UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY


# OPTIMÁLNY NÁVRH EXPERIMENTOV BEZ REPLIKÁCIÍ

DIPLOMOVÁ PRÁCA

2018 Bc. Jakub LETOVANEC

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

# OPTIMÁLNY NÁVRH EXPERIMENTOV BEZ REPLIKÁCIÍ

## DIPLOMOVÁ PRÁCA

| | |
|---|---|
| Študijný program: | Ekonomicko-finančná matematika a modelovanie |
| Študijný odbor: | 1114 Aplikovaná matematika |
| Školiace pracovisko: | Katedra aplikovanej matematiky a štatistiky |
| Vedúci práce: | Mgr. Lenka Filová, PhD. |

Bratislava 2018 **Jakub LETOVANEC**


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. Jakub Letovanec
**Študijný program:** ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
**Študijný odbor:** aplikovaná matematika
**Typ záverečnej práce:** diplomová
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Optimálne návrhy experimentov bez replikácií
*Optimal design of experiments without replications*

**Cieľ:** Cieľom práce je vypracovať teóriu a algoritmy výpočtu optimálnych návrhov experimentov bez replikácií a následne vytvoriť implementáciu navrhnutých algoritmov pre knižnicu OptimalDesign štatistického softvéru R.

**Vedúci:** Mgr. Lenka Filová, PhD.
**Katedra:** FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
**Vedúci katedry:** prof. RNDr. Daniel Ševčovič, CSc.

**Dátum zadania:** 30.01.2017

**Dátum schválenia:** 30.01.2017 prof. RNDr. Daniel Ševčovič, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

**Poďakovanie** Touto cestou chcem poďakovať svojej vedúcej diplomovej práce Mgr. Lenke Filovej, PhD. za trpezlivosť, za podnetné pripomienky a za ochotu odpovedať na každú z množstva mojich otázok. Ďakujem tiež rodine a priateľom za podporu a pomoc.

# Abstrakt

LETOVANEC, Jakub: Optimálny návrh experimentov bez replikácií [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: Mgr. Lenka Filová, PhD., Bratislava, 2018, 64 s.

V práci sa budeme venovať konštrukcii algoritmu pre nájdenie optimálneho návrhu experimentu bez replikácií meraní v jednom bode a s lineárnym ohraničením, napríklad na cenu experimentu. Uvedieme ideu pôvodného algoritmu, od ktorého sa dostaneme až k novému, originálnemu programu. Súčasťou práce je aj nájdenie metódy hľadania inicializačného návrhu, ktorý bude vstupom pre algoritmus hľadajúci optimálny návrh. Na záver otestujeme algoritmus na rôznych typoch príkladov, porovnáme ho s inými algoritmami a vyhodnotíme jeho funkčnosť.

**Kľúčové slová:** optimálny návrh, výmenný algoritmus, optimalita, efektivita, inicializačný návrh, exaktný návrh

# Abstract

LETOVANEC, Jakub: Optimal design of experiments without replications [Master thesis], Comenius university in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: Mgr. Lenka Filová, PhD., Bratislava, 2018, 64 p.

In this thesis, we will deal with the construction of an algorithm finding optimal design of an experiment without replications of observations in a single point, and with linear constraints, e.g. for experiment price. We will introduce an idea of an original algorithm, which will lead to a new, authentic program. Part of the thesis is also about finding a method computing an inicial design, which will be an input of the algorithm looking for an optimal design. In the end we will test our program on various examples, comparing it with different algorithms and evaluate its functionality.

**Keywords:** optimal design, exchange algorithm, optimality, efectivity, inicial design, exact design

# Obsah

# Zoznam obrázkov

# Úvod

Navrhovanie experimentov je súčasťou mnohých výskumov v rôznych oblastiach vedy. Pri všetkých experimentoch vyvstáva potreba tento experiment naplánovať. Tieto experimenty môžu byť však veľmi nákladné a naše zdroje, či už finančné, materiálne alebo časové, môžu byť do značnej miery obmedzené. Za týmto účelom vznikla potreba tieto návrhy optimalizovať tak, aby sme všetky potrebné obmedzenia spĺňali.

Základy modernej teórie navrhovania experimentov položil J. Kiefer [10] v druhej polovici 20. storočia. Za ten čas sa stihli rozvinúť mnohé spôsoby ich riešenia, od jednoduchých až po komplikované algoritmy.

Cieľom našej práce bude naprogramovať a rozvinúť jednu z najpopulárnejších metód na výpočet exaktných návrhov experimentov. KL-výmenný algoritmus sa po prvý krát objavil v publikácii od Atkinsona a Doneva [2] v roku 1989 ako modifikácia klasického Fedorovovho výmenného algoritmu [4] z roku 1972. Podstatou výmenného algoritmu je v každom kroku vymazať "zlý" bod návrhu a nahradiť ho "dobrým"[5]. My sa tento algoritmus pokúsime vylepšiť tak, aby bol schopný zohľadniť aj iné ohraničenia, ako len to na počet experimentov a zapojíme tak aj ohraničenie na cenu, či ľubovoľné iné ohraničenie. Budeme sa tak snažiť dosiahnuť efektívny program a poskytnúť rýchlu, ale aj dostatočne spoľahlivú metódu na hľadanie optimálneho návrhu.

# Kapitola 1

# Optimálny návrh experimentu

V tejto kapitole zadefinujeme štandardnú úlohu optimálneho navrhovania experimentov pozostávajúcu z lineárneho regresného modelu, matice návrhu experimentu a kritéria optimality. Podrobnejšie informácie sa môže čitateľ dozvedieť v zdrojoch [3], [6].

## 1.1 Lineárny regresný model

Predpokladajme, že chceme vykonať experiment, ktorého pozorovania sa dajú zapísať jednoduchým lineárnym regresným modelom

$$\mathbb{Y} = \mathbb{F}\theta + \varepsilon, \tag{1.1}$$

kde

$$\mathbb{Y} = (y_1, ..., y_n)^T$$

je stĺpcový vektor pozorovaní,

$$\mathbb{F}_{n\times m} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & ... & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & ... & f_m(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_n) & f_2(x_n) & ... & f_m(x_n) \end{pmatrix} = \begin{pmatrix} f^T(x_1) \\ f^T(x_2) \\ \vdots \\ f^T(x_n) \end{pmatrix} \tag{1.2}$$

je matica regresorov v bodoch meraní $x_i$, z množiny $\mathcal{X} = \{x_1, ..., x_n\}$ a $f_i(x)$ je známa funkcia. Funkcia $f(x)$ bude teda stĺpcový vektor

$$f(x) = (f_1(x), f_2(x), ..., f_m(x))^T.$$

Vektor $\theta$ obsahuje odhadované parametre

$$\theta = (\theta_1, ..., \theta_m)^T$$

a

$$\varepsilon = (\varepsilon_1, ..., \varepsilon_n)^T$$

je vektor chýb, nezávislých, normálne rozdelených, homoskedastických, s $E(\varepsilon) = 0$ a $Var(\varepsilon) = \sigma^2$.
Pre každé meranie $y_i, i \in \{1, ..., n\}$ teda platí

$$y_i = f^T(x_i)\theta + \varepsilon_i. \tag{1.3}$$

Našim cieľom je čo najpresnejšie odhadnúť neznámy vektor parametrov $\theta$. Jeho odhad $\hat{\theta}$ môžeme nájsť pomocou metódy najmenších štvorcov.

$$\hat{\theta} = \arg\min_{\theta} ||\mathbb{Y} - \mathbb{F}\theta||,$$

kde $||.||$ je Euklidovská norma. Potom platí, že

$$\hat{\theta} = (\mathbb{F}^T\mathbb{F})^{-1}\mathbb{F}^T\mathbb{Y}, \tag{1.4}$$

pričom $\hat{\theta}$ má strednú hodnotu $E(\hat{\theta}) = \theta$ a varianciu $Var(\hat{\theta}) = \sigma^2(\mathbb{F}^T\mathbb{F})^{-1}$.

## 1.2 Návrh experimentu

Našim cieľom je spomedzi všetkých možných meraní vybrať tie, ktoré nám dajú najviac informácie o hodnotách vektora neznámych parametrov. Tento výber nazývame návrhom experimentu. Pod pojmom exaktný návrh rozumieme predpis, ktorý nám hovorí, koľko meraní máme vykonať v jednotlivých bodoch z $\mathcal{X}$. Štandardne zapisujeme exaktný návrh pomocou tzv. matice návrhu $\xi_N$:

$$\xi_N = \begin{pmatrix} x_1 & ... & x_n \\ \omega_1 & ... & \omega_n \end{pmatrix},$$

kde $\omega_i$ je počet meraní v bode návrhu $x_i, i \in \{1, ..., n\}$ a spĺňa podmienku

$$\sum_{i=1}^{n} \omega_i = N. \tag{1.5}$$

$N$ je teda veľkosť návrhu, alebo celkový počet meraní. Body návrhu $x_1, x_2, ..., x_n$ patria do tzv. oblasti návrhu $\mathcal{X}$. Množinu všetkých návrhov na $\mathcal{X}$ označujeme $\Xi$. V našom prípade, keďže sa zaoberáme len návrhmi bez opakovania merania v jednom bode, počet meraní môže byť len 0 alebo 1, teda $\omega_i \in \{0, 1\}$.

Okrem exaktného návrhu zavedieme ešte pojem aproximatívneho návrhu. Matica exaktného návrhu $\xi_N$ môže na rozdiel od aproximatívneho (spojitého) návrhu obsahovať len celočíselné počty meraní. Aproximatívny návrh môžeme teda považovať za akúsi relaxáciu exaktného. Optimálny aproximatívny návrh sa dá použiť ako dobrá referencia k tomu, ako blízko sme sa dostali k teoretickému optimu. Jeho výhodou je tiež to, že sa dá jednoducho nájsť. Keďže predpokladáme, že hľadáme maximum konkávnej (resp. minimum konvexnej) funkcie (viď podkapitolu 1.5) na kompaktnej množine, môžeme použiť silnú teóriu konvexnej optimalizácie. Problém nastáva, keď sa snažíme z aproximatívneho návrhu dostať k exaktnému. Túto tému rozoberieme v kapitole 3.

## 1.3 Ohraničenia experimentu

V praktických situáciách sú často potrebné ohraničenia na počty meraní v experimente. Okrem ohraničenia na veľkosť návrhu $N$ sa môžeme stretnúť aj s inými typmi ohraničení.

Lineárne ohraničenia sa zadávajú v štandardnej forme $A\omega \leq b$, kde $A$ je matica typu $k \times n$ a $b$ je vektor dĺžky $k$. Hodnota $k$ označuje počet rôznych ohraničení. Môžu tu byť zahrnuté napríklad ohraničenia na cenu, množstvo zdrojov, požadovaná "vzdialenosť" medzi bodmi merania a podobne.

Predstavme si, že vykonanie experimentu v bode $x_i$ stojí $c_i$ jednotiek peňazí a celkové náklady na experiment nesmú presiahnuť $C$ peňažných jednotiek. Matica $A$

by mala v tom prípade nasledovný tvar:

$$A = (c_1, c_2, ..., c_n)$$

a druhá strana nerovnosti by bola jednoducho $b = C$.

Pre zakomponovanie už spomínaného ohraničenia na celkový počet meraní by stačilo pridať do matice $A$ riadok samých jednotiek a do vektora $b$ by sme pridali $N$.

$$A = \begin{pmatrix} c_1 & ... & c_n \\ 1 & ... & 1 \end{pmatrix}, b = \begin{pmatrix} C \\ N \end{pmatrix}.$$

Ak by sme chceli, aby počet meraní v $i$-tom bode nepresiahol $b_i$, do matice $A$ vložíme riadok s jednotkou len na $i$-tom mieste a do vektora $b$ pridáme prvok $b_i$.

Harman, Bachratá a Filová [8] uvádzajú ďalší typ ohraničení na zdroje. Uvážme situáciu, v ktorej môžeme v $i$-tom pokuse použiť len isté množstvo $r$-tého zdroja. Takéto obmedzenie zapíšeme nasledovne:

$$\sum_{i=1}^{n} a_{ri}\omega_i \leq b_r, \forall r \in \{1, ..., l\}, \tag{1.6}$$

kde $a_{ri}$ je spotreba $r$-tého zdroja v $i$-tom bode návrhu a $b_r$ je obmedzenie $r$-tého zdroja. Konštanta $l$ označuje počet zdrojov. O rôznych typoch ohraničení píšeme podrobnejšie v kapitole 4.1.

## 1.4 Informačná matica

Pri hľadaní optimálneho návrhu hrá kľúčovú úlohu informačná matica. Informačná matica $M$ návrhu $\xi_N$ je definovaná vzťahom

$$M(\xi_N) = \sum_{i=1}^{n} \omega_i f(x_i) f^T(x_i) \tag{1.7}$$

Variancia odhadovaného merania je

$$Var(\hat{y}(x)) = \sigma^2 f^T(x)(F^T F)^{-1} f(x).$$

Pre naše výpočty však budeme používať štandardizovanú variančnú funkciu vzhľadom na $\sigma^2$, ktorú definujeme nasledovne (viď [3]):

$$d(x, \xi_N) = f^T(x) M^{-1}(\xi_N) f(x) = \frac{N Var(\hat{y}(x))}{\sigma^2}. \tag{1.8}$$

## 1.5 Kritériá optimality

Našim cieľom je z navrhovaného experimentu vyťažiť čo najviac informácie. Bohužiaľ, veľkosť informácie sa v našom prípade nedá jednoznačne vyjadriť. Našu informáciu bude vyjadrovať funkcia $\Phi$, ktorá je závislá od informačnej matice $M$.

$$\Phi : M(\xi) \to \Phi[M(\xi)] \in \langle 0, \infty)$$

Túto funkciu nazývame kritérium optimality. Hovoríme, že návrh $\xi^*$ je $\Phi$-optimálny, ak platí, že

$$\xi^* = \arg\max_{\Xi} \Phi(M(\xi)). \tag{1.9}$$

V našej práci, tak ako v [7], budeme používať špeciálne verzie kritérií, nazývané informačné funkcie. Funkcia $\Phi : \mathcal{S}_+^m \to \langle 0, \infty)$ definovaná na množine pozitívne semidefinitných matíc $m \times m$ $(\mathcal{S}_+^m)$, sa nazýva informačná funkcia, ak to nie je nulová funkcia a spĺňa:

- izotonicita: $D - C \in \mathcal{S}_+^m \Rightarrow \Phi(C) \leq \Phi(D), \forall C, D \in \mathcal{S}_+^m$
- konkávnosť: $\Phi(\alpha C + (1-\alpha)D) \geq \alpha\Phi(C) + (1-\alpha)\Phi(D), \forall C, D \in \mathcal{S}_+^m, \alpha \in (0,1)$
- pozitívna homogénnosť: $\Phi(\alpha C) = \alpha\Phi(C), \forall C \in \mathcal{S}_+^m, \alpha \geq 0$
- polospojitosť zhora: Množiny $\{C \in \mathcal{S}_+^m; \Phi(C) \geq c\}$ sú uzavreté pre všetky $c \in \mathbb{R}$

ďalej uvedieme najpoužívanejšie kritériá optimality v ich konkávnej a pozitívne homogénnej verzii.

- **$D$-optimalita**

  Najpoužívanejším je kritérium $D$-optimality, ktoré je definované nasledovne:

  $$\Phi_D(M(\xi)) = |M(\xi)|^{1/m}, \tag{1.10}$$

  kde symbolom $|.|$ budeme označovať determinant matice a kde $m$ je rozmer vektora neznámych parametrov.

- **$A$-optimalita**

  Kritérium $A$-optimality definujeme ako

  $$\Phi_A(M(\xi)) = \frac{m}{tr[M^{-1}(\xi)]} \tag{1.11}$$

  pre regulárne informačné matice. Pre singulárne $M$, je $\Phi_A = 0$

- **$G$-optimalita**

  Na záver uvedieme $G$-optimalitu. Jej kritérium vyjadríme pomocou vzťahu

  $$\Phi_G(M(\xi)) = f^T(x)M^{-1}(\xi_N)f(x), \tag{1.12}$$

  čo ale nie je nič iné ako štandardizovaná variančná funkcia (1.8). Pre singulárne informačné matice je hodnota kritéria opäť nulová.

### 1.5.1 Efektivita návrhu

Pre porovnanie kvality návrhu je dôležitá aj efektivita, alebo tzv. eficiencia návrhu. Tú popisuje [11] nasledovným vzťahom:

$$\text{eff}(M(\xi)|\Phi) = \frac{\Phi(M(\xi))}{\max_{\zeta\in\Xi}\Phi(M(\zeta))} \tag{1.13}$$

Konkrétne pre $D$-optimalitu to bude

$$\text{eff}(M(\xi)|\Phi_D) = \left\{\frac{|M(\xi)|}{|M(\xi^*)|}\right\}^{1/m}. \tag{1.14}$$

Platí, že $\text{eff}(M(\xi)|\Phi) \in \{0, 1\}$ a hovorí nám o tom, koľko informácie sa nám podarilo získať. Efektivita eff$= 0,5$ teda vyjadruje, že pomocou daného návrhu vieme získať polovicu informácie o parametroch v porovnaní s optimálnym návrhom.

# Kapitola 2

# Výmenné algoritmy

V tejto kapitole popíšeme hlavnú myšlienku výmenných algoritmov. Najprv uvedieme všeobecný princíp výmenných algoritmov a následne sa budeme venovať KL-výmennému algoritmu.

## 2.1 Idea výmenných algoritmov

Výmenné algoritmy sú iteratívne numerické procesy, ktoré vychádzajú z počiatočného $N$-prvkového návrhu. Následne zámenou kandidátov na odobranie a pridanie zlepšujú hodnotu vybraného kritéria optimality. Odobratím a pridaním rovnakého počtu bodov sa teda zachová počet bodov $N$ z počiatočného návrhu. Pre $D$-optimalitu je pri výbere kandidátov a realizácii výmeny dôležitý determinant informačnej matice. Namiesto hľadania determinantu však vieme použiť výpočtovo menej náročný spôsob pre nájdenie optima. Pre nejaké $i \geq 0$, ktoré označuje počet doteraz vykonaných iterácií a pre $j \in \{1, 2, ..., n\}$ platí:

$$det(M(\xi_{i+1})) = det(M(\xi_i))(1 + f(x_j)M^{-1}(\xi_i)f^T(x_j)) = det(M(\xi_i))(1 + d(x_j, \xi_i)), \tag{2.1}$$

kde $d(x_j, \xi_i)$ je variančná funkcia. Keďže $det(M(\xi_i))$ je známy z predošlej iterácie, zo vzťahu (2.1) vyplýva, že pre maximalizáciu determinantu informačnej matice nám stačí maximalizovať variančnú funkciu.

Atkinson a Donev [3] uvádzajú vzťahy, ktorými popisujú zmenu informácie pri jednej výmennej iterácii. Nech $i(i \geq 0)$ je počet doteraz vykonaných iterácií. Nech

$c_k$ a $c_l$ spĺňajú:

1. $c_l = (N+1)^{-1}, c_k = 0$ ak bod $x_l$ je pridaný do návrhu
2. $c_k = (N+1)^{-1}, c_k = 0$ ak bod $x_k$ je odstránený z návrhu
3. $c_k = c_l = (N+1)^{-1}$ ak bod $x_l$ je vymenený za bod $x_k$

Nech príslušné riadky matice regresorov sú $f_k^T = f^T(x_k)$ a $f_l^T = f^T(x_l)$. Potom platí, že

$$M(\xi_{i+1}) = \frac{1-c_l}{1-c_k} M(\xi_i) + \frac{1}{1-c_k}(c_l f_l f_l^T - c_k f_k f_k^T) \tag{2.2}$$

$$|M(\xi_{i+1})| = \left[ \left\{ 1 + \frac{c_l}{1-c_l} d(x_l, \xi_i) \right\} \left\{ 1 - \frac{c_k}{1-c_k} d(x_k, \xi_i) \right\} + \frac{c_k c_l}{(1-c_l)^2} d^2(x_k, x_l, \xi_i) \right] \left( \frac{1-c_l}{1-c_k} \right)^m |M(\xi_i)| \tag{2.3}$$

$$M^{-1}(\xi_{i+1}) = \frac{1-c_k}{1-c_l} \left\{ M^{-1}(\xi_i) - \frac{M^{-1}(\xi_i) A M^{-1}(\xi_i)}{qz + c_l c_k d^2(x_l, x_k, \xi_i)} \right\} \tag{2.4}$$

kde

$$d(x_l, x_k, \xi_i) = f_l^T M^{-1}(\xi_i) f_k$$

$$q = 1 - c_l + c_l d(x_l, \xi_i)$$

$$z = 1 - c_l - c_k d(x_k, \xi_i)$$

a

$$A = c_l z f_l f_l^T + c_l c_k d(x_l, x_k, \xi_i)(f_l f_k^T + f_k f_l^T) - c_l q f_k f_k^T.$$

Teda ak pridáme bod $x_l$ do $N$-bodového návrhu s informačnou maticou $M(\xi_i)$, determinant informačnej matice bude

$$|M(\xi_{i+1})| = \{1 + d(x_l, \xi_i)\} \left( \frac{N}{N+1} \right)^m |M(\xi_i)|,$$

a ak z $N+1$ bodového návrhu s informačnou maticou $M(\xi_i)$ odstránime bod $x_k$, potom

$$|M(\xi_{i+1})| = \{1 - d(x_k, \xi_i)\} \left( \frac{N+1}{N} \right)^m |M(\xi_i)|.$$

Ak spravíme túto výmenu naraz, determinant informačnej matice bude

$$|M(\xi_{i+1})| = [\{1 - d(x_k, \xi_i)\}\{1 + d(x_l, \xi_i)\} + d^2(x_k, x_l, \xi_i)]|M(\xi_i)|.$$

Pôvodný výmenný algoritmus [4] vezme kandidátov na odobranie, kandidátov na pridanie a navzájom ich povymieňa, čím vyskúša všetky možnosti kombinácií. Pre každú dvojicu vypočíta veľkosť zmeny determinantu a vyberie tú, ktorá prinesie najväčšie zvýšenie jeho hodnoty. Kandidátov však môže byť veľa a počítanie determinantu pre každú ich kombináciu môže byť časovo náročné. Tak isto, veľa z nich môže byť úplne zbytočných. O zefektívnenie tejto metódy sa snaží KL-výmenný algoritmus.

## 2.2 KL-výmenný algoritmus

KL-výmenný algoritmus, na rozdiel od pôvodného výmenného algoritmu, neuskutočňuje všetky možné výmeny. Kandidátov na odobranie a pridanie najprv zoradí podľa hodnoty variančnej funkcie danej vzťahom (1.8).

Porovnávanie na základe variančnej funkcie zoradí body rovnako, ale je výpočtovo jednoduchšie ako výpočet kritéria optimality pre každý bod. Keďže sa snažíme výmenou čo najviac zvýšiť determinant matice, kandidátov na odobratie zoradíme tak, aby čo najmenej znižovali jeho hodnotu. Naopak, od kandidátov na pridanie očakávame čo najväčšie zvýšenie hodnoty determinantu. Algoritmus následne vyberie $k$ najlepších kandidátov na odobratie a $l$ najlepších kandidátov na pridanie, a uskutoční výmeny len medzi nimi. Môže sa tak radikálne zmenšiť počet kombinácií a urýchli sa výpočet. Od pomenovania premenných $k$ a $l$ je odvodený aj názov algoritmu.

Výber parametrov $k$ a $l$ závisí od viacerých špecifík problému. Podľa [3] je to napríklad veľkosť návrhu $N$. Nemá zmysel, aby bolo $k > N$, keďže kandidátov na odobratie môže byť maximálne $N$. S narastajúcim počtom parametrov je vhodné zvýšiť aj $k$. Hodnota $l$ narastá s veľkosťou problému, nezvykne však presiahnuť $\frac{N}{2}$.

# Kapitola 3

# Inicializačný návrh experimentu

Pre KL-výmenný algoritmus je jedným z potrebných vstupov inicializačný, alebo počiatočný návrh experimentu, od ktorého sa budeme vylepšovaním snažiť nájsť optimálny návrh. V tretej kapitole rozoberieme metódy, akými sa tento návrh dá nájsť.

## 3.1 Zaokrúhľovanie aproximatívneho návrhu

Riešením relaxovaného problému dostaneme aproximatívny návrh, ten má teda najlepšiu možnú hodnotu kritéria optimality. Jeho problém je však to, že je v praxi nevyužiteľný. V skutočnom svete totiž v podstate každú jednotku diskretizujeme, či už je to jednotka času, dĺžky, hmotnosti, skrátka čohokoľvek. V našom prípade je to o to väčší problém, že vyžadujeme celočíselné hodnoty meraní, ba čo viac, my meranie buď vykonáme, alebo nevykonáme. Preto vyžadujeme hodnoty návrhu len 0 a 1. Na zaokrúhľovanie existuje niekoľko spôsobov, my sa budeme venovať niektorým z nich.

### 3.1.1 Kónické programovanie druhého rádu

Začneme s výpočtom aproximatívneho návrhu. Na to použijeme funkciu `od.SOCP` z knižnice `OptimalDesign` [9], naprogramovanej v prostredí programu `R`. Táto funkcia počíta aproximatívne optimálne návrhy experimentov pre modely s lineárnymi ohraničeniami na váhy pomocou kónického programovania druhého rádu.

Túto metódu rozoberajú G. Sagnol a R Harman [12]. Ich problém je však oproti nášmu všeobecnejší. Zaoberajú sa prípadom, kedy experimentátora zaujíma odhad podsystému parametrov $\vartheta = K^T\theta$, kde $K$ je matica $m \times k (k \leq m)$ plnej hodnosti (hodnosť $K = k$). Informačná matica má v tomto prípade tvar $(K^T M^- K)^{-1}$ (ak stĺpcový priestor $K$ je podmnožinou stĺpcového priestoru $M$). Pod symbolom $M^-$ rozumieme zovšeobecnenú inverznú maticu $M$. Analógiou ku kritériu $D$-optimality v tomto prípade je kritérium $D_K$-optimality definované nasledovne:

$$\Phi_{D|K}(M(\xi)) = (detK^T M^- K)^{\frac{-1}{k}}. \tag{3.1}$$

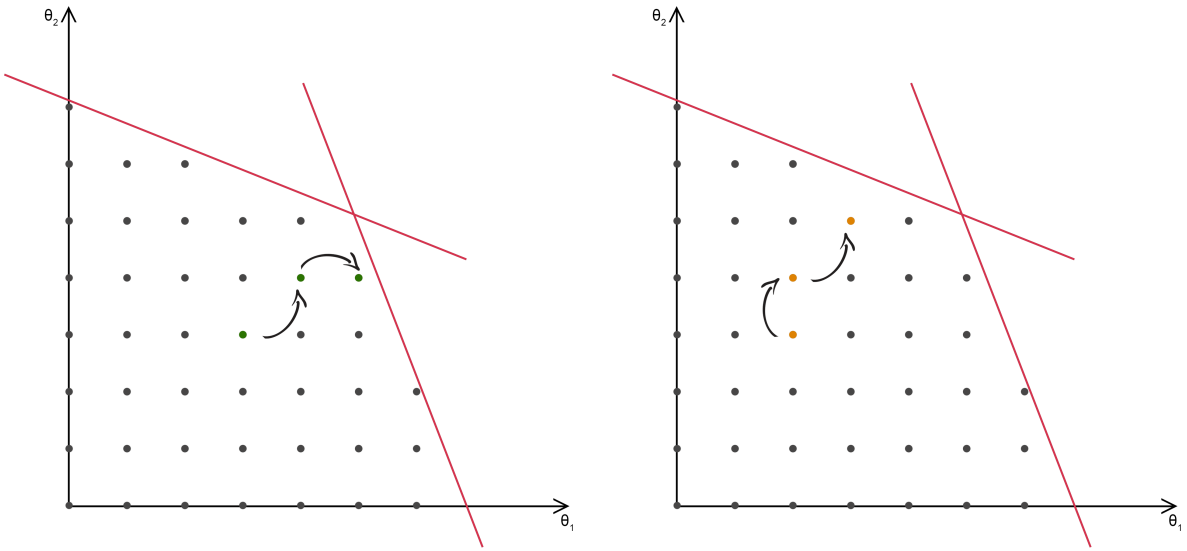
Ak ak stĺpcový priestor $K$ nie je podmnožinou stĺpcového priestoru $M$, hodnota kritéria je rovná nule. Pre náš prípad jednoducho položíme maticu $K$ rovnú identite rozmerov $m \times m$ a dostaneme tak informačnú maticu a kritérium D-optimality zhodné s našim.

Teraz si povedzme niečo o tom, čo je to úloha kónického programovania druhého rádu, alebo Second order cone programing (SOCP). V úlohe SOCP podľa [12] maximalizujeme hodnotu lineárnej funkcie $f^T x$ cez všetky $x$ z množiny $S = \{x \in \mathbb{R}^n : \forall i = 1, 2, ..., N_c, ||G_i x + h_i|| \leq c_i^T x + d_i\}$, kde $G_i, h_i, c_i$ a $d_i$ sú matice (resp. vektory) vhodných rozmerov. Sagnol a Harman [12] uvádzajú definíciu, za akých podmienok pre vlastnosti týchto matíc je úloha tzv. SOC reprezentovateľná. Následne sa takéto úlohy dajú efektívne riešiť pomocou optimalizačných metód vnútorného bodu.

Teraz už môžeme uviesť SOCP formuláciu kritéria $D$-optimality, ktorá vychádza z formulácie pre $D_K$-optimalitu v článku [12]:

$$\begin{aligned}
\max_{\xi \in \Xi} \Phi_D(M(\xi)) = \max_{\omega, Z_i, t_{ij}, J} \quad & \prod_{j=1}^{m} (J_{j,j})^{\frac{1}{m}} \\
s.t. \quad & \sum_{i=1}^{n} f_i^T Z_i = J \\
& J \text{ je dolná trojuholníková matica} \\
& ||Z_i e_j||^2 \leq t_{ij}\omega_i (i \in \{1, ..., n\}, j \in \{1, ..., m\}) \\
& \sum_{i=1}^{n} t_{ij} \leq J_{j,j} (j \in \{1, ..., m\}) \\
& t_{ij} \geq 0 (i \in \{1, ..., n\}, j \in \{1, ..., m\}) \\
& \omega \in \mathcal{W}
\end{aligned} \tag{3.2}$$

Vektor $Z_i$ je $i$-ty stĺpec matice $Z_{n\times k}$. Množina $\mathcal{W}$ je množina všetkých prípustných vektorov váh $\omega$ a $t_{ij}$ je skalár. Riešením tejto úlohy vypočítame aproximatívny návrh. Počet vykonaných meraní v bode $x_i$ ($\omega_i$) preto tento krát nie je iba 0 alebo 1, ale môže nadobudnúť hodnotu z celého intervalu $\langle 0, 1\rangle$. Výsledkom procedúry je ale väčšinou jednoznačne určený návrh, pre lepší výsledok potrebujeme väčšie množstvo štartovacích návrhov. Z rôznych bodov môže totiž algoritmus dokonvergovať k rôznym riešeniam. Problém ilustruje obrázok 3.1.




**Obr. 3.1:** Z rôznych bodov môže algoritmus dokonvergovať k rôznym lokálnym optimám

Z toho dôvodu je nutné do zaokrúhľovania pridať istú náhodnosť. Na zaokrúhľovanie sme vyskúšali niekoľko metód, za všetky môžeme spomenúť napríklad *pipage rounding* [1].Pre potrebu dodržania ohraničení nám však často nekonvergovali k použiteľnému riešeniu. Z toho dôvodu sme boli nútení navrhnúť vlastný algoritmus.

### 3.1.2 Metóda zaokrúhľovania aproximatívneho návrhu

Idea našej metódy zaokrúhľovania tkvie v zachovaní čo najviac informácie z aproximatívneho návrhu. Ten totiž v mnohých prípadoch rozhodne tak, že v niektorých bodoch sa má merať práve jeden krát, alebo je to číslo veľmi blízke jednotke. Tieto body sme sa preto rozhodli zachovať. Merania v zvyšných bodoch následne položíme rovné nule.

V tejto situácii však s najväčšou pravdepodobnosťou nebudeme spĺňať ohraničenie na počet meraní $N$, nakoľko nám bude chýbať niekoľko meraní. Z tohoto dôvodu musíme vybrať niekoľko bodov, v ktorých sa experiment nevykonáva a v týchto bodoch experiment vykonať.

Keďže očakávame, že neskôr budeme na výpočty používať viac takýchto inicializačných návrhov, výstup z nášho algoritmu by nemal byť deterministický. Z tohoto dôvodu dopĺňame body návrhu náhodne. Deje sa to tak, že sa vygeneruje náhodná $p$-tica pozícií vo vektore $\omega$, na ktorých sa nachádza nula, kde $p$ je rozdiel medzi požadovanou veľkosťou návrhu $N$ a veľkosťou zaokrúhleného aproximatívneho návrhu. Na tieto pozície sa následne doplní jednotka a testuje sa, či takýto návrh spĺňa ohraničenia. Ak áno, použijeme ho ako inicializačný, ak nie, generujeme novú $p$-ticu.

Takáto metóda však môže viesť k niekoľkým úskaliam.

- Návrh s náhodne doplnenými bodmi už nemusí spĺňať ohraničenia $A\omega \leq b$ zo zadania. Preto je dôležité kontrolovať, či ich spĺňa a prípadne vygenerovať nové body návrhu.

- Môže sa stať, že aj pri menej ako $N$ bodoch návrhu je nejaký parameter až príliš blízko svojmu ohraničeniu a neexistuje bod, ktorý by sme mohli doplniť tak, aby návrh spĺňal toto ohraničenie. V takom prípade sa ponúka jednoduché riešenie, kde skrátka vymeníme najdrahšie meranie za najlacnejšie možné a doprajeme tak návrhu viac voľnosti.

# Kapitola 4

# Výmenný algoritmus pre modely bez replikácií

Problémom štandardného KL-výmenného algoritmu je, že berie do úvahy len jediné ohraničenie, a to je veľkosť štartovacieho návrhu $N$. Ohraničenia na cenu, či iné obmedzenia zadané v tvare $A\omega \leq b$ nerieši. Našou úlohou bolo preto tieto obmedzenia zakomponovať do algoritmu. Našim cieľom bude skonštruovať algoritmus, ktorý po zadaní štartovacieho návrhu bude výmenou bodov zlepšovať hodnotu kritéria optimality a zároveň dodržiavať ohraničenia:

$$\begin{aligned} \xi^* = \arg\max_{\Xi} \quad & \Phi(M(\xi)) \\ A\omega \quad & \leq b \\ \omega_i \quad & \in \{0,1\}, i \in \{1,...,n\} \end{aligned} \tag{4.1}$$

## 4.1 Lineárne ohraničenia na váhy

Vo všeobecnom prípade môže matica $A$ obsahovať obmedzenia na súčty meraní v nejakých skupinách bodov, rôzne počty meraní v konkrétnych bodoch a podobne. O rôznych typoch ohraničení sme písali už v prvej kapitole. Teraz si rozoberieme niekoľko typov ohraničení, s ktorými sme sa stretávali v našej práci a aplikujeme ich v našom algoritme:

- **Ohraničenie na počet meraní v jednom bode**

  Požadujeme, aby počet meraní v bode $\omega_i$ bol menší, alebo rovný $b_i$. Takéto ohraničenie sa dá zapísať v maticovom tvare pomocou identickej matice, a to nasledovne:

$$\begin{pmatrix} 1 & 0 & & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ & & \ddots & & \\ 0 & & ... & 0 & 1 \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{4.2}$$

  Vektor $b$ obsahuje maximálne počty meraní v danom bode.

  Výhodou nášho typu problému je, že vieme, že $b_i = 1, \forall i = 1, 2, ..., n$. Vďaka tomuto faktu môžeme z nášho programu celú túto časť obmedzení vynechať a v priebehu algoritmu len meniť počty meraní podľa potreby z 0 na 1 a naopak.

- **Ohraničenie na súčet meraní**

  Ďalším typom ohraničenia je ohraničenie na celkový počet meraní (1.5), kde celkový počet meraní nesmie presiahnuť $N$. Vektorovo sa dá takéto ohraničenie zapísať nasledovne:

$$(1, 1, ..., 1)_{1\times n} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{pmatrix} \leq N \tag{4.3}$$

  Takéto obmedzenie môžeme tiež vynechať, keďže náš algoritmus počas výpočtov zachováva počet meraní z inicializačného návrhu, preto pokiaľ už ten spĺňa túto podmienku, nemá pre nás zmysel sa ňou ďalej zaoberať.

- **Všeobecné lineárne ohraničenia**

  Posledným typom ohraničení, ktorými sa budeme zaoberať, sú ostatné - všeobecné ohraničenia, do ktorých spadá napríklad ohraničenie na cenu experimentov a pod. Tieto ohraničenia už zaujímajú aj nás. Označíme ich nasledovne:

$$A'\omega \leq b', \tag{4.4}$$

pričom podotýkame, že $A'$ ani $b'$ už nepodliehajú žiadnemu pravidlu, ako to bolo v predošlých prípadoch. Musia mať ale požadované rozmery.

Vo všeobecnom prípade by sa ohraničenia (4.2), (4.3) a (4.4) skombinovali do jednej podmienky v tvare $A\omega \leq b$, v našom prípade sa však podmienky zredukujú len na (4.4).

# Kapitola 5

# Príklady

V poslednej kapitole uvedieme príklady riešené pomocou nami naprogramovaného KL-výmenného algoritmu a výsledky porovnáme s riešeniami iných algoritmov. Funkcie, ktoré sme naprogramovali pre potrebu výmenného algoritmu sú popísané v prílohe. Experimenty boli vykonané v programe RStudio (Version 1.1.383), s verziou R 3.2.2 na operačnom systéme Windows 10 nainštalovanom na SSD ADATA Premier Pre SP900, s procesorom Inter(R) Core(TM) i5-3230M CPU @ 2.60GHz a RAM 8GB.

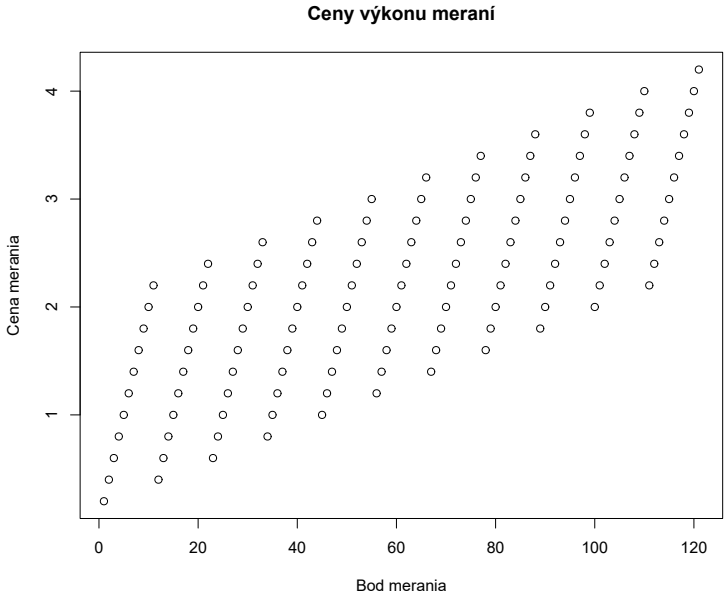
## 5.1 $D$-optimálny návrh na kvadratickom modeli

Ako prvý testovací príklad zvolíme ukážkový príklad z knižnice `OptimalDesign`. Budeme uvažovať kvadratický model $y(x) = x_1 + x_2 + x_1^2 + x_2^2 + x_1x_2$ na štvorcovej sieti $(-1, -0.8, -0.6, ..., 0.8, 1)^2$. Keďže sa pohybujeme v dvojrozmernom priestore, každý bod návrhu $x_i = (x_{i,1}, x_{i,2})$ bude mať dve zložky. V poradí $i$-ty riadok $f^T(x_i)$ matice $F$ (1.2) bude mať tvar

$$f^T(x_i) = (1, x_{i,1}, x_{i,2}, {x_{i,1}}^2, {x_{i,2}}^2, x_{i,1}x_{i,2}).$$

Matica $F$ preto bude mať $n = 121$ riadkov a $m = 6$ stĺpcov. Matica $A$ sa bude skladať z "cien" experimentov v jednotlivých bodoch návrhu, daných predpisom $c_{i_1,i_2} = (i_1 + 1, 1) + (i_2 + 1, 1)$. Ich vizualizáciu môžeme vidieť na obrázku 5.1.

Vektor $b$ bude obsahovať len jeden prvok a to ohraničenie na súčet cien experimentov $B = 28$. Náš algoritmus bude následne riešiť problém (4.1).




**Obr. 5.1:** Graf cien výkonu experimentov v jednotlivých bodoch návrhu

Testovanie algoritmu bude prebiehať nasledovne: Najprv vypočítame optimálny návrh pomocou funkcie `od.MISOCP`. Následne zbehne niekoľko opakovaní experimentov, v ktorých vykoná KL-výmenný algoritmus pre požadovaný počet inicializačných návrhov. Z nich vyberie ten najlepší a zapíše ho. Na záver vypočíta efektivitu tohoto návrhu.

Test sme vykonali so vstupnými hodnotami:

- $k = 10, l = 10$
- $A, b, F$ zo zadania príkladu 1
- Počet experimentov $NumOfExp = 100$
- Maximálny počet inicializačných návrhov $inic.d.max = 10$

Znamená to, že sme vykonali 100 experimentov a v každom jednom sme vybrali najlepší návrh spomedzi 10 štartovacích návrhov pre KL-výmenný algoritmus. S priemerným časom 0,3994 sekundy dosiahol program priemernú efektivitu 98,543%. Výpočet optimálneho návrhu pomocou `od.MISOCP` trval až 93 sekúnd.

Následne môžeme testovať, ako sa zmení výsledok, ak budeme narábať so vstupnými parametrami. Predpokladali sme, že so zvýšením počtu inicializačných návrhov

na 50 môžeme očakávať efektívnejší návrh, ale za cenu zvýšenia výpočtového času. Tento predpoklad sme experimentom potvrdili. Čas výpočtu narástol na 0,95 sekundy, ale efektivita sa zlepšila len minimálne. Skúsili sme preto znížiť počet návrhov na $inic.d.max = 5$. Výsledky boli veľmi podobné tým z prvého pokusu.

Vzhľadom na to, že čas nášho algoritmu je v porovnaní s MISOCP zanedbateľný, skúsili sme mu počet inicializačných návrhov ešte zvýšiť, tento krát na 100. Oproti pokusu s 50 inicializačnými návrhmi sa nám čas zvýšil o polovicu, ale zlepšenie je zanedbateľné. Výsledky sú zhrnuté v tabuľke 5.1.

**Tabuľka 5.1:** Výsledky testov pre rôzne počty inicializačných návrhov

| *inic.d.max* | **priem. čas** | **priem. efektivita** |
|---|---|---|
| 5 | 0,3832 | 98,264% |
| 10 | 0,3994 | 98,543% |
| 50 | 0,95 | 99,018% |
| 100 | 1,461 | 99,102% |

## 5.2 Lokálny $D$-optimálny návrh v nelineárnom modeli

Ďalší príklad, na ktorom sme testovali náš program, je príklad z článku [13] od Wrighta, Sigala a Bailera. Jeho zadanie je nasledovné: Hľadáme optimálne časy vykonávania experimentov v modeli, ktorý dáva do súvislosti čas a vnútornú koncentráciu chemickej látky (fluoraténu) v organizme. Priemerná vnútorná koncentrácia látky v čase $t$ je daná vzorcom

$$\mu_t(\theta_1, \theta_2) = \frac{\theta_1}{\theta_2}(e^{-\theta_2 max\{t-72,0\}} - e^{-\theta_2 t}), \tag{5.1}$$

kde parametre $\theta_1$ a $\theta_2$ zodpovedajú absorbčným a eliminačným pomerom. Sled experimentov sa začne počiatočným experimentom v čase $s$ a bude trvať nasledovných 144 hodín. Preto oblasť návrhu $\mathcal{X} = \{0, 1, 2, ..., 144\}$. Čas $t$ výkonu experimentu bude znamenať počet hodín, ktoré uplynuli od času $s$. Pre potrebu linearizácie modelu navrhuje [13] použiť hodnotu $\theta_2 = 0,2381$, od parametra $\theta_1$ model nezávisí,

keďže je v ňom lineárny. Riadky matice $F$ budú obsahovať hodnoty

$$f_{l(t)} = \nabla \mu_t(\theta_1, \theta_2), \tag{5.2}$$

kde $t \in \mathcal{X}$ je bod návrhu zodpovedajúci času experimentu, $l(t) = t+1$ je index bodu návrhu a symbol $\nabla$ označuje gradient.

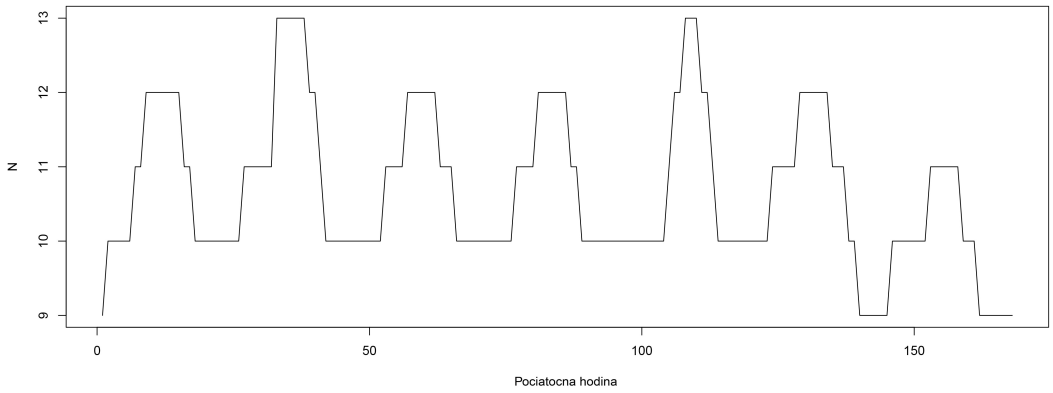
V zadaní príkladu je tiež uvedené, že v časoch $t = 0$, $t = 72$ a $t = 144$ je nutné vykonať experiment. Túto podmienku môžeme do nášho algoritmu implementovať viacerými spôsobmi. Jeden spôsob je taký, že do matice lineárnych ohraničení $A$ pridáme dva ďalšie riadky, pričom jeden by obsahoval samé nuly a na miestach 1, 73 a 145 by mal jednotku, druhý by na rovnakých miestach obsahoval -1. Do vektora $b$ by tiež pribudli dve hodnoty a to 3 a -3. Takáto úprava by zaručila, že v týchto troch bodoch sa experiment určite vykoná. Príklad by sa tým však radikálne skomplikoval (na komplikácie poukazujú experimenty v podkapitole 5.3.3). Druhá možnosť je do algoritmu vložiť počiatočný návrh s jednotkami len na požadovaných miestach. Ostatné jednotky sa doplnia náhodne tak, aby návrh spĺňal ohraničenie na celkovú cenu experimentu a zároveň napĺňal požadovanú veľkosť experimentu $N$. Týmto spôsobom sa teda nahradí krok hľadania počiatočného návrhu našou metódou zaokrúhľovania aproximatívneho návrhu. Výhodou tohoto spôsobu je fakt, že do matice $A$ nemusíme pridávať ďalšie riadky ohraničení a komplikovať tak výpočty.

Matica (vektor) cien $A$ zodpovedá cenám výkonu experimentu v jednotlivých hodinách, respektíve hodinovým mzdám v daných časoch. Možné mzdy sú nasledovné: v štandardnej pracovnej dobe (Pondelok - Piatok, 8:00 - 17:00) je cena za výkon experimentu rovná $1c$. Počas víkendu ( Piatok 19:00 - Pondelok 6:00) je mzda dvojnásobná, teda $2c$. Vo všetkých ostatných časoch je mzda rovná $1{,}5c$. Maximálna povolená cena experimentu vyjadrená ohraničením $b$ je $13c$.

Keďže experiment môže začať v ktorejkoľvek hodine v týždni, náš algoritmus pre hľadanie optimálneho návrhu sme museli spustiť pre každý jeden prípad, teda 168 krát.

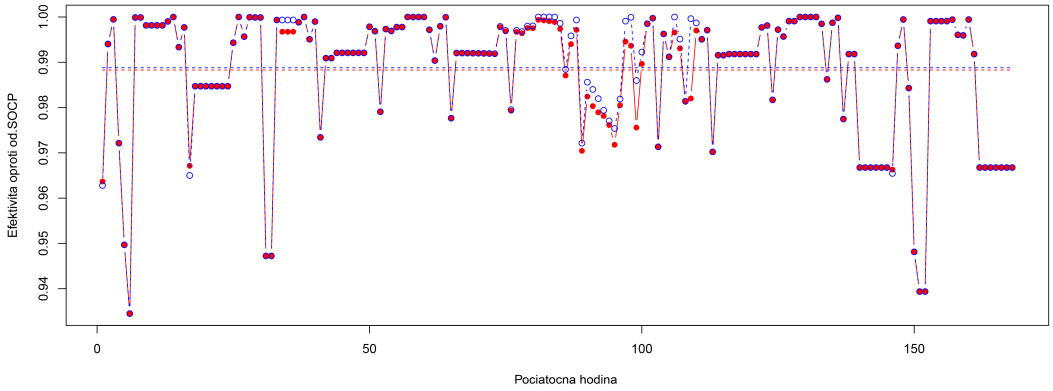
Jednou z vlastností KL výmenného algoritmu je, že mu musíme zadať požadovanú veľkosť návrhu $N$. Tú bohužiaľ nevieme. Vieme, že $N \leq 13$, pričom 13 bodov môže mať sled experimentov vykonávaných len v časoch štandardnej pracovnej doby. Pre dolné ohraničenie sme sa experimentálne dopracovali k hodnote 9. Museli sme

preto hľadať optimálny návrh pre všetky tieto hodnoty $N$. Pre $N = 9$ a $N = 10$ zbehol algoritmus vždy. Pre ďalšie hodnoty $N$ už tomu tak nebolo. V niektorých prípadoch totiž program nevedel nájsť takých $N$ bodov, aby návrh spĺňal ohraničenia a mohol tak byť použitý ako inicializačný. Bolo teda nutné skúšať, aký veľký návrh je pre jednotlivé $s$ možné použiť. Príslušné veľkosti návrhu v závislosti od začiatku experimentu vidíme na obrázku 5.2.




**Obr. 5.2:** Veľkosť návrhu $N$ pre jednotlivé počiatočné časy $s$, v ktorých sme dosiahli najvyššiu efektivitu

Náš algoritmus sme porovnávali s algoritmom `od.MISOCP` [9]. Výsledky sú znázornené na nasledujúcom grafe (Obr. 5.3).



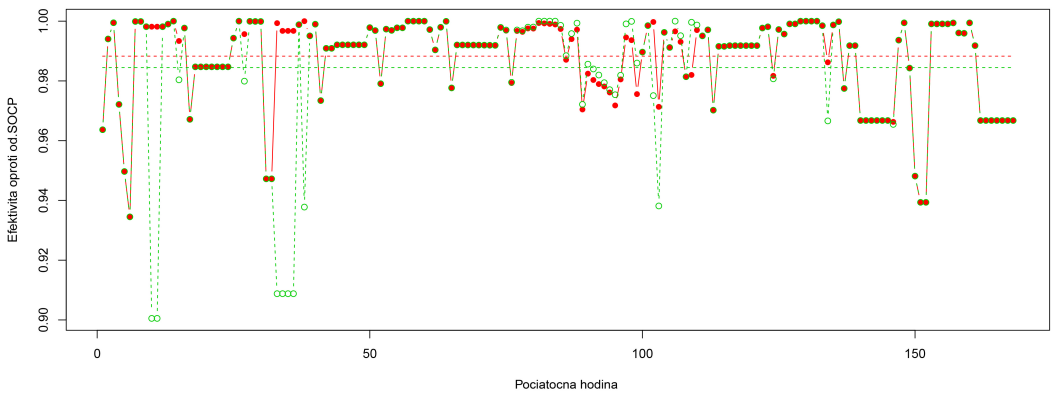

**Obr. 5.3:** Efektivity nášho KL výmenného algoritmu (červenou) a `od.MISOCP` (modrou) v porovnaní s optimálnym aproximatívnym návrhom

Obmedzením pre KL výmenný algoritmus bolo stanovenie maximálneho počtu

inicializačných návrhov na 10. Program `od.MISOCP` mal na výpočty stanovený maximálny čas 120 sekúnd. Môžeme si všimnúť, že priemerná efektivita oboch algoritmov je takmer rovnaká (prerušované čiary sa skoro prekrývajú). V niektorých prípadoch je lepší algoritmus `od.MISOCP`, no prekvapivo v niektorých víťazí náš algoritmus. V týchto prípadoch sa algoritmus využívajúci kónické programovanie určite nedostal do optima v stanovenom čase.

Hovoriť o čase výpočtov pre náš algoritmus je trochu komplikované. Pre každý štartovací čas $s$ sme museli spustiť algoritmus pre rôzny počet veľkostí návrhov $N$. V niektorých prípadoch sme teda spúšťali program len dvakrát, v iných až päťkrát. Ak by sme pre každý počiatočný čas sčítali časy jednotlivých realizácií výpočtov a následne spravili priemer, dostali by sme sa k hodnote 8,4 sekundy. Pre `od.MISOCP` bol priemerný čas 47,5 sekundy, pričom zo 168 pokusov až 53 krát skončil z dôvodu nedostatku času (dostal sa k ohraničeniu 120 sekúnd). Pre všetkých 168 štartovacích časov $s$ bolo vykonaných spolu 550 spustení nášho výmenného algoritmu, teda pre každé $s$ v priemere 3,27 rôznych $N$. Výpočty pre jedno $N$ a jedno $s$ teda v priemere trvali 2,56 sekundy.

Na záver môžeme ešte porovnať náš algoritmus s veľmi efektívnym algoritmom `od.RC`, tiež z knižnice `OptimalDesign` [9]. Ten sme pre každý počiatočný čas $s$ spustili na 120 sekúnd.




**Obr. 5.4:** Efektivity nášho KL výmenného algoritmu (červenou) a `od.RC` (zelenou) v porovnaní s optimálnym aproximatívnym návrhom

Výsledky môžeme pozorovať na obrázku 5.4. Algoritmus dosahuje v niektorých prípadoch rovnaké, alebo o málo lepšie výsledky ako náš algoritmus, v mnohých prípadoch však dosahuje značne horšie výsledky. To sa prejavilo aj na priemernej efektivite vyznačenej prerušovanou čiarou.

## 5.3 Testovanie programu na náhodných maticiach

Pre lepšie a všeobecnejšie odskúšanie nášho programu sme sa rozhodli testovať ho na náhodne vygenerovaných maticiach. Testovali sme, ako sa správa pri rôznych rozmeroch vstupných matíc a pri rôznych hodnotách parametrov.

### 5.3.1 Testovanie zmeny veľkosti návrhu N

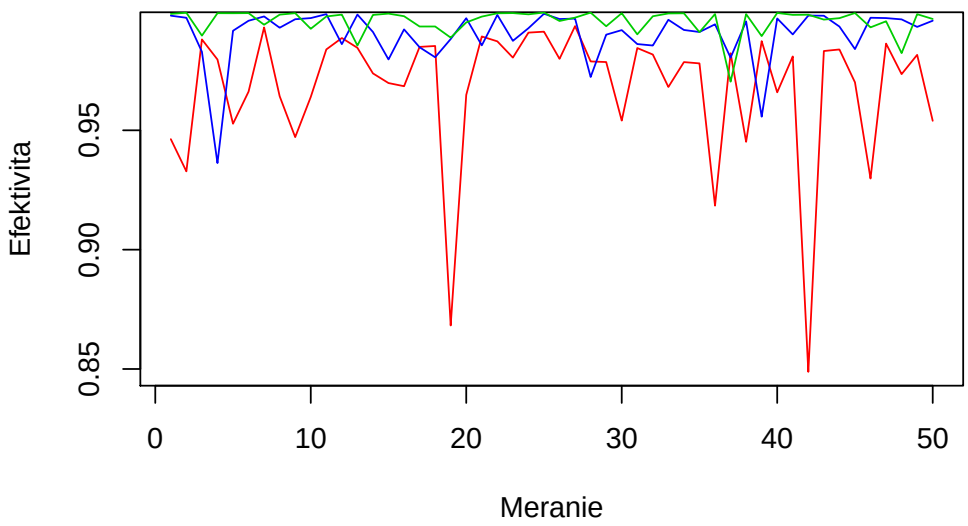
V tejto podkapitole nás bude zaujímať, ako sa zmení rýchlosť a efektivita výpočtov pre rôzne hodnoty veľkosti návrhu $N$.

Do funkcie vstupujú matice $F_{n\times m}$, $A_{r\times n}$ a vektor $b_{r\times 1}$. Počet riadkov matice $A$ sme zvolili $r = 1$, keďže tento prípad je pre našu prácu ťažiskový. Spustili sme funkciu `KL.exchange(k,l,F,A,b,N, inic.d.max = 200)` a merali sme čas výpočtu a efektivitu vzhľadom na optimálny aproximatívny návrh vypočítaný pomocou funkcie `od.SOCP`.

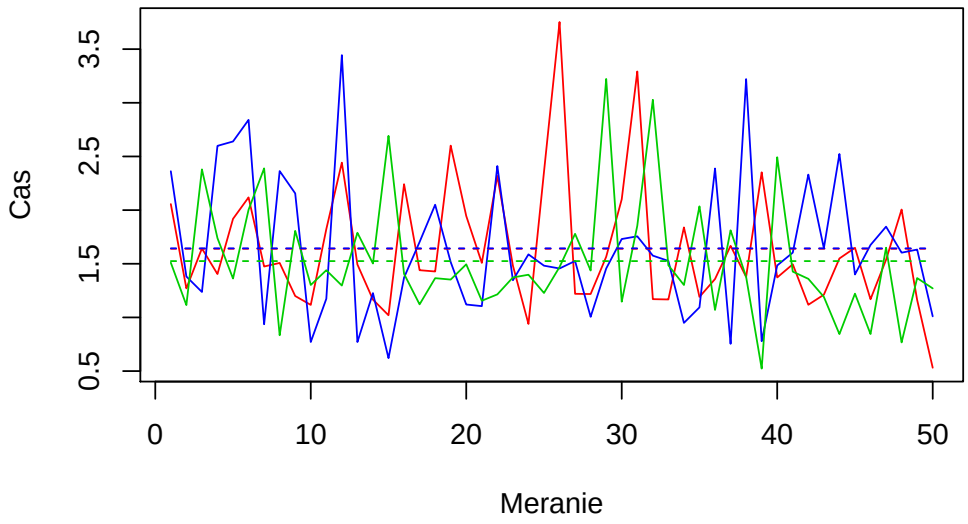
Test sme vykonali najprv pre 50 náhodne vygenerovaných sád matíc $F, A$ a $b$. Generovanie dát bolo z normálneho rozdelenia $N(0, 1)$ so strednou hodnotou 0 a disperziou 1. Následne bola z vygenerovaných čísel spravená absolútna hodnota, aby sme predišli záporným hodnotám. Pre každú sadu matíc bola vykonaná kontrola uskutočniteľnosti experimentu (súčet $N$ "najlacnejších" meraní musí byť menší ako $b$).

Pre ich rozmery platilo $m = 5, n = 100$ a za $N$ sme postupne dosadzovali hodnoty $10, 20$ a $50$. Výsledky sú graficky znázornené na obrázkoch 5.5 a 5.6. Červená farba zodpovedá výsledkom pre hodnotu $N = 10$, modrá pre $N = 20$ a zelená pre $N = 50$.

Z obrázku 5.5 je vidno, že pre vyššie $N$ rastie aj stabilita a priemerná efektivita výpočtu. Z obrázku 5.6 zasa vidno, že priemerné časy vyznačené prerušovanou čiarou sa líšia len minimálne (prerušované čiary pre $N = 20$ a $N = 50$ sa prekrývajú).

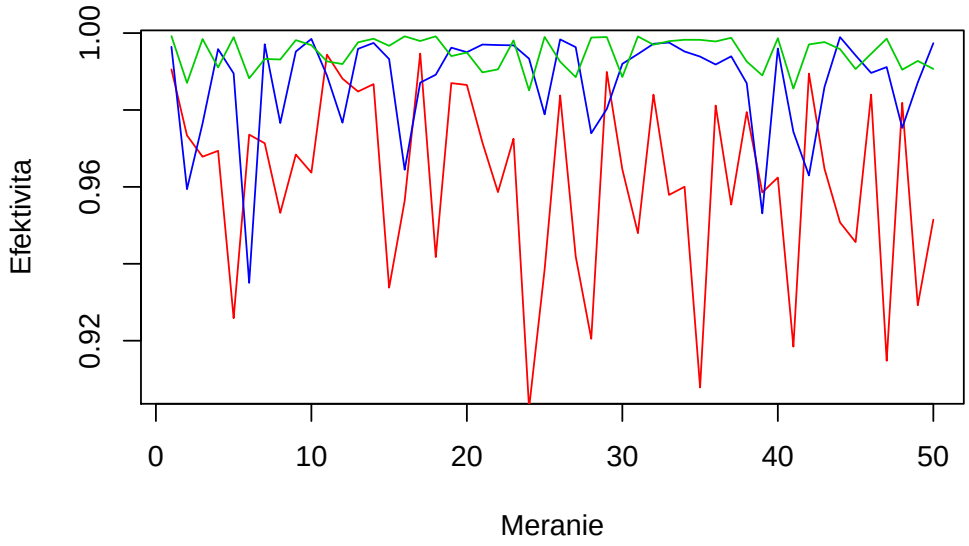



**Obr. 5.5:** Graf efektivít KL algoritmu pre $N$=10 (červená), 20 (modrá), 50 (zelená), $n = 100$




**Obr. 5.6:** Graf časov výpočtu KL algoritmu pre $N$=10 (červená), 20 (modrá), 50 (zelená), $n = 100$

Experiment sme zopakovali aj pre zmenenú hodnotu $n = 200$. Výsledky sú na obrázkoch 5.7, resp. 5.8.



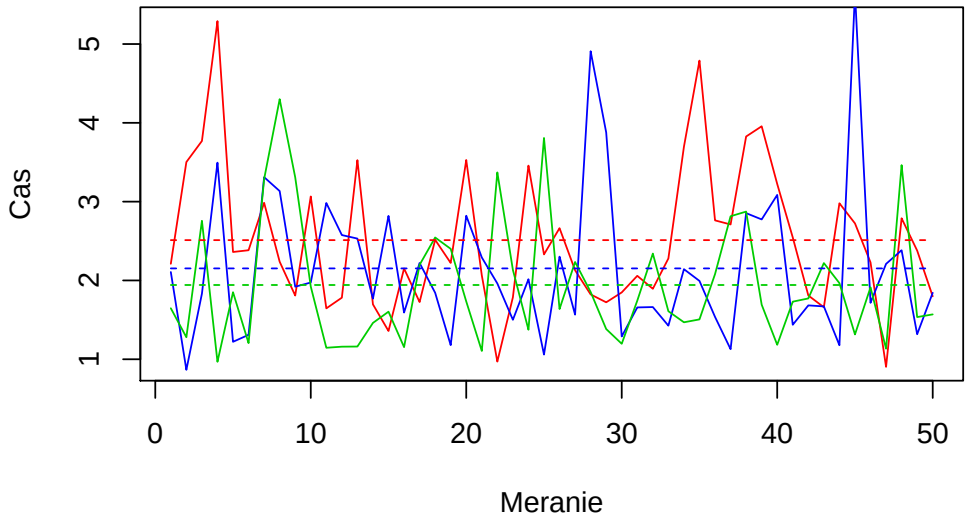

**Obr. 5.7:** Graf efektivít KL algoritmu pre $N$=10 (červená), 20 (modrá), 50 (zelená), $n = 200$

Rozdiely kolísavosti výsledkov sú tu ešte lepšie viditeľné, zväčšili sa aj rozdiely v dĺžke trvania výpočtu. Pre zvyšujúce sa $N$ sa algoritmus zrýchľuje. Rozdiely však opäť nie sú veľké.

### 5.3.2 Porovnanie KL algoritmu s výpočtom optimálneho návrhu

V tejto kapitole ukážeme, že exaktné návrhy, ktoré dáva náš algoritmus majú veľmi vysokú efektivitu. Toto porovnanie vykonáme pre model takých rozmerov, pre ktorý vieme nájsť exaktný optimálny návrh pomocou metód kónického programovania, konkrétne funkciou `od.MISOCP` z knižnice `Optimal Design` [9]. Našim cieľom bolo zistiť, ako sa s pribúdajúcim časom zlepšuje efektivita nášho algoritmu a `od.MISOCP` a následne ich porovnať.

Pre čitateľnosť obrázkov sme testy vykonali vždy len na troch sadách náhodne vygenerovaných matíc (generovaných rovnakým spôsobom ako v podkapitole 5.3.1).



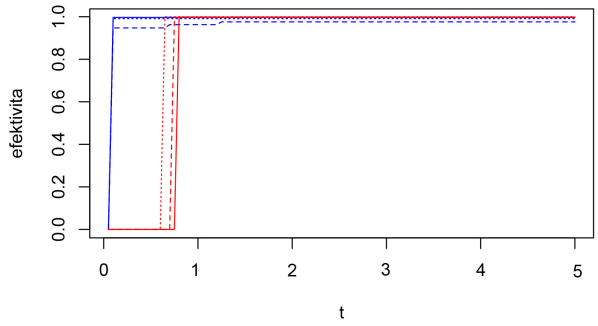

**Obr. 5.8:** Graf časov výpočtu KL algoritmu pre $N$=10 (červená), 20 (modrá), 50 (zelená), $n = 200$

Jednotlivé typy čiar (plná, prerušovaná a bodkovaná) zodpovedajú sadám matíc, modré čiary patria KL algoritmu a červené programu `od.MISOCP`. Efektivity sú porovnávané s aproximatívnym optimálnym návrhom, ktorý je výstupom z funkcie `od.SOCP`.

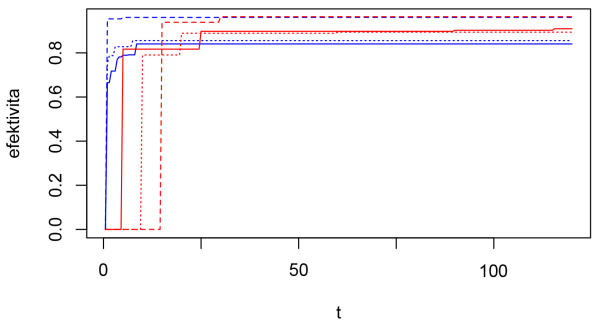
Na prvom grafe z obrázku 5.9 môžeme vidieť výsledky pre výpočtovo najjednoduchšiu kombináciu parametrov. Pre dve sady matíc sú efektivity veľmi vysoké (veľmi blízke 1) a ich rozdiely pre jednotlivé spôsoby výpočtu sú minimálne. Veľký rozdiel je však v čase výpočtu. Pre náš algoritmus sa konečná hodnota nadobudla prakticky okamžite (časový krok bol 0,05s), zatiaľ čo MISOCP sa k nejakému výsledku dopracoval až po vyše pol sekunde.

Pri zvýšení parametra $m$ na 8 sa výpočtová náročnosť zvyšuje. Rozdiel efektivity exaktného a aproximatívneho optimálneho riešenia sa tiež zvýšil. Na druhom grafe môžeme vidieť, že efektivita nášho algoritmu je nižšia ako efektivita `od.MISOCP`, aj keď v jednom prípade sa skoro rovnajú. Nespochybniteľnou prednosťou je však opäť rýchlosť.

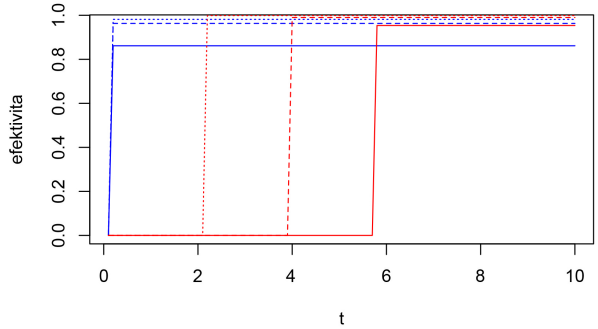
Na treťom obrázku je situácia podobná, iba časová náročnosť MISOCP je nižšia.



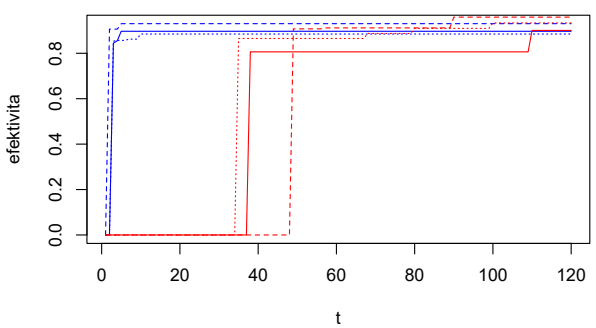

1.) $n = 100, m = 3, \Delta t = 0,05$




2.) $n = 100, m = 8, \Delta t = 0,5$




3.) $n = 100, m = 8, \Delta t = 0,1$




4.) $n = 300, m = 8, \Delta t = 1$

**Obr. 5.9:** Porovnanie efektivít nášho algoritmu a `od.MISOCP`

Posledný graf asi najlepšie ilustruje výhody nášho algoritmu. Síce mu už niekedy trvá aj dve sekundy, kým nájde nejaký návrh, jeho efektivita však veľmi rýchlo stúpne k najvyššej hodnote, ku ktorej sa vie dostať. Druhý algoritmus sa mu začne vyrovnávať až po desiatkach sekúnd.

Problém nášho algoritmu je ten, že sa môže v niektorých prípadoch zaseknúť v nejakom lokálnom optime a už sa z neho nevie dostať. Na elimináciu tohoto problému slúži zvýšenie maximálneho počtu inicializačných návrhov, no ani to nie je záruka ideálneho výsledku. Tento počet bol v našom experimente rovný 200.

Musíme ešte podotknúť, že v poslednom prípade je výpočtová náročnosť taká vysoká, že program `od.MISOCP` nie nutne našiel optimálny návrh. Parameter, ktorý hovoril o maximálnom čase trvania výpočtov bol nastavený na 120s a až tento limit ukončil výpočty. Ak však program nenašiel optimum, predpokladáme, že k nemu bol veľmi blízko.

### 5.3.3 KL výmenný algoritmus s viacerými ohraničeniami

Ďalším výsledkom tejto práce je návrh programu, ktorý by dokázal riešiť problémy s maticou $A$, ktorá má viac ako jeden riadok. Táto komplikácia však predstavuje väčší problém, ako by sa na prvý pohľad mohlo zdať.

Problém je ten, že nevieme jednoznačne povedať, že meranie v jednom bode návrhu je "lacnejšie", ako meranie v druhom bode. Hodnota v matici $A_{1,i}$ môže byť menšia ako hodnota $A_{1,j}$, avšak v druhom riadku už môže byť $A_{2,i}$ väčšia, ako $A_{2,j}$. Táto skutočnosť nám komplikuje program v dvoch prípadoch.

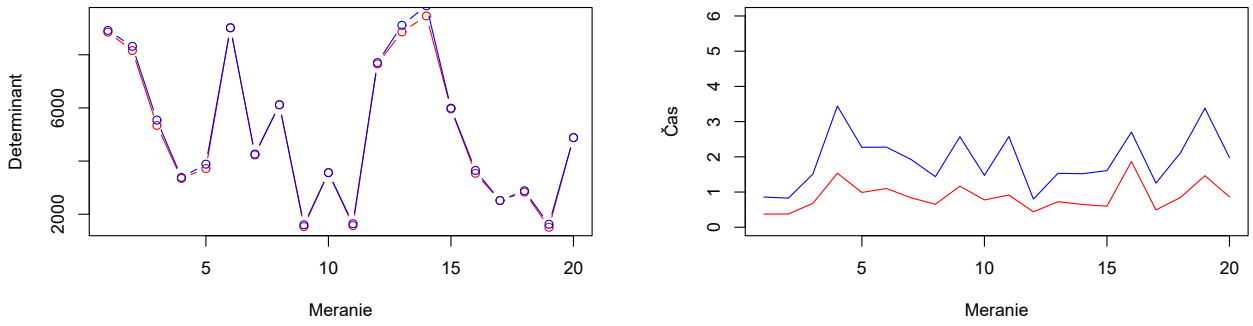
Prvý prípad, ktorý našťastie nepredstavuje zásadný problém, je redukcia kandidátov na pridanie. Keď algoritmus vyberie kandidáta na odobratie a odoberie ho, v pôvodnom algoritme môžeme zúžiť skupinu kandidátov na pridanie len na tých, ktorí nie sú "drahší" ako náš kandidát na odobratie. Toto zúženie výberu kandidátov na pridanie môže zrýchliť výpočty (aj keď za cenu mierneho zníženia efektivity). V niektorých prípadoch je totiž skutočných kandidátov na pridanie menej ako $l$ a stačí tak počítať hodnotu kritéria optimality pre menej kombinácií odobratých a pridaných bodov. Keďže v prípade viacerých ohraničení nevieme takýmto spôsobom zmenšiť počet výpočtov, zaplatíme pomyselnú pokutu v podobe väčšieho výpočtového času.

Druhý problém, na ktorý narážame, už odstrániť nevieme. Týka sa hľadania inicializačného návrhu. V kapitole 3.1.2 sme popísali spôsob zaokrúhľovania aproximatívneho návrhu, ktorý sme použili na nájdenie inicializačného návrhu, teda štartovacieho bodu pre náš algoritmus. Problém v ňom nastal, keď bol zaokrúhlený aproximatívny návrh príliš blízko ohraničenia a nevedeli sme mu doplniť požadovaný počet bodov návrhu. Takýto prípad sme riešili pridaním "najlacnejšieho" bodu. To však teraz nevieme. Vieme si zvoliť jeden riadok a dúfať, že bod bude "lacnejší" vo viacerých riadkoch, a takéto pridanie pomôže. Ináč si v tomto spôsobe hľadania inicializačného návrhu nevieme pomôcť a pre spoľahlivé riešenie tohoto problému je potrebné nájsť inú metódu.

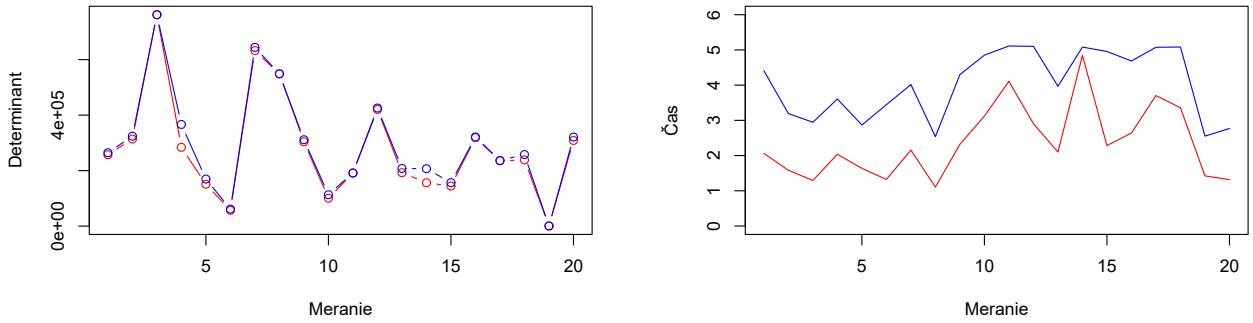
Rozdiel medzi algoritmom, ktorý rieši len jednoriadkové matice $A$ a tým, ktorý rieši aj viacriadkové spočíva iba v zužovaní kandidátov na pridanie. Rozhodli sme sa tieto algoritmy porovnať. Vygenerovali sme 20 náhodných sád matíc $F, A$ a $b$ pre rôzne kombinácie parametrov $m$ a $n$. Hodnota $N$ bola rovná 10 a maximálny počet inicializačných návrhov bol 200.

Na $x$-ovej osi grafov v ľavom stĺpci je tento krát hodnota determinantu a nie efektivita, rozdiely v efektivite by pre tvar vzťahu pre jej výpočet (1.14) boli minimálne a z grafu nečitateľné. Červená farba zodpovedá vždy pôvodnému algoritmu pre jednoriadkové $A$, modrá zasa zovšeobecnenému algoritmu. Z obrázku 5.10 si môžeme všimnúť, že hodnota determinantu z oboch algoritmov je skoro rovnaká, avšak v niektorých prípadoch je všeobecnejší algoritmus lepší. Môže to byť spôsobené tým, že kandidát na pridanie, ktorý je drahší ako kandidát na odobratie, ešte spĺňa ohraničenie a zároveň prináša lepšiu hodnotu kritéria optimality. V druhom stĺpci grafov však vidíme, že cena za jemne zvýšenú efektivitu je čas, ktorý je v prípade druhého algoritmu vždy vyšší, často o niekoľko sekúnd.

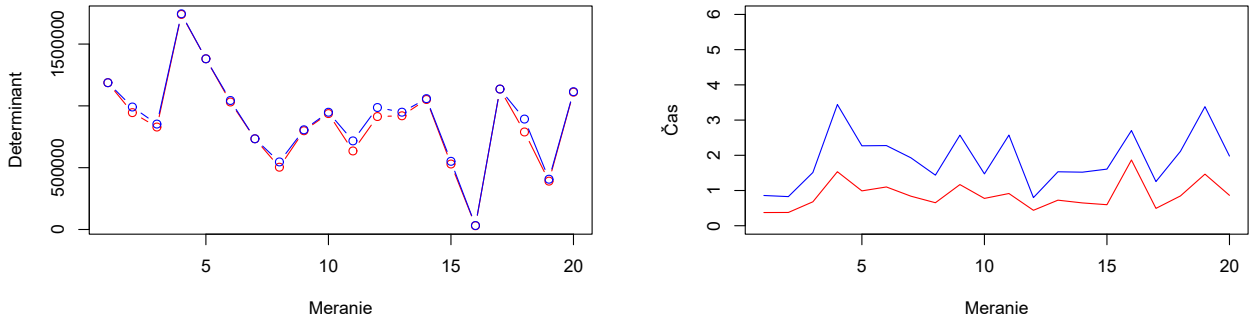
Na záver sme testovali aj to, ako sa menia výsledky algoritmu, ak pridávame riadky do matice $A$. Opäť sme vygenerovali náhodné matice s parametrami $m = 5, n = 100$ a $N$ sme zvolili 10.




1.) $n = 100, m = 3$

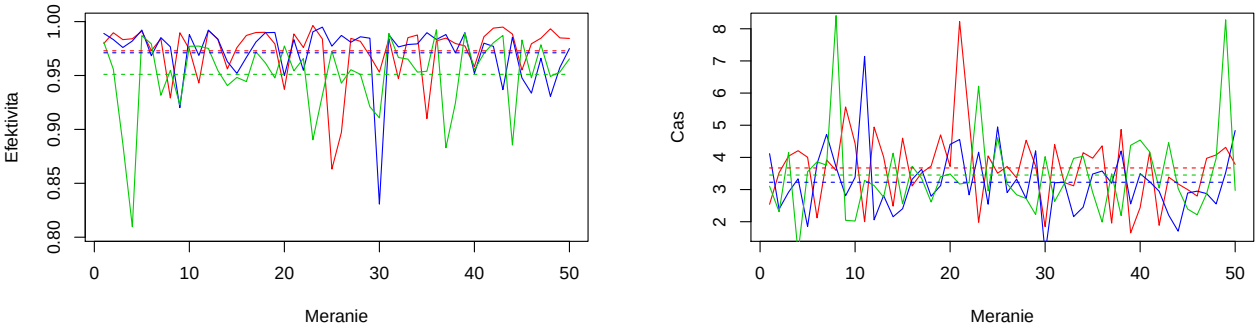



2.) $n = 100, m = 5$




3.) $n = 200, m = 5$

**Obr. 5.10:** Porovnanie determinantov a trvania výpočtov algoritmov určených pre maticu $A$ s jedným (červená) a viacerými (modrá) riadkami.



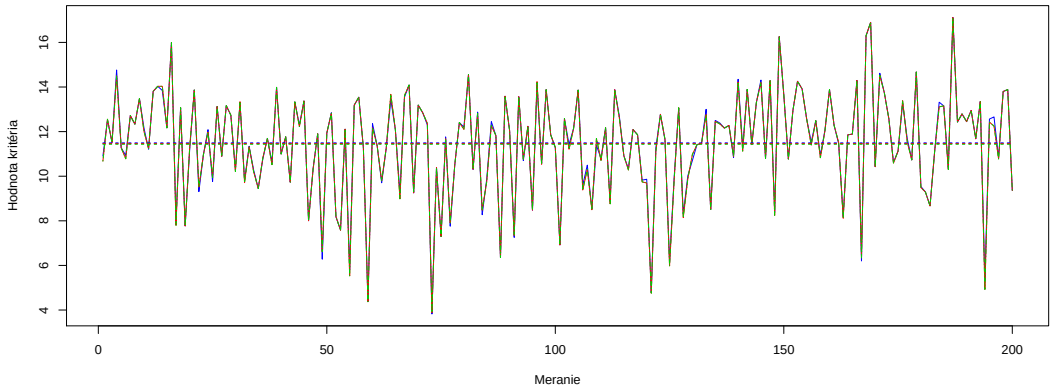

**Obr. 5.11:** Porovnanie efektivity a trvania výpočtov pre rôzne počty riadkov matice $A$ ($A_{1\times100}$ červenou, $A_{2\times100}$ modrou, $A_{3\times100}$ zelenou)

Na obrázku 5.11 zodpovedá červená čiara matici $A$ s jedným riadkom, modrá dvom a zelená trom riadkom. Efektivita je porovnávaná s optimálnym aproximatívnym návrhom. Zatiaľ čo priemerné efektivity znázornené na ľavom grafe prerušovanou čiarou pre jedno a dvojriadkovú maticu $A$ sú veľmi podobné, pre trojriadkovú maticu je už efektivita značne nižšia. Časy výpočtu s počtom ohraničení zdá sa nekorelujú.

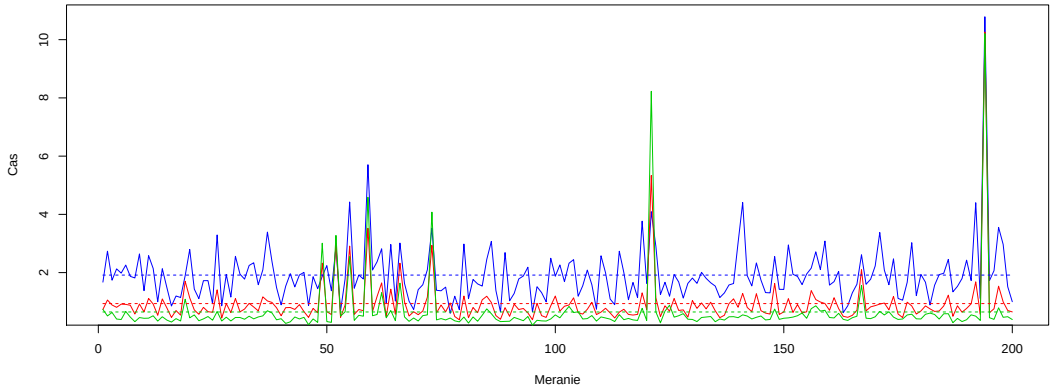
Čo ale z grafov nie je vidno, je neúspešnosť hľadania počiatočných návrhov. Pre jednoriadkové ohraničenia zbehol program vždy. Pre dvojriadkové ohraničenia sa už ale situácia komplikovala. Pre dosiahnutie 50 meraní sme museli vyradiť 4 sady matíc, pre ktoré program nezbehol. Pre trojriadkové ohraničenia sme narazili na problém až 12 krát. S pribúdajúcimi ohraničeniami teda narastá neúspešnosť nášho algoritmu, ktorá je spôsobená neschopnosťou nájsť inicializačný návrh.

### 5.3.4 Testovanie zmeny hodnoty parametrov $k$ a $l$

V doterajších výpočtoch sme mali stanovené hodnoty parametrov $k$ a $l$ na 10. V mnohých prípadoch sme tak ale nevyužili výhodu algoritmu, vďaka ktorej môžeme zmenšiť počet výmen kandidátov na odobratie a pridanie. Ak je veľkosť návrhu $N = 10$ a počet kandidátov na odobratie $k = 10$, narábame vlastne so všetkými bodmi návrhu. Na náhodných maticiach s parametrami $m = 5$, $n = 100$ a $N = 10$ sme preto vyskúšali hodnoty parametrov $k$ a $l$ meniť. Zvolili sme hodnoty 10, 5 a 2.




**Obr. 5.12:** Porovnanie hodnôt kritérií algoritmu pre $k, l = 10$ (modrou), $k, l = 5$ (červenou) a $k, l = 2$ (zelenou)




**Obr. 5.13:** Porovnanie výpočtových časov algoritmu pre $k, l = 10$ (modrou), $k, l = 5$ (červenou) a $k, l = 2$ (zelenou)

Na grafe 5.12 znázorňujúcom hodnoty kritérií je vidno, že ich hodnoty sa líšia len minimálne. Zníženie hodnoty parametrov teda prináša len minimálne zníženie hodnoty kritéria. Na druhej strane, pri znížení hodnoty parametrov dochádza k zásadnému skráteniu trvania výpočtov. Pre $k, l = 10$ bol priemerný čas 1,92s, pre $k, l = 5$ to bolo 0,94s a pre $k, l = 2$ to bolo len 0,65s.

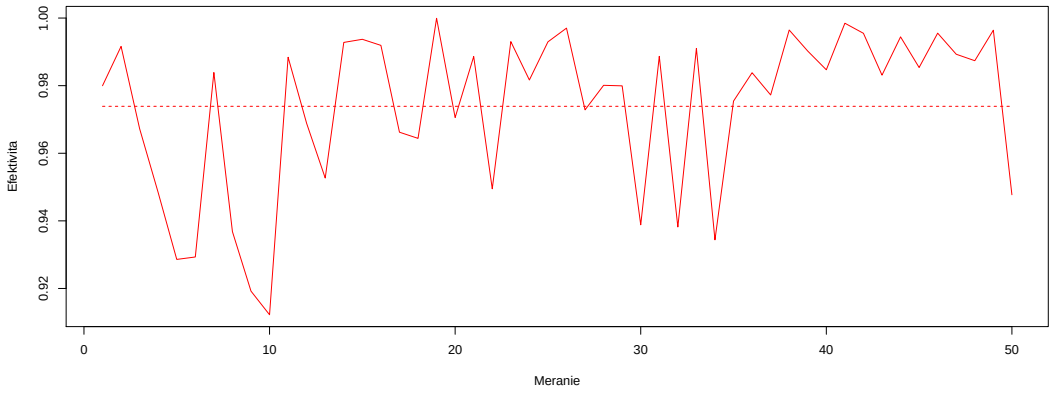
## 5.4 KL výmenný algoritmus pre kritérium $A$-optimality

Do programu sme okrem $D$-optimality zaradili aj kritérium $A$-optimality. Rozdiel oproti pôvodnému algoritmu je nie len v zmene kritéria, na základe ktorého vyberáme najlepšiu výmenu kandidátov na odobratie a pridanie, ale aj v zmene variančnej funkcie. Pri tej nastáva oproti (1.8) jemná zmena [3]:

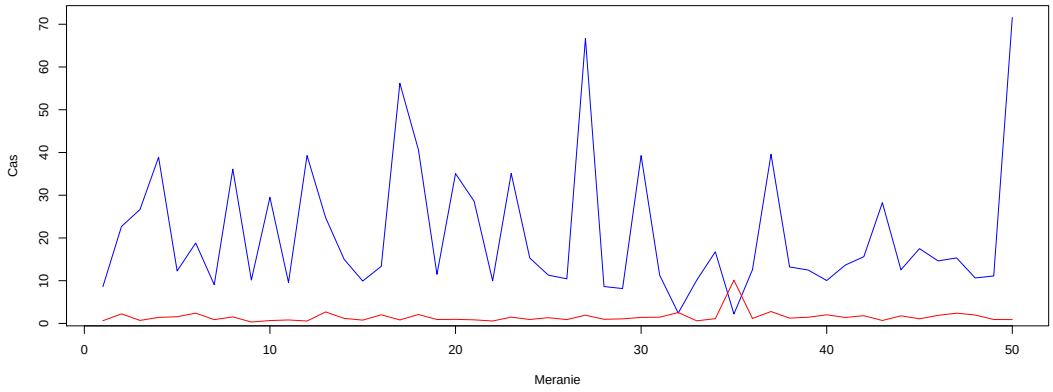
$$a(x, \xi_N) = f^T(x) M(\xi_N)^{-2} f(x) \tag{5.3}$$

Kritérium optimality sme popisovali už v prvej kapitole (1.11).

Algoritmus pre toto kritérium sme testovali na náhodných maticiach. Vygenerovali sme ich podobne ako v predošlých príkladoch, pre $m = 5$, $n = 100$ a $N = 10$. Pre 50 sád takýchto matíc $A, b$ a $F$ sme porovnali výsledky nášho algoritmu s `od.MISOCP`. Hodnoty $k$ a $l$ boli rovné 10. Výsledky testu sú znázornené na nasledujúcich grafoch.




**Obr. 5.14:** Efektivita KL-výmenného algoritmu pre kritérium $A$-optimality




**Obr. 5.15:** Časy KL-výmenného algoritmu pre kritérium $A$-optimality (červenou) v porovnaní s časmi `od.MISOCP` (modrou)

Priemerná efektivita algoritmu oproti optimálnemu exaktnému návrhu sa pohybovala na úrovni 97%. Nespochybniteľnou výhodou však ostáva výpočtový čas.

# Záver

V tejto práci sme sa venovali optimálnemu návrhu experimentov. Za cieľ sme si zvolili naprogramovať KL-výmenný algoritmus a modifikovať ho tak, aby návrh spĺňal aj isté typy ohraničení. Očakávali sme, že jeho prednosťou bude rýchlosť výpočtu.

Na začiatku práce sme zhrnuli teóriu potrebnú k porozumeniu základom navrhovania experimentov. Popísali sme tiež ideu výmenných algoritmov so zvýšenou pozornosťou na KL-výmenný algoritmus.

Súčasťou nášho problému bolo aj nájdenie kvalitného inicializačného návrhu. Našli sme niekoľko existujúcich metód, ale pre naše potreby sme museli vytvoriť vlastný algoritmus na jeho nájdenie. Ukázalo sa, že od počiatočného návrhu do veľkej miery závisí aj výsledný - optimalizovaný návrh a preto je metóda jeho nájdenia veľmi dôležitá.

Naprogramovaný algoritmus sme testovali na niekoľkých príkladoch. Porovnávali sme ho s inými algoritmami a zisťovali sme aj zmeny vo výsledkoch pri zmene vstupných parametrov a matíc. Prešli sme cez príklady, na ktorých boli testované iné algoritmy, ale zaujímalo nás aj správanie programu na príkladoch s náhodne vygenerovanými maticami. Pri testovaniach sa algoritmus ukázal byť veľmi rýchly. Oproti porovnávaným algoritmom bol niekoľkonásobne rýchlejší. Zároveň dosahoval dostatočne vysoké efektivity, čo považujeme za priaznivý výsledok.

Do programu sme zaradili okrem $D$-optimality aj kritérium $A$-optimality. Efektivita v porovnaní s optimálnym exaktným návrhom už nebola taká dobrá, ako pre $D$-optimalitu, rýchlosť výpočtu návrhu však bola vynikajúca.

Náš výmenný algoritmus pre modely bez replikácií sa ukázal ako efektívny nástroj pre hľadanie optimálneho návrhu a aj napriek nedokonalostiam považujeme náš cieľ za splnený. Jeho nedostatky vidíme hlavne v komplikáciách spôsobených

viacerými ohraničeniami, kedy sa nájdenie inicializačného návrhu stáva náročnejším a bolo by vhodné nájsť taký algoritmus, ktorý by bol na tento účel spoľahlivejší. Kód algoritmu je v takom stave, že po miernych formálnych úpravách a precíznejšom testovaní by bolo možné ho implementovať do knižnice `OptimalDesign` [9] .Veríme, že algoritmus nájde uplatnenie alebo poslúži ako inšpirácia pre ďalšie algoritmy.

# Literatúra


[1] Agev, A. A., Donev, A. N.: *The Construction of Exact D-optimum Experimental Designs with Application to Blocking Response Surface Designs*, Biometrika, Londýn, 1989

[2] Atkinson, A. C., Sviridenko, M. I.: *Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee*, Journal of Combinatorial Optimization, Volume 8 (2004), Issue 3, 307-328

[3] Atkinson, A.C., Donev, A. N.: *Optimum Experimental Designs, With SAS*, Oxford University Press, Oxford, 2007

[4] Fedorov V.V.: *Theory Of Optimal Experiments*, Academic Press, New York, 1972

[5] Fedorov V.V.: *OPTIMAL DESIGN WITH BOUNDED DENSITY: Optimization Algorithms Of The Exchange Type*, Journal of Statistical Planning and Inference, Volume 22 (1989), Issue 1, 1-13

[6] Fedorov V.V., Leonov S.L.: *Optimal Design for Nonlinear Response Models*, CRC Press, Boca Raton, 2014

[7] Filová l.: $E_k$-optimal and maximin efficient designs of experiments, dizertačná práca, FMFI UK, Bratislava, 2010

[8] Harman R., Bachratá A., Filová L.: *Construction of efficient experimental designs under multiple resource constraints*, Applied Stochastic Models in Business and Industry, Volume 32(2016), Issue 1, 3-17

[9] Harman R., Filová L.: "Package 'OptimalDesign'", 2016, dostupné na internete(5.12.2018): https://cran.r-project.org/web/packages/OptimalDesign/index.html

[10] Kiefer J.: *General Equivalence Theory for Optimum Designs (Approximate Theory)*, The Annals of Statistics, Volume 2 (1974), 849-879

[11] Pukelsheim F.: *Optimal design of experiments*, John Wiley & Sons, New York, 1993

[12] Sagnol G., Harman R.: *Computing exact D-optimal designs by mixed integer second-order cone programming*, The Annals of Statistics, Volume 43(2015), 2198–2224

[13] Wright S.E., Sigal B.M., Bailer A.J.: *Workweek optimization of experimental designs: exact designs for variable sampling costs.*, Journal of Agricultural Biological and Environmental Statistics, 15(4)(2010),491–509

# Príloha A

## A.1 Hlavný algoritmus

```
#K,L...parametre KL vymenneho algoritmu
  #A...matica A
  #b...vektor b, spolu s A su sucastou systemu Aw <= b
  #F...matica regresorov
  #N...velkost navrhu
  #pevne body inicializacneho navrhu
  #crit...kriterium optimality
  #inic.d.min...minimalny pocet inicializacnych navrhov
  #inic.d.max...maximalny pocet inicializacnych navrhov
  #t.max...maximalna dlzka trvania experimentu [s]
  #d.check...maximalny pocet iterácii algritmu, v ktorych nedojde k
  zlepseniu hodnoty kriteria

  start <- as.numeric(proc.time()[3])  #ulozi pociatocny cas
  finish <- FALSE
  if (is.null(A)) A <- rep(1, dim(F)[1])
  if (is.null(b)) b <- N
  if (is.null(K)) K <- max(c(10, min(ceiling(sqrt(c(N, n))))))
  if (is.null(L)) L <- max(c(10, min(ceiling(sqrt(n)))))
  A<-rbind(A)  #riadkovy vektor ulozi ako riadkovu maticu
  count<- mem <-1  #count pocita pocet roznych inicializacnych navrhov
```

```
all.w <- matrix(NaN, ncol = dim(A)[2] + 1, nrow =  inic.d.max)  #matica
na priebezne zapisovanie navrhov a hodnot kriterii
if(is.null(w1))    w_1 <- Inic.Des.Phase1(F,A,b,N, crit = crit)  #pokial
nie je stanovene inak, vytvori ciastocny inicializacny navrh

while ((!finish & (count <= inic.d.max)) | (count <= inic.d.min & !finish)) {
  #pokial nie je koniec, nenaplnil max. pocet opakovani alebo neprekrocil
  min. pocet opakovani, iteruje
  wy <- Inic.Des.Phase2(w_1,A,b,N) #doplni ciastocny inicializacny navrh
  na kompletny

  if(crit=="D") w <- KL.D.exchanger(K,L, F, A, b ,wy,w1) #vykona prislusny
  vymenny algoritmus
  if(crit=="A") w <- KL.A.exchanger(K,L, F, A, b ,wy,w1)
  all.w[count,] <- c(w$Phi.best ,w$w.best)

  if(count > 1 & (all.w[count,1] > max(all.w[1:(count-1),1]))){
  #ak sa zlepsila hodnota kriteria, resetuje pocitadlo
    mem <- 1
  }

  if(mem > d.check & (all.w[count,1] <= max(all.w[1:(count-1),1]))){
    finish <- TRUE
    break
  }

  count <- count + 1
  mem <- mem + 1
  finish <- as.numeric(proc.time()[3]) > start + t.max  #koniec ak sme
  prekrocili casovy limit
}
```

```
  W <- na.omit(all.w) #odstrani nevyplnene riadky
  if(is.null(W)) {
    print("Algorithm has not found feasible initialising design" )
    break
  } else {
    Phi.best <- W[order(W[,1], decreasing = TRUE),][1,1]
    list(Phi.best = Phi.best, w.best = W[order(W[,1], decreasing = TRUE),]
    [1,2:dim(W)[2]])
  }
}
```

## A.2 Výmenný algoritmus pre D-optimalitu

```
KL.D.exchanger <- function(K,L,F,A,b,w,w1 = NULL,crit){

  A <- rbind(A)  #riadkovy vektor ulozi ako riadkovu maticu
  n <- dim(F)[1]  #nacita parametre
  m <- dim(F)[2]
  M <- od.infmat(F,w)
  eps <- 0.00001

  critNew <- det(M)
  critOld <- critNew - eps  #aby nas to hned na zaciatku nezastavilo

  while(critOld < critNew){  #kym zlepsuje hodnotu kriteria
    critOld <- critNew
    d.fun <- apply((F %*% solve(M)) * F, 1, sum)
    d.fun <- cbind(d.fun, 1:length(d.fun))  #prvy riadok hodnoty var.f,
    druhy riadok prislusne indexy
    if(is.null(w1)) { d_k <- d.fun[which(w==1),]  #ak sme nedefinovali fixne
```

```
body navrhu
} else {
  d_kb <- d.fun[-which(w1==1),]  #z kandidatov na odobratie vyradi fixne
  body
  cand <- d.fun[which(w==1),]
  bad <- match(which(w1==1),cand[,2])
  d_k <- cand[-bad,]  #kandidati na odobratie
}

if(is.vector(d_k)) {d_k<-rbind(d_k)}
d_k <- d_k[order(d_k[,1], decreasing = FALSE),]  #zoradeni kandidati na
odobratie podla hodnoty var. f. od najmensej
d_l <- d.fun[which(w==0),]  #kandidati na pridanie
d_l <- d_l[order(d_l[,1], decreasing = TRUE),]  #zoradeni kandidati na
pridanie podla hodnoty var. f. od najvacsej
criteria <- matrix(-Inf, nrow = K, ncol = L)  #matica ukladajuca
determinanty, z ktorych hladam po vymene ten najvacsi

for(i in 1:min(K , dim(d_k)[1])){
  w[d_k[i,2]] <- 0  #odoberie kandidata
  if(sum(A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[i,2]])!=0){
    d_l_new <- matrix(d_l[A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[i,2]],],
    ncol = 2, byrow = FALSE) #vyberame len tie d_l s mensou alebo
    rovnakou cenou
  }else d_l_new <- matrix(NaN, ncol = 2)  #ak nema ziadneho kandidata
  na pridanie, pocita s maticou NaN

  for(j in 1:min(L, dim(d_l_new)[1])){  #pre kazdeho kandidata na
  odobratie popridava postupne vsetkych moznych kand. na pridanie
    w[d_l_new[j,2]] <- 1
    if( sum(A%*%w <= b) == length(b) ){  #ak splna ohranicenia (pre pripad
```

```
    s jednoriadkovym A je toto zbytocna podmienka)
      M1 <- M - (F[d_k[i,2],])%*%t(F[d_k[i,2],]) + (F[d_l_new[j,2],])
      %*%t(F[d_l_new[j,2],])
      criteria[i,j] <- det(M1)
    }
    w[d_l_new[j,2]] <- 0  #spatne odoberie kandidata na pridanie
  }
  w[d_k[i,2]] <- 1  #spatne prida kandidata na odobratie
}
criteria[is.nan(criteria)] <- -Inf
criteria[is.na(criteria)] <- -Inf
coord <- which(criteria == max(criteria), arr.ind = TRUE)
ki <- coord[1]
li <- coord[2]
if(is.matrix(coord)){
  ki <-coord[1,1]
  li <-coord[1,2]
}
w_control<-w
w_control[d_k[ki,2]] <- 0  #skusobna vymena
#replikujeme vyber kandidatov na pridanie pre vybraneho kandidata na
odobratie
if(sum(A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[ki,2]])!=0){
  d_l_new <- matrix(d_l[A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[ki,2]],],
  ncol = 2, byrow = FALSE)
} else d_l_new <- matrix(NaN, ncol = 2)

w_control[d_l_new[li,2]] <- 1
if( sum(A%*%w_control <= b) == length(b) ){  #ak splna ohranicenia po
vymene, spravi vymenu
  w[d_k[ki,2]] <- 0  #vykonanie vymeny
```

```
      w[d_l_new[li,2]] <- 1
    }
    M <- od.infmat(F,w)
    critNew <- det(M)
  }
  list(Phi.best = max(criteria)^(1/m) , w.best = w)

}
```

## A.3 Výmenný algoritmus pre A-optimalitu

```
KL.A.exchanger <- function(K,L,F,A,b,w,w1 = NULL,crit){

  A<-rbind(A)  #riadkovy vektor ulozi ako riadkovu maticu
  n <- dim(F)[1]  #nacita parametre
  m <- dim(F)[2]
  M <- od.infmat(F,w)
  eps <- 0.00001

  critNew <- m/(sum(diag(solve(M))))
  critOld <- critNew - eps  #aby nas to hned na zaciatku nezastavilo

  while(critOld < critNew){  #kym zlepsuje hodnotu kriteria
    critOld <- critNew
    d.fun <- apply((F %*% solve(M%*%M)) * F, 1, sum)
    d.fun <- cbind(d.fun, 1:length(d.fun))  #prvy riadok hodnoty var.f,
    druhy riadok prislusne indexy
    if(is.null(w1)) { d_k <- d.fun[which(w==1),]  #ak sme nedefinovali fixne
    body navrhu
```

```
} else {
  d_kb <- d.fun[-which(w1==1),]  #z kandidatov na odobratie vyradi fixne
  body
  cand <- d.fun[which(w==1),]
  bad <- match(which(w1==1),cand[,2])
  d_k <- cand[-bad,]  #kandidati na odobratie
}

if(is.vector(d_k)) {d_k<-rbind(d_k)}
d_k <- d_k[order(d_k[,1], decreasing = FALSE),]  #zoradeni kandidati na
odobratie podla hodnoty var. f. od najmensej
d_l <- d.fun[which(w==0),]  #kandidati na pridanie
d_l <- d_l[order(d_l[,1], decreasing = TRUE),]  #zoradeni kandidati na
pridanie podla hodnoty var. f. od najvacsej
criteria <- matrix(-Inf, nrow = K, ncol = L)  #matica ukladajuca
determinanty, z ktorych hlada po vymene ten najvacsi

for(i in 1:min(K , dim(d_k)[1])){
  w[d_k[i,2]] <- 0  #odoberie kandidata
  if(sum(A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[i,2]])!=0){
    d_l_new <- matrix(d_l[A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[i,2]],],
    ncol = 2, byrow = FALSE) #vyberame len tie d_l s mensou alebo
    rovnakou cenou
  }else d_l_new <- matrix(NaN, ncol = 2)  #ak nema ziadneho kandidata
  na pridanie, pocita s maticou NaN

  for(j in 1:min(L, dim(d_l_new)[1])){  #pre kazdeho kandidata na
  odobratie popridava postupne vsetkych moznych kand. na pridanie
    w[d_l_new[j,2]] <- 1
    if( sum(A%*%w <= b) == length(b) ){  #ak splna ohranicenia (pre
    pripad s jednoriadkovym A je toto zbytocna podmienka)
```

```
      M1 <- M - (F[d_k[i,2],])%*%t(F[d_k[i,2],]) + (F[d_l_new[j,2],])
      %*%t(F[d_l_new[j,2],])
      criteria[i,j] <- m/(sum(diag(solve(M1))))
    }
    w[d_l_new[j,2]] <- 0  #spatne odoberie kandidata na pridanie
  }
  w[d_k[i,2]] <- 1  #spatne prida kandidata na odobratie
}
criteria[is.nan(criteria)] <- -Inf
criteria[is.na(criteria)] <- -Inf
coord <- which(criteria == max(criteria), arr.ind = TRUE)
ki <- coord[1]
li <- coord[2]
if(is.matrix(coord)){
  ki <-coord[1,1]
  li <-coord[1,2]
}
w_control<-w
w_control[d_k[ki,2]] <- 0  #skusobna vymena
#replikujeme vyber kandidatov na pridanie pre vybraneho kandidata
na odobratie
if(sum(A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[ki,2]])!=0){
  d_l_new <- matrix(d_l[A[dim(A)[1],d_l[,2]]<=A[dim(A)[1],d_k[ki,2]],],
  ncol = 2, byrow = FALSE)
} else d_l_new <- matrix(NaN, ncol = 2)

w_control[d_l_new[li,2]] <- 1
if( sum(A%*%w_control <= b) == length(b) ){  #ak splna ohranicenia po
vymene, spravi vymenu
  w[d_k[ki,2]] <- 0  #vykonanie vymeny
  w[d_l_new[li,2]] <- 1
```

```
    }
    M <- od.infmat(F,w)
    critNew <- m/(sum(diag(solve(M))))
  }
  list(Phi.best = max(criteria) , w.best = w)

}
```

## A.4 Zaokrúhlenie aproximatívneho návrhu

```
Inic.Des.Phase1 <- function(F,A,b,N,crit){

  A<-rbind(A)
  #zvacsime model pre potreby od.SOCP
  Abig <- rbind(A,rep(1, times = dim(A)[2]), diag(rep(1, times = dim(A)[2])))
  bbig <- c(b, N, rep(1, times = dim(A)[2]))
  sink( tempfile() )  #skrytie zbytocnych vystupov
  w_b <- od.SOCP(F=F,b=bbig,A=Abig,crit=crit)$w.best  #aprox. navrh
  sink()
  eps <- 0.01
  w_b <- floor(w_b + eps)  #zaokruhleny navrh

  return(w_b)
}
```

## A.5 Doplnenie návrhu na požadovanú veľkosť

```
Inic.Des.Phase2 <- function(w_b,A,b,N, t.lim = 1){

  A<-rbind(A)
  pocet <- N - sum(w_b)  #kolko bodov chceme doplnit
  start <- as.numeric(proc.time()[3])

  while (sum(w_b) < N) {  #kym nenaplnime velkost experimentu
    w_bc <- w_b
    w_bc[sample(x = which(w_bc == 0), size = pocet)]<-1
    #nahodne doplnime pozadovany pocet bodov
    if( sum(A%*%w_bc <= b) == length(b) ){
      w_b <- w_bc
    }
    if ((as.numeric(proc.time()[3]) - start ) > t.lim) {   #ak prekrocime
    limit, vymenime najdrahsie meranie za najlacnejsie nezrealizovane
      ceny <- cbind(A[dim(A)[1], w_b == 1], which(w_b == 1))
      ceny <- ceny[order(ceny[,1], decreasing = TRUE),]
      ceny2 <- cbind(A[dim(A)[1], w_b == 0], which(w_b == 0))
      ceny2 <- ceny2[order(ceny2[,1], decreasing = FALSE),]
      w_b[ceny[1,2]] <- 0
      w_b[ceny2[1,2]] <- 1
      start <- as.numeric(proc.time()[3])  #resetujeme casomieru
    }
  }
  return(w_b)
}
```