

**MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY**

Bakalářská práce

BRNO 2017

LENKA HELDOVÁ



**MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY**



Analýza a návrh algoritmů pro optimalizaci vizualizace dat

Bakalářská práce

Lenka Heldová

Vedoucí práce: Mgr. Jiří Zelinka, Dr.

Brno 2017

Bibliografický záznam

Autor:	Lenka Heldová Přírodovědecká fakulta, Masarykova univerzita Ústav matematiky a statistiky
Název práce:	Analýza a návrh algoritmů pro optimalizaci vizualizace dat
Studijní program:	Matematika
Studijní obor:	Modelování a výpočty
Vedoucí práce:	Mgr. Jiří Zelinka, Dr.
Akademický rok:	2016/2017
Počet stran:	ix + 45
Klíčová slova:	PerfCake; testování softvéru; optimalizace; grafy; jadrové odhady; směrodatná odchylka; Nadaraya-Watsonove odhady; redukční algoritmy

Bibliografický záznam

Autor:	Lenka Heldová Prírodovedecká fakulta, Masarykova univerzita Ústav matematiky a štatistiky
Názov práce:	Analýza a návrh algoritmov pre optimalizáciu vizualizácie dát
Študijný program:	Matematika
Študijný odbor:	Modelovanie a výpočty
Vedúci práce:	Mgr. Jiří Zelinka, Dr.
Akademický rok:	2016/2017
Počet strán:	ix + 45
Kľúčové slová:	PerfCake; testovanie softvéru; optimalizácia; grafy; jadrové odhady; smerodajná odchýlka; Nadaraya-Watsonove odhady; redukčné algoritmy

Bibliographic Entry

Author:	Lenka Heldová Faculty of Science, Masaryk University Department of Mathematics and Statistics
Title of Thesis:	Analysis and design of algorithms to optimize data visualization
Degree Programme:	Mathematics
Field of Study:	Modelling and Calculations
Supervisor:	Mgr. Jiří Zelinka, Dr.
Academic Year:	2016/2017
Number of Pages:	ix + 45
Keywords:	Perfcake; software testing; optimization; graphs; kernel estimations; standard deviation; Nadaraya-Watson estimations; reduction algorithms

Abstrakt

V této bakalářské práci se věnujeme návrhu algoritmů pro optimalizaci vizualizace dat, které jsou výstupem nástroje pro testování výkonu PerfCake. V práci jsou popsány dvě části algoritmů, vyhlazovací a redukční. Při vyhlazování jsou používány jádrové odhady, pro redukční část jsou představeny tři různé algoritmy, které jsou na konci práce porovnány. Podle několika aspektů je zvolen nejlepší, reálně použitelný algoritmus na optimalizaci dat, pro vykreslení do grafů.

Abstrakt

V tejto bakalárskej práci sa venujeme návrhu algoritmov pre optimalizáciu vizualizácie dát, ktoré sú výstupom nástroja na testovanie výkonu PerfCake. V práci sú opísané dve časti algoritmov, vyhľadzovacia a redukčná. Pri vyhľadzovaní sú používané jadrové odhady, pre redukčnú časť sú predstavené tri rôzne algoritmy, ktoré sú na konci práce porovnané. Podľa niekoľkých aspektov je zvolený najlepší, reálne použiteľný algoritmus na optimalizáciu dát, pre vykreslenie do grafov.

Abstract

In this bachelor thesis we design algorithms to optimize visualization of data, which are output of the testing tool - PerfCake. These algorithms are composed of two parts, smoothing and reduction. The kernel estimation, that is used for smoothing part, and three different reduction algorithms are introduced in fifth chapter. All three reduction algorithms are afterwards compared in the last, sixth chapter. They are compared in few different aspects, the best algorithm is suitable for real use to optimize data visualization in the PerfCake tool.



MASARYKOVA UNIVERZITA

Přírodovědecká fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Akademický rok: 2015/2016

Ústav: Ústav matematiky a statistiky

Studentka: Lenka Heldová

Program: Matematika

Obor: Modelování a výpočty

Směr: Analýza signálů a dat

Ředitel Ústavu matematiky a statistiky PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje bakalářskou práci s tématem:

Téma práce: Analýza a návrh algoritmů pro optimalizaci vizualizace dat

Téma práce anglicky: Analysis and design of algorithms to optimize data visualization

Oficiální zadání:

Student se seznámí s nástrojem pro testování výkonu software PerfCake a jeho možnostmi záznamu naměřených hodnot. Úkolem studenta bude provést analýzu těchto dat, rozpoznání jejich užitečných a neužitečných částí a navrhnutí několika algoritmů pro zmenšení souboru dat tak, aby bylo možné data v reálném čase vizualizovat. Důraz musí být kladen na zachování celé užitečné informace. Student pak navržené algoritmy implementuje a použije na získaná data. V závěru práce pak zhodnotí jejich vlastnosti a možnosti reálného použití.

Literatura: PerfCake 5.x User Guide, <https://www.perfcake.org/docs/perfcake-user-guide.pdf>

Jazyk závěrečné práce: slovenština

Vedoucí práce: Mgr. Jiří Zelinka, Dr.

Datum zadání práce: 7. 12. 2015

V Brně dne: 3. 2. 2016

Souhlasím se zadáním (podpis, datum):

Lenka Heldová
studentka

Mgr. Jiří Zelinka, Dr.
vedoucí práce

v.z. Prof. J. Slovák
prof. RNDr. Jan Slovák, DrSc.
ředitel Ústavu matematiky a
statistiky

Pod'akovanie

Na tomto mieste by som chcela pod'akovať vedúcemu bakalárskej práce Mgr. Jiřímu Zelinkovi, Dr. a odborným konzultantom Mgr. Martinovi Večeřovi a Ing. Pavlovi Macíkovi za veľkú pomoc, dôkladné korektúry, odborné rady, inšpiráciu a hlavne za čas, ktorý mi pri písaní tejto práce venovali. Veľká vďaka patrí tiež mojej rodine a priateľom, ktorí ma počas štúdia podporovali.

Prohlášení

Prohlašuji, že jsem svoji bakalářskou práci vypracovala samostatně s využitím informačních zdrojů, které jsou v práci citovány.

Brno 3. ledna 2017

.....

Lenka Heldová

Obsah

Úvod	ix
Kapitola 1. PerfCake	1
1.1 O nástroji	1
1.2 Použitie	3
1.3 Výstup	5
1.3.1 Dáta	7
1.3.2 Grafy	7
Kapitola 2. Matematická teória	8
2.1 Regresná analýza	8
2.2 Jadrové odhady	9
2.2.1 Jadrová funkcia	9
2.2.2 Šírka vyhľadzovacieho okna	9
2.2.3 Vhodný výber vyhľadzovacieho parametra	10
2.2.4 Typy jadrových odhadov	12
2.3 Miera polohy a miera variability	13
2.3.1 Aritmetický priemer	13
2.3.2 Rozptyl a smerodajná odchýlka	13
Kapitola 3. Analýza dát	15
Kapitola 4. Vyhľadenie dát	17
Kapitola 5. Redukcia dát	23
5.1 Redukcia pomocou smerodajnej odchýlky a priemeru	23
5.2 Redukcia podobná redukcii pri zrýchľovaní zvuku	29
5.3 Redukcia pomocou jadrových odhadov	31
Kapitola 6. Porovnanie algoritmov	42
Závěr	44
Seznam použité literatúry	45

Úvod

Aplikovanie matematiky je dnes potrebné v nespočetne veľa odvetviach bežného ale aj technického života. Počas štúdia som sa začala viac venovať programovaniu, čo ma priviedlo k tejto práci. Pri vývoji softvéru je potrebné testovanie rôznych druhov. Program PerfCake je nástroj, ktorý slúži na testovanie výkonu, ktorý dokáže vizualizovať výsledky pomocou grafov. V priebehu testu sa ale môže nazhromaždiť desaťsíce záznamov dát. Vykresľovať pomocou grafu takéto množstvo dát je pamäťovo aj časovo náročné a preto sa v tejto práci snažím nájsť optimalizačno redukčný algoritmus, ktoré zredukuje počet takéhoto množstva dát, pričom zachová informáciu vychádzajúcu z daného grafu. Práca je členená do šiestich kapitol.

Prvá kapitola je rýchlym zoznámením sa s nástrojom PerfCake, vyvájaným spoločnosťou Red Hat, a úvodom do testovania výkonu. Popisuje sa v nej tiež ako a načo všetko sa nástroj používa aké sú výstupy jeho testovania.

V druhej kapitole sa čitateľ zoznámi s potrebnou matematickou teóriou, pre správne pochopenie podstaty nájdených optimalizačných algoritmov. Venujeme sa v nej regresívnej analýze a konkrétnej neparametrickej metóde pre nájdenie regresnej krivky - jadrovým odhadom. Na záver kapitoly sú ešte vysvetlené pojmy aritmetický priemer, rozptyl a smerodajná odchýlka.

Tretia kapitola sa venuje analýze viacerých typov dát z nástroja PerfCake a pojednáva o tom, akú dôležitosť informáciu dát, a z nich vykreslený graf nesú.

Štvrtá a piata kapitola opisujú proces optimalizovania dát. V štvrtej kapitole je po- písané prvotné vyhľadenie dát pomocou jadrových odhadov, pre odstránenie nepotrebnnej informácie s grafu. V piatej kapitole je potom aplikovaná redukcia, v tejto kapitole sú uvedené tri redukčné algoritmy. Obidve kapitoly sú doplnené ukážkami grafov s výsledkami vyhľadenia a redukcie na viacerých typoch dát.

Kedže cieľom práce je nájsť najvhodnejší algoritmus, v šiestej kapitole sú popísané algoritmy porovnané z niekoľkých hľadísk a pomocou tohto porovnania je vybraný najvhodnejší optimalizačno redukčný algoritmus pre nástroj PerfCake.

Pre spracovanie dát a aplikovanie algoritmov na dátá bol napísaný program v jazyku Java, grafy boli následne vykreslované v programe Microsoft Excel a práca je vysádzaná v systéme L^AT_EX.

Na záver práce je priložená spätná väzba zo spoločnosti Red Hat.

Kapitola 1

PerfCake

Táto kapitola sa venuje nástroju na testovanie výkonu PerfCake, jeho použitiu a typu dát, ktorými sa budeme zaoberať. Celá kapitola je len povrchným zoznámením sa s nástrojom pre potreby tejto práce, čerpá hlavne z oficiálne stránky tohto nástroja, kde v prípade väčšieho záujmu je všetko o nástroji podrobne rozpisane.



Obr. 1.1: PerfCake

1.1 O nástroji

PerfCake je open-source framework vyvíjaný spoločnosťou Red Hat, ktorý je primárne používaný na testovanie výkonu. Je to generátor zaťaženia s cieľom byť minimalistický a ľahko ovládateľný, poskytujúci stabilné výsledky, majúci minimálny vplyv na testovaný systém, byť nezávislý na platforme, umožňujúci vysokú priepustnosť a používajúci komponentový dizajn.

Nástroj PerfCake môže byť použitý na všetky druhy testovania výkonu, včetne:

- *záťažové testovanie (load testing)* - testuje sa správanie softvéru pri očakávanom zaťažení, pri tomto type testu sa prejavujú takzvané “najužšie miesto”(bottlenecks) v systéme, miesta, ktoré v nejakom smere sú systém limitujú,
- *testovanie hraničnej zátlače (stress testing)* - tento test hľadá hranice možného zaťaženia systému a zistuje správanie sa systému za touto hranicou,

- *testovanie odolnosti (endurance testing, soak testing)* - zistuje sa, či testovaný systém dokáže vydržať sústavnú záťaž po dlhšiu dobu a taktiež, či pri takejto sústavnej záťaži neklesá jeho výkon,
- *testovanie škálovateľnosti (scalability testing)* - testuje sa, či systém dobre škáluje a teda či je možné efektívne zvyšovať jeho kapacity bez nečakaných zmien správania,
- *benchmarking* - zaznamenávanie a porovnávanie výsledkov testov, za účelom získania rôznych metrík.

Pre účely spomínaných testov, podporuje PerfCake radu metrík na testovanie cieľového systému. Napríklad:

- monitorovanie vybraného atribútu správy, roztriedenie a spočítanie podľa konkrétnych hodnôt daného atribútu,
- počítanie priemernej priepustnosti za sekundu,
- sledovanie veľkosti spotrebovanej pamäte cieľovej JVM (Java virtual machine),
- zaznamenávanie času odpovede (response time) v milisekundách, pomocou HDR (High Dynamic Range) histogramu, ktorý dokáže riešiť problém Koordinovaného vynechania (Coordinated omission), viac o tomto probléme a riešení pomocou HRD histogramu sa môžeme dočítať na oficiálnej stránke nástroja v sekcii 4.7. Reporting [],
- počítanie štatistik, ako maximum, minimum, priemer z hodnôt priepustnosti alebo časov odpovedí, za čas od začiatku trvania testu, až po reportovanie výsledku alebo zo špecifikovaného časového okna,
- detekovanie pripravenosti systému, ktorá sa určuje pomocou splnenia 3 podmienok: momentálna priepustnosť sa mení o menej ako nastaviteľná hranica, špecifikovaný čas bol dosiahnutý a špecifikovaný počet iterácií bol vykonaný,
- rozoznávanie geo-lokácie tretích strán.

Užívateľ má niekoľko možností reportovania výsledkov, ako napríklad výpis na konzolu, súbor logov, CSV formát, ktorý môže byť importovaný ako tabuľkový zošit (*spreadsheet*) pomocou nástrojov ako Excel alebo LibreOffice Calc, alebo tiež dokáže generovať grafy výsledkov už v priebehu testovania.

Jeden z problémov podobných testovacích nástrojov je prípad, kedy testovaný softvér zlyhá a vzápätí odošle chybovú správu. Pre tento prípad sú v nástroji PerfCake zabudované validátory, ktoré dokážu takúto správu rozpoznať a nezaviesť tým chybové správy do výsledku výkonu softvéru. Podobné softvéry, ktoré takéto validácie neprevádzajú potom vytvárajú veľmi skreslený výsledok.

1.2 Použitie

Pre spustenie nástroja PerfCake je potrebné vytvoriť, takzvané scenáre (scenarios). V týchto súboroch sú definované testovacie bloky, ktoré hovoria PerfCaku čo a ako robiť. Scenáre sú rozdelené do niekolkých sekcií:

- *vlastnosti (properties)* - táto sekcia slúži na pridanie akýchsi meta-dát o scenárii, nie je to povinný parameter a slúži na komplexnejšie využitie testovacích scenárov,
- *beh (run)* - slúži na definovanie dĺžky trvania testu,
- *generátor (generator)* - špecifikuje ako a kolko zaťaženia sa má vytvárať,

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <scenario
3      xmlns="urn:perfcake:scenario:8.0"
4      <!-- Scenario properties (optional) -->
5      <properties>
6          <property name="..." value="..."/>
7          ...
8      </properties>
9      <!-- Run section (required) -->
10     <run ... >
11         ...
12     </run>
13     <!-- Generator section (required) -->
14     <generator ... >
15         ...
16     </generator>
17     <!-- Sequences section (optional) -->
18     <sequences>
19         ...
20     </sequences>
21     <!-- Sender section (required) -->
22     <sender ... >
23         ...
24     </sender>
25     <!-- Receiver section (optional) -->
26     <receiver>
27         ...
28     </receiver>
29     <!-- Reporting section (optional) -->
30     <reporting>
31         ...
32     </reporting>
33     <!-- Messages section (optional) -->
34     <messages>
35         ...
36     </messages>
37     <!-- Validation section (optional) -->
38     <validation>
39         ...
40     </validation>
41 </scenario>
```

Obr. 1.2: Štruktúra scenára

- *odosielateľ'(sender)* - táto sekcia je o spôsobe prenosu, o tom, kde zaťaženie posielat', poprípade o prijímaní odozvy z testovaného systému,

- *prijímateľ'(receiver)* - ďalšia rozširujúca sekcia, ktorá sa používa v prípade, že odozva z testovaného systému prichádza iným kanálom (protokolom) ako bola poslaná správa,
- *reportovanie (reporting)* - definuje metriky, spôsob a výstup reportovania výsledkov testu,
- *správy (messages)* - sekcia slúži na definovanie formátu a obsahu správ posielaných na testovaný systém,
- *sekvencie (sequencies)* - táto sekcia umožňuje používať sekvenciu hodnôt, ktoré sa menia pre každú správu (prípadne množinu správ, špecifikovaných v jednom scenárii), čo dáva možnosť vytvárať unikátné správy,
- *validácia (validation)* - dovoľuje validovať odpoveď testovaného systému.

Štruktúru testovacieho scenára vo formáte XML (eXtensible Markup Language) a povinnosť konkrétnych parametrov zobrazuje obrázok 1.2. Okrem tohto formátu podporuje PerfCake aj iné možnosti scenárov, napríklad DSL (Domain Specific Language) formát alebo použitie špeciálneho rozhrania pre programovanie aplikácií (API) v jazyku Java, kde si užívateľ môže nadefinovať celý scenár.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <scenario
3      xmlns="urn:perfcake:scenario:7.0">
4      <run type="${perfcake.run.type:time}"
5          value="${perfcake.run.duration:30000}" />
6      <generator class="DefaultMessageGenerator"
7          threads="${perfcake.thread.count:100}" />
8      <sender class="HttpSender">
9          <target>http://${server.host}/post</target>
10         <property name="method" value="POST"/>
11     </sender>
12     <reporting>
13         <reporter class="IterationsPerSecondReporter">
14             <destination class="ConsoleDestination">
15                 <period type="time" value="1000" />
16             </destination>
17         </reporter>
18     </reporting>
19     <messages>
20         <message uri="plain.txt" />
21     </messages>
22 </scenario>
```

Obr. 1.3: Príklad scenára

Pre jednoduchú ukážku si uvedieme príklad scenára XML. Máme vyvinutý softvér, ktorý asynchrónne spracováva správy prichádzajúce pomocou POST metódy a HTTP protokolu. Následne tieto správy spracúva (validuje, vytvára záznamy do databázy, generuje

reporty, ...) a po úspešnom spracovaní pošle potvrdzujúcu správu späť. Na obrázku 1.3 sa nachádza scenár pre jednoduchý testovací blok, ktorý by sme mohli na takýto nami vytvorený softvér použiť. Zo scenára je možné vidieť, že tento test bude bežať 3000 ms = 3 sekundy, generuje správy použitím 100 vlákien, ktoré posiela pomocou HTTP protokolu a metódy POST. Text správy je špecifikovaný v súbore "plain.txt" a výsledky sú zaznamenávané na konzolu, každú 1 sekundu. Ako reporter je použitý "IterationsPerSecondReporter" a teda sa každú sekundu zapíše, koľko správ bolo softvérom spracovaných.

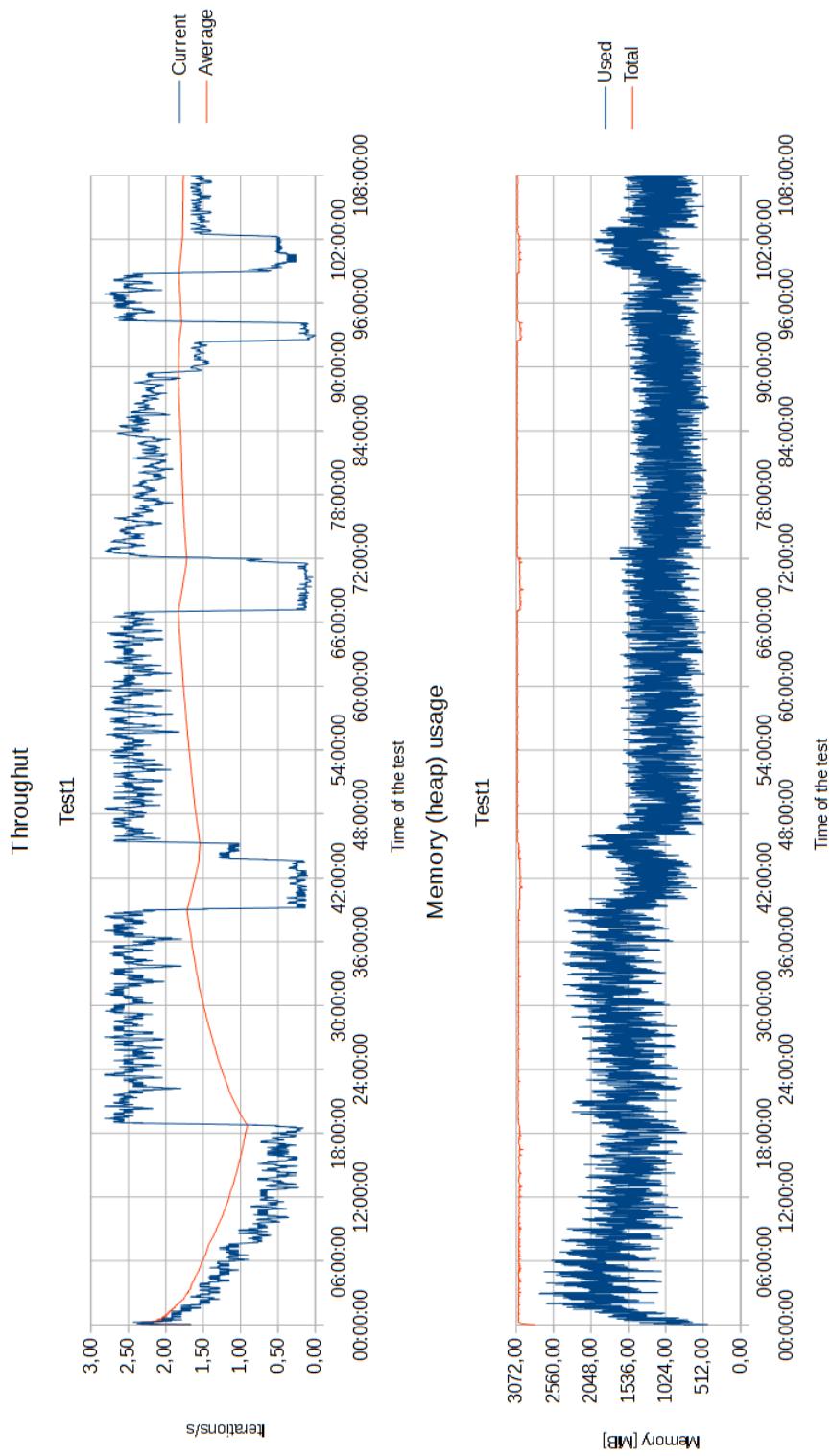
Veľmi podobným scenárom by sme mohli testovať, koľko softvér používa pamäte, stačilo by zmeniť atribút reporter na "MemoryUsageReporter". Týmto spôsobom sa veľmi ľahko odhalujú úniky pamäte. Scenáre sa púšťajú pomocou nástroja Maven alebo pomocou shell skriptu.

1.3 Výstup

Ako bolo uvedené v prvej podkapitole, PerfCake má bohaté možnosti východzích súborov. Pri vytváraní optimalizačného algoritmu budeme ale pracovať iba s dvoma typmi. Dáta budeme spracovať z CSV súboru a následne z nich budeme generovať grafy.

Throughput			Memory		
Time	Current	Average	Time	Used	Total
00:01:00	1,67	1,67	00:01:00	777,30	2817,06
00:02:00	1,67	1,67	00:02:00	528,81	2835,94
00:03:00	2,22	2,22	00:03:00	450,90	2901,19
00:04:00	2,08	2,08	00:04:00	585,94	2911,63
00:05:00	2,33	2,33	00:05:00	979,96	2953,00
00:06:00	2,22	2,22	00:06:00	855,63	2960,25
00:07:00	2,38	2,38	00:07:00	1193,20	2992,13
00:08:00	2,29	2,29	00:08:00	1175,80	3001,75
00:09:00	2,22	2,22	00:09:00	1197,66	3009,88
00:10:00	2,17	2,17	00:10:00	831,64	3013,38
00:11:00	2,27	2,27	00:11:00	1380,88	3019,63
00:12:00	2,22	2,22	00:12:00	815,44	3031,56
00:13:00	2,18	2,18	00:13:00	950,54	3035,38
00:14:00	2,26	2,26	00:14:00	726,44	3036,44
00:15:00	2,44	2,33	00:15:00	659,87	3042,88
00:16:00	2,18	2,19	00:16:00	715,86	3042,88
00:17:00	2,18	2,16	00:17:00	909,47	3042,44
00:18:00	2,05	2,13	00:18:00	1016,83	3042,31
00:19:00	2,14	2,19	00:19:00	697,72	3043,13
00:20:00	2,05	2,17	00:20:00	844,43	3043,75

Tabuľka 1.1: Dáta



Obr. 1.4: Grafy

1.3.1 Dáta

Pri optimalizácii dát pre potreby vykreslovania do grafov, bude používaný formát súboru CSV, ktorý môžeme previesť do podoby tabuľky.

Tabuľka 1.1 ukazuje dátá vygenerované nástrojom PerfCake za 20 minút, testovala sa prieplustnosť softvéru v iteráciách za sekundu a využitie pamäte v megabajtoch. Pri vytváraní algoritmu budeme pracovať len s dátami, ktoré boli vytvorené pri testovaní týchto dvoch vlastností softvéru. Pri prieplustnosti sa počítava priemer za časovú jednotku (current) a tiež priemer za niekoľko časových jednotiek (average), časová jednotka a veľkosť okna, s ktorými sa bude počítať sa dá nastaviť v scenárii, pri dátach s tabuľky je to minúta a 15 záznamov. Priemer sa začína počítať, keď sa naplní okno a potom už pre každý ďalší záznam.

Pri testovaní využívania pamäte sa zaznamenáva použitá (used) a celková pamäť (total). Celková pamäť, je pamäť, ktorú si JVM alokuje a použitá pamäť je pamäť, ktorú naozaj využije - celková mínus voľná pamäť. Viac o pamäti v JVM sa čitateľ dozvie v oficiálnej dokumentácii (<http://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html>).

V popisovanej tabuľke sú dátá zaznamenané za 20 min, v praxi ale tie testy bežia väčšinou niekoľko desiatok hodín a teda tabuľky môžu dosahovať tisíce záznamov. Ukladať a následne spracovať takéto množstvo dát je pamäťovo aj časovo náročné a preto budem hľadať algoritmus, ktorý zmenší objem dát a zároveň neznehodnotí informáciu z nich vyplývajúcemu, pri vykreslení do grafov.

1.3.2 Grafy

Pomocou grafu je často jednoduchšie pochopiť a spracovať veľké množstvo informácií, ako zo surových čísel v obrovskej tabuľke. Na obrázku 1.4 sú znázornené grafy z testov, ktoré bežali 108 hodín, a ktorých prvých 20 záznamov sme si ukázali v tabuľke 1.1. Pre analýzu výsledku nie sú potrebné tak presné grafy, väčšinou je dôležitá hodnota okolo ktorej dátá v určitej časti grafu konvergujú, alebo či hodnoty veľmi "skáču". V bakalárskej práci budú tieto grafy upravené a následne porovnávané s grafmi, ktoré budú vykreslené z dát, na ktoré sa použije optimalizačný algoritmus.

Kapitola 2

Matematická teória

V tejto kapitole uvedieme znalosti z matematickej teórie, ktoré sú potrebné pre zstrojenie optimalizačných algoritmov, ktorými sa budeme zaoberať v následujúcich kapitolách. Najprv sa oboznámime s úvodom do regresnej analýzy a s neparametrickou metódou odhadu regresnej funkcie - metódou jadrových odhadov, ktorú použijeme na vyhľadenie dát. Na konci kapitoly si ešte uvedieme vzorce pre aritmetický priemer, rozptyl a smerodajnú odchýlku, ktoré budú použité na redukciu dát. Pojmy a definície sú prebraté zo zdrojov...

2.1 Regresná analýza

Pre pevné hodnoty nezávisle premennej X (v našom prípade čas) máme k dispozícii nameenané hodnoty závisle premennej Y (priepustnosť, použitá pamäť,...). Takýmito dvojicami bodov $(x_i, Y_i), i = 1, \dots, n$ chceme preložiť vhodnú krivku, tak aby boli odfiltrované výkyvy a bolo možné lepšie poznať štruktúru dát. Táto krivka sa nazýva *regresná krivka* a jej príslušný regresný vzťah zapisujem v tvare

$$Y_i = m(x_i) + \varepsilon_i, i = 1, \dots, n, \quad (2.1)$$

kde m je neznáma regresná funkcia a $\varepsilon_i, i = 1, \dots, n$, sú chyby merania. Cieľom regresnej analýzy je nájsť vhodnú approximáciu \hat{m} neznámej funkcie m . Hľadanie tejto regresnej krivky sa tiež nazýva *vyhľadzovanie* a je možné použiť dva spôsoby odhadu, *parametricky* a *neparametricky*:

- *Parametrický prístup* - predpokladá, že regresná funkcia je nejakého predpísaného tvaru. Odhadnutá regresná funkcia bude teda určitého tvaru a bude ju popisovať množina parametrov - to je dôvod pre názov *parametricky*.
- *Neparametrický prístup* - nepredpokladá predpísany tvar regresnej funkcie. Tento prístup sa vyhýba parametrizácii a tvar funkcie sa odhaduje priamo z dát. Predpokladá sa jedine istá hladkosť hľadanej funkcie.

Jedna z najjednoduchších neparametrických metód je *metóda klízavých priemerov*. Pre odhad hodnoty Y_i sa používa priemer niekolkých hodnôt $Y_j, j \in [i-h, i+h]$ v centrovanom okolí príslušného bodu x_i .

Konkrétnie,

$$\hat{m}(x) = \frac{\sum_{i=1}^n Y_i I_{[x-h,x+h]}(x_i)}{\sum_{i=1}^n I_{[x-h,x+h]}(x_i)}. \quad (2.2)$$

Na vyjadrenie

$$I_{[j,k]}(x) \begin{cases} 1 & x \in [j, k], \\ 0 & \text{inak.} \end{cases} \quad (2.3)$$

sa používa *indikátorová funkcia* $I_{[j,k]}(x)$.

Jadrové odhady sa považujú za zovšeobecnenie *metódy klízavých priemerov*.

2.2 Jadrové odhady

Pri odhadovaní regresnej funkcie pomocou jadrových odhadov, sa používajú vážené hodnoty Y v centrovanom okolí príslušného bodu x_i . Váhy hodnôt Y sú závislé na vzdialosti príslušných x bodov od konkrétneho x_i , bližšie hodnoty - väčšia váha. Toto nám pomáha dosiahnuť *jadrová funkcia*. Vzorec pre jadrové odhady vo všeobecnosti, môžme zapísť nasledovne

$$\hat{m}(x, h) = \sum_{i=1}^n W_i(x, h) Y_i, \quad (2.4)$$

kde $W_i(x, h)$ je váhová funkcia s vyhľadzovacím parametrom h . Ide teda o akýsi vážený súčet pozorovaní, ktorý v jednom z najjednoduchších prípadov môže mať tvar

$$W_i(x, h) = K_h(x - x_i) = \frac{1}{h} K\left(\frac{x - x_i}{h}\right), \quad (2.5)$$

kde K je jadrová funkcia. Váhová funkcia by mala dávať súčet váh 1.

2.2.1 Jadrová funkcia

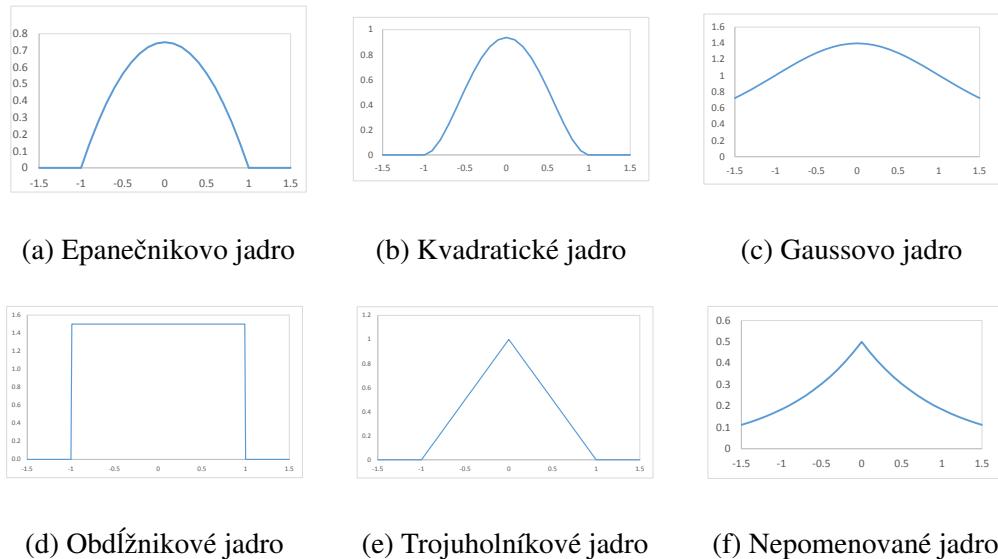
Jadrová funkcia determinuje tvar vyhľadzovacej funkcie. Na obrázku 2.1 môžeme vidieť niekoľko najpoužívanejších jadrových funkcií. V popise funkcie sa používa indikátorová funkcia (2.3).

Vo všeobecnosti hocijaká integrovateľná, obmedzená funkcia, ktorá spĺňa kritériá 2.6 môže byť jadrom.

$$\begin{aligned} a) \int K(z) dz &= 1 & b) \int z K(z) dz &= 0 \\ c) \int z^2 K(z) dz &< \infty & d) K(x) &\geq 0 \text{ pre všetky } x. \end{aligned} \quad (2.6)$$

2.2.2 Šírka vyhľadzovacieho okna

Okrem jadrovej funkcie, alebo jadra, je ďalším dôležitým parametrom tejto metódy šírka vyhľadzovacieho okna h . Šírka vyhľadzovacieho okna alebo aj vyhľadzovací parameter h udáva šírku vyhľadzovacej funkcie a teda aj silu vyhľadenia.



Obr. 2.1: Rôzne tvary jadrových funkcií:

$$\begin{array}{lll}
 \text{a)} K(x) = \frac{3}{4}(1-x^2)I_{[-1,1]}(x), & \text{b)} K(x) = \frac{15}{16}(1-x^2)^2I_{[-1,1]}(x), & \text{c)} K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}, \\
 \text{d)} K(x) = \frac{1}{2}I_{[-1,1]}(x), & \text{e)} K(x) = (1-|x|)I_{[-1,1]}(x), & \text{f)} K(x) = \frac{1}{2}e^{-|x|}
 \end{array}$$

Malá šírka vyhľadzovacieho okna znamená, že odhad závisí na úzkom okolí bodu x_i a teda odhad do veľkej miery reprodukuje pôvodné dátá. Naopak, ak zvolíme vysokú hodnotu h , aj veľmi vzdialené hodnoty majú vysoký dopad na odhad, čo vedie k “prehladeniu” a pri dostatočne veľkej šírke h až k priemeru dát.

Spomínané rozdieli v šírke vyhľadzovacieho okna môžeme vidieť na obrázku 2.2. Pre ilustráciu vplyvu vyhľadzovacieho parametra sú použité reálne dátá z merania použitej pamäte nástroja PerfCake. Aplikované sú Nadaraya-Watsonove odhady, ktoré sú popísané v nasledujúcej podkapitole, s Epanečníkovou jadrovou funkciou.

2.2.3 Vhodný výber vyhľadzovacieho parametra

Hoci je v praxi bežné vybrať vhodnú šírku vyhľadzovacieho okna na základe niekoľkých pokusov a následného subjektívneho výberu, existuje niekoľko metód pre výber optimálnej šírky vyhľadzovacieho okna.

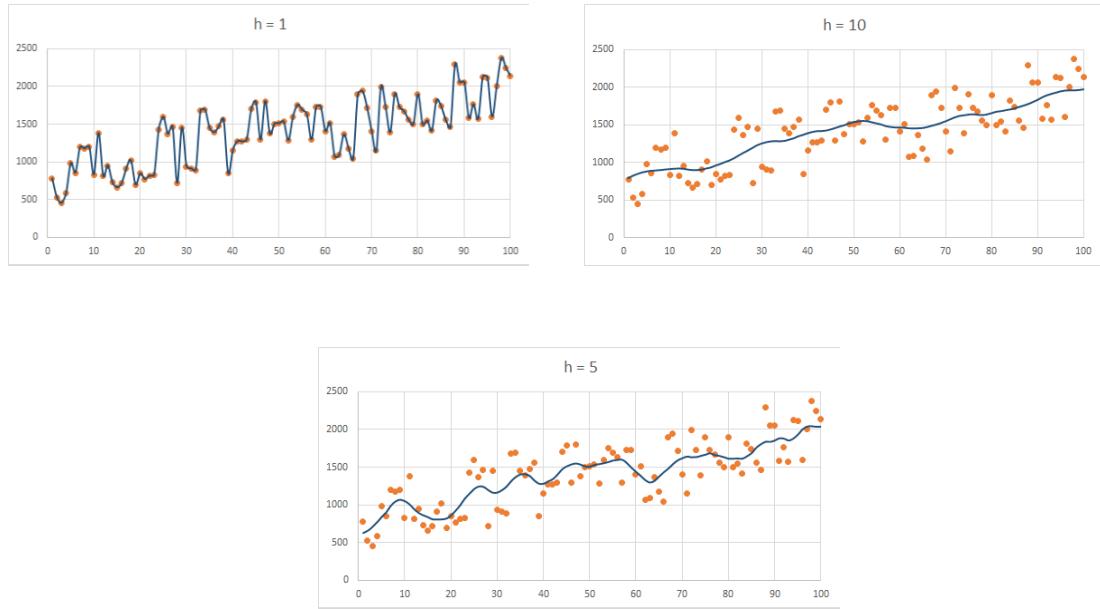
Kvalita jadrových odhadov regresnej funkcie môže byť lokálne popísaná pomocou *srednej kvadratickej chyby (MSE - Mean Square Error)*

$$MSE\{\hat{m}(x, h)\} = E\{\hat{m}(x, h) - m(x)\}^2.$$

Alebo tiež pomocou globálnej chyby - *priemernej strednej kvadratickej chyby (AMSE - average MSE)*

$$AMSE\{\hat{m}(., h)\} = \frac{1}{n}E \sum_{i=1}^n \{\hat{m}(x_i, h) - m(x_i)\}^2.$$

Minimalizovaním týchto chýb pre hodnotu parametra h , dostávame jeho optimum.



Obr. 2.2: Porovnanie šírky vyhľadzovacieho okna, bodky sú pôvodné dátá, súvislá čiara odhadnutá regresná funkcia

Kedže vzorec pre AMSE obsahuje neznáme hodnoty regresnej funkcie, použijeme pre výpočet optimálnej hodnoty parametra h , reziduálny súčet štvorcov (RSS), kde tieto neznáme hodnoty nahradíme nameranými hodnotami Y_i . Teda

$$RSS_n(h) = \frac{1}{n} \sum_{i=1}^n \{Y_i - \hat{m}(x_i, h)\}^2. \quad (2.7)$$

RSS ale nie je nestranným odhadom AMSE, preto sa metódy pre odhad optimálnej šírky vyhľadzovacieho okna snažia RSS upraviť tak, aby bol nestranný.

Jedna z najpoužívanejších metód tohto typu je *metóda krížového overenia* (*Cross-validation method*). Táto metóda spočíva vo vyniechaní pozorovania x_j pre spočítanie odhadu v bode x_j

$$\hat{m}_{-j}(x_j, h) = \sum_{\substack{i=1 \\ i \neq j}}^n W_i(x_j, h) Y_i.$$

S použitím tejto modifikácie, môžme RSS prepísat do tvaru

$$CV(h) = \frac{1}{n} \sum_{i=1}^n \{Y_i - m_{-i}(x_i)\}^2, \quad (2.8)$$

funkcia CV je tiež známa ako “cross-validačná” funkcia. Odhadnuté optimálne h dostaneme minimalizovaním (2.8)

$$\hat{h}_{CV} = \arg \min_{h \in H_n} CV(h). \quad (2.9)$$

Aj keď je táto metóda jedna z najpoužívanejších, negarantuje nestrannosť odhadu, ani že \hat{h}_{CV} minimalizuje AMSE (alebo hocijakú inú chybovú mieru). Dôsledkom toho je, že \hat{h}_{CV} nadobúda väčšinou jemne nižšiu hodnotu ako je optimálna šírka h .

Viac o metóde krížového overenia, ale aj iných metódach na odhad optimálnej šírky vyhľadzovacieho okna, napr. Malloowsova metóda, Penalizačná metóda, alebo metóda založená na Fourierovej transformácii, môžme nájsť v [(Kernel smoothing)].

2.2.4 Typy jadrových odhadov

V tejto podkapitole si uvedieme niektoré najznámejšie typy jadrových odhadov, ktoré sú uvedené v zdrojoch ...

Ak ako váhovú funkciu použijeme (2.5) dostávame **Priestley-Chaove odhady**

$$\hat{m}_{PCH}(x, h) = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) Y_i. \quad (2.10)$$

Modifikovaním tohto estimátora získame ďalší známy typ jadrových odhadov, **Gasser-Müllerove odhady**

$$\hat{m}_{GM}(x, h) = \sum_{i=1}^n Y_i \int_{s_{i-1}}^{s_i} K_h(t - x) dt, \quad (2.11)$$

kde $s_0 = 0, s_i = \frac{x_i + x_{i+1}}{2}, s_n = 1$, pre x_i ekvidistantne rozdelené na $[0,1]$. Tento estimátor sa dá vhodne použiť pre odhad derivácie regresnej funkcie.

Nasledujúce typy jadrových odhadov vychádzajú z takzvaných *lokálne polynomiálnych jadrových odhadov*. Odhad neznámej regresnej funkcie v bode x je získaný preložením polynómu stupňa d váženou metódou najmenších štvorcov. Nech tento polynom má tvar

$$P(u) = \beta_0 + \beta_1(u - x) + \dots + \beta_d(u - x)^d. \quad (2.12)$$

Váhy sú dané pomocou jadrovej funkcie

$$K_h(x - x_i) = \frac{1}{h} K\left(\frac{x - x_i}{h}\right). \quad (2.13)$$

Aplikujeme váženú metódu najmenších štvorcov, to znamená, že minimalizujeme výraz

$$\sum_{i=1}^n \{Y_i - \beta_0 - \beta_1(x_i - x) - \dots - \beta_d(x_i - x)^d\}^2 K_h(x_i - x) \quad (2.14)$$

v závislosti na parametroch $\boldsymbol{\beta} = (\beta_0, \dots, \beta_d)'$. Označme $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \dots, \hat{\beta}_d)'$ vektor, pre ktorý (2.14) nadobúda minimum. Odhad regresnej funkcie získaný popísanou metódou je hodnota parametra $\hat{\beta}_0$.

Špeciálnym prípadom lokálne polynomiálnych odhadov, kde stupeň polynómu $d = 0$ sú **Nadaraya-Watsonove odhady**

$$\hat{m}_{NW}(x, h) = \frac{\sum_{i=1}^n K_h(x - x_i) Y_i}{\sum_{i=1}^n K_h(x - x_i)}. \quad (2.15)$$

V tomto prípade prekladáme dátu konštantnou funkciou. Môžeme si tiež všimnúť, že ak použijeme Nadaraya-Watsonove odhady s obdĺžnikovou jadrovou funkciou, dostaneme metódu kľzavých priemerov (2.2).

Druhým špeciálnym prípadom lokálne polynomiálnych odhadov, kde stupeň polynómu $d = 1$ a ide teda o priamku, sú **Lokálne lineárne estimátory**

$$\hat{m}_{LL}(x, h) = \frac{1}{n} \sum_{i=1}^n \frac{\hat{s}_2(x; h) - \hat{s}_1(x; h)(x_i - x)K_h(x - x_i)Y_i}{\hat{s}_2(x; h)\hat{s}_0(x; h) - \hat{s}_1(x; h)^2}, \quad (2.16)$$

kde

$$\hat{s}_r(x, h) = \frac{1}{n} \sum_{i=1}^n (x_i - x)^r K_h(x - x_i). \quad (2.17)$$

2.3 Miera polohy a miera variability

Ako bolo uvedené na začiatku kapitoly, v tejto podkapitole sa oboznámime ešte s niektorými pojмami zo štatistiky.

Majme *súbor hodnôt* nejakej náhodnej veličiny

$$x_1, x_2, x_3, \dots, x_n, \quad (2.18)$$

kde n je rozsah súboru a jeho hodnoty sú intervalového alebo pomerového typu. To znamená, že je možné stanoviť vzdialenosť medzi meranými hodnotami, a teda počítať s ich rozdielami. Tento súbor hodnôt môžeme analyzovať niekoľkými spôsobmi. Oboznámime sa niekoľkými charakteristikami takýchto náhodných veličín.

2.3.1 Aritmetický priemer

Na určenie hodnoty, okolo ktorej sa hodnoty jednotlivých pozorovaní nachádzajú je vhodný aritmetický priemer daného súboru.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.19)$$

Aritmetický priemer je tiež nazývaný miera polohy štatistických znakov.

2.3.2 Rozptyl a smerodajná odchýlka

Miery variability určujú, spôsob akým sú merané hodnoty usporiadane okolo strednej hodnoty. Najpoužívanejšie miery variability sú rozptyl a smerodajná odchýlka.

Rozptyl je často označovaný ako *stredná kvadratická odchýlka* a je definovaný

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (2.20)$$

V prípade, že rozptyl počítame iba zo vzorky hodnôt, a nie z celého súboru hodnôt, vzorec sa zmení na

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2, \quad (2.21)$$

vďaka tejto úprave bude vypočítaný rozptyl jemne väčší, čím sa stáva nestranným odhadom skutočného rozptylu. Táto korektúra sa používa na zmiernenie zkreslenia výsledku pri výpočte zo zmenšeného počtu dát.

Smerodajná odchýlka je daná ako odmocnina z rozptylu

$$s = \sqrt{s^2}. \quad (2.22)$$

Čím väčšie hodnoty rozptyl a smerodajná odchýlka naberajú, tým viac sú hodnoty rozpísané od priemeru, naopak menšie hodnoty indikujú “tesnejšie” usporiadanie meraných hodnôt.

Kapitola 3

Analýza dát

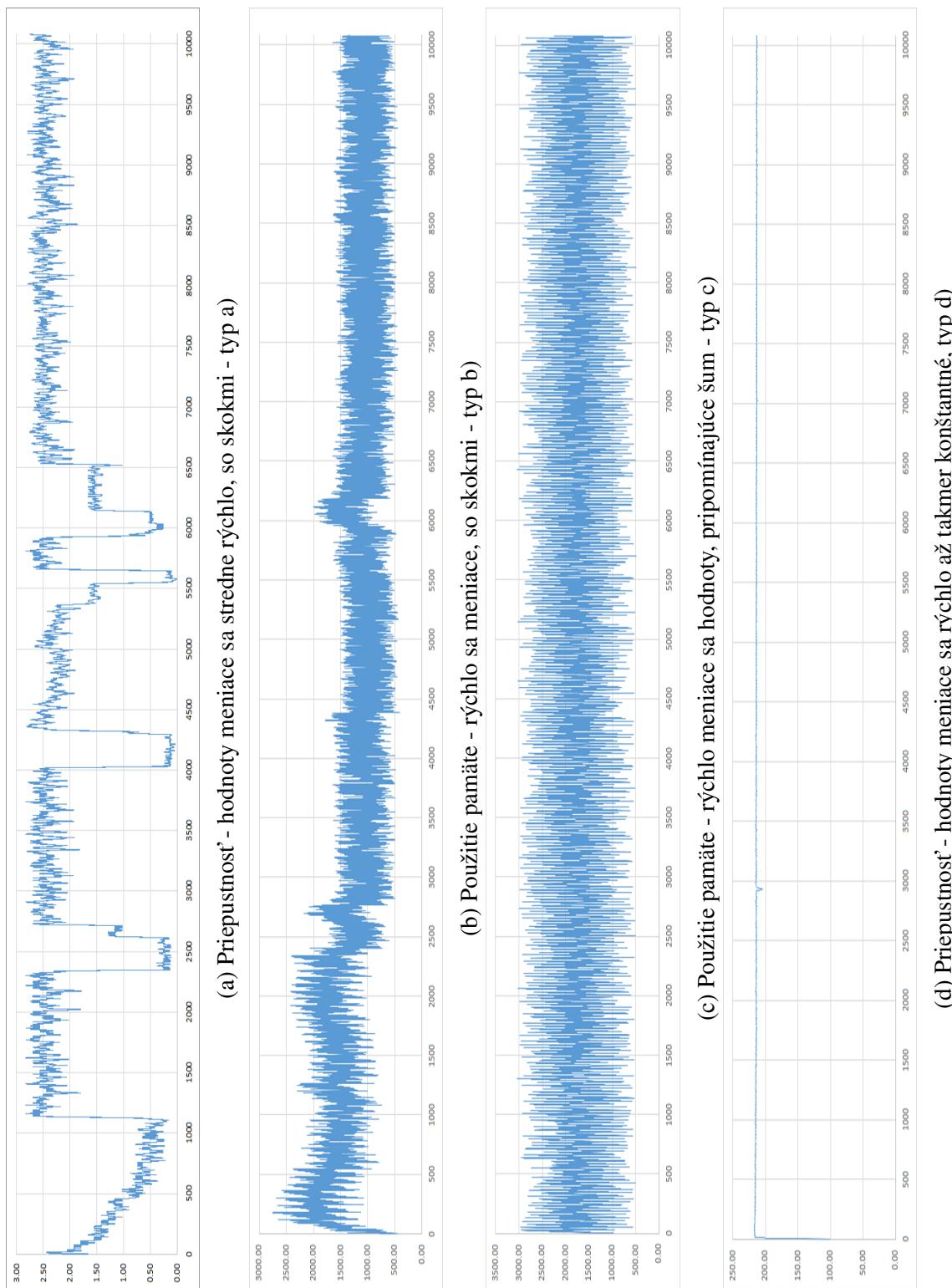
V tejto kapitole si ukážeme a zanalyzujeme konkrétné dáta vykreslené do grafov, s ktorými budeme pri vytváraní optimalizačného algoritmu pracovať.

Nástroj PerfCake môže testovať radu funkčností, ako bolo opísané v kapitole 1. Pre vytvorenie vhodného algoritmu máme k dispozícii štyri typy dát, na ktorých by mal algoritmus vhodne fungovať. Od stredne rýchlo sa meniacich hodnôt dát so “skokmi”, typ a), cez rýchlo sa meniace hodnoty dát so “skokmi”, typ b), a rýchlo sa meniace hodnoty dát (pripomínajúce šum), typ c), až po pomaly sa meniace, takmer konštantné hodnoty dát, typ d). Na obrázku 1.4 môžeme z vymenovaných vidieť dva typy dát, vykreslené do grafov z redukovaného počtu hodnôt, do podoby v akej sú potrebné pre analýzu výsledkov PerfCaku. V tejto práci, budeme pracovať s grafmi všetkých štyroch typov, každý pôvodne vykreslený z 10080 hodnôt, ktoré budeme následne redukovať, v podobe nami vyhovujúcej k následnému spracovaniu a demonštrovaniu výsledkov. Tieto grafy môžeme vidieť na obrázku 3.1, sú to dáta vygenerované nástrojom PerfCake z testov, ktoré bežali 168 hodín, dva grafy z testov na použitú pamäť a dva grafy z testov na meranie prieplustnosti.

Grafy z nástroja PerfCake sú určené ľudským užívateľom. Voľným okom nie je možné rozoznať desaťtisíce hodnôt v jednom grafe, navyše 10080 hodnôt pochádzajúce z testov, ktoré bežali sedem dní a zaznamenávali dátu každú minútu, tieto testy môžu byť spustené na dlhší čas, poprípade zaznamenávať dátu častejšie. V takých prípadoch, môže byť záznamov niekoľko násobne viac, čo je aj v dnešnej dobe pamäťovo náročné a v prípade vykreslenia do grafu, náročné časovo. V neposlednom rade náročné na analýzu.

Z grafu je možné vyčítať niekoľko dôležitých informácií. Okolo akej hodnoty sa dátu pohybujú, ako veľmi a často “skáču” a teda ako veľmi sú stabilné, prípadne tesnosť ich rozptylenia. Samozrejme analýza grafu sa mení v závislosti na vlastnosti softvéru, ktorú meriame. Pri testovaní použitej pamäte je rozptyl hodnôt obvykle väčší ako pri testovaní prieplustnosti, čo môžeme vidieť aj na obrázku 3.1. Vytvorené optimalizačné algoritmy dbajú na zachovanie, čo možno najmenšieho skreslenia týchto informácií pri redukovaní počtu dát. Algoritmy sú konfigurovatelné, čiže veľkosť redukcie, na úkor čo najmenšieho skreslenia je nastaviteľná užívateľom, pomocou niekoľkých parametrov. Samozrejme sa predpokladá, že užívateľ, ktorý analyzuje výsledky je oboznámený s aplikovaním optimalizačného algoritmu, poprípade si ho sám nakonfiguruje.

Postupné aplikovanie optimalizačných algoritmov na dátu, spolu s ich porovnaním si ukážeme v nasledujúcich kapitolách.



Obr. 3.1: Typy dát v grafoch

Kapitola 4

Vyhľadenie dát

V tejto kapitole aplikujeme jadrové odhady opisované v podkapitole 2.2 na dáta, čím ich vyhľadíme. Touto úpravou sa nestratí takmer žiadna podstatná informácia. Pre aplikovanie jadrových odhadov sme sa rozhodli po dôkladnej analýze dát a vyskúšaní kľzavých priemerov.

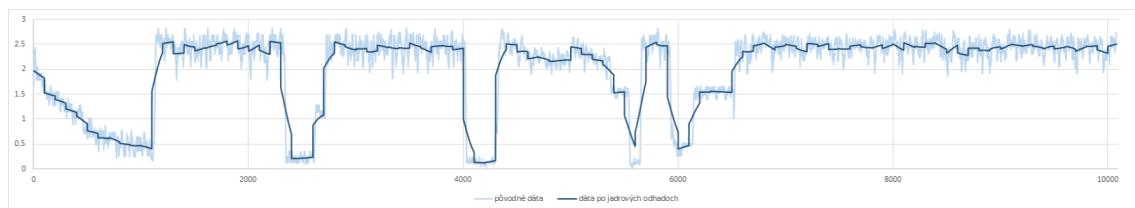
Pri počítaní jadrových odhadov budeme používať Nadaraya-Watsonove odhady po- písané v rovnici (2.15) s Epanečníkovým jadrom, ktoré môžme vidieť na obrázku 2.1, konkrétnie obrázok (a), a s použitím (2.13).

Použitý vzorec teda vyzerá

$$\hat{m}_{MW}(x; h) = \frac{\sum_{i=1}^n \frac{1}{h} \frac{3}{4} \left(1 - \left(\frac{x-x_i}{h}\right)^2\right) Y_i}{\sum_{i=1}^n \frac{1}{h} \frac{3}{4} \left(1 - \left(\frac{x-x_i}{h}\right)^2\right)}, \quad (4.1)$$

kde $\frac{3}{4} \left(1 - \left(\frac{x-x_i}{h}\right)^2\right)$ dáva 0, keď $|x - x_i| \geq h$.

Optimalizačný algoritmus, by mal byť spustený naraz s testovaním nástrojom PerfCake a mal by pracovať v reálnom čase, poprípade s nejakých zdržaním. Určite ale pri aplikovaní jadrových odhadov, nebudú k dispozícii celé dátá. Budeme teda počítať jadrové odhady vždy zo 100 záznamov. Čo by spôsobilo zdržanie maximálne 100 záznamov plus výpočet.

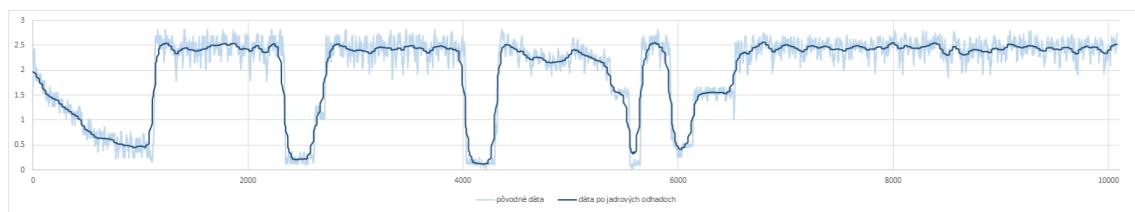


Obr. 4.1: Jadrové odhady bez prelínania

Na obrázku 4.1 môžme vidieť aplikovanie jadrových odhadov (4.1) na dátá, vždy pre každých 100 záznamov a dĺžkou vyhľadzovacieho okna 50. Graf, ktorý je výsledkom (tmavá čiara), je ale príliš "hranatý", čo je následkom rozdelenia dát na 100 záznamov

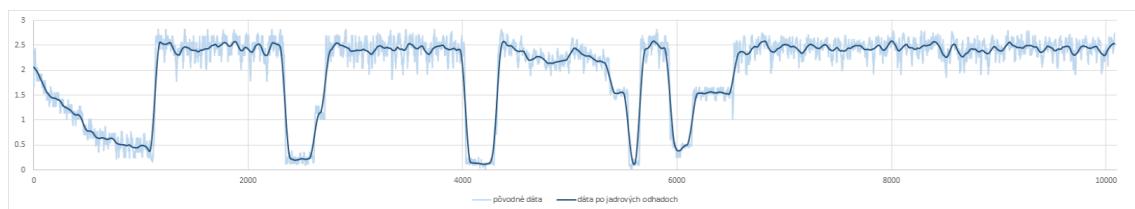
dlhé bloky. Tento algoritmus tiež z teoretického hľadiska zavádza mnoho chýb do odhadu, pretože na okraji intervalu nie je k dispozícii dostatočný záznamov pre spočítanie nestranného odhadu, čo sa pri opísanej aplikácii stáva na obidvoch okrajoch intervalu o dĺžke 100 záznamov.

Skúsmo upraviť vyhladzovací algoritmus tak, že budeme sice aplikovať jadrové odhady vždy na 100 záznamov dlhé bloky, ale tieto bloky sa budú prekrývať. Po aplikovaní na prvý blok záznamov, sa posunieme o 30 záznamov a aplikujeme jadrové odhady znova. To znamená, že pre väčšinu záznamov (okrem prvých 30 a posledných 30 záznamov) bude spočítaných niekoľko odhadov, pre výpočet výslednej hodnoty, tieto odhady spriemerujeme. Takto upravený algoritmus s dĺžkou vyhladzovacieho okna $h = 50$ aplikovaný na dátu môžeme vidieť na obrázku 4.2.



Obr. 4.2: Jadrové odhady s prekrývaním

Takto upravený algoritmus na vyhladenie dát je ale skoro niekoľkokrát pomalší, pretože takmer všetky hodnoty odhadu sa počítajú viac krát. Skúsmo teda algoritmus upraviť ešte inak. Nebudeme dátu rozdeľovať na bloky z fixnou dĺžkou, keďže pri použití Epaničkovho jadra sa každá hodnota odhadu spočíta z množstva záznamov, ktoré určí dvojnásobok dĺžky vyhladzovacieho parametra. To znamená, že v prípade dĺžky okna 50, zoberieme pre každý odhad 50 záznamov naľavo, 50 napravo a z nich spočítame jadrový odhad. V prípade prvých 50 a posledných 50 nastáva skreslovanie, kvôli nedostatku záznamov pre spočítanie nestranného odhadu, nazýva sa tiež hraničné efekty. Toto hraničné skreslovanie nastáva iba na začiatku a na konci celých dát. Týmto výpočtom simulujeme počítanie jadrových odhadov pre všetky dátu naraz, a zároveň spočítame odhad v každom bode iba raz. Použitie takéhoto vyhladenia je zobrazené na obrázku 4.3, použitá dĺžka vyhladzovacieho okna je taktiež 50.

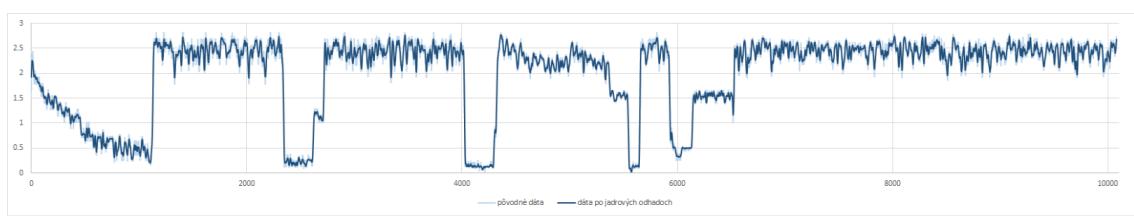
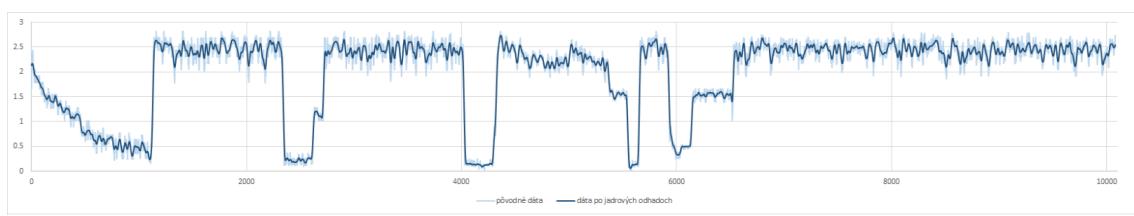
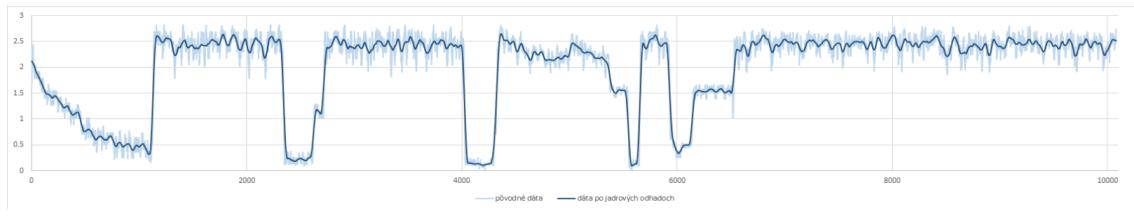


Obr. 4.3: Jadrové odhady s posúvaním bloku záznamov

Zdržanie takéhoto vyhladzovacieho algoritmu je naplnenia okna o dĺžke parametra h

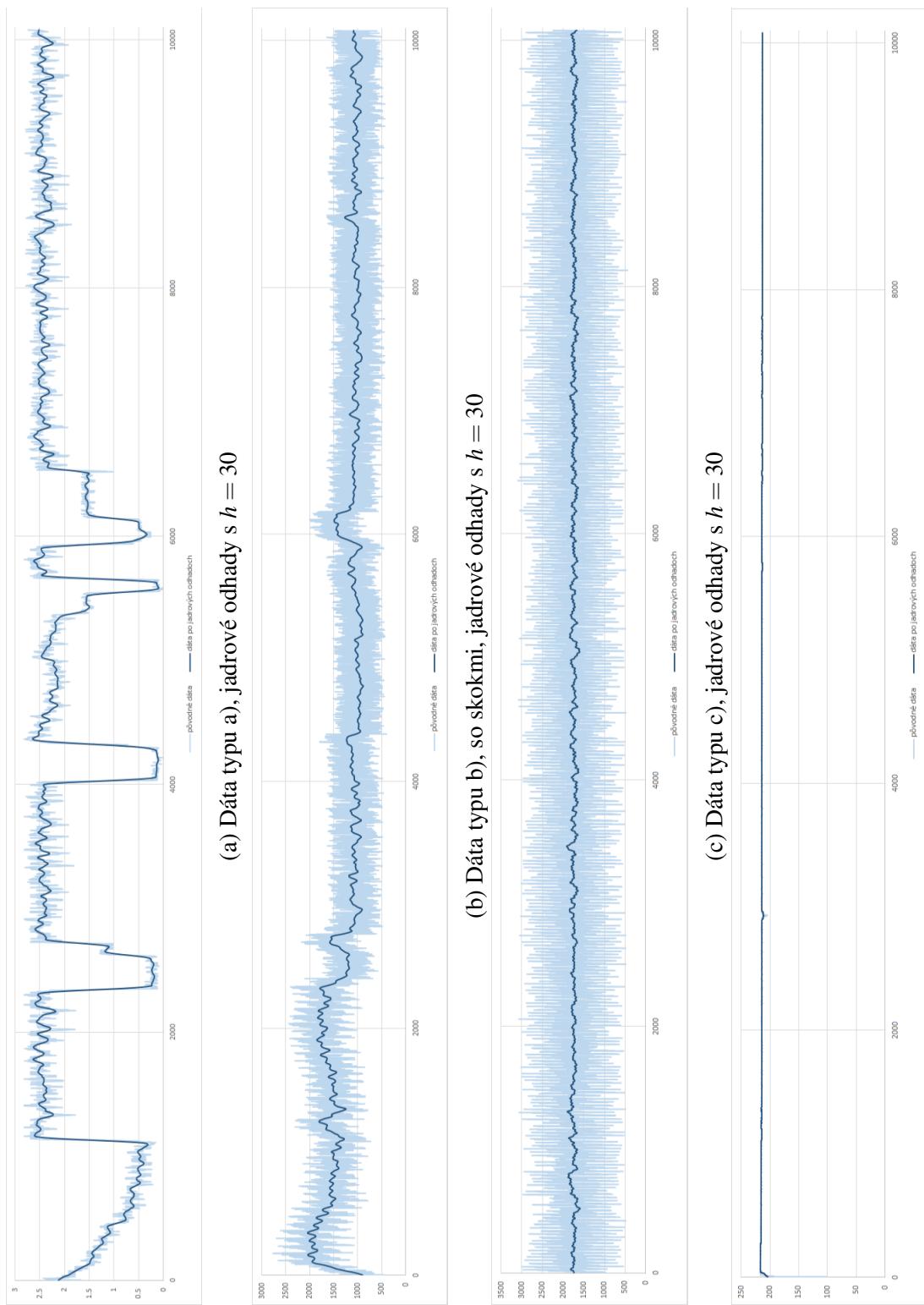
plus výpočet, a potom už iba výpočet pri pridaní ďalšieho záznamu. Dobu výpočtu je možné pri použití Nadaraya-Watsonových odhadov skrátiť. V 4.1 môžeme vidieť, že váhová funkcia je závislá iba na parametri h a vzdialenosť danej hodnoty Y od bodu, pre ktorý sa odhad počíta. Preto si tieto vähy môžme spočítať iba raz a potom ich vhodne používať pre výpočet ostatných odhadov. Tento vyhľadzovací algoritmus s počítaním jadrových odhadov bude základom optimalizačných algoritmov, ktorých druhou časťou bude redukcia dát opísaná v ďalšej kapitole.

Ako už bolo spomínané dĺžka vyhľadzovacieho okna je veľmi dôležitým parametrom pri jadrových odhadoch. Počítať optimálnu dĺžku vyhľadzovacieho okna dynamicky pre každý nový záznam, by bolo veľmi náročné a navyše dáva tento parameter možnosť nastavenia veľkosti vyhľadzovacieho efektu. Preto bude dĺžka vyhľadzovacieho okna konfigurovateľný vstupný parameter pre algoritmus. Čím väčšiu hodnotu užívateľ zvolí, tým viac vyhľadené grafy budú výsledkom. Na obrázku 4.4 môžeme vidieť aplikovanie upravených jadrových odhadov s rôznymi dĺžkami vyhľadzovacieho okna, s dĺžkami 5, 15 a 30. Vyhľadzovanie s dĺžkou okna 50 bolo ukázané na obrázku 4.3.

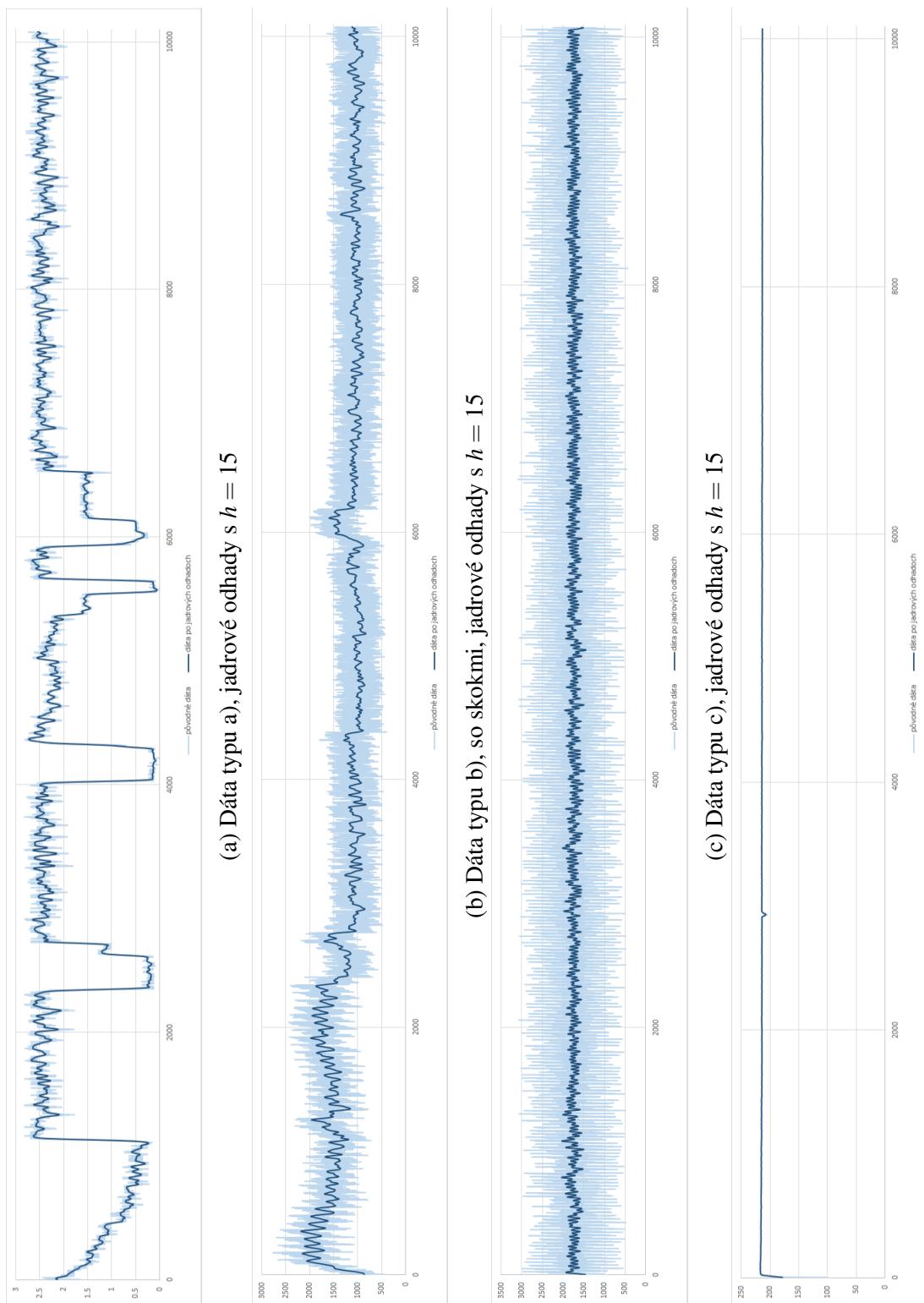


Obr. 4.4: Jadrové odhady s rôznymi hodnotami vychladzovacieho parametra

Vidíme, že zmena dĺžky vyhľadzovacieho okna nám naozaj dáva možnosť riadiť silu vyhľadenia. Vplyv jadrových odhadov na jede typ dát, typ a), sme si už ukázali, na ďalších



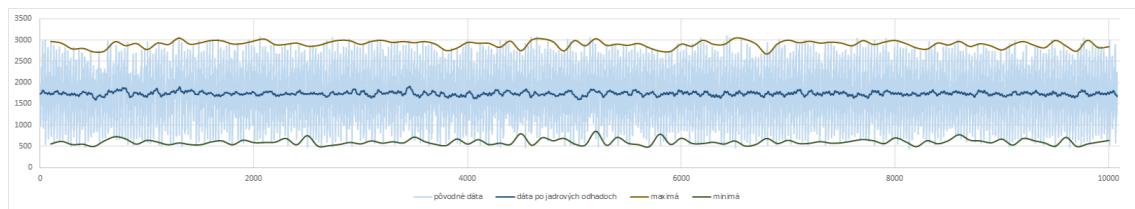
Obr. 4.5: Typy dát v grafoch s jadrovými odhadmi



Obr. 4.6: Typy dát v grafoch s jadrovými odhadmi

obrázkoch, si ukážeme vyhladené aj ostatné typy dát. Pre vyhladenie na obrázku 4.5 bola zvolená dĺžka okna 30 a na obrázku 4.6 dĺžka okna 15.

Pri type c) sa hodnoty menia veľmi rýchlo a bez akéhokoľvek trendu, pripomínajú biely šum. Pri takýchto dátach, by vplyvom vyhladenia mohlo dôjsť k strate informácie o veľkosti rozptylu dát. Samozrejme to závisí na testovanej vlastnosti a type softvéra. Obvykle sa pri testovaní použítej pamäti zameriava na hľadanie “únikov pamäte” (skôr známe pod anglickým pojmom - *memory leak*). V tomto prípade je takéto vyhladenie viac ako prospešné, pretože ukazuje, že nenastali žiadne výkyvy. Naopak ak by sa pri testovaní zameriaval na rozptyl hodnôt použítej pamäti, poprípade na nejaké lokálne maximá, vyhladenie by takéto informácie zničilo. Pre takéto prípady môžeme pridať možnosť ohraňujúcich kriviek, ktoré doplnia stratené informácie. Použitie týchto kriviek je na obrázku 4.7.



Obr. 4.7: Ohraničujúce krivky

Dáta, z ktorých sú krivky vykreslené sú tvorené maximami a minimami z každých 100 vzoriek. To znamená, že pri 10080 záznamov potrebujeme ďalších 202 záznamov pre tieto krivky. Vykreslovanie ohraničujúcich kriviek, ktoré ukazujú akési lokálne minimá a maximá, bude ďalší konfigurovatelný parameter pre výsledný optimalizačný algoritmus.

Kapitola 5

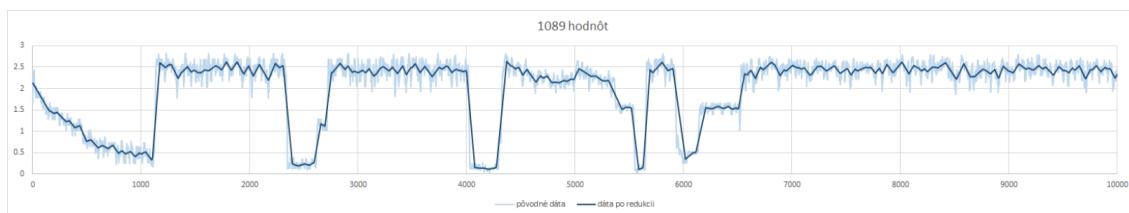
Redukcia dát

V tejto kapitole aplikujeme na už vyhadené dátá redukciu. V každej podkapitole si popíšeme a použijeme iný redukčný algoritmus.

5.1 Redukcia pomocou smerodajnej odchýlky a priemeru

Pri aplikovaní tejto redukcie je hlavná myšlienka taká, že ak sa dátá veľmi nemenia môžme ich redukovať. Na túto redukciu nám poslúži smerodajná odchýlka.

Algoritmus pre každú hodnotu spočíta smerodajnú odchýlku z posledných 10 hodnôt včetne danej hodnoty, ak je smerodajná odchýlka spočítaná s danou hodnotou niekoľko krát väčšia ako smerodajná odchýlka spočítaná bez tejto hodnoty pre predchádzajúci záznam, hodnota sa uloží. Číslo, ktoré určuje koľkokrát sa musí zmeniť smerodajná odchýlka aby sa daná hodnota uložila, bude ďalším voliteľným parametrom, ktorý riadi silu redukcie (ďalej len parameter redukcie). Vplyv tohto parametru na dátá ukazuje obrázok 5.1, redukcia bola aplikovaná na dátá vyhadené oknom o šírke 30.



(a) Redukcia pomocou smerodajnej odchýlky, parameter = 1,2



(b) Redukcia pomocou smerodajnej odchýlky, parameter = 1,3



(c) Redukcia pomocou smerodajnej odchýlky, parameter = 1,4

Obr. 5.1: Redukcia pomocou smerodajnej odchýlky, použitá pôvodná hodnota

Malou obmenou tohto algoritmu môže byť, ukladanie priemeru všetkých 10 záznamov namiesto danej hodnoty, ktorá rozhoduje o uložení, respektíve vyhodení záznamu. Nasledujúci obrázok 5.2 ukazuje túto obmenu na rovnakých dátach ako obrázok 5.1.



(a) Redukcia pomocou smerodajnej odchýlky, parameter = 1,2



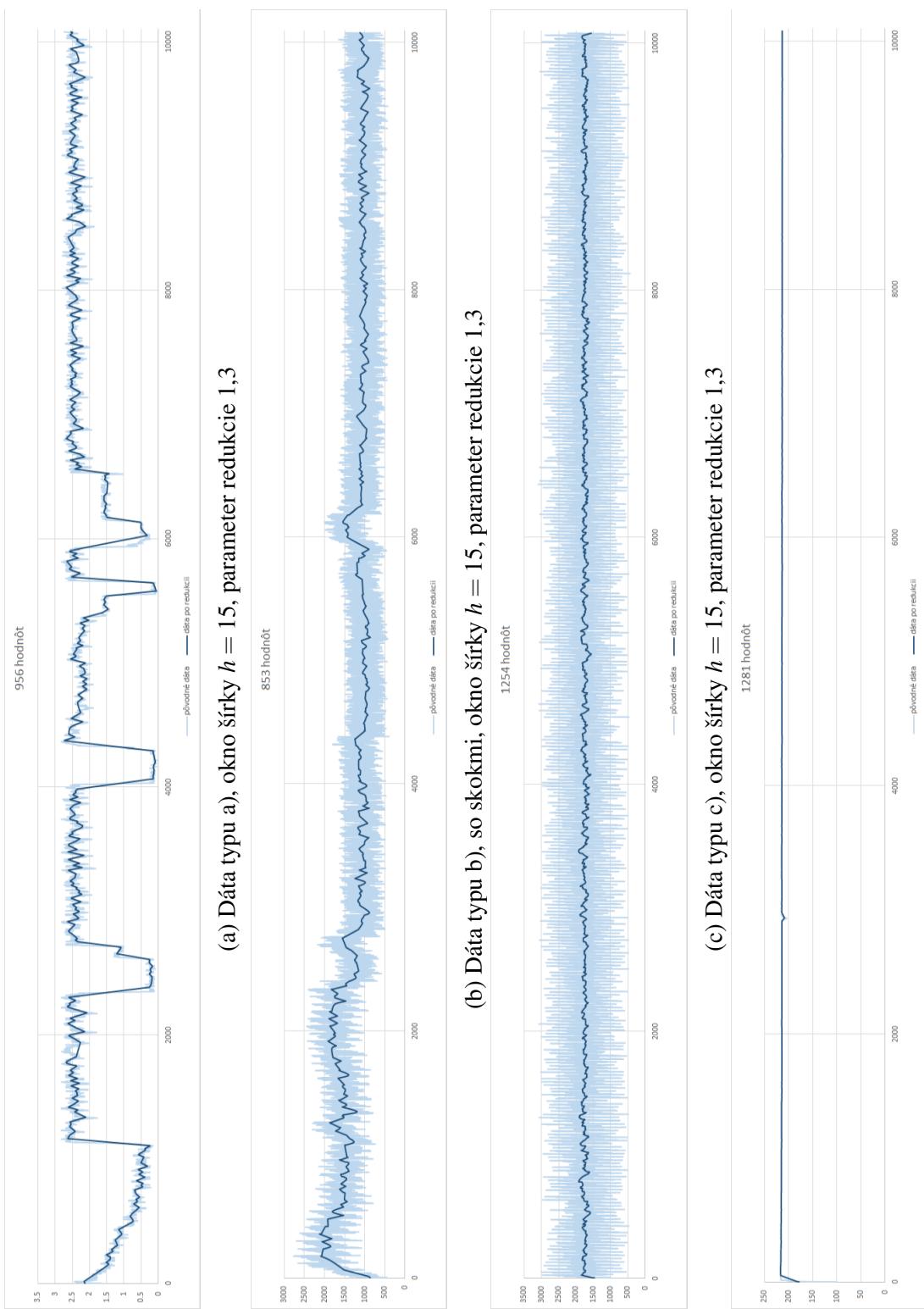
(b) Redukcia pomocou smerodajnej odchýlky, parameter = 1,3



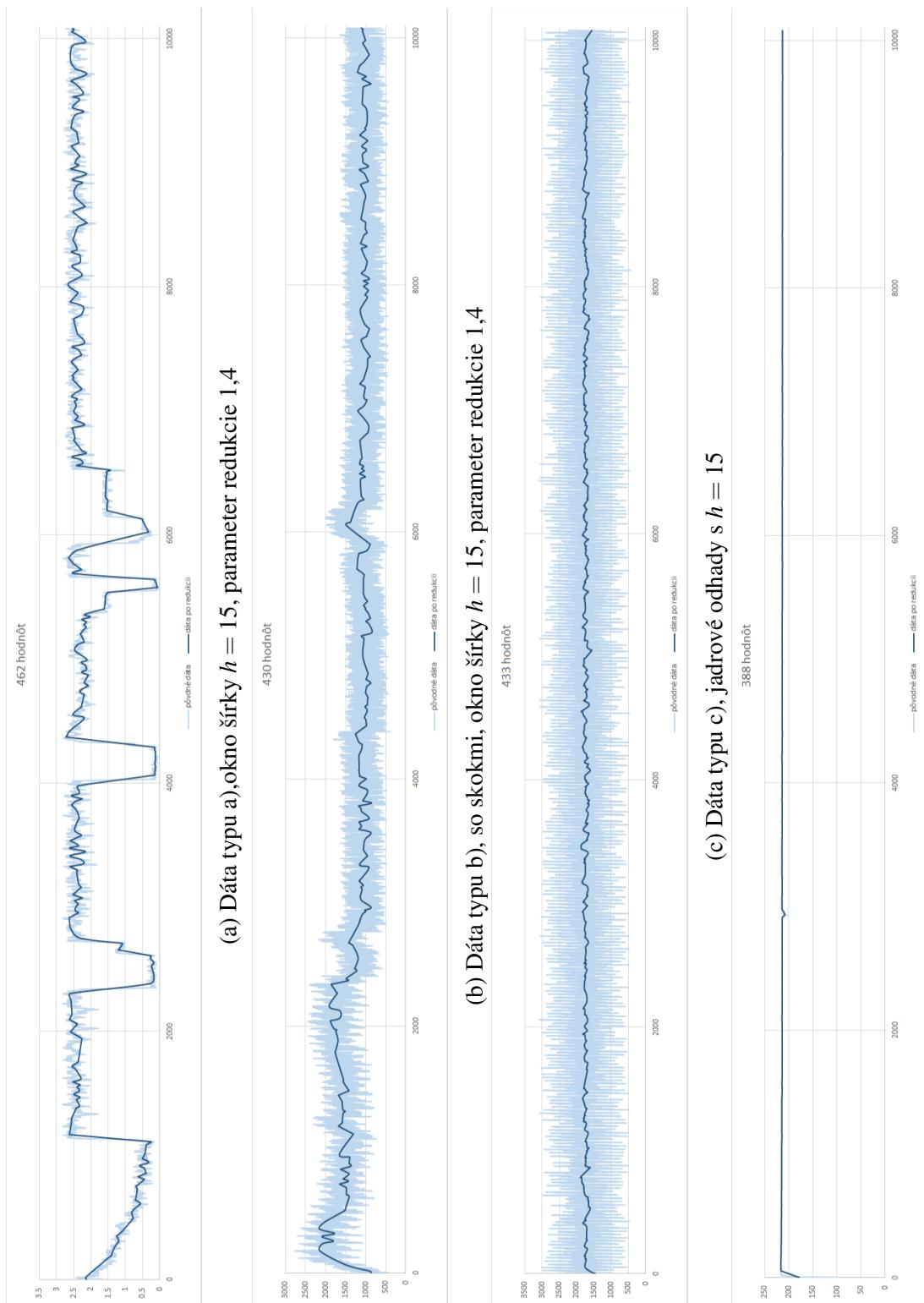
(c) Redukcia pomocou smerodajnej odchýlky, parameter = 1,4

Obr. 5.2: Redukcia pomocou smerodajnej odchýlky, použitý priemer hodnôt

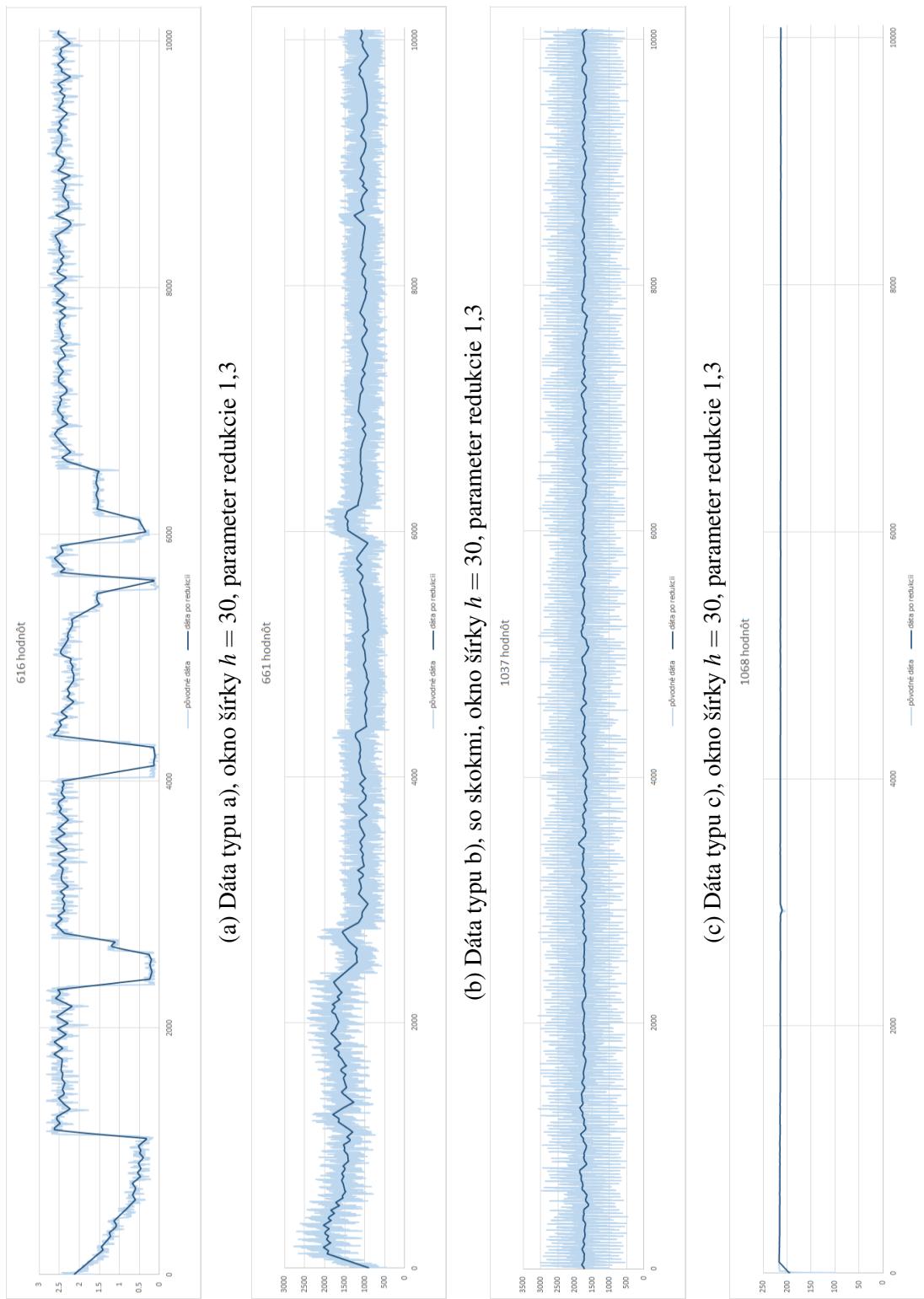
Takto upravený algoritmus nezahadzuje informáciu z vylúčených záznamov, pretože ju použije v najbližšej uloženej hodnote v podobe priemeru. Preto tento algoritmus s používaním priemeru dáva reálnejšie výsledky. Na nasledujúcich obrázkoch si ukážeme



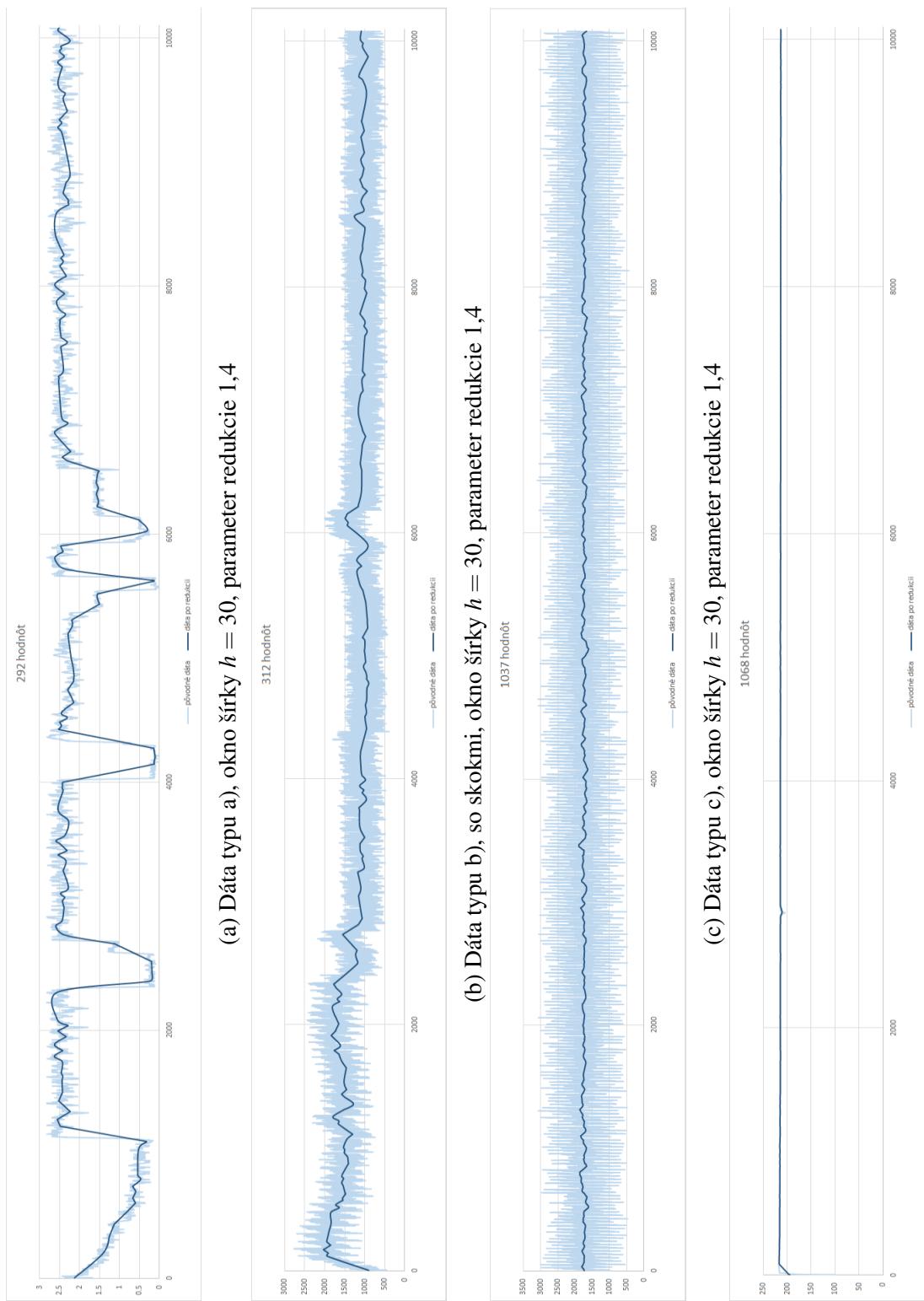
Obr. 5.3: Typy dát v grafoch po redukcii pomocou smerodajnej odchýlky



Obr. 5.4: Typy dát v grafoch po redukcii pomocou smerodajnej odchýlky



Obr. 5.5: Typy dát v grafoch po redukcii pomocou smerodajnej odchýlky



Obr. 5.6: Typy dát v grafoch po redukcii pomocou smerodajnej odchýlky

aplikovanie algoritmu s použitím priemeru na všetkých typoch dát. Obrázok 5.3 ukazuje dátá po vyhladení oknom šírky 15 a po redukcii pomocou smerodajnej odchýlky s parametrom redukcie 1,3. Obrázok 5.4, okno šírky 15 a parameter redukcie 1,4, obrázok 5.5 okno šírky 30 a parameter redukcie 1,3 a obrázok 5.6 okno šírky 30 a parameter redukcie 1,4.

5.2 Redukcia podobná redukcii pri zrýchľovaní zvuku

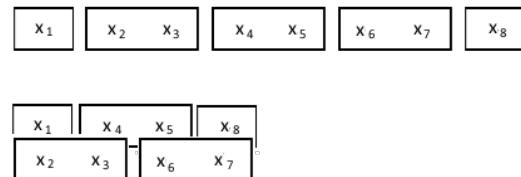
Pri zrýchľovaní zvuku je potrebné modifikovať časovú os, aby sme zachovali frekvenciu zvukového signálu. Ak by sme niečo také neurobili, pri zrýchľovaní by sa zvukový signál začal zvyšovať, naopak pri spomaľovaní znižovať. Typicky sa niečo také deje keď si chceme zrýchliť alebo spomalitiť video, zvuk sa samozrejme musí zrýchliť, resp. spomalitiť tiež a teda aj jeho frekvencia. Toto spôsobuje, že už pri malej zmene rýchlosť videa je ľudská reč nezrozumiteľná. Modifikácia časovej osi sa používa napríklad v prehrávači VLC. [zdroják VLC] V tomto prehrávači môžeme videá pozerať aj niekoľko krát rýchlejšie, či pomalšie, bez zmeny výšky zvuku. V dnešnej dobe to už dokáže viac prehrávačov, nie je to ale samozrejmostou.

Zjednodušene sa takáto modifikácia aplikuje tak, že sa signál rozdelí na rovnako dlhé úseky, ktoré sú následne posunuté po časovej osi, tak aby bola dosiahnutá požadovaná dĺžka signálu. To ako sa z prekryvajúcich úsekov dostane výsledný signál vedie k niekoľkým metódam tejto problematiky, ktoré sú popísané v [1].

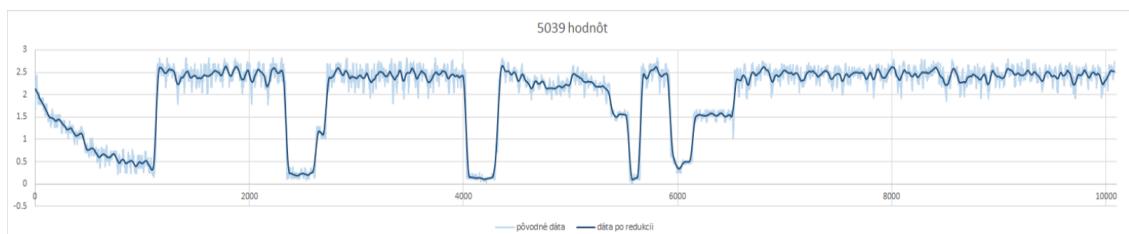
My na dátá použijeme rovnaký algoritmus, prekryté hodnoty ale budeme prieberovať. Aby sa priemerovali čo najblížšie záznamy, rozdelíme dátá na úseky o dĺžke

2 záznamy (okrem prvého a posledného, tie majú dĺžku 1 záznam). Tieto cez seba prekryjeme, toto rozdelenie a prekrytie je znázornené na obrázku 5.7. Tie záznamy, ktoré sa prekryjú priemerujeme. Takto zredukujeme dátá na polovicu. Aplikovaním rovnakej modifikácie napríklad 4-krát, môžeme dátá zredukovať na šestnásťinu.

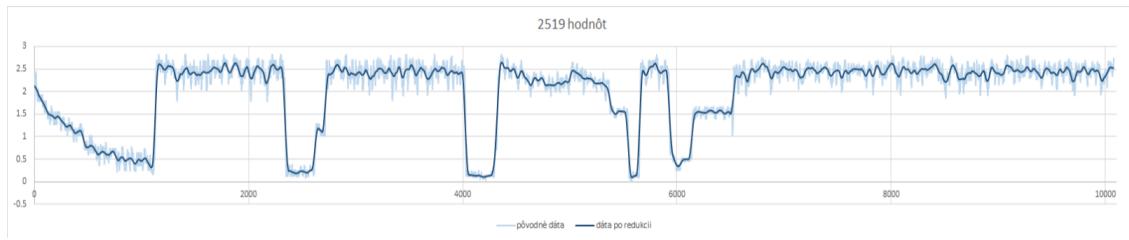
Na nasledujúcom obrázku 5.8 si ukážeme opísaný algoritmus použitý na dátá vyhľadené šírkou okna 30, a to raz, dvakrát, trikrát, štyrikrát a pätkrát.



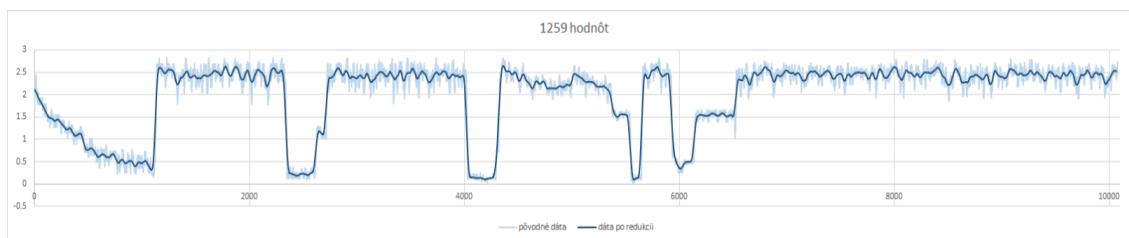
Obr. 5.7: Rozdelenie a prekrytie dát



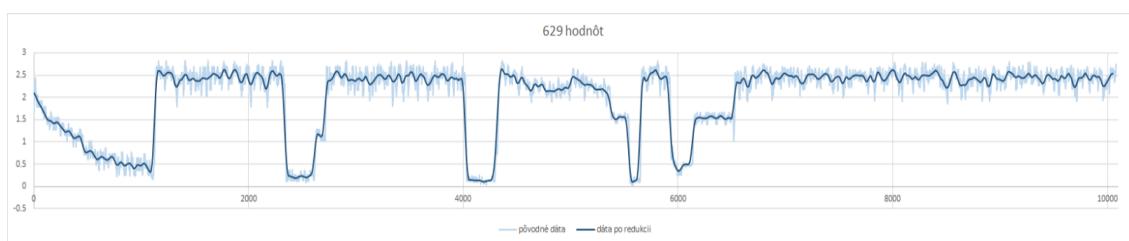
(a) Redukcia priemerovaním aplikovaná raz



(b) Redukcia priemerovaním aplikovaná dvakrát



(c) Redukcia priemerovaním aplikovaná trikrát



(d) Redukcia priemerovaním aplikovaná štyrikrát



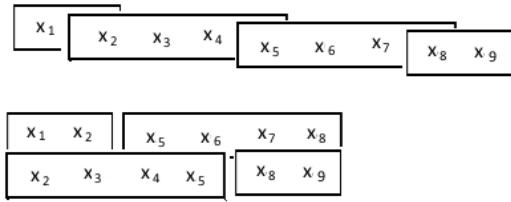
(e) Redukcia priemerovaním aplikovaná päťkrát

Obr. 5.8: Redukcia priemerovaním aplikovaná niekoľkokrát

Možná obmena opísaného redukčného algoritmu je rozdeliť dátu na úseky, ktoré by obsahovali duplicity. Napríklad v nasledujúcom úseku bude prvý záznam totožný s posledným záznamom v predchádzajúcim úseku. Takéto rozdelenie a prekrytie je zobrazené na obrázku 5.9. Tu sú dátu rozdelené na úseky o dĺžke 4 záznamy (prvý a posledný úsek iba 2 záznamy) a následne prekryté. Aj pri tejto obmene sa priemerujú iba najbližšie záznamy,

vďaka používaniu duplicit. Kvôli týmto duplicitám, ale po takejto redukcii zostávajú dve tretiny z pôvodného množstva záznamov.

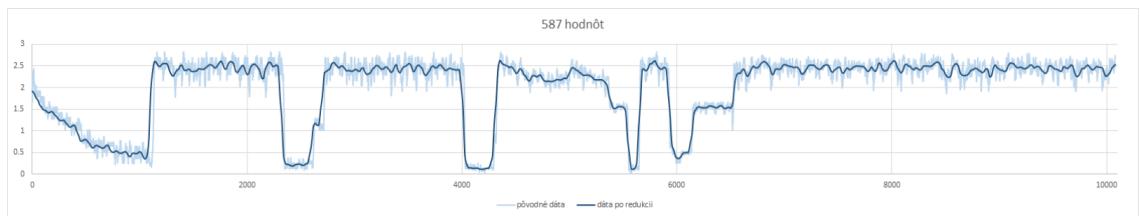
To znamená, že aby sme sa pri množstve záznamov 10079 dostali na 587 hodnôt, musíme takýto algoritmus aplikovať sedemkrát. Pôvodný algoritmus dáva 629 hodnôt po štvrtom aplikovaní, a teda je obmenený algoritmus pomalší, náročnejšie a zavádza väčšie skreslenie kvôli väčšiemu počtu priemerov. Kedže tento obmenený algoritmus dáva veľmi podobné výsledky ako pôvodný, kde sa duplicity nevyskytovali, je používanie duplicit zbytočné. Tento obmenený algoritmus použijeme na obrázku 5.10, na dátach vyhľadených oknom šírky 30 a to šesť a sedemkrát.



Obr. 5.9: Rozdelenie a prekrytie dát



(a) Redukcia priemerovaním, s použitím duplicit aplikovaná šestkrát



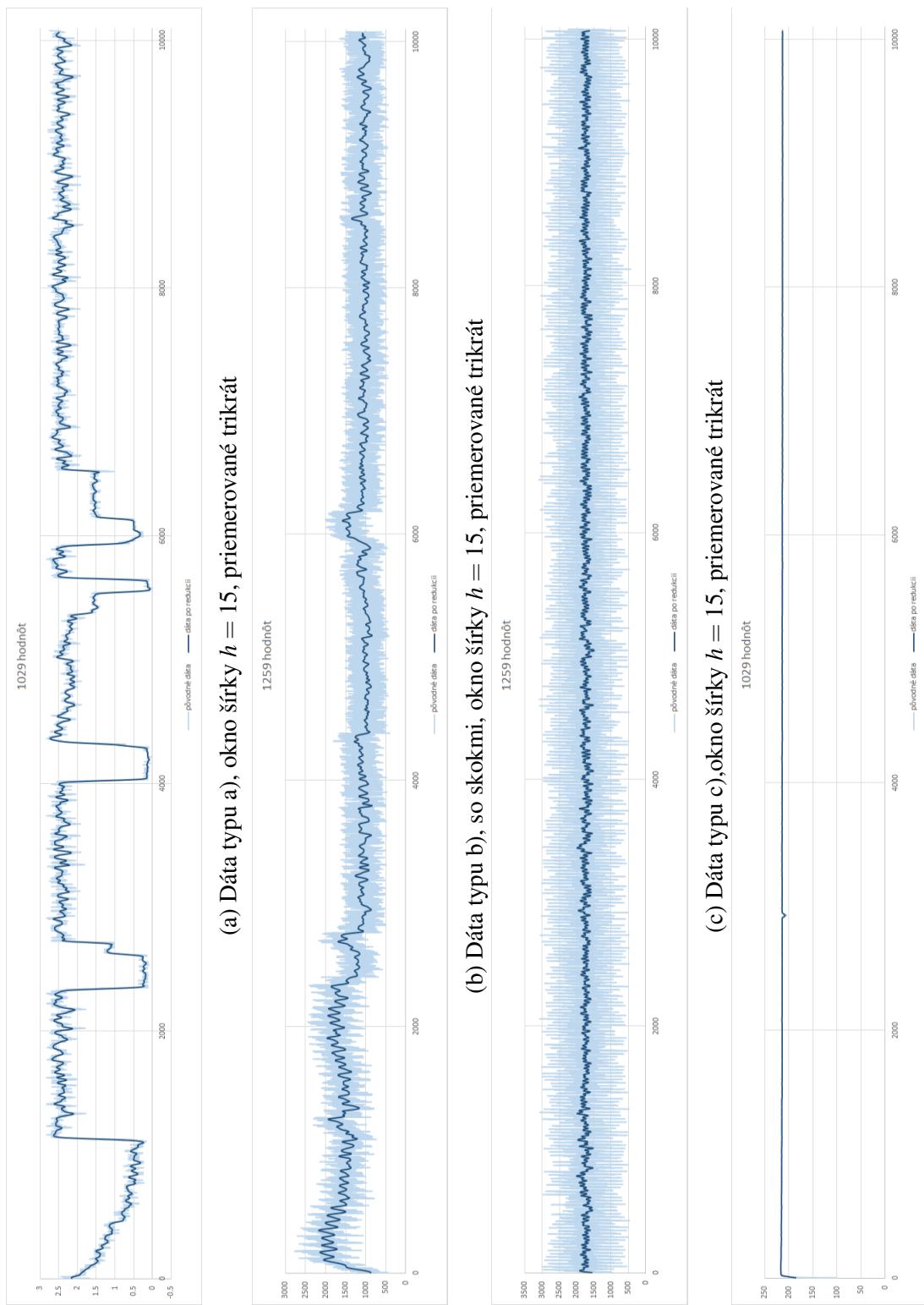
(b) Redukcia priemerovaním, s použitím duplicit aplikovaná sedemkrát

Obr. 5.10: Redukcia priemerovaním, spoužitím duplicit aplikovaná niekoľkokrát

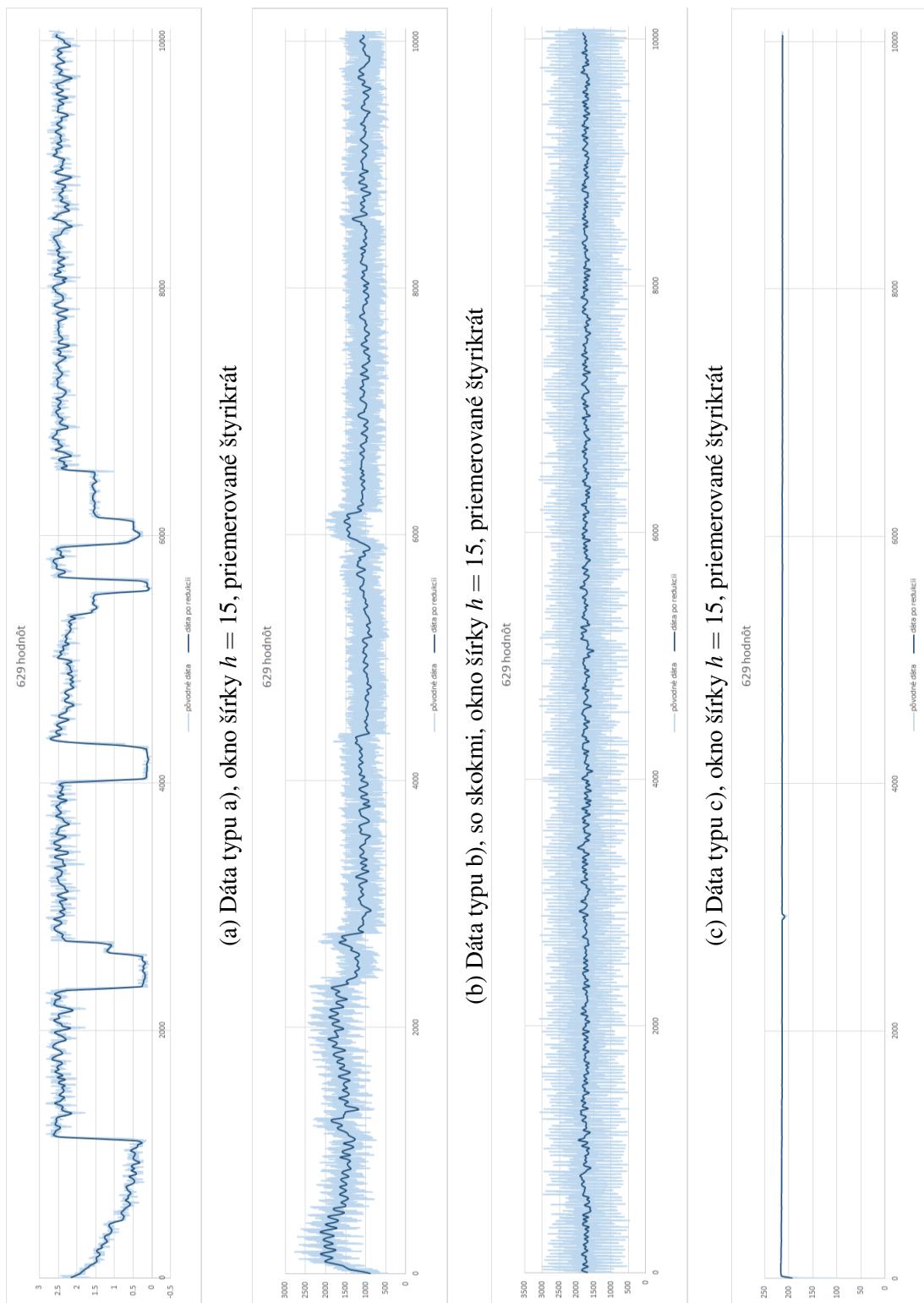
Na zvyšné typy dát použijeme iba pôvodný algoritmus, bez použitia duplicit, trikrát a štyrikrát, použité šírky okien sú 15 a 30. Výsledky sú znázornené na obrázkoch 5.11, 5.12, 5.13 a 5.14.

5.3 Redukcia pomocou jadrových odhadov

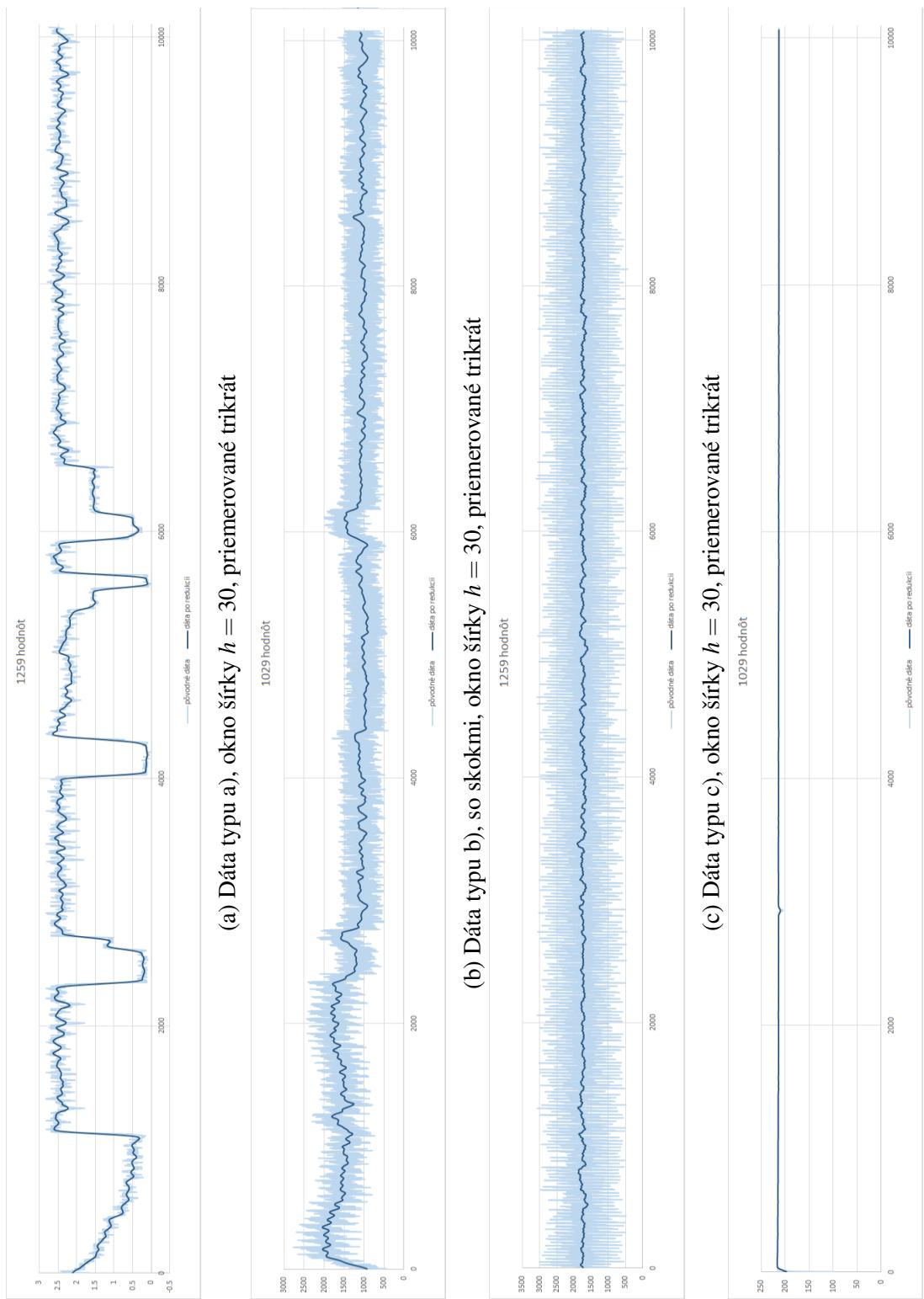
Posledný redukčný algoritmus opísaný v tejto kapitole, vychádza zo samotných jadrových odhadov. Táto redukcia spočíva v tom, že pri počítaní jadrových odhadov sa odhad spočíta



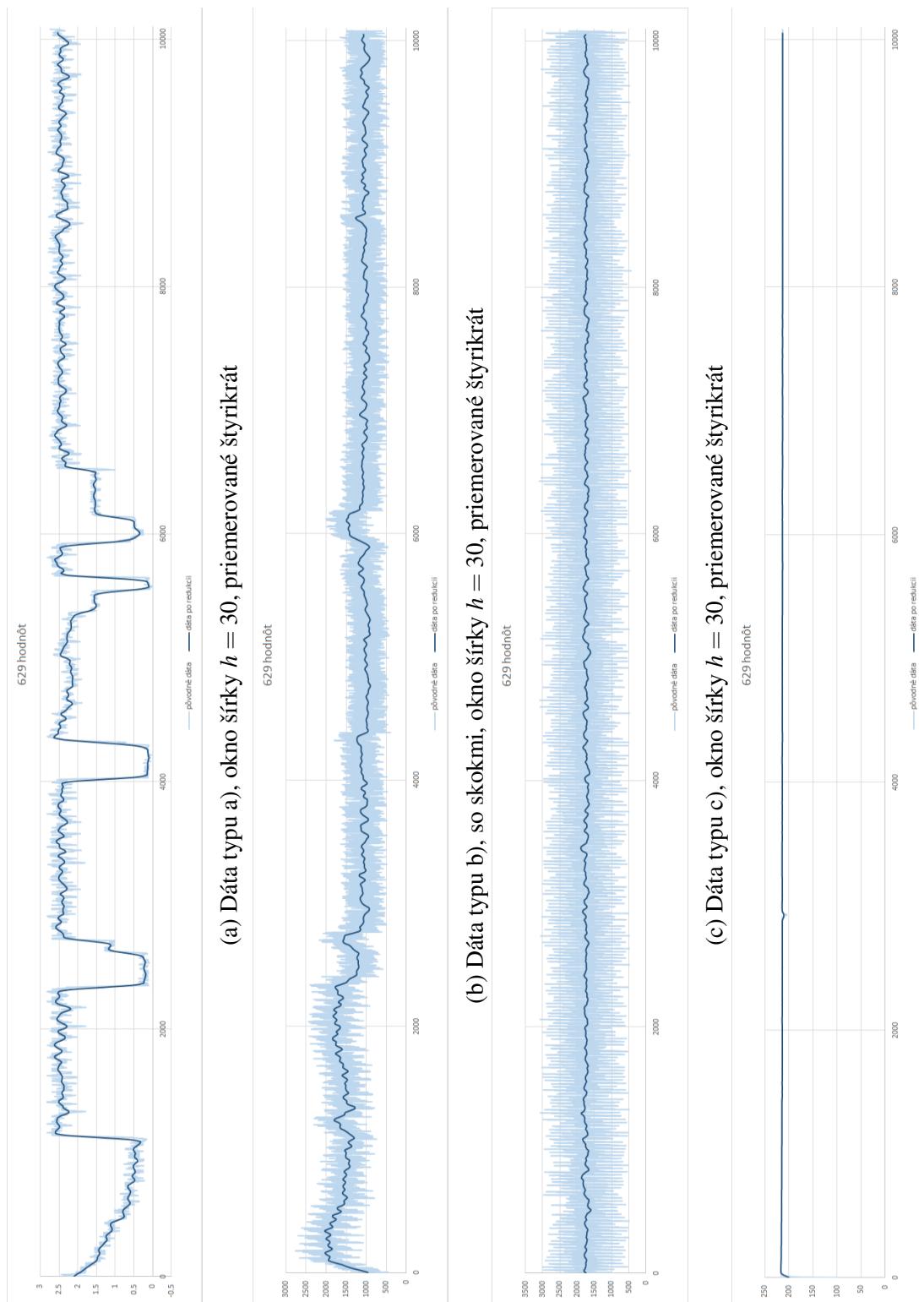
Obr. 5.11: Typy dát v grafoch po redukcii podobnej zrýchľovaniu zvuku



Obr. 5.12: Typy dát v grafoch po redukcii podobnej zrýchľovaniu zvuku



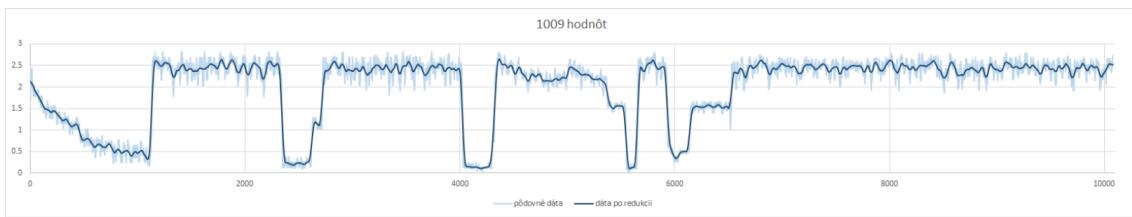
Obr. 5.13: Typy dát v grafoch po redukcii podobnej zrýchľovaniu zvuku



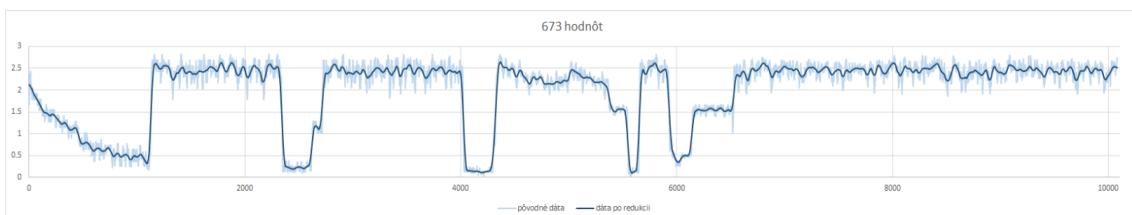
Obr. 5.14: Typy dát v grafoch po redukcii podobnej zrýchľovaniu zvuku

iba pre každý n -tý člen, odhad sa bude počítať algoritmom opísaným v kapitole 4. Konfigurovateľný parameter, ktorý bude riadiť silu redukcie, bude v tomto prípade n . V grafoch vykreslených neskôr v tejto kapitole, je vždy spočítaný odhad aj pre poslednú hodnotu. To znamená, že v prípade, že $n = 10$, sa odhad spočíta pre záznamy číslo 1, 11, 21, 31, ..., 10071, 10079. Pri takomto redukovaní sa tiež zmenšuje počet odhadov spadajúcich do intervalu s hraničnými efektmi.

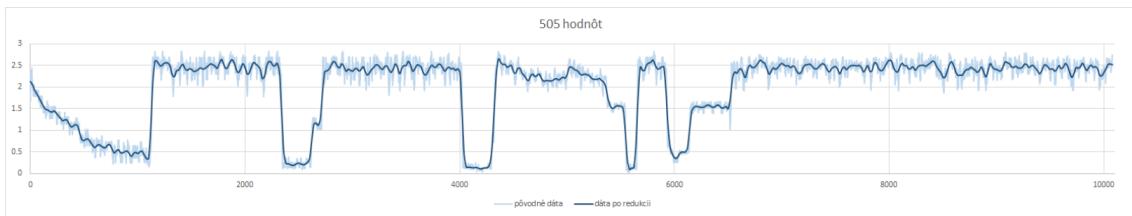
Na obrázku 5.15 si ukážeme vplyv redukčného parametra n , grafy sú vykreslené po redukcii jadrovými odhadmi so šírkou vyhladzovacieho okna 30, pre $n = 10, 15, 20, 30$, a 40.



(a) Redukcia pomocou jadrových odhadov, $n = 10$



(b) Redukcia pomocou jadrových odhadov, $n = 15$



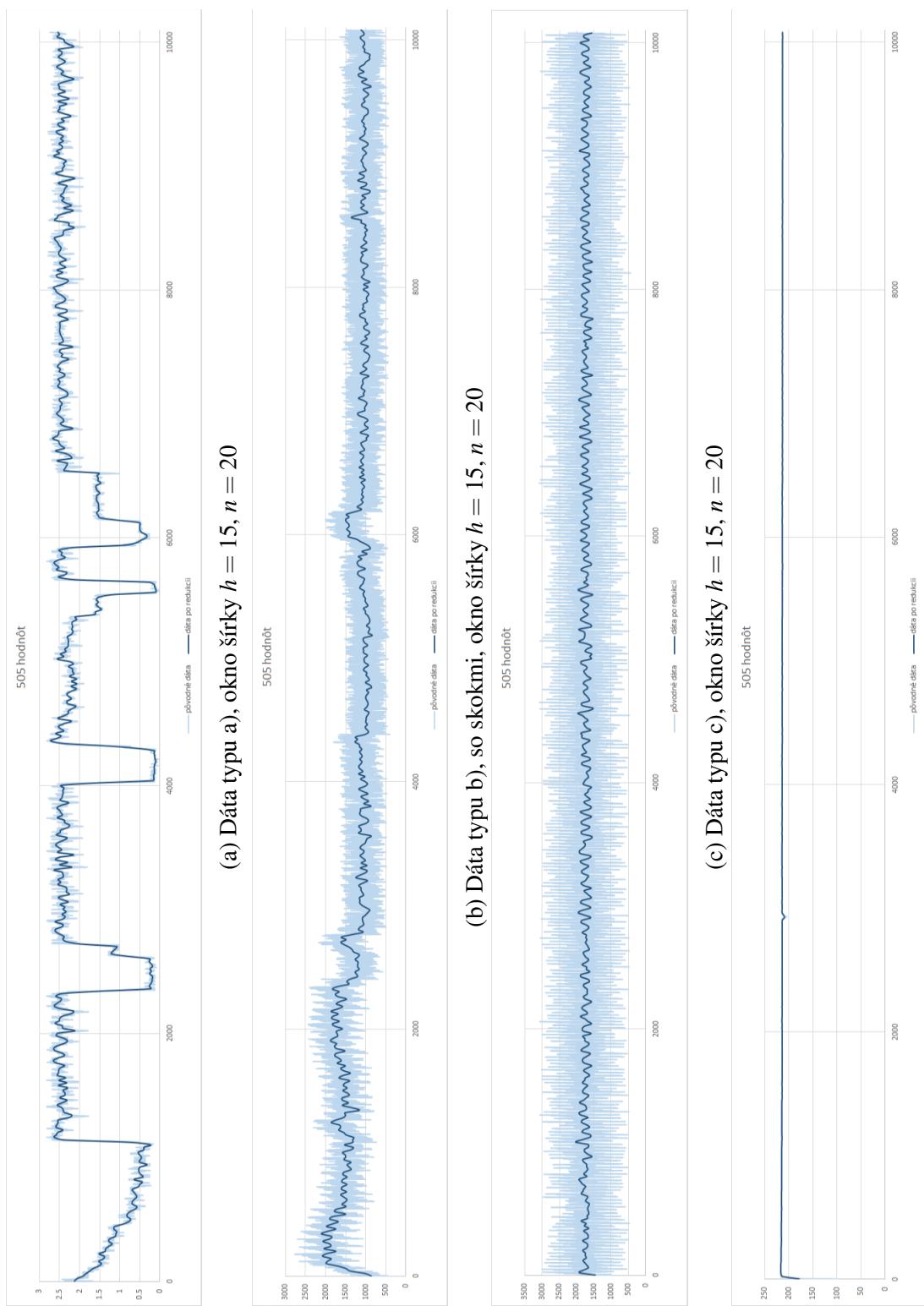
(c) Redukcia pomocou jadrových odhadov, $n = 20$



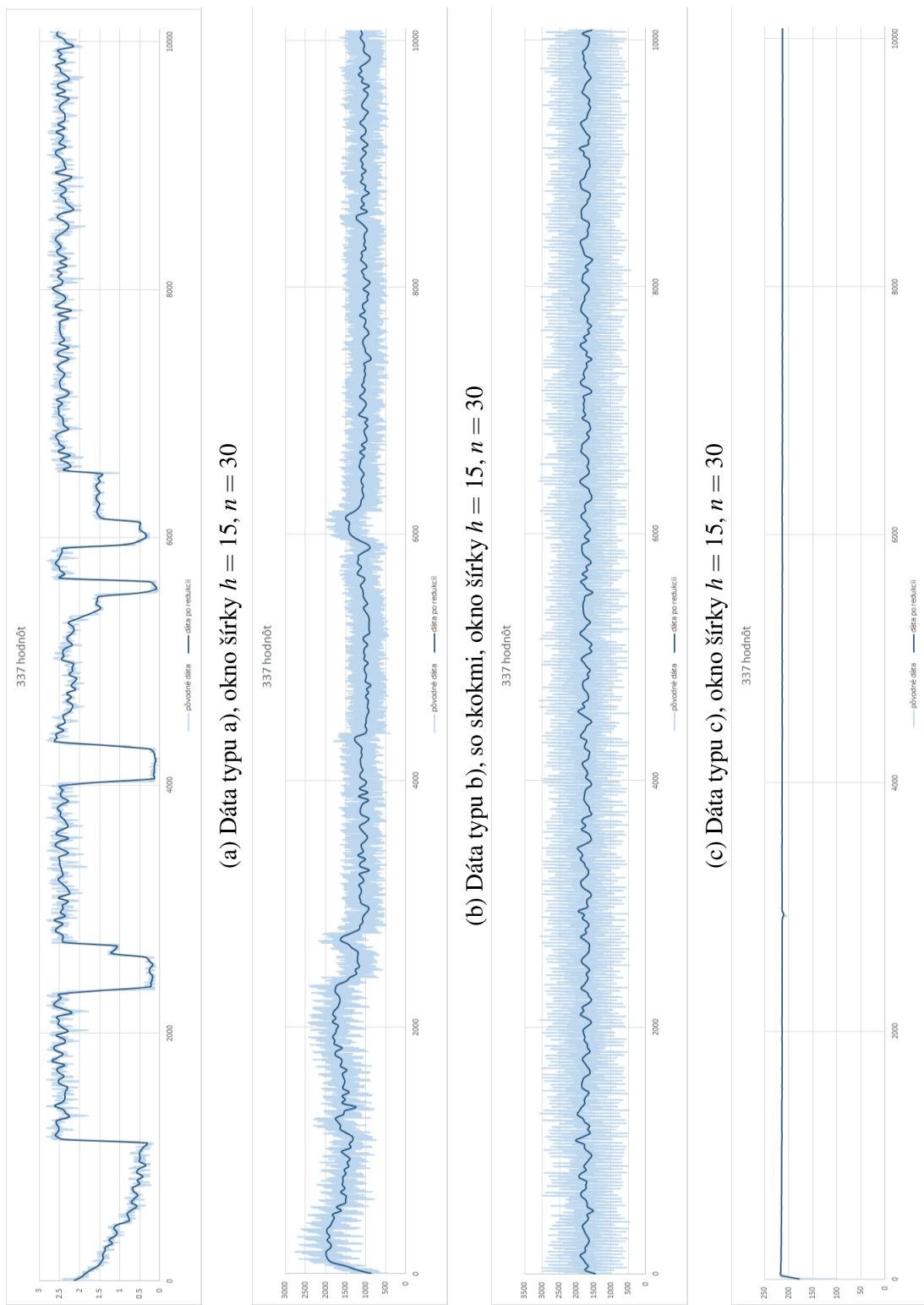
(d) Redukcia pomocou jadrových odhadov, $n = 30$

(e) Redukcia pomocou jadrových odhadov, $n = 40$ Obr. 5.15: Redukcia pomocou jadrových odhadov, pre rôzne hodnoty parametra n

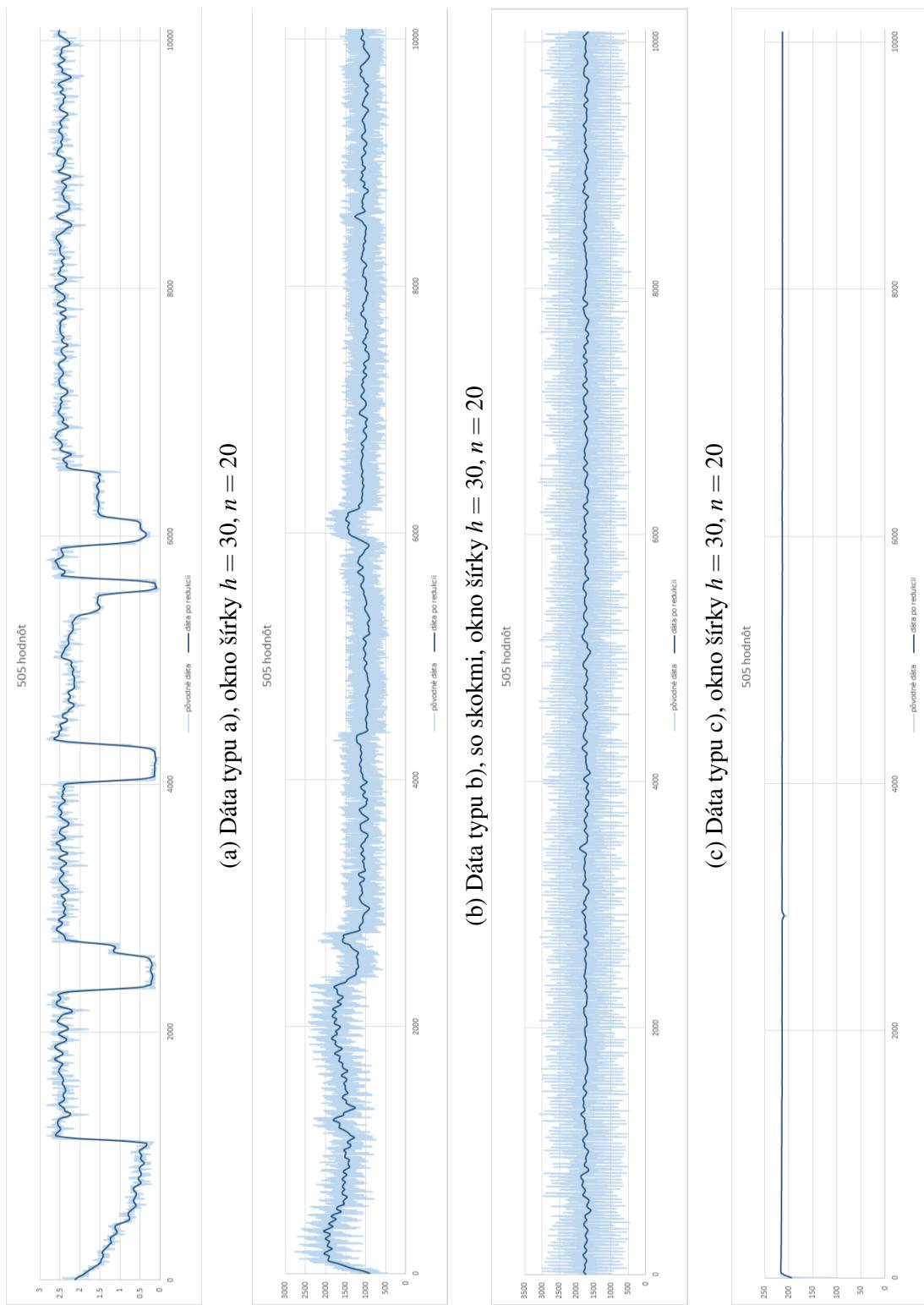
V neposlednom rade si ukážeme redukciu pomocou jadrových odhadov na všetkých typoch dát. Pre obrázok 5.16 so šírkou vyhladzovacieho okna 15 a parametrom $n = 20$, pre obrázok 5.17 šírka okna 15 a parameter $n = 30$, pre obrázok 5.18 šírka okna 30 a parameter $n = 20$, a pre obrázok 5.19 šírka okna 15 a parameter $n = 30$.



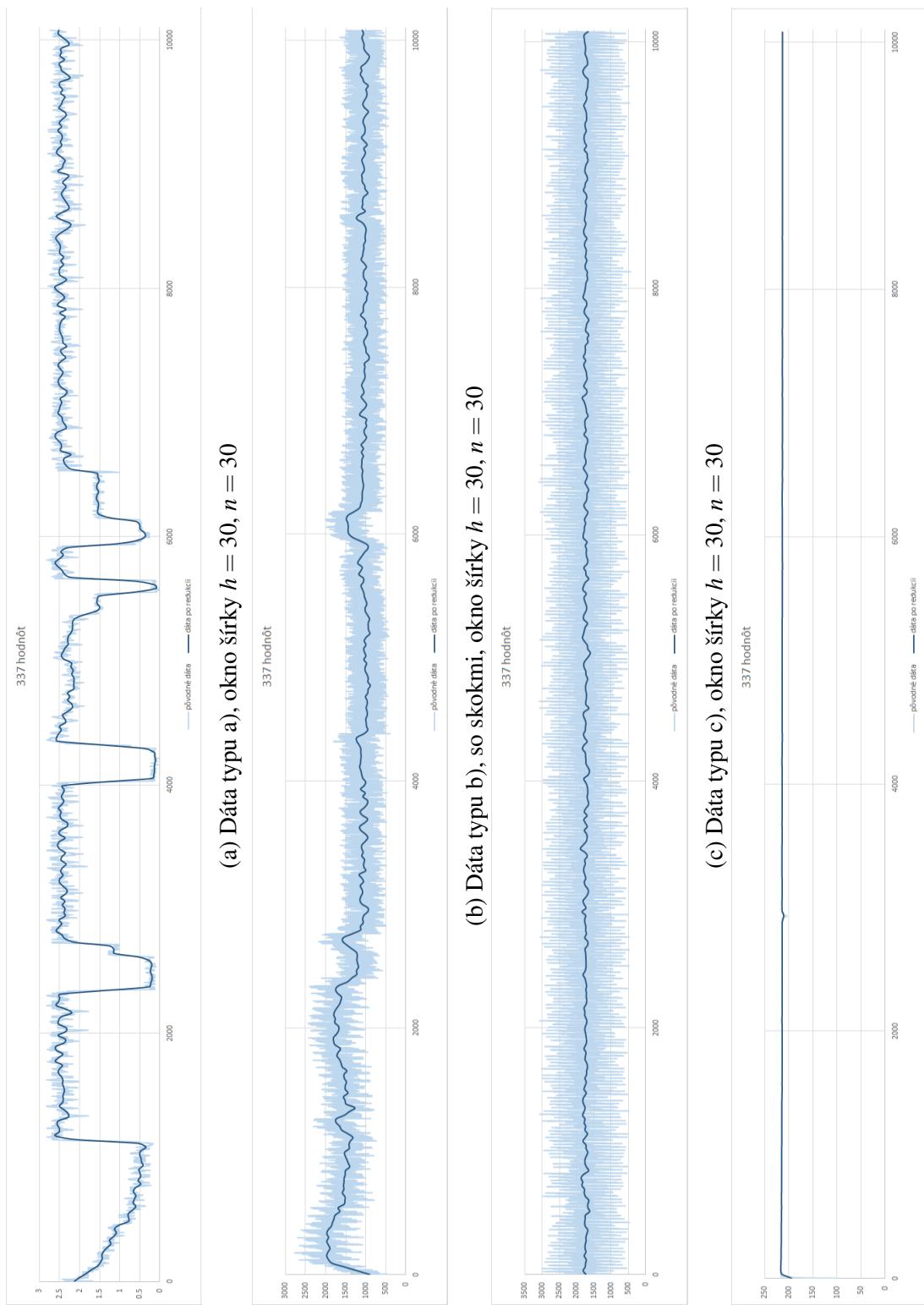
Obr. 5.16: Typy dát v grafoch po redukcii pomocou jadrov7ch odhadov.



Obr. 5.17: Typy dát v grafoch po redukcii pomocou jadrových odhadov



Obr. 5.18: Typy dát v grafoch po redukcii pomocou jadrových odhadov



Obr. 5.19: Typy dát v grafoch po redukcii pomocou jadrových odhadov

Kapitola 6

Porovnanie algoritmov

V tejto kapitole si uvedené redukčné algoritmy porovnáme z niekoľkých hľadísk. Pre najlepší algoritmus chceme aby bol rýchly, uchovával čo najlepšiu vizuálnu informáciu, primerane citlivý na redukčný parameter a aby bolo možné graf vykresľovať v priebehu behu performačných testov.

Z hľadiska rýchlosťi je jednoznačne najrýchlejší algoritmus na redukovanie pomocou jadrových odhadov (algoritmus 2). Dáta sa redukujú už pri vyhľadzovaní, čiže po vyhľadení už neprichádza žiadny výpočet, navyše odhad sa nepočítá pre každý záznam ale iba pre n -tý záznam. Algoritmus pre redukciu pomocou smerodajnej odchýlky (algoritmus 1) je mierne pomalší ako redukcia podobná zrýchľovaniu zvuku (algoritmus 3), pretože obsahuje mierne náročnejší výpočet. Tento rozdiel je ale zanedbateľný.

Porovnanie algoritmov z hľadiska uchovávania vizuálnej informácie je veľmi subjektívne a veľmi závisí na konkrétnych potrebách performačného testu. Pre porovnanie budeme teda vychádzat z grafov vykreslených v kapitole 5. Algoritmus 1 pri porovnateľných množstvách hodnôt, kopíruje pôvodné dátá najhoršie, čo môžeme vidieť hlavne pri dátach typu a). Zvyšné dva algoritmy dávajú veľmi podobné výsledky skoro na všetkých grafoch. Zo týchto vykreslených grafov však môžeme určiť, že ako najkritickejšie miesto všetkých algoritmov, sú dátá typu a), konkrétnie hodnoty v okolí 250 na ose x. Tu sa hodnoty dostávajú na chvíľu k hodnote medi 1 a 1,5 na ose y. Algoritmus 2 ešte pri redukcii na 253 hodnôt túto zmenu zachoval, obrázok 5.15 d), pričom algoritmus 3 pri redukcii na 314 hodnôt už toto miesto skoro vyhľadil, obrázok 5.8 e).

O citlivosti na redukčný parameter má zmysel baviť sa iba pri algoritme 1. Pri algoritme 2 a algoritme 3 presne vieme koľko hodnôt bude výsledkom pri zvolení konkrétnej hodnoty redukčného parametra. Pri algoritme 1 to tak ale nie je, pri určovaní hodnoty redukčného algoritmu je ľahšie odhadnúť silu redukcie a aj veľkosť stratenej informácie. Toto môžeme vidieť na obrázku 5.1, kde už zmena hodnoty o 0,2 spôsobí zmenu zo 1089 výsledných hodnôt na 292 hodnôt. Táto citlivosť je vnímaná ako negatívna vlastnosť algoritmu 1.

Posledné kritérium, ktoré budeme pri algoritmoch sledovať je možnosť vykreslovať grafy v priebehu behu performačného testu. Tu je želaným výsledkom, že v hociktorej fáze testu si užívateľ môže vykresliť graf z doterajších výsledkov. Znova je algoritmus 1 najmenej vhodný, a to z toho dôvodu, že trpí nepravidelnou distribúciou dát. Bežne sa môže stať, že na úseku nejakej dĺžky je pári hodnôt a na inom úseku tej istej dĺžky že ich niekoľko krát viac. Pri vykresľovaní sa teda môže stať, že posledná hodnota bude niekoľko

hodín stará. Pri algoritmoch 2 a 3 sú dátá rozdelené ekvidistančne a teda bude vždy dané aká je najhoršie možné oneskorenie vykresleného výsledku.

Z popisaného vyplýva, že algoritmus pomocou smerodajnej odchýlky je najmenej vhodný pre potreby nástroja PerfCake. Naopak algoritmus na redukciu pomocou jadrových odhadov sa javí ako najlepší. Je najrýchlejší, najmenej náročný na výpočet, aj pri veľkej redukcii veľmi dobre uchováva vizuálnu informáciu, jeho citlivosť na redukčný parameter je primeraná a predvídateľná a navyše vždy je jasné aké maximálne oneskorenie budú mať vykreslené grafy aj v priebehu testu. Toto oneskorenie je dané redukčným parametrom n a vyhľadzovacím parametrom h . Ak je $h > n$ tak maximálne oneskorenie je $h - (h - n)$ záznamov, naopak ak $h < n$ tak je to $n - h + h$ záznamov.

Pôvodné dátá a dátá po redukcii som sa v rámci bakalárskej práce snažila porovnať aj niektorými z metód spektrálnej analýzy, z dôvodu modifikovania osi x, ale tieto metódy nedávali žiadne použiteľné výsledky.

Závěr

Cieľom tejto bakalárskej práce bolo zoznámenie sa s nástrojom pre testovanie výkonu PerfCake a dátami, ktoré sú jeho výstupom. Následne po dôkladnej analýze nájsť vhodný redukčný algoritmus pre tieto dátu z dôvodu optimálnej vizualizácie daných dát vo forme grafov, ktorý by na základe niekoľkých vlastností mal reálne využitie.

Aby sme vedeli pochopíť a správne interpretovať dátu namerané nástrojom PerfCake, na začiatku práce sme venovali práve tomuto nástroju a hlavne jeho použitiu v oblasti testovania výkonu. Následne sme uviedli matematickú teóriu, ktorej poznatky sú dôležité pre pochopenie nájdených algoritmov. Najdôležitejšia časť tejto teórie sú jadrové odhady a vplyv šírky vyhľadzovacieho okna na ich výsledky. Práve táto šírka vyhľadzovacieho okna je použitá ako konfigurovatelný parameter výsledného algoritmu.

V tretej kapitole práce sú popísané možné typy dát, ktoré môžu byť výstupom nástroja PerfCake. Nad týmito dátami je urobená detailná analýza, ktorá bola prevedená aj na základe odborných rád konzultantov z firmy Red Hat, Mgr. Martina Večeřa a Ing. Pavla Macíka. Táto analýza určila potrebnú a nepotrebnú časť informácie nesenú v dátach a príslušných grafoch. Na základe tejto vedomosti je na dátu aplikovaná vyhľadzovacia metóda jadrových odhadov. Ako presne aplikuje algoritmus jadrové odhady na dátu je popísané v štvrtej kapitole. Na grafoch v tejto kapitole je znázornené uchovanie dôležitej informácie po aplikovaní vyhľadenia. Na konci štvrtej kapitoly sa pojednáva o riešení v špecifickom prípade, kedy by sme vyhľadením mohli stratiť časť informácie.

Samotná redukcia je popísaná v kapitole piatej, a to tromi rôznymi spôsobmi. Opisuje sa tu aplikovanie redukcii pomocou smerodajnej odchýlky, pomocou jadrových odhadov a aj redukčný algoritmus, ktorý vychádza z modifikácie časovej osi pri zrýchľovaní zvuku. Každý z algoritmov ponúka špecifický redukčný parameter, ktorý riadi silu redukcie a je použitý ako ďalší konfigurovatelný prvk výsledného optimalizačno redukčného algoritmu. Každý z opísaných algoritmov je použitý na všetky typy dát vyhľadené rôznymi šírkami vyhľadzovacieho okna a s rôznymi hodnotami redukčných parametrov, výsledky sú potom vykreslené v grafoch v rámci piatej kapitoly.

V poslednej šiestej kapitole nastáva porovnávanie nájdených redukčných algoritmov. Sledujeme rýchlosť algoritmu, jeho uchovávanie, respektíve neuchovávanie užitočnej informácie, citlivosť na redukčný parameter a možnosť vykreslovanie v priebehu behu performačných testov. Z tohto porovnania vychádza, že najlepší nájdený algoritmus je algoritmus, ktorý používa redukciu pomocou jadrových odhadov.

Algoritmus, ktorý je výsledkom tejto práce, vykazuje veľmi dobré výsledky vo všetkých sledovaných aspektoch a preto je reálne použiteľný a vhodný pre vykreslovanie výsledkov v nástroji PerfCake.

Seznam použité literatury

- [1] MOLYNEAUX, Ian. *Art of application performance testing : from strategy to tools*. 2. vydanie. Sebastopol (USA) : O'Reilly Media, 2014. 245 s. ISBN: 978-1-491-90054-3.
- [2] WAND, M.P. - JONES, M.C. *Kernel Smoothing*. 1. vydanie. London : Chapman&Hall, 1995. 208 s. ISBN 0 412 55270 1.
- [3] KONCZIOVÁ, Anita. *Neparametrické odhad regresní funkce : Bakalářská práce*. Brno : Masarykova univerzita, Přírodovědecká fakulta, 2013. 35 s. Vedoucí bakalářské práce Mgr. Jan Koláček, Ph.D.
- [4] KONCZIOVÁ, Anita. *Splajny a neparametrické odhad regresní funkce : Diplomová práce*. Brno : Masarykova univerzita, Přírodovědecká fakulta, 2015. 60 s. Vedoucí diplomové práce Mgr. Jan Koláček, Ph.D.

