Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Generovanie realizácií rovnomerného rozdelenia pravdepodobnosti na mnohorozmerných polyédroch

Bakalárska práca

2018
Slavomír Hanzely

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Generovanie realizácií rovnomerného rozdelenia pravdepodobnosti na mnohorozmerných polyédroch

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej matematiky a štatistiky
Školiteľ:	doc. Mgr. Radoslav Harman, PhD.

Bratislava, 2018
Slavomír Hanzely

55658269

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Slavomír Hanzely
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Generovanie realizácií rovnomerného rozdelenia pravdepodobnosti na mnohorozmerných polyédroch
Random sampling from the uniform distribution on multidimensional polyhedra

Anotácia: V Monte-Carlo metódach výpočtu pravdepodobností a v znáhodnených optimalizačných metódach je často potrebné generovať realizácie z rovnomerného rozdelenia na mnohorozmerných polyédroch. Tieto polyédre môžu byť zadané buď systémom konečného počtu lineárnych nerovníc (takzvaná H-reprezentácia), alebo ako konvexný obal konečnej množiny bodov (takzvaná V-reprezentácia). V prípade oboch typov reprezentácií je rovnomerné generovanie vo vnútri všeobecného polyédra netriviálna úloha, kombinujúca techniky a poznatky z matematiky, štatistiky a informatiky.

Cieľ: Cieľom bakalárskej práce je: Po prvé vypracovať prehľad existujúcich prístupov generovania realizácií z rovnomerného rozdelenia na polyédroch (priame generovanie pre špeciálne polyédre, zamietacie algoritmy, MCMC algoritmy a iné); po druhé vypracovať a programovo implementovať vlastnú metódu založenú na elipsoide najmenšieho objemu obsahujúceho zadaný polyéder.

Vedúci: doc. Mgr. Radoslav Harman, PhD.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Daniel Ševčovič, DrSc.

Dátum zadania: 14.10.2018

Dátum schválenia: 24.10.2018 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie:

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Úvod

V rámci tejto práce sa budeme zaoberať metódami na generovanie z rovnomerného rozdelenia vo veľarozmernom polyédri (konvexnom mnohostene). Rovnomernosť rozdelenia znamená, že pravdepodobnosť, že pri generovaní dostaneme bod vnútri ľubovoľnej oblasti polyédra je lineárne závislá iba od objemu danej oblasti. Predstavíme si známe algoritmy z tried Markov Chain Monte Carlo a zamietacích metód, ktoré možno použiť na rovnomerné generovanie a ako ich špeciálny prípad generovanie v polyédri.

Vo všeobecnosti možno polyéder reprezentovať viacerými spôsobmi, napríklad ako konvexný obal bodov (V–reprezentácia) alebo ako sústavu lineárnych nerovníc (H–reprezentácia). Obidve spomenuté reprezentácie možno v prípade potreby previesť na tú druhú. Prevod medzi nimi síce nie je jednoduchý, no daný výpočet je nutné spraviť len raz pred začatím generovania.

Rovnomerné generovanie bodu v polyédre je problém s prirodzeným uplatnením v praxi. Mnoho algoritmov, napríklad z triedy Monte Carlo alebo z triedy znáhodnených optimalizačných metód, je závislých na rovnomernom generovaní bodov spĺňujúcich určité požiadavky. Generovanie bodov v polyédre možno vnímať ako generovanie bodov, ktoré spĺňajú sústavu lineárnych obmedzení H–reprezentácie polyédru.

Cieľom tejto práce je jednak poskytnúť prehľad známych metód, ktoré je možné použiť na rovnomerné generovanie v polyédroch a na základe porovnania implementovať čo najefektívnejší generátor.

V prvej kapitole sa budeme zaoberať známymi metódami, ktoré možno použiť na generovanie na polyédroch. Medzi ne patria Metropolis–Hastings metódy (z triedy Markov Chain Monte Carlo), ktoré sa snažia simulovať realizácie z komplexných rozdelení vhodne konštruovanou "náhodnou prechádzkou". To možno použiť aj v našom prípade, keď je cielené rozdelenie uniformné. V triede Metropolis–Hastings metód sa špecificky zameriame na Hit–and–Run generátor a Gibbsov generátor, ktoré možno jednoducho implementovať práve pre generovanie na mnohorozmerných polyédroch. Okrem toho sa budeme zaoberať aj zamietacími metódami, ktoré namiesto generovania bodov priamo v polyédri vygeneruju bod na jednoduchšej nadmnožine polyédra rovnomerne náhodne. Po vygenerovaní bodu overia, či leží v polyédre. Ak nie, tak generujú znovu.

Druhá kapitola je venovaná problému optimálneho návrhu experimentov (optimal design problem), pomocou ktorého predstavíme algoritmus Randomized Exchange Al-

gorithm. Daný algoritmus vieme použiť aj na nájdenie elipsoidu s minimálnym objemom obaľujúci zadaný polyéder. Tento elipsoid možno jednak priamo použiť ako nadmnožinu pri zamietacej metóde, no taktiež možno použiť jeho vlastnosti využiť na zistenie natočenia polyédra v priestore a obalenie polyédra kvádrom s malým objemom.

Tretia kapitola obsahuje technické podrobnosti implementácie, výsledky porovnaní algoritmov spomenutých v prvej kapitole a zdrojový kód čo najrýchlejšieho algoritmu na generovanie bodov vnútri polyédru.

Kapitola 1

Metódy generovania vnútri polyédru

V tejto kapitole sa budeme zaoberať známymi metódami na generovanie z určitého rozdelenia, ktoré je v našom prípade konštantné vnútri polyédra a nulové mimo polyédra. V prvej podkapitole sa budeme zaoberať triedou Metropolis–Hastings algoritmov, v druhej podkapitole sa budeme zaoberať zamietacími metódami.

1.1 Metropolis–Hastings metódy

V tejto sekcii si predstavíme triedu Metropolis–Hastings algoritmov na generovanie bodov z ľubovoľného rozdelenia. Na postupnosť bodov generovnaných algoritmami z triedy Metropolis–Hastings sa dá pozerať ako na postupnosť stavov markovovského reťazcu (Markov Chain Monte Carlo, ďalej MCMC). Pri danej cieľovej hustote Q je Markovovský reťazec konštruovaný tak, aby mal jediné stacionárne rozdelenie, ktoré je totožné s Q. Pri použití vhodného nastavenia parametrov algoritmu bude rozdelenie vygenerovaných bodov s rastúcim počtom bodov konvergovať ku žiadanému stacionárnemu rozdeleniu Q.

V nasledujúcej časti stručne predstavíme všeobecný (abstraktný) Metropolis–Hastings algoritmus, podrobnejšie vysvetlenie možno nájsť napríklad v [3]. V následných častiach predstavíme Hit–and–Run generátor a Gibbsov generátor ako jeho konkrétne realizácie.

1.1.1 Všeobecný Metropolis–Hastings algoritmus

Majme cieľovú hustotu Q z ktorej chceme generovať, v prípade rovnomerného generovania vnútri polyédra je konštantná v polyédri a nulová mimo neho.

Metropolis–Hastings algoritmus [3] sa nachádza v stave $\mathbf{x}^{(i)}$ reprezentovanom bodom v polyédri, stav určuje *kandidátsku hustotu* $q(\cdot|\mathbf{x}^{(i)})$ závislú na $\mathbf{x}^{(i)}$. Táto kandidátska hustota (angl. "proposal density") je volená tak, aby z nej bolo možné jednoducho

generovať ďalšie body. Môže byť značne odlišná od cieľovej hustoty Q, avšak je nutné, aby limitné rozdelenie vygenerovaných bodov konvergovalo ku Q.

Algoritmus postupuje iteratívne, v jednom kroku vygeneruje ďalší potenciálny stav $\mathbf{y}$ podľa hustoty $q(\cdot|\mathbf{x}^{(i)})$. Ďalší stav algoritmu $\mathbf{x}^{(i+1)}$ bude $\mathbf{y}$ s pravdepodobnosťou $\alpha(\mathbf{y}|\mathbf{x}^{(i)})$ (algoritmus sa "pohne"), inak to bude $\mathbf{x}^{(i)}$ (algoritmus "ostane stáť"). Funkcia α sa označuje ako *pomer akceptovania* (angl. "acceptance ratio"), je definované ako

$$\alpha(\mathbf{y}|\mathbf{x}^{(i)}) = \min(\frac{Q(\mathbf{y}|\mathbf{x})}{q(\mathbf{y}|\mathbf{x})}, 1).$$

Pri tejto definícii platí

$$Q(\mathbf{y}|\mathbf{x}) = q(\mathbf{y}|\mathbf{x})\alpha(\mathbf{y}|\mathbf{x}).$$

Pravdepodobnosť $\alpha(\mathbf{y}|\mathbf{x}^{(i)})$ môže byť vo všeobecnosti zložitá. V prípade, že je hustota Q rovnomerná a postupnosť stavov markovovského reťazca časovo reverzibilná, t.j.

$$q(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) = q(\mathbf{x}^{(i)}|\mathbf{x}^{(i+1)}),$$

tak je daná pravdepodobnosť pohybu konštante jedna, $\alpha(\mathbf{y}|\mathbf{x}^{(i)}) = 1$. Tieto predpoklady budú splnené pri ďalej spomínaných algoritmoch (Hit–and–Run generátore a Gibbsovom generátore). Teda naše MCMC metódy, ktoré budeme používať pre generovanie z rovnomerného rozdelenia, sa v každom kroku "pohnú", na rozdiel od všeobecných MCMC generátorov, ktoré často "stoja", lebo neakceptujú kandidátov $\mathbf{y}$ na prechod.

Algorithm 1 Všeobecný Metropolis–Hastings algoritmus [3]

1: inicializuj $\mathbf{x}^{(0)}$
2: **for** $i = 0, 1, \ldots, N-1$ **do**
3: Vygeneruj bod $\mathbf{y}$ z $q(\cdot|\mathbf{x}^{(i)})$
4: Vygeneruj u z $U(0,1)$.
5: **if** $u \leq \alpha(\mathbf{y}|\mathbf{x}^{(i)})$ **then**
6: Nastav $\mathbf{x}^{(i+1)} = \mathbf{y}$
7: **else**
8: Nastav $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$
9: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$.

Môžeme si všimnúť, že v Metropolis–Hastings algoritmoch, je hustota bodu $\mathbf{x}^{(i)}$ závislá od predchádzajúceho bodu $\mathbf{x}^{(i-1)}$. Podľa [3] je pri vhodnej voľbe kandidátskej hustoty $q(\cdot|\mathbf{x}^{(i)})$ a pravdepodobnosti α možné dokázať, že napriek závislosti po sebe idúcich bodov je pre $N \to \infty$ limitné rozdelenie náhodneho vektora $\mathbf{x}^{(N)}$ rovné Q. Potrebná veľkosť N na dosiahnutie dostatočne presného odhadu hustoty Q sa nazýva burn–in period.

V ďalších častiach si ukážeme niekoľko konkrétnych realizácii Metropolis–Hastings algoritmu. Každá z tých metód obsahuje určité predpoklady na distribúciu, z ktorej chceme generovať, no dá použiť aj na rovnomerné generovanie bodov v polyédri.

1.1.2 Hit–and–Run generátor

Ako jedna z možností na realizáciu Metropolis–Hastings algoritmu prichádza do úvahy Hit–and–Run generátor. Algoritmus je analogický s algoritmom Metropolis–Hasting.

Označme si S zadaný polyéder. Kandidátska hustota $q(\cdot|\mathbf{x}^{(i)})$ je určená priamkou $\mathbf{d}_i$ s náhodným smerom cez bod $\mathbf{x}^{(i)}$. Hustota $q(\cdot|\mathbf{x}^{(i)})$ je konštantná na úsečke $S \cap \mathbf{d}_i$ a nulová inde. Pri danej hustote je markovovský reťazec časovo reverzibilný, t.j. $q(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) = q(\mathbf{x}^{(i)}|\mathbf{x}^{(i+1)})$, [2] preto je funkcia α konštantne 1. Algoritmus sa teda každým krokom pohne do ďalšieho bodu.

Hit–and–Run generátor funguje nasledovne:

Algorithm 2 Hit–and–Run generátor [2],[8]

1: Inicializuj $\mathbf{x}^{(0)}$
2: **for** $i = 0, \ldots, N-1$ **do**
3: Vygeneruj smer $\mathbf{d}_i$ z distribúcie D na povrchu sféry
4: Nájdi množinu $S_i(\mathbf{d}_i, \mathbf{x}^{(i)}) = \{\lambda \in \mathbb{R}; \mathbf{x}^{(i)} + \lambda \mathbf{d}_i \in S\}$
5: Vygeneruj $\lambda_i \in S_i$ podľa hustoty $q(\lambda|\mathbf{d}_i, \mathbf{x}^{(i)})$
6: Nastav $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \lambda_i \mathbf{d}_i$
7: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$.

Použiteľnosť Hit–and–Run generátora závisí od toho, ako rýchlo vieme generovať smery $\mathbf{d}_i$ z rozdelenia D. Ak by dimenzia priestoru bola príliš veľká, nebolo by možné generovať rýchlo z rozdelenia D a preto celý algoritmus by bol pomalý.

Na rovnomerné vygenerovanie d–rozmerného smerového vektoru vygenerujeme bod v d–rozmernom rotačne symetrickom rozdelení a následne ho zobrazíme z počiatku sústavy na jednotkovú sféru. Takto dostaneme rovnomerné rozdelenie na d–rozmernej sfére. Takto zložitosť vygenerovania smeru rastie len asymptoticky lineárne s dimenziou, preto aj rýchlosť generovania bodov rastie len lineárne s dimenziou.

Podľa [5] ako d–rozmerné rotačné symetrické rozdelenie možno zvoliť mnohorozmerné normálne rozdelenie so zložkami z nezávislých jednorozmerných normálnych rozdelení s priemerom 0 a výchylkou 1.

1.1.3 Gibbsov generátor

V tejto podsekcii sa budeme zaoberať Gibbsovým generátorom, metódou generovania z triedy MCMC vhodnou na generovanie vo viacrozmernom priestore. Našou úlohou je generovať z d–rozmernej distribúcie Q, pričom z cielenej hustoty Q nevieme generovať priamo. Predpokladajme, že nevieme použiť Hit–and–Run generátor, lebo

kandidátska hustota $q(\cdot|\mathbf{x}^{(i)}) = q(\cdot|(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}))$ je kvôli veľkému rozmeru priestoru príliš zložitá na generovanie. Pred podrobným popísaním vlastností algoritmu si najprv ukážme, ako Gibbsov generátor funguje.

Gibbsov generátor určí kandidátsku hustotu $q(\cdot|\mathbf{x}^{(i)})$ tak, že bude možné generovať z $q(\cdot|\mathbf{x}^{(i)})$ po súradniciach. Gibbsov generátor bude generovať bod $\mathbf{x}^{(i+1)} = (x_1^{(i+1)}, x_2^{(i+1)}, \ldots, x_d^{(i+1)})$ postupne po súradniciach, j–tú súradnicu $x_j^{(i+1)}$ vygeneruje z kandidátskej hustoty

$$q(x_j^{(i+1)}|x_1^{(i+1)}, x_2^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, x_{j+2}^{(i)}, \ldots, x_d^{(i)}).$$

Označme si cieľový polyéder S a $\mathbf{d}_{i,j}$ priamku rovnobežnú s j–tou osou prechádzajúcou cez bod $(x_1^{(i+1)}, \ldots, x_j^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_d^{(i)})$. V prípade rovnomerného generovania na polyédroch je pre Gibbsov generátor kandidátska hustota $q(x_j^{(i+1)}|x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_d^{(i)})$ konštantná na úsečke $\mathbf{d}_{i,j} \cap S$ a nulová inde.

Gibbsov generátor funguje nasledovne:

Algorithm 3 Gibbsov generátor [9]

1: inicializuj $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_d^{(0)})$
2: **for** $i = 0, \ldots, N-1$ **do**
3: **for** $j = 0, 1, \ldots, n$ **do**
4: $x_j^{(i)} \sim q(x_j^{(i+1)}|x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_d^{(i)})$
5: $\mathbf{x}^{(i+1)} = (x_1^{(i+1)}, x_2^{(i+1)} \ldots, x_d^{(i+1)})$
6: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$.

Gibbsov generátor predpokladá, že možno rýchlo generovať jednotlivé súradnice z rozdelenia $q(x_j^{(i+1)}|x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_d^{(i)})$. V prípade rovnomerného generovania v polyédri je daný predpoklad splnený, vďaka linearite nerovníc pri H–reprezentácii polyédra možno ľahko generovať na úsečke $\mathbf{d}_{i,j} \cap S$. Hraničné body úsečky vieme zo systému nerovností $Ax \geq b$ vypočítať priamo bez lineárneho programovania, čiže generovať z týchto rozdelení je obzvlášť jednoduché.

Ako špeciálny prípad Metropolis–Hastings algoritmu má Gibbsov generátor podobné vlastnosti ako Metropolis–Hastings algoritmus. Jeho hlavnou výhodou je, že je jednoduchý a neobsahuje žiadne parametre. Môžeme si všimnúť, že generovanie bodu z $q(\cdot|\mathbf{x})$ trvá lineárne dlho od rozmeru priestoru, preto pri zväčšovaní počtu rozmerov priestoru rastie čas potrebný na vygenerovanie bodu asymptoticky kvadraticky. Preto očakávame, že v porovnaní s Hit–and–Run generátorom bude Gibbsov generátor pri vyššom počte rozmerov pomalší.

1.2 Zamietacie metódy

Zamietacie metódy nám poskytujú ďalší zo spôsobov rovnomerného generovania bodov na určitej množine. Myšlienka za nimi je nasledovná: Označme si S množinu, na ktorej chceme rovnomerne náhodne generovať prvky. Predpokladajme, že nevieme priamo rovnomerne generovať body na S (S je veľarozmerná alebo komplikovane zadaná), no vieme rovnomerne generovať na množine T, $S \subset T$.

Náš generátor G_T bude pracovať nasledovne:

Algorithm 4 Zamietacia metóda

1: **for** $i = 1, \ldots, N$ **do**
2: **repeat**
3: Vygeneruj bod $\mathbf{x}^{(i)} \in T$ rovnomerne náhodne
4: **until** $\mathbf{x}^{(i)} \in S$
5: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$

Generátor G_T vygeneruje bod $\mathbf{x}$ na množine T rovnomerne náhodne, ak je ten bod aj z S, tak ho vráti ako výstup, inak vygeneruje nový bod $\mathbf{x} \in T$. Všimnime si, že generátor G_T je závislý iba od T a že generuje body na $S \cap T = S$ rovnomerne náhodne.

Označme si $\lambda(M)$ objem množiny M. Očakávaná rýchlosť generovania závisí od toho, koľkokrát G_T vygeneruje bod z $T \setminus S$. Z rovnomernosti G_T je tá pravdepodobnosť rovná $\frac{\lambda(T \setminus S)}{\lambda(T)} = 1 - \frac{\lambda(S)}{\lambda(T)}$. Označme si p_k pravdepodobnosť, že G_T vygeneruje bod z S na k–ty pokus, t.j. najprv $k-1$ krát vygeneruje bod z$T \setminus S$ a potom vygeruje bod z S. Platí $p_k = (1 - \frac{\lambda(S)}{\lambda(T)})^{k-1} \frac{\lambda(S)}{\lambda(T)}$, preto p_k patrí geometrickému rozdeleniu pravdepodobnosti. Očakávý počet generovaní G_T je $E(G_T) = \sum_0^\infty k p_k = \frac{\lambda(S)}{\lambda(T)} \sum_0^\infty k(1 - \frac{\lambda(S)}{\lambda(T)})^{k-1} = \frac{\lambda(S)}{\lambda(T)} \frac{1}{((1-\frac{\lambda(S)}{\lambda(T)})-1)^2} = \frac{\lambda(T)}{\lambda(S)}$.

Táto metóda generovania je vhodná, ak je $\frac{\lambda(T)}{\lambda(S)}$ dostatočne malé, t.j. ak je obal T relatíve malý oproti polyédru S. Ak je $\frac{\lambda(T)}{\lambda(S)} \sim \infty$, tak je táto metóda nepoužiteľná.

Navyše, ak poznáme objem T, tak táto metóda nám ako vedľajší produkt poskytne aj štatistické intervaly spoľahlivosti I_S pre objem S. Totiž $\frac{\lambda(S)}{\lambda(T)}$ je presne parameter binomického rozdelenia náhodnej premennej S, ktorá zodpovedá vygenerovaní bodu z S. Preto možno odhadnúť objem S ako $\lambda(S) = \lambda(T) I_S$.

1.2.1 Použitie na generovanie bodu vnútri polyédru

Zamyslime sa nad tým, ako by sme vedeli použiť túto metódu na generovanie bodu vnútri polyédru. Ako množinu možných T, $S \subset T$ môžeme použiť najmenší kváder

so stranami rovnobežnými s osami. Vypočítať súradnice kvádra je ľahké, stačí nám to spraviť raz pred (začatím generovania) pomocou lineárneho programovania.

Žiaľ, pre takúto množinu T môže byť podiel $\frac{\lambda(T)}{\lambda(S)}$ byť ľubovoľne veľký. Ako príklad na takú množinu S uveďme kváder s obsahom k, ktorý "leží pozdĺž diagonály" kocky $[0,1]^n$ a dotýka sa každej steny kocky $[0,1]^n$. Zrejme najmenšia množina T (kváder so stranami rovnobežnými s osami) obaľujúca S je kocka $[0,1]^n$, ktorá má obsah 1. Platí $\frac{\lambda(T)}{\lambda(S)} = \frac{1}{k}$. Keďže vieme nájsť polyéder (napr. kváder) taký, že sa dotýka stien kocky $[0,1]^n$ a k je ľubovoľne malé, tak očakávaná dĺžka generovania touto metódou (pre danú množinu T) vie byť ľubovovoľne veľká.

Ako ďalšia možná množina T prichádza do úvady elipsoid obaľujúci polyéder. Keďže chceme, aby bol podiel $\frac{\lambda(T_{MVEE})}{\lambda(S)}$ čo najmenší, budeme skúmať elipsoid s najmenším obsahom obaľujúci polyéder — Minumum Volume Enclosing Elipsoid (ďalej MVEE). Môžeme si všimnúť, že MVEE elipsoid obsahuje veľa informácie o tom, ako vyzerá polyéder. Keďže elipsoid je jednotková guľa zobrazená lineárnou transformáciou, možno generovať body vnútri elipsoidu ako obrazy bodov vygenerovaných vnútri jednotkovej gule v lineárnej trasformácii. Nájsť daný elipsoid rýchlo vieme pomocou REX algoritmu, ktorému je venovaná ďalšia kapitola.

Pre túto metódu vieme dokonca odhadnúť rýchlosť generovania. Na odhadnutie veľkosti $\lambda(T_{MVEE})$ možno využiť fakt, že ku každému konvexnému telesu C existuje (unikátny) vpísaný elipsoid s najväčším objemom (takzvaný Johnov elipsoid). Podľa [1] Johnov elipsoid pre konvexné d–rozmerné teleso C (v našom prípade polyéder S) zobrazený rovnoľahlosťou so stredom s centre elipsoidu a koeficientom d obsahuje celé teleso C. Zobrazením danou rovnoľahlosťou sa objem d–rozmerného telesa zväčší d^d–krát. Tým pádom je objem MVEE nanajvýš d^d–krát väčší ako objem Johnovho elipsoidu, preto je nanajvýš d^d–krát väčší ako objem telesa C.

$$\frac{\lambda(T_{MVEE})}{\lambda(S)} \leq \frac{d^d \lambda(T_{JE})}{\lambda(S)} < \frac{d^d \lambda(S)}{\lambda(S)} = d^d,$$

kde T_{JE} je Johnov elipsoid prislúchajúci ku polyédru S. Očakávaný počet generovaní pomocou nadmnožiny T_{MVEE} je teda najviac d^d (kde d je počet rozmerov).

Poďme sa zamyslieť nad inými množinami T. Keďže najjednoduchšia množina T, v ktorej vieme generovať, je kváder, ako ďalšiu uvažovanú množinu T možno zvážiť najmenší kváder obaľujúci MVEE elipsoid (bez podmienky, že jeho strany sú rovnobežné s osami sústavy). Zrejme osi kvádra budú zhodné s osami MVEE elipsoidu. Označme si daný kváder T_K a metódu generovania založenú na T_K *kvádrova metóda*. Pre T_K platí, že je obrazom kocky $[0,1]^n$ v zobrazení, ktoré zobrazí jednotkovú guľu na MVEE elipsoid.

Avšak, keďže tento kváder T_K nie je rovnobežný s osami sústavy, ako prirodzený spôsob rovnomerného generovania vnútri tohoto kvádra možno použiť generovanie vnútri jednotkovej hyperkocky s osami rovnobežnými s osami sústavy a následne zobrazené lineárnou transformáciou.

Tento prístup je analogický prístupu s MVEE elipsoidom. Dokonca, lineárne zobrazenia pri T_K a T_{MVEE} sú rovnaké. Rozdiel je jedine v tom, že pri T_{MVEE} rovnomerne generujeme na d–rozmernej jednotkovej guli (kde d je dimenzia priestoru), čo následne zobrazujeme lineárnym zobrazením. Pri T_K rovnomerne generujeme na d–rozmernej jednotkovej kocke (obaľujúcej d–rozmernú jednotkovú guľu), čo zobrazujeme lineárnym zobrazením na kváder T_K obaľujúci MVEE elipsoid.

Ukážme si, že pri dostatočne rýchlom rovnomernom generovaní bodov v guli kvádrova metóda nie je rýchlejšia ako MVEE metóda. Porovnajme očakávaný čas týchto prístupov. Označme si t_f očakávaný čas výpočtu lineárneho zobrazenia, T_G d–rozmernú jednotkovú guľu a t_G očakávaný čas vygenerovania bodu v nej, T_K d–rozmernú jednotkovú kocku a t_K očakávané čas vygenerovanie bodu v nej. Nakoniec si označme T_P množinu bodov hľadaného polyédra.

Očakávaný čas na vygenerovanie bodu pomocou zamietacej metódy pomocou MVEE je $\frac{\lambda(T_G)}{\lambda(T_P)} t_f t_G = \frac{t_f}{\lambda(T_P)} t_G \lambda(T_G)$, očakávný čas na vygenerovanie pomocou kvádrovej metódy je $\frac{\lambda(T_K)}{\lambda(T_P)} t_f t_K = \frac{t_f}{\lambda(T_P)} t_K \lambda(T_K)$. Podiel očakávaných časov je

$$\frac{\text{očakávaný čas kvádrovej metódy}}{\text{očakávaný čas MVEE metódy}} = \frac{\frac{\lambda(T_K)}{\lambda(T_P)} t_f t_K}{\frac{\lambda(T_G)}{\lambda(T_P)} t_f t_G} = \frac{\lambda(T_K) t_K}{\lambda(T_G) t_G}.$$

Ak pri MVEE metóde použijeme na rovnomerné generovanie bodov vnútri gule s očakávaným časom $t_G \leq \frac{\lambda(T_K) t_K}{\lambda(T_G)}$, tak podiel očakávaných trvaní metód bude menší–rovný ako jedna, teda kvádrová metóda bude pomalšia.

Všimnime si, že zamietacia metóda na generovanie bodov v T_G pomocou nadmnožiny T_K má očakávaný počet generovaní rovný $\frac{\lambda(T_K) t_K}{\lambda(T_G)}$, preto očakávané trvanie vygenerovania bodu pomocou nej je $\frac{\lambda(T_K) t_K}{\lambda(T_G)}$. Týmto sme ukázali, že ak by sme pri generovali body v polyédri pomocou MVEE metódy, pričom rovnomerné generovanie bodov v T_G by sme realizovali pomocou zamietacej metódy s nadmnožinou T_K, dostali by sme presne rýchlosť kvádrovej metódy. Kedže existujú aj rýchlejšie metódy rovnomerného generovania vnútri gule, kvádrová metóda je pomalšia ako MVEE metóda. Ďalej sa ňou nebudeme zaoberať.

Treba podotknúť, že zamietacia metóda na generovanie bodov v T_G pomocou T_K v d rozmeroch má zásadný problém, že pomer objemov $\frac{\lambda(T_K)}{\lambda(T_G)}$ je už pre malé d obrovský. Totiž v d–rozmeroch je $\lambda(T_K) = 2^d$ a $\lambda(T_G) = \Gamma(\frac{d}{2} + 1)$ (kde Γ je Eulerová Gamma

funkcia), preto

$$\frac{\lambda(T_K)}{\lambda(T_G)} = \frac{2^d}{\Gamma(\frac{d}{2}+1)} \sim \frac{2^d}{\frac{d+1}{2}!} \sim \frac{2^d}{\sqrt{(d+1)\pi}(\frac{d+1}{2e})^{(d+1)/2}} = \frac{2^d(2e)^{(d+1)/2}}{\sqrt{(d+1)\pi}(d+1)^{(d+1)/2}}.$$

V čítateli je c^d (pre vhodnú konštantu c), menovateľ je rádovo $d^{d/2}$, čo rastie so zväčšujúcou sa dimenziou priestoru asymptoticky oveľa rýchlejšie ako čítateľ.

1.3 Rovnomerné generovanie bodov v polyédri pomocou MVEE elipsoidu

V tejto podkapitole sa budeme bližšie zaoberať MVEE metódou, ako pomocou už vypočítaného MVEE elipsoidu možno rovnomerne generovať body vnútri zadaného polyédra. Budeme používať zamietaciu metódu, vygerujeme bod v elipsoide a overíme, či je daný bod tiež v polyédre.

Nakoľko každý elipsoid je jednotková guľa zobrazená lineárnym zobrazením, na rovnomerné generovanie v MVEE elipsoide najprv rovnomerne vygenerujeme bod v jednotkovej guli a následne ho zobrazíme daným lineárnym zobrazením do bodu v MVEE elipsoide. Kedže zobrazenie je lineárne, jeho jakobián je konštantný, preto rovnomernosť hustoty generovania nezávisí od x. A teda rovnomernosť generovania sa zachová.

Na rovnomerné generovanie bodu $\mathbf{x}$ v d–rozmernej jednotkovej guli so stredom v nule najprv vygenerujeme d–rozmerný smerový vektor rovnomerne náhodne a následne vygenerujeme vzdialenosť bodu $\mathbf{x}$ od nuly tak, aby mali oblasti rôzne vzdialené od 0 rovnakú pravdepodobnosť na vygenerovanie.

Na rovnomerné vygenerovanie d–rozmerného smerového vektoru vygenerujeme bod v d–rozmernom centrálne symetrické rozdelenie a následne ho zobrazíme na jednotkovú sféru (rovnako ako pri generovaní smeru Hit–and–Run generátora). Následne vygenerujeme vzdialenosť od nuly podľa rozdelenia daného hustotou $h : [0,1] \to [0,1]$:

$$h(\mathbf{x}) = \frac{\mathbf{x}^d}{\int_0^1 y^d dy}.$$

Pri generovaní z danej hustoty získame rovnomerné rozdelenie na guli. Zobrazenie bodu $\mathbf{x}$ z d–rozmernej sféry do d–rozmernej gule možno vyjadriť pomocou generátora $U(0,1)$ z rovnomerného rozdelenia na $(0,1)$ ako

$$G_d(\mathbf{x}) = \mathbf{x} \cdot U(0,1)^{\frac{1}{d}}$$

.

Pri danej funkcii má generovaný bod $\mathbf{x}$ rovnakú pravdepodobnosť, že padne do ľubovoľne vzdialenej oblasti s rovnakým obsahom, dané rozdelenie je rovnomerné. **TODO najst elegantne zdovodnenie**

Elipsoid $E(\overline{\mathbf{H}}, \overline{\mathbf{z}})$ je definovaný ako

$$E(\overline{\mathbf{H}}, \overline{\mathbf{z}}) = \{\mathbf{z} \in \mathbb{R}^d \mid (\mathbf{z} - \overline{\mathbf{z}})' \overline{\mathbf{H}} (\mathbf{z} - \overline{\mathbf{z}}) \leq 1\},$$

pričom matica $\overline{\mathbf{H}}$ je pozitívne semidefinitná.

Parametre $\overline{\mathbf{H}}$ a $\overline{\mathbf{z}}$ získame pomocou REX algoritmu. Vygenerovať bod $\mathbf{x}_{MVEE}$ z rovnomerného rozdelenia v MVEE možno pomocou bodu $\mathbf{x}_G$ z rovnomerného rozdelenia v jednotkovej d–rozmernej guli pomocou transformácie $\mathbf{x}_{MVEE} \leftarrow C\mathbf{x}_G + \overline{\mathbf{z}}$, kde C je matica z Cholského rozkladu matice $\overline{\mathbf{H}}^{-1}$, t.j. $\overline{\mathbf{H}} = (CC^T)^{-1}$. Na overenie, ozaj platí

$$E(\overline{\mathbf{H}}, \overline{\mathbf{z}}) = \{\mathbf{x}_{MVEE} | (\mathbf{x}_{MVEE} - \overline{\mathbf{z}})^T \overline{\mathbf{H}} (\mathbf{x}_{MVEE} - \overline{\mathbf{z}}) \leq 1\} = \{\mathbf{x}_{MVEE} | (C\mathbf{x}_G)^T (CC^T)^{-1} (C\mathbf{x}_G) \leq 1\}$$

$$E(\overline{\mathbf{H}}, \overline{\mathbf{z}}) = \{\mathbf{x}_{MVEE} \mid \mathbf{x}_G^T \mathbf{x}_G \leq 1\} = \{\mathbf{x}_{MVEE} \mid \|\mathbf{x}_G\| \leq 1\},$$

preto je množina bodov $\{\mathbf{x}_{MVEE} \mid \mathbf{x}_{MVEE} = C\mathbf{x}_G + \overline{\mathbf{z}}$ pre $\mathbf{x}_G$ z jednotkovej d-rozmernej gule$\}$ ozaj množina bodov elipsoidu $E(\overline{\mathbf{H}}, \overline{\mathbf{z}})$.

MVEE metóda funguje nasledovne:

Algorithm 5 MVEE metóda

1: Vypočítaj $E(\overline{\mathbf{H}}, \overline{\mathbf{z}})$ pomocou REX algoritmu
2: Vypočítaj $C \leftarrow \text{chol}(\overline{\mathbf{H}}^{-1})$
3: **for** $i = 1, \ldots, N$ **do**
4: **repeat**
5: Vygeneruj $\mathbf{x}_G$ v jednotkovej guli rovnomerne náhodne
6: $\mathbf{x}_{MVEE} \leftarrow C\mathbf{x}_G + \overline{\mathbf{z}}$ (zobraz bod $\mathbf{x}_G$ do MVEE)
7: **until** $\mathbf{x}_{MVEE} \in S$
8: $\mathbf{x}^{(i)} \leftarrow \mathbf{x}_{MVEE}$
9: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$

V MVEE metóde vygenerujeme bod $\mathbf{x}_G$ v jednotkovej guli, ktorý následne zobrazíme lineárnou transformáciou $f(\mathbf{x}_G) = Cx_G + \overline{\mathbf{z}}$ na bod $\mathbf{x}_{MVEE}$ v elipsoide. Potom overíme, či je daný bod v polyédri S. Takto zobrazujeme linárnym zobrazením taktiež aj body, ktoré ležia v $S_{MVEE} \setminus S$.

1.3.1 Zrýchlená MVEE metóda

Na zrýchlenie tejto metódy môžeme namiesto overovania, či je bod $\mathbf{x}_{MVEE}$ v S môžeme overovať, či je bod $f^{-1}(\mathbf{x}_{MVEE}) = \mathbf{x}_G$ v $f^{-1}(S)$. Zrejme ak $\mathbf{x}_G \in f^{-1}(S)$, tak aj $\mathbf{x}_{MVEE} \in S$. Ak $\mathbf{x}_G \notin f^{-1}(S)$, tak ani $\mathbf{x}_{MVEE} \notin S$.

Zrýchlená verzia MVEE metódy vyzerá nasledovne:

Algorithm 6 Zrýchlená MVEE metóda

1: Vypočítaj $E(\overline{\mathbf{H}}, \overline{\mathbf{z}})$ pomocou REX algoritmu
2: Vypočítaj $C \leftarrow \text{chol}(\overline{\mathbf{H}}^{-1})$
3: **for** $i = 1, \ldots, N$ **do**
4: **repeat**
5: Vygeneruj $\mathbf{x}_G$ v jednotkovej guli rovnomerne náhodne
6: **until** $\mathbf{x}_G \in f^{-1}(S)$
7: $\mathbf{x}^{(i)} \leftarrow f(\mathbf{x}_G) = C\mathbf{x}_G + \overline{\mathbf{z}}$
8: Vráť $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$

Takýmto overovaním pred transformáciou f sa vyhneme zobrazovaniu bodov $\mathbf{x}_G$, ktorých obraz v zobrazení f patrí do $S_{MVEE} \setminus S$. Kedže zobrazením f zobrazujeme v zrýchlenej verzii pre každý bod len raz, týmto sme pri každom ušetrili $\frac{\lambda(T_G)}{\lambda(T_P)} - 1$ zobrazovaní f. Dosiahli sme takmer t_f násobné zrýchlenie algoritmu (kde t_f je čas potrebný na použitie zobrazenia f).

Kapitola 2

Metódy na riešenie problému optimálneho návrhu

V tejto kapitole si predstavíme Randomized exchange algoritmus [4] (ďalej REX) ako metódu na riešenie problému optimálneho návrhu (optimal design problem). Táto kapitola je čerpaná z článku [4].

Cieľom problému optimálneho návrh je nájsť návrh minimalizujúci kritérium optimality. Pri návrhoch experimentov to zodpovedá nájdeniu návrhu experimentov minimalizujúceho výchylku parametrov.

Vďaka ekvivalencii problému D–optimálneho návrhu a minimum volume enclosing elipsoidu (MVEE) možno optimálny návrh vhodného modelu experimentov využiť aj na riešenie MVEE problému. Ak vrcholy polyédra P vo V–reprezentácii stotožníme s lineárnymi regressormi v probléme D–optimálneho návrhu, z riešenia D–optimálneho návrhu pre dané regressory môžno vypočítať MVEE elipsoid prislúchajúci k polyédru P.

Zaveďme si označenia využívané pri probléme optimálneho návrhu. Označme si množinu zvolených bodov návrhov $\mathfrak{X}$. Označme návrh ako n–rozmerný vektor $\mathbf{w}$ s nezápornými prvkami so súčtom 1. Pri probléme optimálneho návrhu, komponent w_x vektoru $\mathbf{w}$ predstavuje počet pokusov v bode $x \in \mathfrak{X}$. Označme si $\mathbf{f}(x) \in \mathbb{R}^m$ lineárny regresor prislúchajúci ku x. Predpokladajme, že model je nesingulárny v zmysle, že $\{\mathbf{f}(x) | x \in \mathfrak{X}\}$ generuje $\mathbb{R}^m$. Označme nosič návrhu $\mathbf{w}$ ako $\text{supp}(\mathbf{w}) = \{x \in \mathfrak{X} | w_x > 0\}$. Množina všetkých návrhov tvorí pravdepodobnostný simplex v $\mathbb{R}^m$, označme ju Ξ (je kompaktná a konvexná). Označme si $\mathbf{M}(\mathbf{w})$ informačnú maticu prislúchajúcu k návrhu $\mathbf{w}$, platí

$$\mathbf{M}(\mathbf{w}) = \sum_{x \in \mathfrak{X}} w_x \mathbf{f}(x) \mathbf{f}'(x).$$

Taktiež si označme výchilku $\mathbf{w}$ ako $\mathbf{d}(\mathbf{w})$, v pozorovanom bode x má hodnotu

$$d_x(\mathbf{w}) = \mathbf{f}'(x)\mathbf{M}^{-1}(\mathbf{w})\mathbf{f}(x), x \in \mathfrak{X}.$$

Z predpokladu, že model je nesingulárny vyplýva, že $\mathbf{M}^{-1}$ existuje. Ďalej si označme $\Phi_D : S_+^m \to \mathbb{R} \cup \{-\infty\}$, kritérium D–optimality,

$$\Phi_D(\mathbf{w}) = (\det(\mathbf{M})^{1/n}).$$

Cieľom probému optimálneho návrhu je maximalizovať $\Phi_D(\mathbf{M}(\mathbf{w}))$ vzhľadom na $\mathbf{w}$, t.j. nájsť optimálny návrh

$$\mathbf{w}^* = \arg\max_{\mathbf{w}\in\Xi} \in \{\Phi_D(\mathbf{M}(\mathbf{w}))\}.$$

Podľa [4], pre verziu D–optimality Ψ splňujúcu $\Psi(\mathbf{M}) = \log\det(\mathbf{M})$ platí, že

$$d_x(\mathbf{w}) = \lim_{\alpha\to 0_+} \frac{\Psi[(1-\alpha)\mathbf{M}(\mathbf{w}) + \alpha\mathbf{M}(\mathbf{e}_x)] - \Psi(M(w))}{\alpha} + n,$$

kde $\mathbf{e}_x$ je singulárny návrh v x. Z tohoto tvaru vidno, že pri iterátívnom spôsobe rátania D–optimality možno použiť $\mathbf{d}$ na určenie smeru, v ktorom hľadať ďalší návrh.

Pre polyéder v H–reprezentácii zadaný bodmi $\{\mathbf{z_1}, \ldots, \mathbf{z_n}\}$ ($\mathbf{z_i} \in \mathbb{R}^d$ pre $i = 1, \ldots, n$) je možno MVEE vypočítať ako

$$P(\overline{\mathbf{H}}, \overline{\mathbf{z}}) = \{\mathbf{z} \in \mathbb{R}^d | (\mathbf{z} - \overline{\mathbf{z}})'\overline{\mathbf{H}}(\mathbf{z} - \overline{\mathbf{z}}) \leq 1\},$$

kde $\overline{\mathbf{z}} = \sum_{i=1}^n w_i^* \mathbf{z_i}$, $\overline{\mathbf{H}} = \frac{1}{m-1}\left[\sum_{i=1}^n w_i^*(\mathbf{z_i} - \overline{\mathbf{z}})(\mathbf{z_i} - \overline{\mathbf{z}})'\right]^{-1}$, pričom $\mathbf{w}^*$ je D–optimálny návrh pre regressory $\mathbf{f}_i = (1, \mathbf{z_i}')'$ pre $i = 1, \ldots, n$.

Vzhľadom na dôležitosť MVEE elipsoidu pre túto prácu a kvôli lepšiemu pochopeniu REX algoritmu sa najprv pozrieme na jednoduchšie metódy riešenia problému optimálneho návrhu.

2.1 Základné metódy na riešenie problému optimálneho návrhu

Najprv si predstavíme metódu Subspace Ascend Method (ďalej SAM) ako všeobecnú iteratívnu metódu na riešenie problému optimálneho návrhu a Vertex Exchange Method (VEM) ako jej konkrétnu realizáciu. Následne sa pozrieme na REX algoritmus ako na špeciálny prípad SAM, ktorý kombinuje VEM metódu s pažravým prístupom.

2.1.1 Subspace Ascend Method

SAM algoritmus postupuje iteratívne. V každej iterácii si vyberie podpriestor v ktorom sa bude hýbať a následne spraví optimálny krok v danom podpriestore:

Algorithm 7 Subspace Ascend Method (SAM) [4]

1: Zvoľ regulárny n rozmený návrh $\mathbf{w}^{(0)}$
2: **while** $\mathbf{w}^{(k)}$ nespĺňa podmienky zastavenia **do**
3: Zvoľ podmnožinu bodov $S_k \subset \mathfrak{X}$
4: Nájdi aktívny podpriestor Ξ ako $\Xi_k \leftarrow \{\mathbf{w} \in \Xi | \forall x \notin S_k, w_x = w_x^{(k)}\}$
5: Vypočítaj $\mathbf{w}^{(k+1)}$ ako riešenie $\max_{\mathbf{w}\in\Xi_k} \Phi_D(\mathbf{M}(\mathbf{w}))$
6: Nastav $k \leftarrow k + 1$
7: Vráť $\mathbf{w}$

Pozrime sa na to, ako sa bude meniť hodnota cieľovej funkcie D–optimality $\Phi_D(\mathbf{M}(\mathbf{w}^{(k)}))$. SAM algoritmus v kroku ako ďalší návrh zvolí $\mathbf{w}^{(k+1)} = \max_{\mathbf{w}\in\Xi_k} \Phi_D(\mathbf{M}(\mathbf{w}))$ (krok 5). Nakoľko $\mathbf{w}^{(k)} \in \Xi_k$, nový návrh spĺňa $\Phi_D(\mathbf{M}(\mathbf{w}^{(k+1)})) \geq \Phi_D(\mathbf{M}(\mathbf{w}^{(k)}))$. Preto je postupnosť hodnôt $\Phi_D(\mathbf{M}(\mathbf{w}^{(k)}))$ neklesajúca, preto žiadnym krokom nezhoršíme hodnotu cieľovej funkcie.

2.1.2 Vertex Exchange Method

Algoritmus VEM možno vnímať ako konrétnu realizáciu SAM algoritmu. Postupuje taktiež iteratívne, v jednom kroku z návrhu $\mathbf{w}$ nájde index k minimalizujúci $\text{supp}(\mathbf{w})$, l minimalizujúci $\mathfrak{X}$. Ako ďalší návrh $\mathbf{w}'$ zvolí návrh z úsečky $[w_k w_l]$ maximalizujúci $\Phi_D(\mathbf{M}(\mathbf{w}'))$.

Algorithm 8 Vertex Exchange Method (VEM) [4]

1: Zvoľ regulárny n rozmerný návrh $\mathbf{w}$
2: **while** $\mathbf{w}$ nespĺňa podmienky zastavenia **do**
3: Vypočítaj $k \leftarrow \arg\min_{u\in supp(\mathbf{w})}\{d_u(\mathbf{w})\}$
4: Vypočítaj $l \leftarrow \arg\max_{v\in\mathfrak{X}}\{d_v(\mathbf{w})\}$
5: Vypočítaj $\alpha^* \leftarrow \arg\max_{\alpha\in[-w_l,w_k]}\{\Phi_D(\mathbf{M}(\mathbf{w} + \alpha\mathbf{e}_l - \alpha\mathbf{e}_k))\}$
6: Nastav $w_k \leftarrow w_k - \alpha^*$
7: Nastav $w_l \leftarrow w_l + \alpha^*$
8: Vráť $\mathbf{w}$

Veľkosť optimálneho kroku α^* možno podľa [4] vyjadriť explicitne. Nech pre $k, l \in \mathfrak{X}$ platí, že w_k, w_l sú kladné a regresory $\mathbf{f}(k)$ a $\mathbf{f}(l)$ nezávislé. Označme si $d_{k,l} = \mathbf{f}'(k)\mathbf{M}^{-1}(\mathbf{w})\mathbf{f}(l)$. Podľa [4] pre veľkosť optimálneho kroku do návrhu na úsečke $[w_k w_l]$

platí

$$\alpha^*_{k,l} = \min\left(w_k, \max\left(-w_l, \frac{d_k(\mathbf{w}) - d_l(\mathbf{w})}{2[d_k(\mathbf{w})d_l(\mathbf{w}) - d^2_{k,l}(\mathbf{w})]}\right)\right).$$

Krok VEM algoritmu sa označuje ako leading Bohning exchange (ďalej LBE). Dvojica (k, l) sa označuje ako pár LBE.

2.2 Radomized Exchange Algoritmus

V tejto podkapitole popíšeme randomized exchange algoritmus (REX) predstavený v [4]. Dá sa na neho pozerať ako na špeciálny prípad SAM algoritmu. Ako už bolo spomenuté, REX algoritmus kombinuje kroky VEM algoritmu a pažravých algoritmov. Podľa [4] je REX algoritmus v praxi rýchlejší ako všetky s ním porovnané state–of–the–art algoritmy na riešenie problému optimálneho riadenia. Na uvedenie predstavy o rýchlosti REX algoritmu, medzi známe algoritmy na riešenie problému optimálneho riadenia patria algoritmy konvergujúce v lineárnom čase (napr. Khachiyanov algoritmus [7]). Konvergencia v lineárnom čase znamená, že do vzialenosti ϵ od optima dôjdu v čase $\mathcal{O}(log(\frac{1}{\epsilon}))$

Prejdime k samotnému algoritmu. Nech $\mathbf{w}$ je regulárny návrh, nech $\mathbf{d}(\mathbf{w})$ je n–rozmerný vektor s komponentami $d_x(\mathbf{w})$. Hlavná myšlienka REX algoritmu je, počnúc inicializovaným regulárnym návrhom $\mathbf{w}$, iteratívne vyberať niekoľko bodov aktuálneho návrhu (ich počet sa bude líšiť v rámci iterácii) a v náhodnom poradí vykonať optimálnu výmenu váh medzi vybranými bodmi. Optimálna výmena váh je analogická LBE kroku VEM algoritmu. Voľba nodov závisí od $\mathbf{d}(\mathbf{w})$. Kroky REX algoritmu budú nasledovné:

- **Krok LBE.** Pri danom návrhu $\mathbf{w}$, vypočítaj $\mathbf{d}(\mathbf{w})$ a urob LBE krok daný nasledovne:

  $$\alpha^* \leftarrow \underset{\alpha\in[-w_l,w_k]\}}{\arg\max}\ \{\Phi_D(\mathbf{M}(\mathbf{w} + \alpha\mathbf{e}_l - \alpha\mathbf{e}_k)),$$

  kde $k \in \arg\min_{u\in\mathrm{supp}(\mathbf{w})}\{d_u(\mathbf{w})\}, l \in \arg\max_{v\in\mathfrak{X}}\{d_v(\mathbf{w})\}$. Optimálny krok $\alpha^*_{k,l}(\mathbf{w})$ nazvime *nulujúci*, ak je rovný buď $-w_l$ alebo w_k. To zodpovedá prípadu, keď sme sa optimálnym krokom pohli do niektorého z bodov w_l alebo w_k.

- **Výber aktívneho podpriestoru.** Podpriestor $S \subset \mathfrak{X}$, v ktorom sa pohneme bude zvolený ako zjednotenie dvoch množín. Jednou vybranou pažravým procesom (S_{greedy}) a druhou ako nosič návrhu $\mathbf{w}$ (S_{support}).

– **Pažravá množina.** Nech $L = \min(\gamma m, n)$ je počet návrhov, ktoré vyberieme. Potom zvoľ S_{greedy} ako

$$S_{\text{greedy}} = \{l_1^*, \ldots, l_L^*\} \subset \mathfrak{X},$$

kde l_i^* je najväčšia zložka vektoru $\mathbf{d}(\mathbf{w})$. Platí $L = |S_{\text{greedy}}|$.

– **Nosič.** Nastav

$$S_{\text{support}} = \text{supp}(\mathbf{w}).$$

Označme K veľkosť nosiča, $K = |\text{supp}(\mathbf{w})| = |S_{\text{support}}|$.

– **Aktívny podpriestor.** Aktívny podpriestor S je definovaný ako

$$S = S_{\text{greedy}} \cup S_{\text{support}}.$$

Váhy návrhu $\mathbf{w}$ mimo aktívneho podpriestoru nebudú upravované v tejto iterácii.

- **Krok v aktívnom podpriestore.** Teraz vykonáme krok, v ktorom aktualizujeme hodnoty w_v pre $v \in S$. Body w_v návrhu $\mathbf{w}$ pre $v \notin S$ ostanú nezmenené.

– **Tvorba párov.** Nech $(k_1, \ldots, k_K)$ je uniformne náhodná premutácia S_{support} a nech $(l_1, \ldots, l_L)$ je uniformne náhodná permutácia S_{greedy}. Potom postupnosť aktívnych dvojíc návrhov, na ktorých budú vykonané optimálne výmeny váh je

$$(k_1, l_1), (k_2, l_1), \ldots, (k_L, l_1), (k_1, l_2)(k_2, l_L), \ldots, (k_K, l_L).$$

– **Aktualizácia.** Vykonaj postupne všetky Φ_D–optimálne LBE kroky medzi návrhmi z $(k_1, l_1), \ldots, (k_K, l_L)$ s prisluchajúcimi aktualizáciami $\mathbf{w}$ a $\mathbf{M}(\mathbf{w})$.

REX algoritmus vyzerá nasledovne:

Algorithm 9 REX algoritmus [4]

1: Zvoľ regulárny n–rozmerný návrh $\mathbf{w}$
2: **while** $\mathbf{w}$ nespĺňa podmienky zastavenia **do**
3: Urob LBE krok vo $\mathbf{w}$
4: Nech k je vektor zodpovedajúci náhodnej permutácii prvkov supp($\mathbf{w}$)
5: Nech l je vektor zodpovedajúci náhodnej permutácii $L = \min(\gamma m, n)$ indexov prvkov $\mathbf{d}(\mathbf{w})$
6: **for** $l = 1 \dots L$ **do**
7: **for** $l = 1 \dots K$ **do**
8: $\alpha^* \leftarrow \arg\max_{\alpha \in [-w_l, w_k]} \{\Phi_D(\mathbf{M}(\mathbf{w} + \alpha \mathbf{e}_l - \alpha \mathbf{e}_k))\}$
9: **if** LBE krok bol nulujúci alebo $\alpha^* = -w_l$ alebo $\alpha^* = w_k$ **then**
10: $w_k \leftarrow w_k - \alpha^*$
11: $w_l \leftarrow w_l + \alpha^*$
12: Vráť $\mathbf{w}$

V rámci jednotlivých krokoch REX algoritmu, počas LBE krokov je na výpočet ďalšieho bodu potrebné invertovať maticu $\mathbf{M}(\mathbf{w})$. Ako potenciálny problém by mohlo nastať, že v rámci behu programu sa algoritmus bude nachádzať v bode $\mathbf{w}^{(q)}$ takom, že matica $\mathbf{M}(\mathbf{w}^{(q)})$ bude singulárna a teda neeexistuje inverzná matica $\mathbf{M}(\mathbf{w}^{(q)})^{-1}$. To však nastať nemôže, ak pri inicializácii bude zvolený počiatočný návrh $\mathbf{w}^{(0)}$ tak, že matica $\mathbf{M}(\mathbf{w}^{(0)})$ bude regulárna. Kedže $\Phi_D(\mathbf{w}) = (\det(\mathbf{M})^{1/d})$ a postupnosť dosiahnutých návrhov $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots$ spĺňa $\Phi_D(\mathbf{M}(\mathbf{w}^{(0)})) \leq \Phi_D(\mathbf{M}(\mathbf{w}^{(1)})) \leq \Phi_D(\mathbf{M}(\mathbf{w}^{(2)})) \leq \dots$, postupnosť determinantov matíc $\mathbf{M}(\mathbf{w}^{(k)})$ je teda neklesajúca a preto sa regulárnosť matíc $\mathbf{M}(\mathbf{w}^{(k)})$ zachová.

Poznámka: popísali sme REX algoritmus vzhľadom na kritérium D–optimality. Podľa [4] REX algoritmus funguje aj vzhľadom na A–optimalitu pri nahradení funkcie d_x funkciou a_x definovanou ako

$$a_x(\mathbf{w}) = \mathbf{f}'(x)\mathbf{M}^{-2}(\mathbf{w})\mathbf{f}(x), x \in \mathfrak{X}.$$

a nahradení funkcie Φ_D funkciou Φ_A definovanou ako

$$\Phi_A(\mathbf{w}) = (\mathrm{tr}(\mathbf{M})^{-1})^{-1}.$$

Pri tejto verzii problému však článok [4] nedokazuje konvergenciu algoritmu. Pre účely tejto práce to však nie je podstatné.

Kapitola 3

Porovnanie metód

Táto kapitola obsahuje praktické porovnania Metropolis–Hastings algoritmov spomenutých v prvej kapitole so zamietacou metódou pomocou MVEE elipsoidu. Porovnania boli naprogramované v jazyku Julia a spustené na počítači s procesorom Intel Core i5-6200U s frekvenciou 2.3GHz a RAM pamäťou veľkosti 8 GB pod systémom Linux.

Ako základ náhody bude náš generátor bodu v polyédre používať rovnomerný generátor čísel na $[-1, 1]$ (ďalej $U[0, 1]$). Pomocou $U[0, 1]$ možno triviálne generovať bod na $[0, k]$ (prenásobením konštantou k), tiež možno generovať bod na $[a, b]$ (vygenerovaním bodu na $[0, -a + b]$ a pripočítaním konštanty a, alebo bod na $[0, 1]^n$ (postupným vygenerovaním súradníc). Treba podotknúť, že generovanie na iných polyédroch, najmä v priestoroch vysokej dimenzie, je však vo všeobecnosti netriviálny problém.

Okrem generátora $U[0, 1]$ budeme používať generátor z jednorozmerného normálneho rozdelenia $N(\mu, \sigma)$ s priemerom $\mu = 0$ a odchýlkou $\sigma = 1$. Vďaka rotačnej symetrickosti normálneho rozdelenia možno generovaním po zložkách pomocou $N(\mu, \sigma)$ získať d–rozmerné viacrozmerné normálne rozdelenie. Keďže viacrozmerné normálne rozdelenie je centrálne symetrické, možno ho použiť na generovanie na d–rozmernej sfére ako bolo popísané v 1.3.

Ako možné parametre porovnania možno uvažovať rýchlosť generovania bodov a vierohodnosť vygenerovaného rozdelenia (ako sa rozdelenie vygenerovaných bodov podobá rovnomernému rozdeleniu v polyédri). Nakoľko v praxi pri generovaní malého počtu bodov nezáleží na efektivite generátora, možno použiť hocijakú zo spomenutých metód. Kedže tento prípad je v praxi nezaujímavý, v rámci testovania budeme porovnávať generovanie veľkého počtu bodov.

Pozrime sa na vierohodnosť vygenerovaných bodov. Zamietacie metódy, špeciálne aj MVEE metóda, generujú presne rovnomerné rozdelenia v polyédri. Metropolis–

Hastings metódy sú v tomto smere nepresnejšie, generujú postupnosť, ktorá sa limitne blíži rovnomernému rozdeleniu v polyédri. Nakoľko uvažujeme generovanie veľkého počtu bodov, postupnosť vygenerovaných bodov sa bude aj pre Metropolis–Hastings metódy blížiť rovnemernému rozdeleniu v polyédri. Preto ak náhodne preusporiadame postupnosť takto vygenerovaných bodov, dostaneme rozdelenie bodov podobné rovnomernému rozdeleniu v polyédri, kde navyše po sebe idúce body nie sú korelované (ako pri postupnosti priamo vygenerovaných bodov). Preto možno predpokladať, že vierohodnosť vygenerovanej postupnosti bodov bude v oboch prístupoch vysoká. Ďalej sa ňou nebudeme zaoberať, budeme porovnávať výlučne rýchlosť generovania veľkého počtu bodov. Taktiež, kedže nás zaujíma generovanie veľkého počtu bodov, do rýchlosti generovanie nebudeme zarátavať rýchlosť inicializácii algoritmov, v prípade zamietacej metódy pomocou MVEE nebudeme brať do úvahy čas potrebný na vypočítanie MVEE.

Pred spustením testovania sme predpokladali, že najrýchlejší generátor bude generátor pomocou zamietacej metódy využívajúcej MVEE, táto práca bude obsahovať implementáciu REX algoritmu ako rýchly nástroj na hľadanie MVEE (namiesto hľadania elipsoidu inou pomalšou, no jednoduchšou metódou).

3.1 Generovanie polyédrov

Metódy boli porovnávané na veľarozmerných polyédroch. Nakoľko predpokladáme, že v praxi pri generovaní bodov vnútri polyédrov sa štruktúra polyédrov od problému k problému líši, v rámci tejto práce sme na testovanie algoritmov zvolili množinu polyédrov ako množinu náhodne vygenerovaných polyédrov. Týmto sme sa vyhli patologickým prípadom polyédrov.

3.1.1 Definícia náhodných polyédrov

Zadefinovať náhodný polyéder je samo o sebe ťažký problém. Ako jeden z možných spôsobov definovania prichádza do úvahy definovanie cez V–reprezentáciu polyédru, ako konvexný obal náhodných bodov v priestore. Táto definícia je síce nepresná, no poskytuje návod, ako aj možno takto definovaný polyéder generovať — vygenerovať množinu bodov V–reprezentácie a následne, v prípade záujmu o zlepšenie efektivity, možno z danej množiny vyhodiť body neležiace na obale polyédra. Nepresnosť spomenutej definície spôsobuje dva zásadné problémy. Po prvé, nie je jasné, koľko bodov polyédru je potrebné vygenerovať. Ak sa obmedzíme na vygenerovanie v bodov, tak potom takto nemožno vygenerovať polyédre s viac ako v vrcholmi. Na odstránenie tohoto problému by bolo nutné generovať číslo v z nejakého, pravdepodobne zložitého rozdelenia. Ako druhý, závažnejší problém je, že nemožno generovať body v celom priestore

rovnomerne náhodne. Už jednorozmerný prípad je problematický — nemožno rovnomerne náhodne generovať reálne čísla. To súvisí s tým, že pri rovnomernom generovaní reálnych čísel na $(-\infty, \infty)$ majú sú všetky intervaly (a, b) nulovú hustotu.

Ako iný spôsob, možno definovať náhodný polyéder pomocou H–reprezentácie, ako množinu bodov x splňujúcu množinu náhodných lineárnych obmedzení (nadrovín), pomocou náhodnej matice A a náhodného vektoru b. Táto definícia nám tiež dáva návod, ako generovať polyédre. Stačí náhodne vygenerovať lineárne nerovnosti — maticu A a vektor b, polyéder bude množina bodov splňujúce nerovnosť $Ax \geq b$. Podobne ako pri predchádzajúcom pokuse o definovanie náhodných polyédrov, aj tu nepresnosti definície spôsobujú problémy. Prvý problém je, že nie je jasné, koľko nerovností je potrebné vygenerovať. Analogický problému predchádzajúcej definície, ak sa obmedzíme na vygenerovanie h nadrovín, tak potom takto nemožno vygenerovať polyédre s viac ako h stenami. Ďalší netriviálny problém je, ako náhodne zvoliť nadroviny. Nadrovina je definovaná otočením (prislúchajúcim riadkom matice A) a posunutímod počiatku sústavy (daným prislúchajúcim prvkom vektoru b). Na to, aby sme dostali náhodné otočenie nadroviny s rovnakou pravdepodobnosťou, stačí vygenerovať prvky matice A z nezávislých normálnych rozdelení s priemerom 0 a výchilkou 1, vďaka rotačnej symetrii normálneho rozdelenia bude každá nadrovina cez počiatok sústavy rovnako pravdepodobná. Avšak náhodne vygenerovať posunutie nadroviny od počiatku sústavy je úloha ekvivalentná rovnomernému vygenerovaniu reálneho čísla. To tiež nejde spraviť. Ako ďalší problém tohoto prístupe je, že takto možno vygenerovať prázdny polyéder (množina bodov splňujúcich $Ax \geq b$ vie byť ľahko prázdna).

Keďže zameranie práce nie na generovanie polyédrov ako také, ďalej v práci upustíme od formálne presnej definície náhodneho polyédra, uspokojíme sa s menšími nepresnosťami.

3.1.2 Reprezentácia generovania polyédra

Na generovanie bodov pomocou Metropolis–Hastings metód potrebujeme mať polyéder zadaný v H–reprezentácii, no REX algoritmus na nájdenie MVEE elipsoidu potrebuje ako vstup polyéder vo V–reprezentácii. Preto je nutné v rámci generovania vygenerovať polyéder zároveň v H–reprezentácii aj vo V–reprezentácii. Potrebujeme vygenerovať polyéder v jedenej reprezentácii a previesť ho do druhej. Daný prevod nie je súčasťou žiadnej metódy generovania bodov v polyédri, nebude súčasťou nášho generátora. Kedže bude pre každý polyéder spravený len raz, nie je nutné, aby bol rýchly.

Ako alternatíva ku generovaniu polyédrov v jednej reprezentácii a prevádzaniu do

druhej reprezentácie možno porovnávať metódy aj inak. Predpokladajme, že máme generátor G_H polyédrov v H–reprezentácii a generátor G_V polyédrov vo V–reprezentácii, pričom G_H a G_V generujú z rovnakého rozdelenia polyédrov. Ak by sme testovali Metropolis–Hastings metódy na veľkom množstve polyédrov vygenerovaných pomocou G_H a zamietaciu metódu pomocou MVEE elipsoidu na veľkom množstve polyédrov vygenerovaných pomocou G_V, výsledky budú podobné ako keby sme spomínané metódy testovali na rovnakých polyédroch. Nakoľko nájdenie generátorov G_H a G_V s rovnakým rozdeleným polyédrov je potenciálne ťažší problém (už len zabezpečenie rovnakého rozdelenia obsahov polyédrov je netriviálne), tomuto alternatívnemu prístupu sa v tejto práci venovať nebudeme.

Poďme sa zamyslieť, v akej reprezentácii sa nám oplatí generovať polyéder. Teoreticky by mohlo byť možné, že pre nejakú triedu polyédrov je jedna ich reprezentácia exponenciálne väčšia ako ich druhá reprezentácia. T.j., že by mali jednu reprezentáciu veľkosti n a druhú veľkosti $\mathcal{O}(2^n)$. Polyédre z danej triedy by mohli byť pri testovaní problematické, nakoľko algoritmy využívajúce spomenutú reprezentáciu by boli značne znevýhodnené. Zišlo by sa nám ukázať, že niečo také nemôže nastať, t.j., že možno vzájomne polynomiálne odhadnúť veľkosti reprezentácii.

V prípade trojrozmerných polyédrov to možno dokonca ohraničiť lineárnym faktorom 2. Kedže topologickú štruktúru trojrozmerného polyédra možno zapísať ako planárny graf, možno jednoducho ukázať, že minimálna množina nerovností (stien) v H–reprezentácii a minimálna množina vrcholov vo V–reprezentácii sú asymptoticky rovnako veľké. Presnejšie, medzi počtom stien F a počtom vrcholov V v polyédra platí vzťah $F + V = E - 2$ (Eulerová charakteristika), kde E je počet hrán polyédra. Keďže E možno odhadnúť pomocou vzťahu $E \leq 3V - 6$ (pre $V > 2$), platí $F = E - 2 - V \leq 2V - 6$, preto $F \leq 2V - 6$. Taktiež, keďže v nedegenerovanom polyédri majú vrcholy stupňe aspoň 3, E možno odhadnúť pomocou súčtov stupňov vrcholov polyédra ako $E \geq \frac{3}{2}V$, z čoho získavame odhad $F = E - 2 - V \geq \frac{1}{2}V - 2$, preto $F \geq \frac{V}{2} - 2$.

Týmto sme ukázali, že pre ľubovoľný nedegenerovaný polyéder v troch rozmeroch sú jeho H–reprezentácia a V reprezentácia asymptoticky rovnako veľké, až na lineárny faktor 2. Preto v rámci porovnania metód by nie je podstatné, či najprv náhodne vygenerujeme polyéder v H–reprezentácii, ktorú prevedieme do V–reprezentácie alebo či najprv vygenerujeme V–reprezentáciu, ktorú prevedieme do H–reprezentácie. V obidvoch prípadoch dostaneme asymptoticky rovnako veľké reprezentácie.

Pre vyšší počet rozmerov sa nám nič podobné nepodarilo ukázať. Navyše, testovanie generátoru polyédrov (viď 10) v praxi nasvedčuje, že podiel $\frac{F}{V}$ rastie exponenciálne s rastúcim počtom rozmerov. Preto možno predpokladať, že polynomiálny odhad počtu vrcholov polyédra od jeho počtu stien a rozmeru priestoru nemusí existovať.

Ak máme danú aj stenovú aj vrcholovú reprezentáciu polyédra, ktoré nie sú nutne minimálne (vzhľadom na počet stien v H–reprezentácii a počet vrcholov vo V–reprezentácii), môžeme jednoducho dané reprezentácie minimalizovať odstránením prebytočných nadrovín a vrcholov. Z H–reprezentácie možno odstrániť tie nadroviny, ktoré neprechádzajú aspoň tromi vrcholmi V–reprezentácie. Z V–reprezentácie možno odstrániť tie vrcholy, ktoré neležia na aspoň troch nadrovinách z H–reprezentácie. Takto získané reprezentácie sú s pravdepodobnosťou 1 minimálne. Všimnime si, že pravdepodobnosť, že náhodná nadrovina, ktorá nie je stena polyédru, prechádza aspoň jedným vrcholom V–reprezentácie je 0. Preto v rámci implementácie budeme odstráňovať nadroviny, ktoré neprechádzajú žiadnym vrcholom V–reprezentácie. Overiť, či bod x leží na nadrovine danej vektorom a a konštantou c je triviálne, stačí overiť rovnosť $a^T x = c$.

Generovanie polyédru vo H–reprezentácii

V rámci tejto práce je použitý mierne upravený algoritmus na generovanie náhodných polyédrov popísaný v [10]. Výstupom algoritmu je polyéder v H–reprezentácii, taký, že každý bod z jednotkovej hyperkocky má rovnakú pravdepodobnosť byť vnútri.

Algoritmus popísaný v [10] využíva prístup Monte Carlo, funguje nasledovne: Najprv náhodne zvolí m nadrovín $p_1, \ldots, p_m$, tie rozdeľujú priestor na niekoľko nie nutne ohraničených oblastí. Následne rovnomerne náhodne vygeneruje bod c v jednotkovej hyperkocke. Ako polyéder P_c zvolí algoritmus oblasť vymedzenú nadrovinami p_i, v ktorej leží c. Nadrovina p_i je reprezentovaná i–tým riadkom matice A a i–tým prvkom vektoru b. Polpriestor vymedzený p_i je množina bodov x spĺňajúca nerovnosť $(Ax - b)_i \geq 0$. Ak po vygenerovaní A, b, c neleží c v priestore vymedzenom p_i (platí $(Ax - b)_i \leq 0$), tak algoritmus otočí i–tú nerovnosť na $\leq$ prenásobením i–teho riadku A a i–teho prvku b hodnotou -1. Potom bude zrejme platiť $(Ax - b)_i \geq 0$.

Na záver algoritmus overí, či je vygenerovaný polyéder P_c ohraničený vopred zvolenou hyperkockou. Ak áno, tak vráti P_c. Ak nie je, tak daný polyéder zahodí a generuje znovu.

Algorithm 10 Generátor náhodných polyédrov [10]

1: Vygeneruj prvky matice A z nezávislých $N(0,1)$
2: Vygeneruj prvky vektoru b z nezávislých $N(0,1)$
3: Vygeneruj bod c v jednotkovej hyperkocke
4: Nastav jednotlivé nerovnosti na $\geq$ alebo $\leq$ tak, aby c spĺňal $Ac \geq b$.
5: Nastav $y = Ac - b$
6: **for** $i = 0, 1, \ldots, n$ **do**
7: **if** $y_i > 0$ **then**
8: Nastav i–tu nerovnosť na $\geq$
9: **else**
10: Nastav i–tu nerovnosť na $\leq$
11: **if** P_c nie je celý v hyperkocke **then**
12: Zamietni polyéder P_c, vráť sa na 1
13: **else**
14: Odstráň obmedzenia hyperkocky, vráť P_c

Všimnime si, že c leží v polyédri, teda polyéder je neprázdny. Navyše, bod c je vygerovaný z rovnomerného rozdelenia na prieniku polyédra a jednotkovej hyperkocky. Tento bod použijeme na inicializáciu Hit–and–Run generátora a Gibbsovho generátora.

Pri takomto generovaní v rámci P_c môžeme získať aj nadroviny p_i, ktoré neobsahujú žiadnu stenu polyédra. Tieto nadroviny sú zrejme nadbytočné, preto ich môžeme odtiaľ odstrániť postupom popísaným vyššie.

Prvky matice A a vektoru b boli generované z nezávislých normálnych rozdelení $N(0,1)$. Na získanie V–reprezentácie z H–reprezentácie použili knižničné funkcie knižníc Polyhedra a CDDLib (v jazyku Julia). Nakoľko daný prevod bol pre rozmery $d \geq 10$ príliš pomalý, obmedzili sme sa na rozmery $d < 10$.

Na overenie, či je polyéder ohraničený v k–tom rozmere bolo použité lineárne programovanie s maximalizačným (a minimalizačným) cieľom x_k (maximalizuje/minimalizuje sa k–tá súradnica x) pre body x splňujúce $Ax \geq b$. V rámci implementácie boli použité knižničné funkcie knižníc Convex a SCS (v jazyku Julia).

V rámci generovania polyédrov trvalo príliš dlho, kým sme vygenerovali polyéder ohraničený jednotkovou hyperkockou. Problémom bolo, že bolo potrebné zahodiť príliš veľa ohraničených polyédrov, ktoré neležali celé v jednotkovej hyperkocke. Preto sme kvôli zrýchleniu programu namiesto overenia, či je polyéder ohraničený jednotkovou hyperkockou (riadok 11 algoritmu) overili iba, či je polyéder ohraničený. Po tejto zmene náš generátor polyédrov taktiež vracia ohraničené polyédre, ktoré nie sú celé v

jednotkovej hyperkocke.

3.2 Inicializácia algoritmov

Na inicializáciu Metropolis–Hastings metód (Hit–and–Run generátora a aj Gibbsovho generátora) je potrebný počiatočný bod vnútri polyédra a veľkosť burn–in periódy (iterácii algoritmu bez vracania vygenerovaných bodov ako výsledkov). Ako počiatočný bod sme zvolili bod c určený pri generovaní polyédra, ktorý je vygenerovaný na prieniku polyédra a jednotkovej hyperkocky rovnomerne náhodne. Vzhľadom k tomu, že generujeme veľké množstvo bodov v polyédri, je burn–in perióda nepotrebná. Symbolicky bola zvolená hodnota 100.

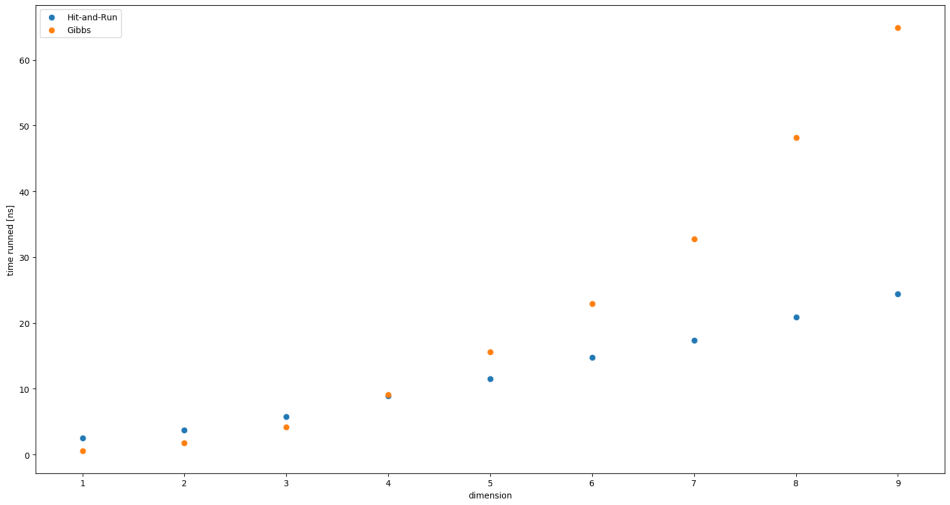
Na inicializáciu zamietacej metódy pomocou MVEE elipsoidu je potrebné vypočítať MVEE elipsoid. Pri inicializácii REX algoritmu je potrebné zvoliť počiatočný nosič tak, aby bola informačná matica M regulárna. V rámci práce bol nosič počiatočného návrhu zvolený ako $n+1$ náhodných vrcholov. Vzhľadom na náhodnosť polyédra, dané vrcholy budú s pravdepodobnosťou 1 určovať regulárnu maticu M.

3.3 Testovací algoritmus

TODO ráno doplniť

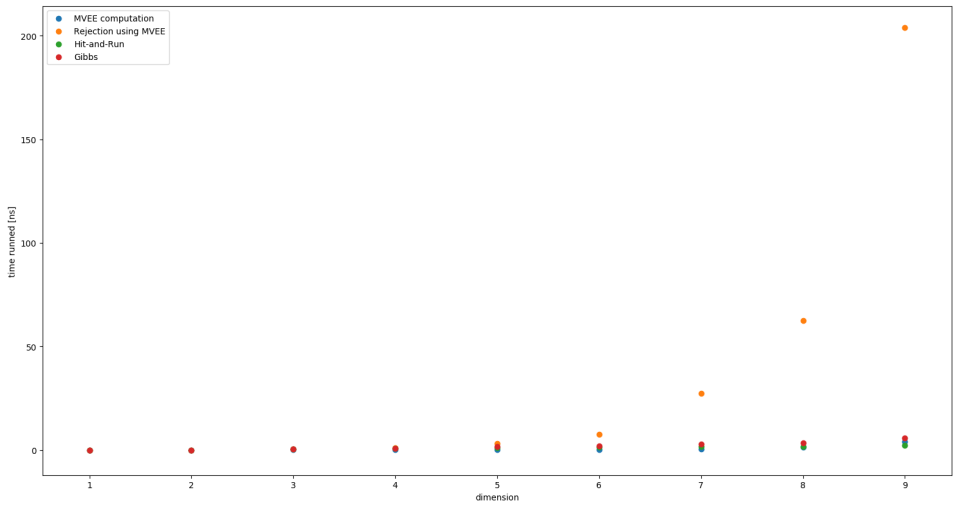
3.4 Porovnanie rýchlostí generovania

V rámci testovania boli metódy porovnávané na rozmeroch $d = 1, \ldots, 9$, v každom rozmere bolo vygenerovaných 100 náhodných polyédrov. Na každom z daných polyédrov boli spustené všetky metódy, každá z nich vygenerovala 10^5 bodov vnútri polyédru. Ako výstup testovania boli pre každú metódu zobrazené priemerné rýchlosti generovaní bodov v jednotlivých rozmeroch.



Obr. 3.1: Priemerný čas trvania generovania bodov Metropolis–Hastings metód

Porovnanie Metropolis–Hastings metód dopadlo podľa očakávaní. Čas potrebný na vygenerovanie bodu pomocou Gibbsovho generátor rástol asymptoticky kvadraticky a čas potrebný na vygenerovanie bodu pomocou Hit–and–Run generátora rástol asymptoticky lineárne (viď 3.1).



Obr. 3.2: Priemerný čas trvania všetkých metód **TODO toto je predbežný obrázok (porovnávaný na 3 polyedroch), dat dobry obrazok po vygenerovani (uz to generuje 12. hodinu)**

Metóda MVEE pri vyššom počte rozmerov generuje body výrazne pomalšie ako Metropolis–Hastings metódy (viď 3.2). To môže byť jednak spôsobené zväčšením po-

meru veľkostí V–reprezentácie ku H–reprezentácie alebo taktiež zväčšeniu pomeru objemov $\frac{\lambda(T_{MVEE})}{\lambda(S)}$. Ako druhý možný dôvod uveďme, že náš horný odhad d^d očakávaného počtu generovaní (kde d je počet rozmerov priestoru) podľa Johnovho elipsoidu nemusí byť tak voľný, ako sa na prvý pohľad zdá.

TODO graf počtu generovaní MVEE metódy vzhľadom na dimenziu

TODO graf (počet stien polyédra)/(počtu vrcholov) vzhľadom na dimenziu

Záver

V rámci tejto práci sa nám podarilo splniť všetky stanovené ciele. Dokonca sme mali čas venovať sa aj súvisiacim oblastiam, ktoré nie sú priamo nutné pre túto prácu. Menovite to boli problém optimálneho návrhu, náhodné polyédre, vzájomné odhady veľkostí H–reprezentácie a V–reprezentácie.

3.5 Prínos práce

V kapitole 1 vytvorili prehľad existujúcich metód na generovanie bodov z rovnomerného rozdelenia v polyédri.

Popísali sme triedu Metropolis–Hasting metód. Predstavili sme všeobecný Metropolis–Hasting generátor a jeho konkrétne realizácie Hit–and–run generátor a Gibbsov generátor ako známe algoritmy.

Taktiež sme sa venovali zamietacím metódam, ukázali sme, že spomedzi prípustných nadmnožín polyédra, výber kvádra so stranami rovnobežnými s osami sústavy môže viesť k ľubovoľne veľkému očakávanému počtu generovaní. Okrem toho sme ukázali, že MVEE je lepšia nadmnožina polyédra na generovanie zamietaciou metódou ako kvádер bez obmedzení na natočenie v priestore. Pomocou Johnovho elipsoidu sme spravili horný odhad d^d očakávaného počtu generovaní (kde d je počet rozmerov priestoru), kým nájdeme bod v polyédri. Taktiež sme vymysleli zrýchlenú verziu MVEE metódy, ktorá overuje príslušnosť vygenerovaných bodov v polyédri ešte pred zobrazením z jednotkovej gule do MVEE.

Okrem metód na generovanie bodov v polyédri sme sa v kapitole 2 zaoberali problémom optimálneho návrhu. Podľa článku [4] sme predstavili samotný problém a stručne predstavili algoritmy SAM, VEM, REX na riešenie problému optimálneho návrhu. Tiež sme podľa [4] sme čítateľovi ukázali súvis D–optimálneho návrhu a nájdenia MVEE pre daný polyéder.

Na záver sme v kapitole 3 popísali testovanie metód a výsledky testovania. V rámci implementácie testovača sme zaoberali výberom polyédrov, na ktorých budú algoritmy

testované. Nakoľko sme chceli zvoliť náhodné polyédre, v rámci diskusie o náhodnosti polyédrov sme ukázali, že už samotné poriadné formálne definovanie "náhodného" polyéhru je zaujímavé.

Metódy boli testované na "skoro náhodných" polyédroch vygenerovaním veľkého počtu bodov. Nakoľko pri veľkom počte vygenerovaných bodov vierohodnosť vygenerovaných rozdelení všetkých metód je vysoká, ako najdôležitejší ukazovateľ sme porovnávali rýchlosť generovania. Ukázali sme, že čas potrebný na vygenerovanie veľkého počtu bodov pomocou Metropolis–Hastings metód je so zväčšujúcim sa počtom rozmerov asymptoticky menší ako pri všetkých nami uvažovaných zamietacích metódach. Praktickým testovaním sme ukázali, že pre počet rozmerov do 10 s rastúcim počtom rozmerov rastie pomer $\frac{\lambda(T_{MVEE})}{\lambda(S)}$ aspoň exponenciálne, preto možno na prvý pohľad úplne voľný horný odhad d^d môže byť do určitej miery tesný.

3.6 Vplyv implementačných volieb

V tejto podkapitole odôvodníme niekoľko implementačných rozhodnutí, ktoré sme na začiatku spravili a ich dopad na prácu.

Výber jazyku: Jazyk Julia bol zvolený najmä kvôli rýchlosti výpočtov a prehľadnosti zdrojového kódu. Napriek týmto výhodám, počas implementácie testovača bolo potrebné vyriešiť niekoľko chýb spôsobených nedostatočnou a neaktuálnou dokumentáciou jazyka. Taktiež implementácia jednotlivých algoritmov nám trvala oveľa dlhšie ako sa pôvodne očakávalo. Obzvlášť pri prepisovaní implementácie algoritmu REX z jazyku R [4] sme narazili na niekoľko nepríjemností spôsobených nedostatočnou znalosťou jazyku R.

Avšak kedže po úspešnej implementácii niektoré výpočty trvali vyše desiatky hodín, považujeme jazyk Julia za vhodnú voľbu.

Voľba definície polyédrov: V rámci práce sme si zvolili definíciu polyédrov v H–reprezentácii ako $\{\mathbf{x} \mid A\mathbf{x} \geq b\}$. Táto voľba bola vcelku nešťastná, nakoľko všetky algoritmy, s ktorými sme pracovali používali ekvivalentnú definíciu $\{\mathbf{x} \mid A\mathbf{x} \leq b\}$. Touto voľbou bolo potrebné prepisovať jednotlivé vzorce Hit–and–Run generátora, Gibbsovho generátora, generovania polyédrov, čo vytvorilo priestor na výskyt chýb.

3.7 Možné rozšírenia práce

Vzhľadom k obmedzenému časovému rámcu nebolo možné venovať sa všetkým súvisiacim oblastiam do hĺbky. V tejto podkapitole si uvedieme niekoľko možných smerov, ako by sa dala daná práca rozšíriť.

Uvádzané algoritmy boli testované na "skoro náhodných" polyédroch. Ako možné rozšírenia práce prichádza do úvahy testovať spomínané algoritmy na iných polyédrov. Pri nahradení "skoro náhodných" polyédrov polyédrami z nejakej konkrétnej triedy prakticky relevantných polyédrov by bolo možné získať iné výsledky.

Ako súvisiace rozšírenie sa núka predefinovanie náhodných polyédroch. Možno vygenerovanie polyédru vo V–reprezentácii a následný prevod do H–reprezentácie môže teoreticky viesť ku triede polyédrov s väčšou H–reprezentáciou ako V–reprezentáciou. **TODO isto? zamyslieť sa ráno** Generovanie polyédrov vo V–repreznetácii iným spôsobom s následným prevodom do H–reprezentácie podľa nášho názoru nebude viesť k inému výsledku.

Ako ďalšie rozšírenie (spomenuté pri generovaní polyédrov, viď 3.1.2) možno namiesto prevádzania polyédrov hľadať dvojicu generátorov z rovnakého rozdelenia polyédrov, pričom jeden generuje polyédre v H–reprezentácii a druhý vo V-reprezentácii.

Literatúra

[1] Keith Ball. Ellipsoids of maximal volume in convex bodies. *arXiv:math/9201217*, September 1990. arXiv: math/9201217.

[2] Ming-Hui Chen and Bruce W. Schmeiser. General Hit-and-Run Monte Carlo sampling for evaluating multidimensional integrals. *Operations Research Letters*, 19(4):161–169, October 1996.

[3] Siddhartha Chib and Edward Greenberg. Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, 49(4):327–335, November 1995.

[4] Radoslav Harman, Lenka Filová, and Peter Richtárik. A Randomized Exchange Algorithm for Computing Optimal Approximate Designs of Experiments. *arXiv:1801.05661 [stat]*, January 2018. arXiv: 1801.05661.

[5] Radoslav Harman and Vladimír Lacko. On decompositional algorithms for uniform sampling from n-spheres and n-balls. *Journal of Multivariate Analysis*, 101(10):2297–2304, November 2010.

[6] Radoslav Harman and Vladimír Lacko. On decompositional algorithms for uniform sampling from n-spheres and n-balls. *Journal of Multivariate Analysis*, 101(10):2297–2304, November 2010.

[7] Leonid G. Khachiyan and Michael J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Mathematical Programming*, 61(1):137–159, August 1993.

[8] Viktor Lukáček and Radoslav Harman. Použitie metód analýzy zhlukov na konštrukciu návrhov experimentov. 2017.

[9] D. J. C. Mackay. Introduction to Monte Carlo Methods. In Michael I. Jordan, editor, *Learning in Graphical Models*, NATO ASI Series, pages 175–204. Springer Netherlands, Dordrecht, 1998.

[10] Jerrold H. May and Robert L. Smith. Random polytopes: Their definition, generation and aggregate properties. *Mathematical Programming*, 24(1):39–54, December 1982.

[11] Gareth O. Roberts and Jeffrey S. Rosenthal. Convergence of Slice Sampler Markov Chains. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):643–660, January 1999.

Zdrojový kód najrýchlejšieho generátora

return zeros(n,d)

TODO nahradiť menej rýchlim, ale správnym generátorom