2=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp3=dec2bin(eeg_block(ni),rest);
   ni=ni+1;
   eeg_dec_new(i)=bnr2dcml(strcat(tmp1,tmp2,tmp3));
end
% naplnění výsledného pole nulami z důvodu přidělení paměti
r_k=zeros(1,length(eeg_dec_new));
% rekvantizace
for i=1:length(eeg_dec_new)
r_k(i)=(eeg_dec_new(i)*delta)+mn;
end;
eeg=r_k;
```

Příloha E – Huffmanovo kódování a 32 bitové vyjádření čísla s plovoucí řádovou čárkou - dekomprese

```
% určení délky posledního bitového řetězce
len_last=length(zeros(1,rest));
% určení délky EEG dat
len_eeg=length(eeg);
% naplnění výsledného pole nulami z důvodu přidělení paměti
dec_eeg=zeros(1,(8*length(eeg(1:end-1)))+len_last);
fb=1;
lb=fb+7;
% cyklus pro převod decimálních hodnot do pole 0 a 1
for i=1:len_eeg
   if i~=len_eeg
      tmp=dec2bin(eeg(i),8);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   else
      lb=fb+len_last-1;
      tmp=dec2bin(eeg(i),len_last);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   end
   fb=lb+1;
   lb=fb+7;
end
% volání dekompresního skriptu
eeg_block=huffmandeco(dec_eeg,dict);
lgt=length(eeg_block)/4;
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_dec_new=zeros(1,lgt);
ni=1;
% cyklus pro sestavení 32 bitového řetězce a jeho převedení na číslo s
% plovoucí řádovou čárkou
```

```
for i=1:length(eeg_dec_new)
   tmp1=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp2=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp3=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp4=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   bin2float=strcat(tmp1,tmp2,tmp3,tmp4);
   expon=bnr2dcml(bin2float(2:9))-127;
   mnts=bin2float(10:32);
   v=mnts-'0';
   twos = pow2(-1:-1:-numel(v));
   mnts = sum(v .* twos(ones(1,1),:));
   sgn=bin2float(1)-'0';
   if sgn == 1
      eeg_dec_new(i)=-1*((1+mnts)*2^expon);
   else
      eeg_dec_new(i)=(1+mnts)*2^expon;
   end

end
eeg=eeg_dec_new;
```

Příloha F – Aritmetické kódování a kvantování hodnot – dekomprese

```
% určení délky posledního bitového řetězce
len_last=length(zeros(1,rest));
% určení délky EEG dat
len_eeg=length(eeg);
% naplnění výsledného pole nulami z důvodu přidělení paměti
dec_eeg=zeros(1,(8*length(eeg(1:end-1)))+len_last);
fb=1;
lb=fb+7;
% cyklus pro převod decimálních hodnot do pole 0 a 1
for i=1:len_eeg
   if i~=len_eeg
      tmp=dec2bin(eeg(i),8);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   else
      lb=fb+len_last-1;
      tmp=dec2bin(eeg(i),len_last);
      tmp=tmp-'0';
      dec_eeg(fb:lb)=tmp;
   end
   fb=lb+1;
   lb=fb+7;
end
% volání dekompresního skriptu
eeg_block=arithdeco(dec_eeg,counts,len);
eeg_block=eeg_block-1;
% určení délky binárního řetězce posledního bytu
if mod(q_k_mx_l,8)~=0
   rest=mod(q_k_mx_l,8);
else
   rest=8;
end
```

```
lgt=length(eeg_block)/3;
% naplnění výsledného pole nulami z důvodu přidělení paměti
eeg_dec_new=zeros(1,lgt);
ni=1;
% cyklus pro sestavení binárního řetězce ze 3 decimálních hodnot
for i=1:length(eeg_dec_new)
   tmp1=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp2=dec2bin(eeg_block(ni),8);
   ni=ni+1;
   tmp3=dec2bin(eeg_block(ni),rest);
   ni=ni+1;
   eeg_dec_new(i)=bnr2dcml(strcat(tmp1,tmp2,tmp3));
end
% naplnění výsledného pole nulami z důvodu přidělení paměti
r_k=zeros(1,length(eeg_dec_new));
% rekvantizace
for i=1:length(eeg_dec_new)
r_k(i)=(eeg_dec_new(i)*delta)+mn;
end;
eeg=r_k;
```