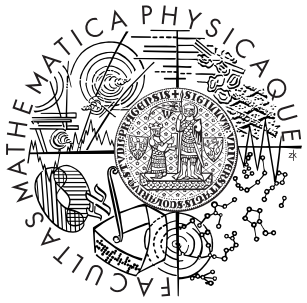
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# DIPLOMOVÁ PRÁCE




Bc. Karel Murgaš

## Úlohy pravděpodobnostního programování s diskrétním rozdělením

Katedra pravděpodobnosti a matematické statistiky

Vedoucí diplomové práce: prof. RNDr. Jitka Dupačová, DrSc.
Studijní program: Matematika, ekonometrie

2010

Děkuji paní prof. Dupačové za její trpělivost a vstřícné konzultace, děkuji Honzovi za program na počítání distribuční funkce a za výpočetní kapacitu, děkuji Argel, která mi pomáhala soustředit se na práci, děkuji babičce za jazykovou korekturu a děkuji všem, kteří se mnou vydrželi především období dokončování práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 2.8.2010 Bc. Karel Murgaš

# Obsah

Název práce: Úlohy pravděpodobnostního programování s diskrétním rozdělením
Autor: Bc. Karel Murgaš
Katedra (ústav): Katedra pravděpodobnosti a matematické statistiky
Vedoucí diplomové práce: prof. RNDr. Jitka Dupačová, DrSc.
e-mail vedoucího: Jitka.Dupacova@mff.cuni.cz

Abstrakt: Tato práce se zabývá úlohami stochastického programování s pravděpodobnostními omezeními s diskrétním rozdělením. Ukazuji konečnost a korektnost algoritmu pro výpočet $p$-level eficientních bodů, který také implementuji v prostředí R. Pomocí těchto bodů pak uvolňuji množinu přípustných řešení, abych získal úlohu konvexního programování, a zkoumám vlastnosti množiny vzniklé tímto uvolněním.Výsledky jsou prezentovány pro lineární, celočíselné a nelineární programování. V závěrečném příkladu je porovnán diskrétní přístup k náhodě se spojitým případem.

Klíčová slova: stochastické programování, pravděpodobnostní omezení, diskrétní rozdělení, $p$-level eficientní body, algoritmy

Title: Probabilistic programs with discrete probability distributions
Author: Bc. Karel Murgaš
Department: Department of Probability and Mathematical Statistics
Supervisor: prof. RNDr. Jitka Dupačová, DrSc.
Supervisor's e-mail address: Jitka.Dupacova@mff.cuni.cz

Abstract: This thesis deals with stochastic programming problems with probabilistic constraits with discrete distribution. Finitness and corectness of algortithm for finding $p$-level efficient points is proved and I implement this algorithm in R. I relax the feasible set to get convex problem and I study properties of the relaxed set. Results for linear, integer and nonlinear problems are presented. In en example I compare discrete approach with the continuous one.

Keywords: stochastic programming, probabilistic constraints, discrete probabilistic distributions, $p$-level efficient points, algorithms

# Kapitola 1

# Úvod

Náhoda je v optimalizaci důležitým faktorem a může do jejích modelů vstupovat několika způsoby. Úloha může mít náhodu v účelové funkci, může mít náhodné koeficienty v omezujících funkcích, náhodné pravé strany nebo může jít o kombinaci těchto vlivů. V této práci se zpočátku zaměřuji na lineární úlohu stochastického programování s deterministickou účelovou funkcí, deterministickou maticí koeficientů a náhodnými pravými stranami. Náhodná omezení se pak snažím splnit s předem danou pravděpodobností.

Samozřejmě velmi záleží na rozdělení, které popisuje pravé strany omezujících nerovnic. Úlohy se řeší převedením na deterministický ekvivalent, ve kterém ale obecně není zajištěna konvexita úlohy a tedy řešitelnost standardními prostředky. Pro spojitá rozdělení je známá třída rozdělení, pro která tento problém odpadá, ale u diskrétních rozdělení je situace složitější.

Řešení nabízejí takzvané p-level eficientní body (PLEP), které představují jakési "krajní" body. Po jejich spočtení se dá množina přípustných řešení uvolnit a úloha je pak již řešitelná obvyklými metodami. PLEP se však běžně používají jen pro lineární úlohy s reálnými rozhodujícími proměnnými, proto se zabývám zobecněními této teorie pro úlohy s celočíselnými proměnnými a pro nelineární úlohy.

Při uvolnění úlohy se zvětší množina přípustných řešení, což má vliv na spolehlivost řešení. Nejhorší možnou spolehlivost lze hrubě apriorně odhadnout.Uvolnění množiny přípustných řešení však není vždy optimální, a to i v případě lineárních úloh.

Příklad z vodohospodářské praxe ilustruje, jak se dají popsané metody aplikovat na úlohu s lineárními omezeními, a srovnává výsledky s prací uvažující spojité rozdělení. Ukazuje se, že při dostatečném počtu dat jsou

výsledky srovnatelné, ale daný způsob generování scénářů je neekonomický z hlediska výpočetních prostředků.

V příloze jsou implementovány algoritmy pro prostředí R a GAMS určené pro výpočet úloh pravděpodobnostního programování s diskrétním rozdělením, s jejichž pomocí je také výše zmíněný příklad vyřešen. CD příloha obsahuje i konkrétní aplikaci kódu na vstupní data odpovídající příkladu.

# Kapitola 2

# Matematické programování a náhoda

## 2.1 Základní úloha

Základní úloha, se kterou budu v této práci pracovat, je úloha lineárního programování. Bez újmy na obecnosti budu uvažovat minimalizační úlohu.

$$\begin{aligned} &\min c^T x \\ &Ax \geq b \\ &x \in X, \end{aligned} \tag{2.1}$$

kde $x$ je vektor rozhodovacích proměnných, $c$ parametry účelové funkce, $A$ matice koeficientů, $b$ vektor omezení a $X$ pevná polyhedrická množina. Jde o jednoduchou úlohu dobře známou z teorie lineárního programování a v této práci budou vystupovat její modifikace zahrnující náhodu.

## 2.2 Možný výskyt náhodnosti

Náhoda se může do úlohy (2.1) dostat několika způsoby. Pokud je náhoda v účelové funkci a ostatní veličiny jsou nenáhodné, pak může jít například o minimalizaci (případně maximalizaci) střední hodnoty vhodně zvolené užitkové funkce. Úloha pak může vypadat takto:

$$\begin{aligned} &\min \mathrm{E}(c^T x) \\ &Ax \geq b \\ &x \in X, \end{aligned} \tag{2.2}$$

kde $c$ je náhodný vektor. Příkladem takové úlohy je třeba maximalizace výnosu portfolia, kde $c$ představuje náhodné výnosy jednotlivých aktiv, $x$ počty zakoupených aktiv a $Ax \geq b$ omezení (například na objem investice).

Další z možností je úloha, ve které je náhoda umístěna v rovnicích omezení, ať už v koeficientech u rozhodovacích proměnných, v náhodné pravé straně nebo v obou najednou. Opět je možné požadovat, aby byly podmínky splněny pro střední hodnoty, nebo penalizovat odchylky při nesplnění rovnice. Jiným přístupem je snažit se zajistit splnění náhodou ovlivněných podmínek na předem zvolené hranici spolehlivosti $p$. Takováto úloha má následující tvar:

$$\begin{aligned} &\min c^T x \\ &\mathrm{P}(Ax \geq b) \geq p \\ &x \in X, \end{aligned} \tag{2.3}$$

kde je buď $A$ náhodná matice, nebo $b$ náhodný vektor (případně jsou náhodné $A$ i $b$). Příkladem náhodného vektoru $b$ může být uspokojení náhodné poptávky $b$ po produktech $x$, kdy se snažíme minimalizovat náklady $c^T x$.

Různé typy úloh s náhodou lze samozřejmě kombinovat, tedy je možné například maximalizovat očekávané náhodné výnosy při splnění náhodných podmínek a podobně.

V této práci se budu zabývat jen úlohami, kde náhodné koeficienty vystupují na pravých stranách omezení, zatímco $A$ je pevná matice.

## 2.3 Náhodné pravé strany

Úloha s náhodnou pravou stranou vychází z myšlenky, že hodnoty, kterých musí $Ax$ dosáhnout, jsou náhodné - například množství pacientů, kteří budou potřebovat během jednoho dne ošetřit. Úloha (2.1) pak má tvar

$$\begin{aligned} &\min c^T x \\ &Ax \geq \xi \\ &x \in X, \end{aligned} \tag{2.4}$$

kde $\xi$ je náhodný vektor.

Splnit takováto omezení s jistotou (tedy pro nejhorší možný případ) pak obvykle bývá velmi neekonomické (mít v nemocnici dost pokojů, kdyby onemocnělo celé město najednou) nebo přímo nemožné (mít dostatečné

zásoby, ať je výpadek zásobování libovolně dlouhý). Je tedy zapotřebí udělat ústupek od podmínek $Ax \geq \xi$.

Možností, jak to udělat, je vícero. Lze například zavést penalizační funkci, pokud omezení nebudou splněna, či pracovat s možností kompenzace (například pokud elektrárna nevyrobí dostatek elektřiny, může draze dokoupit, aby uspokojila své klienty).

V této práci však budu pracovat s tím, že budu požadovat splnění podmínek s předem danou pravděpodobností $p$. Výše pravděpodobnosti je volena podle toho, jak moc je důležité, aby byla podmínka splněna. Zatímco ve finaních může být dostačující hladina 0.95, v technických úlohách může být požadována i 0.9999 a vyšší.

Speciálním případem je úloha s individuálními omezeními. V této úloze požadujeme spolehlivost splnění nerovnic pro každé omezení zvlášť ($\mathrm{P}(Ax_i \geq \xi_i) \geq p_i$). V takovémto případě je vidět, že úlohu lze snadno přepsat v deterministickém a přehledném tvaru $\mathrm{F}_i(Ax_i) \geq p_i$, kde $\mathrm{F}_i$ je distribuční funce náhodné veličiny $\xi_i$. Toto se pak přepíše za pomoci kvantilu $Ax_i \geq \mathrm{F}_i^{-1}(p_i)$. Takto získáváme klasickou úlohu lineárního programování, kde nehrozí nekonvexní množina přípustných řešení ani jiné obtíže.

Při obecnější úloze se sdruženými omezeními dostáváme úlohu ve tvaru:

$$\begin{aligned} \min c^T x \\ \mathrm{P}(Ax \geq \xi) \geq p \\ x \in X, \end{aligned} \tag{2.5}$$

což je úloha (2.3) (jen s jinak pojmenovaným náhodným vektorem).

Když chceme znát pravděpodobnost, že pro dané $x$ bude $Ax \geq \xi$, musíme přirozeně znát rozdělení náhodné veličiny $\xi$. Není-li k dispozici, musíme použít nějakou jeho aproximaci (například odhadnout z historických dat). V praktických úlohách rozdělení zpravidla neznáme přesně, ale v úlohách teoretických (například při porovnávání nové metody s jinou již známou) naopak bývá mnohdy přesně určeno (aby bylo porovnání výsledků smysluplné, musí obě metody pracovat se stejným rozdělením a to může být samozřejmě popsáno přesně). Když známe rozdělení a jeho distribuční funkci, můžeme toho využít a přepsat úlohu (2.5) do následujícího tvaru:

$$\begin{aligned} \min c^T x \\ Ax - z \geq 0 \\ \mathrm{F}(z) \geq p \\ x \in X \end{aligned} \tag{2.6}$$

kde $\mathrm{F}(z)$ je zmiňovaná distribuční funkce, tj. $\mathrm{F}(z) = \mathrm{P}(z \geq \xi)$. Tato úloha je s předchozí úlohou (2.5) ekvivalentní, ale všechny proměnné jsou již deterministické. Problém náhodnosti je tak vyřešen (za využití znalosti distribuční funkce náhodného vektoru pravé strany), ale tento postup s sebou přináší také riziko ztráty konvexnosti úlohy, protože množina $\{z|\, \mathrm{F}(z) \geq p\}$ nemusí být (a u diskrétních rozdělení často nebývá) konvexní.

**Poznámka:** Na deterministická omezení se dá pohlížet jako na speciální případ náhodných omezení. Dobře je to vidět z posledních formulací ((2.5) a (2.6)). Pokud je $\xi_i$ pro nějaké $i$ konstantní a rovno $b$, pak můžeme psát $\mathrm{P}(\xi_i = b) = 1$ a tedy $\mathrm{F}(z) = 0$ pro $z_i < b$ a $\mathrm{F}(z) = \hat{\mathrm{F}}(z_{-i})$ pro $z_i \geq b$, kde $\hat{\mathrm{F}}(z_{-i})$ je sdružená distribuční funkce vektoru $\xi$ bez $i$-té složky v bodě $z$ bez $i$-té složky. V úlohách (2.5) a (2.6) pak není nutné vyčleňovat $x \in X$ a lze s těmito omezeními zacházet stejně jako s ostatními aniž by to zvedlo náročnost výpočtu $p$-level eficientních bodů (představených v kapitole 3.3).

## 2.4 Spojitá rozdělení

V případě spojitých rozdělení je klíčem ke konvexnosti množiny $\mathrm{F}(z) \geq p$ kvazikonkavita případně log-konkavita funkce $\mathrm{F}$.

**Definice 1** *Funkce $f(z)$ se nazývá kvazikonkávní, pokud pro všechna $z_1, z_2 \in \mathbb{R}^n$ a $\lambda \in (0,1)$ platí:*

$$f(\lambda z_1 + (1-\lambda)z_2) \geq \min(f(z_1), f(z_2))$$

**Definice 2** *Funkce $f(z)$ se nazývá log-konkávní, pokud pro všechna $z_1, z_2 \in \mathbb{R}^n$ a $\lambda \in (0,1)$ platí:*

$$f(\lambda z_1 + (1-\lambda)z_2) \geq f(z_1)^{\lambda} f(z_2)^{1-\lambda}$$

**Věta 1** *Je-li $f(z)$ kvazikonkávní, pak je pro každé $p$ množina $Z := (z|f(z) \geq p)$ konvexní.*

**Důkaz:** Vezměme $x, y \in Z$ a $\lambda \in (0,1)$. Označme konvexní kombinaci $x$ a $y$ $q = \lambda x + (1-\lambda)y$.

$$f(q) = f(\lambda x + (1-\lambda)y) \geq \min f(x,y) \geq p,$$

tedy $q \in Z$ a tudíž $Z$ je konvexní množina. Q.E.D.

Následující věta a její důkaz jsou převzaty z [10].

**Věta 2** *Je-li $f(z)$ log-konkávní, pak je i kvazikonkávní.*

**Důkaz:** Ukážeme, že pro libovolná $z_1, z_2 \in \mathbb{R}^n$ a $\lambda \in (0,1)$ z log-konkavity vyplývá kvazikonkavita. Log-koknavita znamená

$$f(\lambda z_1 + (1-\lambda)z_2) \geq f(z_1)^{\lambda} f(z_2)^{1-\lambda},$$

po zlogaritmování a drobných úpravách dostaneme tvar

$$\log(f(\lambda z_1 + (1-\lambda)z_2)) \geq \lambda \log(f(z_1)) + (1-\lambda)\log(f(z_2)).$$

Konvexní kombinace dvou čísel je větší nebo rovna menšímu z nich, tedy

$$\log(f(\lambda z_1 + (1-\lambda)z_2)) \geq \min(\log(f(z_1)), \log(f(z_2))).$$

Díky monotonii logaritmu je minimum z logaritmu dvou čísel rovno logaritmu minima těchto čísel a tedy

$$\log(f(\lambda z_1 + (1-\lambda)z_2)) \geq \log(\min(f(z_1), f(z_2))).$$

Stejně jako na začátku důkazu využijeme toho, že logaritmus je rostoucí funkce a odlogaritmujeme, čímž dostáváme

$$f(\lambda z_1 + (1-\lambda)z_2) \geq \min(f(z_1), f(z_2)).$$

Funkce je tedy kvazikonkávní. Q.E.D.

Díky těmto větám víme, že pro konvexitu úlohy (2.6) stačí, aby $\mathrm{F}(z)$ byla log-konkávní nebo kvazikonkávní. Úloha je pak řešitelná standardními prostředky. Tyto podmínky splňují například distribuční funkce normálního a exponenciálního rozdělení.

Dále víme, že pro náhodný vektor $\xi$ se spojitým rozdělením k log-konkávitě jeho distribuční funkce stačí, aby byla log-konkávní hustota. Tato je jako věta obecněji formulované v [8].

# Kapitola 3

# Diskrétní rozdělení

## 3.1 Využití

Diskrétní rozdělení mají využití jednak samostatně pro veličiny, které mohou nabývat pouze diskrétních hodnot, jako například počet nemocných, jednak jako aproximace spojitých náhodných veličin (například množství vody aproximované na stovky hektolitrů).

## 3.2 Problémy diskrétního rozdělení

U diskrétních rozdělení je však množina $\mathrm{F}(z) \geq p$ často nekonvexní. I zde se dají vyčlenit úlohy, kde pravděpodobnostní omezení nezkomplikují úlohu, ale na rozdíl od široké skupiny u spojitých rozdělení nepokryjí mnoho praktických úloh (potíže nevzniknou například tehdy, pokud je vektor pravých stran náhodných omezení jednorozměrný).

Pro ilustraci si představme následující úlohu:

**Příklad:**

$$\begin{array}{rcrl} & & \min x_2 & \\ x_1 & + & \frac{1}{2}x_2 & \geq \xi_1 \\ -x_1 & + & x_2 & \geq \xi_2 \\ & & x & \geq 0, \end{array} \tag{3.1}$$

kde $\xi$ je náhodný vektor, jehož rozdělení je popsáno tabulkou 3.1. Požadovaná hladina spolehlivosti je $p = 0.5$.

Úlohu si přepíšeme jako

| $\xi_1$ | $\xi_2$ | $\mathrm{P}(\xi_1, \xi_2)$ | $\mathrm{F}(\xi_1, \xi_2)$ |
|---|---|---|---|
| 0 | 0 | 0.1 | 0.1 |
| 0 | 1 | 0.1 | 0.2 |
| 0 | 2 | 0.05 | 0.25 |
| 1 | 0 | 0.1 | 0.2 |
| 1 | 1 | 0.05 | 0.35 |
| 1 | 2 | 0.15 | 0.55 |
| 2 | 0 | 0.15 | 0.35 |
| 2 | 1 | 0.1 | 0.6 |
| 2 | 2 | 0.2 | 1 |

Tabulka 3.1: Rozdělení vektoru $\xi$

$$\begin{array}{rcrcrl} & & & & \min x_2 & \\ x_1 & + & \frac{1}{2}x_2 & - & z_1 & \geq 0 \\ -x_1 & + & x_2 & - & z_2 & \geq 0 \\ & & & & \mathrm{F}(z) & \geq p \\ & & & & x & \geq 0 \\ & & & & z & \in \{0,1,2\} \times \{0,1,2\} \end{array} \tag{3.2}$$

S využitím distribuční funkce F můžeme snadno zakreslit množinu přípustných řešení, na obr. 3.1 je vyznačená modře.

Je vidět, že množina přípustných řešení i pro takto jednoduchou úlohu není konvexní. Tuto nepříjemnost je tedy zapotřebí řešit.

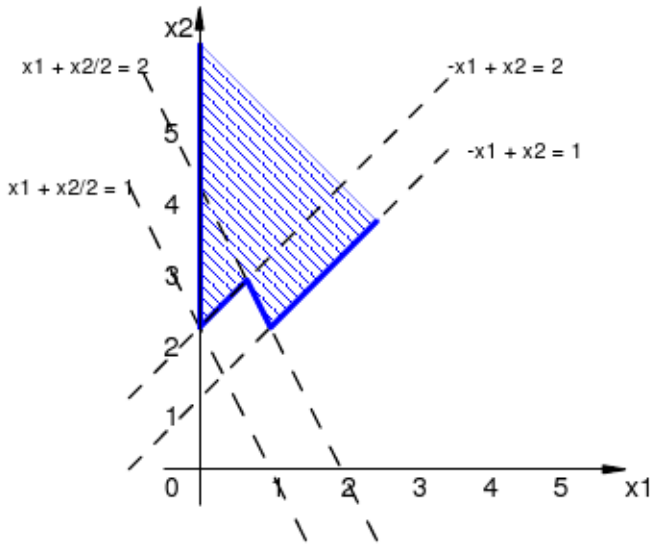
I v diskrétním případě se dají definovat log-konkávní funkce. Pro jednorozměrné náhodné veličiny dostáváme výsledky analogické ke spojité variantě. Máme-li pravděpodobnosti $\{p_i\}$, že náhodná veličina $\xi$ nabude hodnot $x_i$, pak řekneme, že rozdělení je log-konkávní, pokud

$$p_i^2 \geq p_{i-1} p_{i+1}, \ \forall i \tag{3.3}$$

V [8] je ukázané, že se tato log-konkavita přenáší i na distribuční funkci.

Pro vícerozměrné diskrétní rozdělení ale už nastávají problémy. Log-konkavita se dá definovat například rozšířením vztahu (3.3), ale pro mnohorozměrná diskrétní rozdělení již neplatí, že z log-konkavity pravděpodobností plyne log-konkavita distribuční funkce (viz [8]).

Na konvexní množinu přípustných řešení se tedy nelze spoléhat a je zapotřebí vypořádat se s její nekonvexností.




Obrázek 3.1:

## 3.3 p-level eficientní body

Optimální body $z$ v úloze (2.6) se nazývají $p$-level eficeintní body (zkráceně PLEP). Tyto body musí vyhovovat podmínce $\mathrm{F}(z) \geq p$ a zároveň mít co nejnižší hodnoty, aby byly optimální (kvůli podmínce $Ax - z \geq 0$). Tedy pokud by byl nějaký bod ve všech souřadnicích menší nebo roven $p$-level eficientnímu bodu a alespoň v jedné souřadnici ostře, pak hodnota distribuční funkce v tomto bodě bude ostře menší než $p$.

**Definice 3 ($p$-level eficientní body)** *Nechť* $\mathrm{F}(z)$ *je distribuční funkce a* $p \in [0, 1]$. *Pak* $\hat{z}$ *je* $p$*-level eficientním bodem funkce* $\mathrm{F}(z)$ *právě tehdy, když* $\mathrm{F}(\hat{z}) \geq p$ *a zároveň neexistuje* $\bar{z}$, $\bar{z} \leq \hat{z}$, $\bar{z} \neq \hat{z}$, *takové, že* $\mathrm{F}(\bar{z}) \geq p$.

Jak je patrné, PLEP jsou založené na eficienci známé z vícekriteriální optimalizace. Zde minimalizuji souřadnice vektoru $z$ a zároveň maximalizuji $\mathrm{F}(z)$. Zavedením "$\varepsilon$-omezení" v podobě $F(z) \geq p$ pak získáme $p$-level eficientní body.

Pokud máme spočítané PLEP, získáme na množinu přípustných řešení mnohem lepší pohled. Úlohu (2.6) zapíšeme s využitím znalosti PLEP, k če-

muž si zavedeme následující značení: $p$-level eficientní body budeme značit $z_s,\ s \in S$ a definujme $\forall s \in S,\ C_s := \{z | z \geq z_s\},\ C := \bigcup_{s \in S} C_s$. $C_s$ v podstatě značí množinu, vůči které je daný bod $z_s$ eficientní. S využitím těchto symbolů můžeme úlohu přepsat na

$$\begin{aligned} \min c^T x \\ Ax - z &\geq 0 \\ z &\in C \\ x &\in X. \end{aligned} \tag{3.4}$$

V této úloze jsou rozhodovací proměnné $x$ a $z$, úloha však jde zjednodušit. Protože pro každé $z \in C$ musí existovat index $s \in S$ takový, že $z$ leží v $C_s$, můžeme psát:

$$\begin{aligned} \min c^T x \\ Ax - \textstyle\sum_{s \in S} \lambda_s z_s &\geq 0 \\ \textstyle\sum_{s \in S} \lambda_s &= 1 \\ x &\in X, \end{aligned} \tag{3.5}$$

kde $\forall s \in S,\ \lambda_s \in \{0, 1\}$ je rozhodovací proměnná, zatímco $z_s$ jsou pevné. Z tohoto zápisu je vidět, že lze k úloze přistupovat tak, že minimalizuji účelovou funkci na množinách vytyčených jednotlivými $z_s,\ s \in S$ a poté vyberu minimum z těchto hodnot. Řešit ale větší množství úloh, které by takto vznikly, by bylo v mnoha případech výpočetně neúnosné. Existují postupy, jak neprocházet "neperspektivní" $C_s$ (viz například [3]), ale i odlišné přístupy. Jedním z nich je, že místo množiny vytyčené PLEP budeme pracovat s množinou vytyčenou jejich konvexním obalem (viz [10]). Lze snadno nahlédnout, že pokud uvolníme podmínku celočíselnosti $\lambda_s \in \{0, 1\}$ na $\lambda_s \in [0, 1],\ \forall s \in S$, pak již množina přípustných řešení bude konvexní a úlohu lze snadno řešit běžnými postupy. (Stačí si uvědomit si, že v takto modifikované úloze vymezují všechna omezení konvexní množiny a že průnik konvexních množin je konvexní množina.)

Uvolněná množina přípustných řešení je tedy konvexní a obsahuje původní množinu přípustných řešení (každé z původních přípustných řešení je přípustným řešením uvolněné úlohy s tím, že $\exists i,\ \lambda_i = 1$ a $\forall j \neq i,\ \lambda_j = 0$), ale bohužel nemusí vždy jít o její konvexní obal, jak ukazuje následující příklad.

| $\xi_1$ | $\xi_2$ | $\xi_3$ | $\mathrm{P}(\xi_1, \xi_2, \xi_3)$ | $\mathrm{F}(\xi_1, \xi_2, \xi_3)$ |
|---|---|---|---|---|
| 1 | 5 | 2 | 0.5 | 0.5 |
| 1 | 5 | 6 | 0.01 | 0.51 |
| 1 | 13 | 2 | 0.01 | 0.51 |
| 1 | 13 | 6 | 0.01 | 0.53 |
| 1 | 15 | 2 | 0.01 | 0.52 |
| 1 | 15 | 6 | 0.1 | 0.64 |
| 2 | 5 | 2 | 0.01 | 0.51 |
| 2 | 5 | 6 | 0.01 | 0.53 |
| 2 | 13 | 2 | 0.1 | 0.62 |
| 2 | 13 | 6 | 0.025 | 0.675 |
| 2 | 15 | 2 | 0.01 | 0.64 |
| 2 | 15 | 6 | 0.025 | 0.82 |
| 5 | 5 | 2 | 0.1 | 0.61 |
| 5 | 5 | 6 | 0.01 | 0.64 |
| 5 | 13 | 2 | 0.01 | 0.73 |
| 5 | 13 | 6 | 0.025 | 0.82 |
| 5 | 15 | 2 | 0.01 | 0.76 |
| 5 | 15 | 6 | 0.025 | 1 |

Tabulka 3.2: Rozdělení vektoru $\xi$

**Příklad:**

$$\begin{array}{rcrl} & & & \min x_1 + x_2 \\ x_1 & & & \geq \xi_1 \\ x_1 & + & 2x_2 & \geq \xi_2 \\ x_2 & & & \geq \xi_3, \end{array} \tag{3.6}$$

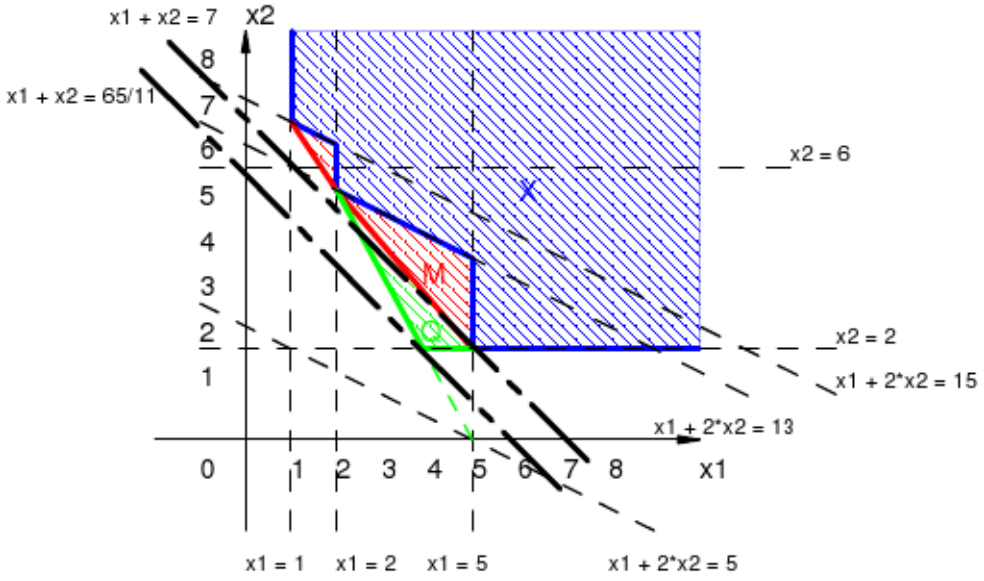
kde $\xi$ je náhodný vektor, jehož rozdělení je popsáno tabulkou 3.2. Požadovaná hladina spolehlivosti je $p = 0.6$.

V této úloze jsou $p$-level eficientní body tři, a to $(1, 15, 6)$, $(2, 13, 2)$ a $(5, 5, 2)$. Přeformulováním úlohy do tvaru (3.5) získáme toto:

$$\begin{array}{rcrcl} & & & & \min x_1 + x_2 \\ x_1 & & & - & \sum_{i=1}^{3} \lambda_i z_{i1} \geq 0 \\ x_1 & + & 2x_2 & - & \sum_{i=1}^{3} \lambda_i z_{i2} \geq 0 \\ & & x_2 & - & \sum_{i=1}^{3} \lambda_i z_{i3} \geq 0, \end{array} \tag{3.7}$$

kde $z_{ij}$ označuje $j$-tou složku $i$-tého $p$-level eficientního bodu.

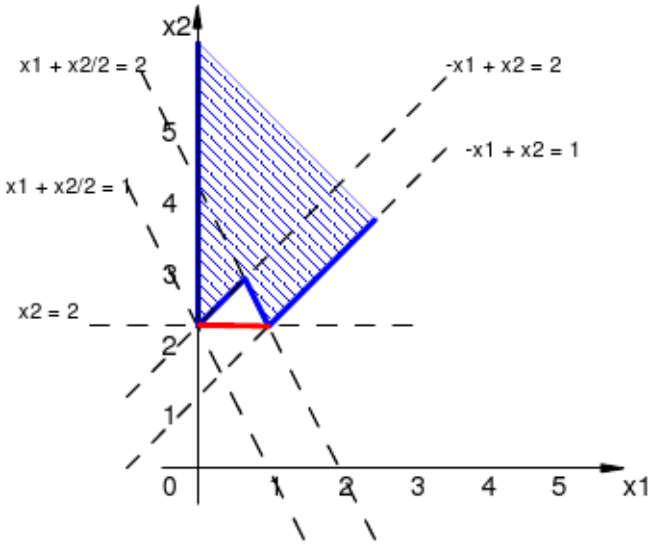
Na obr. 3.2 je pak vidět, jak vypadá množina přípustných řešení původní úlohy (označena $X$ a modrou barvou), jak vypadá konvexní obal této množiny (rozšířen o červenou množinu $M$) a jak vypadá uvolněná množina přípustných řešení (ke konvexnímu obalu přibude ještě zelená část označená $Q$).




Obrázek 3.2:

Jak je vidět, místo správného optimálního řešení $x_1 = 5, x_2 = 2$ vyjde jako optimální řešení $x_1 = \frac{43}{11}, x_2 = 2$ a hodnota účelové funkce klesne o více než 1. Koeficienty $\lambda_i$ vyšly $\lambda_1 = 0, \lambda_2 = 0.364$ a $\lambda_3 = 0.636$, řešení tedy naplňovalo omezení dané druhým a třetím $p$-level eficientním bodem. Jejich třetí složka je stejná (má hodnotu 2) a tedy je to stejné, jako bychom dělali spojnici bodů $(5, 0)$ a $(2, 5)$ (na obrázku (3.2) je vyznačena zelenou přerušovanou čarou) a až tu omezovali nerovnicí $x_2 \geq 2$.

Otázkou je, jak se projeví uvolnění úlohy na její optimální řešení. Z teorie lineárního programování víme (např. [7]), že optimální řešení leží v krajních

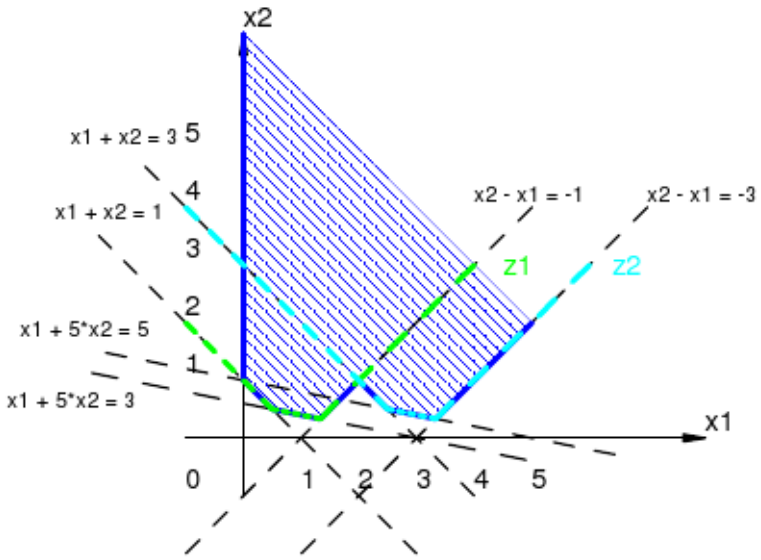



Obrázek 3.3: Zvětšení množiny optimálních řešení při relaxaci

bodech množiny přípustných řešení a pokud je krajních bodů optimálních více, pak jsou optimální i všechny body jejich konvexního obalu. V úlohách, kde nedochází k nadměrnému rozšíření množiny přípustných řešení, ale uvolňuje se jen v mezích konvexního obalu, je tedy vše v pořádku, neboť žádné nové krajní body nepřibývají. Mohou sice přibýt optimální řešení, ale optimální řešení původní úlohy mezi nimi zůstanou, stačí vzít krajní body množiny optimálních řešení (takové, kde vektor $\lambda$ má jednu složku rovnou jedné a ostatní nule). Ukázka nadbytečných řešení je na datech z příkladu (3.1) vyobrazena na obrázku 3.3, kde červeně vyznačena úsečka značí množinu optimálních řešení uvolněné ulohy. Původní úlohu přitom řeší jen její krajní body $(0, 2)$ a $(1, 2)$.

Pokud je důsledkem uvolnění množina větší než konvexní obal, může dojít i k tomu, že optimální řešení uvolněné úlohy není přípustným řešením původní úlohy, jako v příkladě (3.6). V takovém případě má výsledné řešení nižší než požadovanou spolehlivost.

**Poznámka:** Je dobré si uvědomit, že PLEP nejsou hodnoty proměnné $x$, ale $z$. Pokud si vyznačíme množinu přípustných řešení, tak PLEP na




Obrázek 3.4: Množina přípustných řešení

ní nejsou zobrazeny jako body, ale jako přímky vyznačující omezení (pravé strany nerovností). Jako body jsou zobrazeny v prostoru v němž se realizuje vektor $\xi$. Názorně to ukazuje následující příklad. Je na něm mimo jiné například vidět, že řešení leží v prostoru dimenze 2 a PLEP v prostoru dimenze 3.

**Příklad:**

$$\begin{array}{rcrl} & & \min 2x_1 + x_2 & \\ x_1 & + & x_2 & \geq \xi_1 \\ x_1 & + & 5x_2 & \geq \xi_2 \\ -x_1 & + & x_2 & \geq \xi_3 \\ & & x & \geq 0, \end{array} \tag{3.8}$$

kde $\xi$ je náhodný vektor, jehož rozdělení je popsáno tabulkou 3.3. Požadovaná hladina spolehlivosti je $p = 0.6$.

Hodnoty $p$-level eficientních bodů jsou $z_1 = (1, 3, -1)$ a $z_2 = (3, 5, -3)$. Na obrázku 3.4 jsou naznačeny množiny vymezené jim odpovídajícími ome-

| $\xi_1$ | $\xi_2$ | $xi_3$ | P($\xi_1, \xi_2, \xi_3$) | F($\xi_1, \xi_2, \xi_3$) |
|---|---|---|---|---|
| 1 | 3 | -3 | 0.4 | 0.4 |
| 1 | 3 | -1 | 0.2 | 0.6 |
| 1 | 5 | -3 | 0.05 | 0.45 |
| 1 | 5 | -1 | 0.04 | 0.69 |
| 3 | 3 | -3 | 0.1 | 0.5 |
| 3 | 3 | -1 | 0.05 | 0.75 |
| 3 | 5 | -3 | 0.06 | 0.61 |
| 3 | 5 | -1 | 0.1 | 1 |

Tabulka 3.3: Rozdělení vektoru $\xi$

zeními a je tedy vidět, jak se $z_1$ a $z_2$ podílejí na tvaru množiny přípustných řešení.

Praktický výpočet úlohy (2.5) pak probíhá tak, že se nejprve spočtou $p$-level eficientní body, k čemuž lze použít algoritmus z následující kapitoly. Pak se provede uvolnění, čímž se získá úloha ve tvaru (3.5). Na tuto úlohu jdou už použít obvyklé algoritmy (třeba simplexová metoda) a řešiče pro lineární úlohy (například CPLEX a další z [5]).

Další metoda pracuje se souborem deterministických úloh $U_s,\ s \in S$. V úloze $U_s$ bereme vektor omezení roven $s$-tému $p$-level eficientnímu bodu, zatímco $A$ a $c$ zůstávají stejné. Tyto úlohy vyřešíme každou zvlášť (jde o obyčejné deterministické úlohy lineárního programování) a označíme $M_s^*$ množinu optimálních řešení každé úlohy $U_s$. Pak stačí najít celkové optimální řešení $\hat{x} = \min_{s \in S} c^T x, x \in \bigcup_{s \in S} M_s^*$. Nevýhodou tohoto postupu je, že $p$-level eficientních bodů může být mnoho a pro každý z nich musíme řešit potenciálně velkou úlohu, může se značně prodloužit doba výpočtu. Pro lineární úlohy je tedy tato metoda pomalejší než uvolnění množiny přípustných řešení, ale může přinést větší přesnost.

## 3.4 Algoritmus na výpočet PLEP

Dále popsaný algoritmus je (bez důkazu) převzat z [8]. Jeho aplikací získáme $p$-level eficientní body pro konečně-rozměrný náhodný vektor nabývající konečně mnoha hodnot. Možné hodnoty vektoru v algoritmu nefigurují, je důležitý jen počet možných hodnot v každé složce, proto budu nadále bez

újmy na obecnosti předpokládat, že je vektor $\xi$ definován na $\{1, 2, \ldots, k_1\} \times \{1, 2, \ldots, k_2\} \times \ldots \times \{1, 2, \ldots, k_r\}$. Toto zjednodušení usnadní implementaci algoritmu a každé jiné zadání lze na tento tvar přepsat.

**Algoritmus:**

1. krok: $k = 0$

2. krok:

$$\begin{aligned} z_1 &= \operatorname{argmin}\{y |\, \mathrm{F}(y, k_2, k_3, \ldots, k_r) \geq p\} \\ z_2 &= \operatorname{argmin}\{y |\, \mathrm{F}(z_1, y, k_3, \ldots, k_r) \geq p\} \\ &\vdots \\ z_r &= \operatorname{argmin}\{y |\, \mathrm{F}(z_1, z_2, z_3, \ldots, y) \geq p\} \end{aligned}$$

3. krok: $E = \{z_1, z_2, \ldots, z_r\}$

4. krok: $k = k + 1$. Pokud $z_1 + k \geq k_1$ tak přeskoč na 6. krok.

5. krok: Spočti všechny $p$-level eficientní body funkce $\mathrm{F}(z_1 + k, y)$, kde $y \in \mathbb{R}^{r-1}$. Přidej do $E$ takové z nich, které nedominují žádný prvek z $E$ (tedy neexistuje takové $z \in E$, že pro daný bod $h$ by platilo $z \leq h$ a $z \neq h$). Pak se vrať ke 4. kroku.

6. krok: Konec algoritmu. Množina $E$ obsahuje všechny $p$-level eficientní body distribuční funkce $\mathrm{F}(x)$.

Následující věta ukazuje, že algoritmus je kvalitní v tom ohledu, že skutečně spočte $p$-level eficientní body, a to v konečném čase.

**Věta 3** *Algoritmus na hledání p-level eficientních bodů skončí po konečném počtu kroků a najde všechny p-level eficientní body.*

**Důkaz:** Nejprve konečnost algoritmu. Jediný krok, který by mohl dělat problémy je 5. krok, kde dochází k rekurzi. Hledání p-level eficientních bodů se ale provádí na funkci, která má o jednu proměnnou méně (ostatní jsou fixované parametry). Postupně tedy rekurze dorazí do bodu, kdy má funkce už jen jednu proměnnou a dále už nepostupuje. Tedy k rekurzivnímu počtu volání dochází jen konečně mnohokrát a každé se skládá z konečně mnoha

kroků (hledání minima na končné množině, přiřazování proměnných, porovnávání konečně mnoha bodů). Konečnost algoritmu je tedy dokázána.
Dále tvrdím, že algoritmus najde právě všechny $p$-level eficientní body. Bod $\hat{z} = \{z_1, z_2, \ldots, z_r\}$ nalezený v 1. kroku zřejmě definici vyhovuje (žádná složka nemůže klesnout, aby byla stále dodržena hladina $p$). Rekurzivní část pak zkouší zvedat první složku a zmenšovat ostatní. Pokud se to nepodaří, výsledek se do množiny $E$ nezařadí kvůli podmínce s dominancí. Pokud existuje takový bod, že jeho první složka je vyšší než u bodu $\hat{z}$ a jedna z následujících nižší (s tím, že jsou stále minimální při zachování hladiny $p$), pak jsme našli další PLEP. Pro takový pak zkouším zvednout druhou složku a snižovat ostatní. Tento postup opakuji dokud to lze. Je zřejmé, že všechny takto získané body jsou $p$-level eficientní. Zároveň algoritmus projde všechny body $z$ takové, že $\mathrm{F}(z) \geq p$ tedy žádný PLEP nemůže vynechat.
Algoritmus tedy skončí po konečně mnoha krocích a najde právě všechny $p$-level eficientní body. Q.E.D.

# Kapitola 4

# Implementace algoritmu v R

## 4.1 Úvod

Algoritmus jsem implementoval v programu R. Důvodem bylo pohodlné zadávání vstupních dat a snadná manipulace s vektory a proměnnými, včetně používání předdefinovaných funkcí pro jednoduché operace. Kromě výhod jsem se však setkal i s několika nevýhodami (nemožnost předávat proměnné do funkcí odkazem například) a je známé, že programy v R neběží tak rychle jako v jiných jazycích (matlab, C). Přepsání zdrojového kódu do jiného jazyka by tedy mohlo přinést větší čistotu kódu a vyšší rychlost. Více o R se lze dozvědět z [9].

## 4.2 Obsluha programu

Program se obsluhuje snadno. Pro přehlednost je rozdělen na dvě části. V první se zadávají data a vypočítává sdružená distribuční funkce. V druhé pak probíhá samotný výpočet PLEP.

```
##################
## Priprava dat ##
##################
```

První příkaz `source("PLEPS.R")` načte funkce potřebné pro běh programu. Je zapotřebí mít v R nastavený adresář ve kterém se nachází soubor PLEPS.R jako pracovní. Do proměnné `slozky` pak uživatel uloží přípustné hodnoty jednotlivých složek $\xi_1, \xi_2, \ldots, \xi_r$. Program si pak spočte pomocný údaj `max` -

počet hodnot, kterých mohou jednotlivé složky nabývat. V dalším kroku se vytvoří vícerozměrná "matice" pro pravděpodobnosti nabytí jednotlivých bodů $(\xi_1, \xi_2, \ldots, \xi_r)$ a všechny se předvyplní hodnotou 0. Tu pak může uživatel snadno předefinovat pro vybrané body: příkaz `psti[`$i_1, i_2, \ldots, i_r$`] <- <- q;` dosadí pravděpodobnost $q$ pro bod, jehož hodnota je v první složce na pozici $i_1$ (řazeno od nejmenší možné hodnoty po největší), ve druhé složce na pozici $i_2$ a tak dále. Tedy `psti[1,4,2]` znamená pravděpodobnost, že $\xi_1$ nabude své nejnižší možné hodnoty, $\xi_2$ své čtvrté nejnižší možné hodnoty a $\xi_3$ zároveň své druhé nejnižší možné hodnoty. Takovéto neobratné zadávání je způsobeno reprezentací "matice" v programu R. Pokud místo hodnoty $i_j$ napíše uživatel $(i_{j_1} : i_{j_2})$, vyplní se ta samá pravděpodobnost pro body, kde na $j$-té pozici jsou postupně čísla $i_{j_1}, i_{j_1+1}, \ldots, i_{j_2}$. Následuje kontrolní součet, který má ověřit, zda součet všech pravděpodobností je opravdu 1.

Poslední příkaz v této části spočte sdruženou distribuční funkci vektoru $\xi$ ze zadaných pravděpodobností. Pokud uživatel zná přímo distribuční funkci, nemusí ji přirozeně počítat. V takovém případě do pole `psti` vyplňuje rovnou hodnoty distribuční funkce a značení srovná příkazem `F <- as.array(psti)`. V obou případech je distribuční funkce uložena jako vektor, zobrazit jako "vícerozměrná matice" se dá příkazem `array(F,max)`.

```
########################################
## Vypocet p-level eficientnich bodu: ##
########################################
```

Do proměnné `p` pak uživatel zadá hladinu $p \in (0, 1)$, na které se budou $p$-level eficientní body hledat. Program funguje i pro hodnoty 0 a 1, ale pro ně je zbytečné používat takto složitý postup. Následující příkaz stačí zavolat a $p$-level eficietní body se vypočítají. Následuje série příkazů, které vytvářejí výstup kompatibilní s GAMSem a ukládají ho do textových souborů `mnozina.inc` a `plepsy.inc`. Je zapotřebí vyplnit cestu do adresáře, kam se mají uložit a odkud si je bude poté GAMS brát (například `C:/Vypocty /GAMS/Pravdepodobnostni_omezeni`). Po posledním příkazu se vypíší hodnoty PLEP přímo na obrazovku, pokud si je chce uživatel zkontrolovat.

Pokud uživatel zadá někde nesmyslné hodnoty ($p \notin [0, 1]$, součet pravděpodobností různý od 1 a podobně), program ho nijak nezastaví a pokusí se provést výpočet, který samozřejmě nedopadne správně a nejspíš skončí chybou. Cílem práce nebylo vytvořit dokonalý program, ten je jen nástrojem, který se musí správně používat.

## 4.3 Popis fungování programu

Funkce budu popisovat podle toho, jak jsou v programu volány, ne podle jejich pořadí při definování.

První funkcí je `dimenze <- function(slozky)`, která jen projde jednotlivé vektory uložené v seznamu `slozky` a spočte jejich délky. Takto získané hodnoty `max` jsou široce využívány v celém programu, neboť program pracuje pouze s počty možných hodnot jednotlivých složek, ne s jejich hodnotami (kvůli snažší a přehlednější implementaci algoritmu).

Prvním větším celkem je výpočet sdružené distribuční funkce ze zadaných pravděpodobností. Tato část není nezbytně nutná - pokud uživatel distribuční funkci zná, může tento výpočet přeskočit. Většinou však zná pravděpodobnosti a sdruženou distribuční funkci, bez níž se výpočet PLEP neobejde, je nejprve nutné spočítat. Aby nebylo zapotřebí předávat velká pole mnoha funkcím a tím je kopírovat, funkce `distribuuj(psti,max)` tvoří prostředí, ve kterém jsou definované všechny funkce, které využívá. Ty tak mají přístup do jejích proměnných, jako by byly globální, ale přitom nehrozí kolize s globálními proměnnými, pokud by byla tato implementace začleněna do většího programu. Protože si uživatel může navolit počet rozměrů (v kódu označováno proměnnou `r`), zvolil jsem přepis vícerozměrného pole na jednorozměrné se složkami řazenými za sebe (označeno `ps`). Pro každý prvek `ps` se pak spočítají souřadnice v původním mnohorozměrném poli a uloží do proměnné `s`. Tento výpočet řídí funkce `spocti_souradnice()`, což je vlastně jen cyklus volající funkci `souradnice(i)` pro všechny prvky pole `ps`.

Struktura vektoru `ps` je taková, že nejprve při pevných souřadnicích $2, 3, \ldots, r$ prochází hodnoty první souřadnice, pak o 1 zvedne druhou souřadnici a proces opakuje. Takto postupně projde všechny hodnoty. Tedy aby se poslední ($k_r$-tá) složka zvedla o 1, musí se index `i` posunout o $\prod_{l=1}^{r-1} k_l$. Proto když `i` celočíselně vydělíme tímto součinem, dostaneme hodnotu $r$-té složky. Když pak index očistíme o vliv poslení složky, můžeme analogicky spočítat předposlední a tak dále. Počítání dobře funguje pro indexování od 0, proto si na začátku index o 1 zmenším a na závěr při výstupu indexy opět o 1 zvětším.

S touto úpravou pak již lze přikročit k samotnému výpočtu distribuční funkce, který řídí funkce `distribuuj(ps,max)` (poté, co si definuje používané funkce). Ta si vytvoří vektor `F`, přes jehož prvky pak prochází cyklus, během kterého funkce `distribuuj(ps,max)` zapisuje hodnoty distribuční

funkce v daném bodě. Tyto hodnoty mu dodá funkce `secti_ps(i)`, která využívá pole `s` pro porovnávání souřadnic prvku na `i`-tém místě. Prochází totiž celé pole `ps` až po prvek `i` a sčítá pravděpodobnosti pro prvky, jejichž souřadnice jsou menší nebo rovny těm na místě `i`. Tak se samozřejmě spočte sdružená distribuční funkce v daném bodě. Tato část algoritmu by pravděpodobně šla zrychlit, aby se rovnou procházely pouze body, které mají menší souřadnice, ale to není předmětem této práce.

Hlavní úkol programu, výpočet $p$-level eficientních bodů, řídí funkce `plepsuj(F,max,p)`. Stejně jako v minulém případě jsou jí využívané funkce definované uvnitř, aby bylo snažší předávání proměnných. Funkce kromě poskytnutí proměnných pro své podfunkce volá jen dvě funkce. První je rekurzivní funkce `spocti_pleps(zacatek)` s počáteční hodnotou `zacatek` $= \emptyset$. Druhá funkce, `prepocitej(PLEPs)`, spočte hodnoty $p$-level eficientních bodů ze znalosti pořadí jejich hodnot (jak jsem psal již výše, program počítá PLEP nezávisle na hodnotách náhodného vektoru $\xi$, stačí znát jejich pořadí v jednotlivých složkách).

Funkce `spocti_pleps(zacatek)` je pak již konečně implementací algoritmu na výpočet $p$-level eficientních bodů. 1. a 3. krok jsou triviální, proto nejsou dále podrobněji popsány.

2. krok algoritmu zajišťuje funkce `najdi_plep(zacatek)`. Ta předpokládá, že prvních několik složek je fixovaných (na začátku je to samozřejmě 0 složek) a přes ostatní postupně minimalizuje. Na minimalizaci si volá funkci `minimalizuje(m,vek,i)`, kde `vek` je vektor sestávající se ze tří částí - fixovaného začátku, složky `i` (přes kterou se minimalizuje) a konce fixovaného nejvyššími hodnotami v daných složkách. Protože distribuční funkce je ve složkách neklesající, stačí při minimalizaci `i`-té složky vždy najít první hodnotu, pro kterou je překročena hladina $p$ a vyššími hodnotami se není třeba zabývat. Protože souřadnice jsou udávány vektorem a hodnoty distribuční funkce `F` jsou uloženy v jednorozměrném poli, musí se v průběhu funkce přepočítat, což zajišťuje funkce `souradnicim_cislo(souradnice)`, postupem analogickým k vypočítávání souřadnic z pozice v jednorozměrném poli.

Poté se rozeběhne while cyklus (4. krok), který pomocí binární proměnné `do` kontroluje, zda nepřekračujeme maximální možnou hodnotu v dané složce vektoru $\xi$. Pokud ne, spustí funkci `spocti_pleps(zacatek)` pro vektor o jednu volnou složku kratší (tedy s jednou fixovanou složkou navíc), což je 5. krok algoritmu. Výslednou množinu potenciálních PLEP označenou `H` sloučí se současnou (označenou `E` - pozor, toto `E` nemusí být finální množinou $E$, pokud jsme uvnitř rekurze!) tak, aby žádný z bodů `H` nedominoval žádný z

bodů `E`. O to se stará funkce `sluc(E,H)`, která postupně přidává do množiny `E` body z množiny `H` pomocí funkce `pridej(E,bod)`, která bod opravdu přidá, jen pokud nedominuje žádný z množiny `E`. Rekurze se zastaví ve chvíli, kdy bychom chtěli hledat PLEP a všechny složky vektoru $\xi$ už byly fixované.

Funkce `spocti_pleps(zacatek)` pak vrací jako svoji hodnotu množinu bodů podezřelých z toho, že jsou PLEP. Ta se slučuje s množinou získanou v dřívější části rekurze. Až se takto dostaneme až k původnímu volání této funkce, budou v množině `E` právě všechny $p$-level eficientní body distribuční funkce `F` (6. krok).

Převedení výsledků do množiny původních hodnot vektoru $\xi$ je již triviální záležitost. Stačí projít všechny PLEP a jejich hodnoty nahradit hodnotami v příslušných vektorech možných hodnot složek $\xi$ (uložených v proměnné `slozky`).

## 4.4 Zpracování výstupu programem GAMS

Pro GAMS neuvádím zdrojový kód, neboť ten je již zcela vázán na konkrétní řešenou úlohu. Důležité je jen to, jak se do něj $p$-level eficientní body začlení. Program psaný v R poskytuje výstup do souborů, které se pak dají do GAMSu načíst příkazem `$include`. Pro načtení je důležité znát cestu k souborům, kterou uživatel zadal při výpočtu v R-ku, označme si ji "Cesta". Při samotném programování v GAMSu pak stačí při definici množin přidat příkaz `$include "Cesta\mnozina.inc"`. Tento příkaz přidá do GAMSU následující řádek:

```
s PLEPsy /1*S/
```

Místo `S` pak bude konkrétní počet $p$-level eficientních bodů. Je také dobré si uvědomit, že `s` v daném programu se tím stává vyhrazeným názvem pro množinu indexů $p$-level eficientních bodů. Toto se dá samozřejmě změnit změnou výstupu z R-ka (buď na jiný název proměnné, nebo nechat vypsat jen samotné množiny `1*S` a zbytek doplnit v GAMSu) či úpravou souboru "mnozina.inc" v textovém editoru.

Druhý vstup je pak při definování parametrů. Příkazem `$include "Cesta\plepsy.inc"` pak GAMS načte konkrétní hodnoty PLEP, které mohou vypadat například takto:

```
z(k,s) plepsy /
2.1 18.8
2.2 38.6
2.3 38.6
3.1 62.6
3.2 23.3
3.3 62.6
4.1 20.3
4.2 20.3
4.3 3.6
/
```

Tento vstup počítá s tím, že množina indexující rovnice, ve kterých se vyskytují pravděpodobnostní omezení, se jmenuje `k` (a že množina indexující PLEP se opět jmenuje `s`). Stejně jako v minulém případě se dá toto změnit ve výstupu z R-ka jednoduchými úpravami ve zdrojovém kódu nebo samotém souboru `plepsy.inc` případně v kombinaci s GAMSem. Je dobré si povšimnout toho, jaký formát byl zvolen - první index označuje rovnici, druhý pak označuje, o který PLEP se jedná. V tomto příkladu jsou tedy PLEP tři, a to $(18.8, 62.6, 20.3)$, $(38.6, 23.3, 20.3)$ a $(38.6, 62.6, 3.6)$.

Pomocí těchto příkazů se tedy do programu zahrnou $p$-level eficientní body spočtené v R-ku a s těmi se zachází z hlediska jazyka jako s obyčejnými parametry. Pro modelování úlohy v GAMSu viz [6].

# Kapitola 5

# Zobecnění

## 5.1 Celočíselná úloha

Při zobecňování úlohy (2.5) jsem se nejprve zaměřil na celočíselné omezení proměnné $x$. Úlohu nejprve zformulujeme s využitím $p$-level eficientních bodů do tvaru (3.5). Pokud úlohu neuvolníme, řešíme ji pro $\lambda_i \in \{0, 1\}\ \forall i$, tedy hledáme minima na jenotlivých množinách vytyčených $p$-level eficientními body stejně, jako jsme to mohli udělat v lineárním případě. Rozdíl je jen v tom, že je třeba použít algoritmy pro celočíselné programování (viz např. [5]). Takovýto postup dá přesné řešení, nicméně jak bylo řečeno výše, při velkém množství PLEP bude výpočet úlohy trvat příliš dlouho.

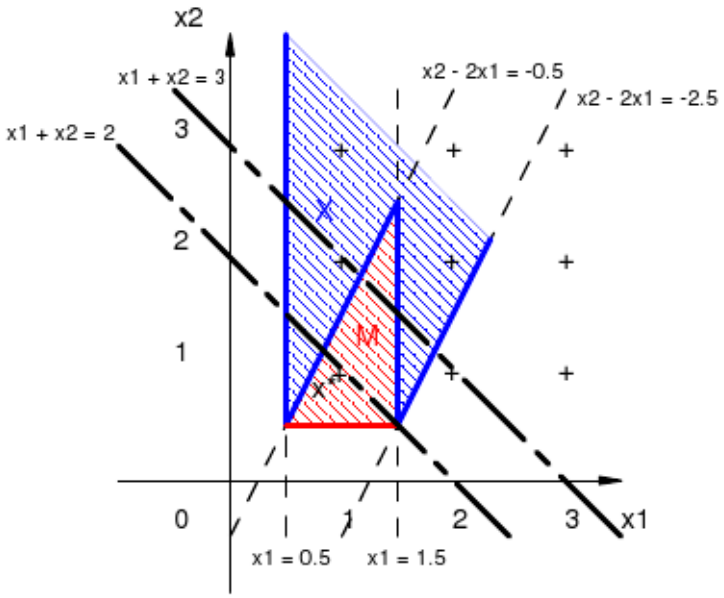
Budu tedy zkoumat relaxaci množiny přípustných řešení popsanou pomocí uvolnění $\lambda_i \in [0, 1]\ \forall i$, stejně jako v kapitole 3, a zjišťovat důsledky jejího použití. Na následujícím příkladě je vidět, že i pro jednoduchou situaci mohou oproti úloze se spojitými proměnnými přibýt další obtíže.

**Příklad:**

$$\begin{array}{rcl} & & \min x_1 + x_2 \\ x_1 & & \geq \xi_1 \\ -2x_1 & + & x_2 \geq \xi_2 \\ & & x \geq 0,\ x \text{ celočíselné} \end{array} \tag{5.1}$$

kde $\xi$ je náhodný vektor, jehož rozdělení je popsáno tabulkou 5.1. Požadovaná hladina spolehlivosti je $p = 0.9$.

Množina přípustných řešení $X$ původní úlohy při vynechání podmínek celočíselnosti je na obrázku 5.1 vyšrafována modře. Při uvolnění podmínek




Obrázek 5.1: Řešení uvolněné a neuvolněné úlohy

| $\xi_1$ | $\xi_2$ | $P(\xi_1, \xi_2)$ | $F(\xi_1, \xi_2)$ |
|---|---|---|---|
| 0 | -2.5 | 0.18 | 0.18 |
| 0 | -0.5 | 0.05 | 0.23 |
| 0.5 | -2.5 | 0.65 | 0.83 |
| 0.5 | -0.5 | 0.03 | 0.91 |
| 1.5 | -2.5 | 0.07 | 0.9 |
| 1.5 | -0.5 | 0.02 | 1 |

Tabulka 5.1: Rozdělení vektoru $\xi$

s nerovnostmi se rozšíří o červeně šrafovanou množinu $M$. Jak je vidět, uvolněná úloha má jediné optimální celočíselné řešení a to $x^* = (1, 1)$. Ale $x^* \notin X$, není tedy řešením původní úlohy. Optimálním řešením neuvolněné úlohy by v tomto příkladě byl zřejmě bod $(1, 2)$.

Vyvstává přirozená otázka, jak velká chyba může vzniknout, pokud použijeme řešení uvolněné úlohy $x^*$. V takovémto případě nemusí být splněna podmínka $Ax^* \geq \xi$ s pravděpodobností $p$ ani vyšší. Skutečnou pravděpodobnost snadno spočteme ze znalosti distribuční funkce, protože $\mathrm{P}(Ax^* \geq \xi) = F(Ax^*)$. Tuto pravděpodobnost si označím $p^*$ a podívám se, jak vzdálená může být od požadované pravděpodobnosti $p$. Tato analýza může být užitečná, protože nám dává představu o nejhorší možné spolehlivosti ještě před samotným výpočtem optimálního řešení.

Definujme $\hat{z},\ \hat{z}_j = \min_s z_j^s$. Potom triviální mez je $p^* \geq \mathrm{F}(\hat{z})$, protože platí $\{x|Ax - \sum_{s\in S} \lambda_s z_s \geq 0, \sum_{s\in S} \lambda_s = 1, \lambda_s \in [0, 1]\ \forall s\} \subset \{x|Ax - \hat{z} \geq 0\}$ (konvexní kombinace vektorů je větší nebo rovna vektoru složenému z minim jednotlivých složek).

Například v úloze (5.1), kde jsou PLEPs $z_1 = (0.5, -0.5)$ a $z_2 = (1.5, -2.5)$ je vektor $\hat{z} = (0.5, -2.5)$. $\mathrm{F}(\hat{z}) = 0.83$ a tedy spolehlivost řešení bude nejméně 0.83, což opravdu pro $x^* = (1, 1)$ platí (zde je dokonce $\mathrm{F}(1, 1) = 0.83$). Je dobré si uvědomit, že $p^*$ závisí silně na typu úlohy. Čím větší hodnoty distribuční funkce jsou na bodech ve všech složkách menších nebo rovných složkám PLEPs, tím menší chyby se při uvolnění úlohy dopouštíme. Naopak pokud pro všechna $z \leq z_s,\ \forall s \in S$, $z_s$ $p$-level eficientní platí $\mathrm{F}(z) = 0$, pak pro optimální řešení uvolněné úlohy může platit i $P(Ax^* \geq 0) = 0$. V takovém případě ale bude patrně i nízká hodnota spolhlivosti $p$, aby mohlo vůbec být PLEP více.

Praktický výpočet probíhá analogicky s případem obyčejné lineární úlohy (2.5). Nejprve se pomocí algoritmu z kapitoly 3.4 spočtou $p$-level eficientní body, úloha se uvolní a následně se aplikuje některý ze známých algoritmů zabudovaných v řešičích (např. z [5]). U výsledného řešení se pak pro ověření spočte spolehlivost.

I zde existuje možnost výpočtu, který umožní najít řešení splňující podmínky s požadovanou pravděpodobností (přesný výpočet), ale i zde je jeho aplikace přes soubor celočíselných deterministických úloh (pro každý PLEP samostatná úloha) potenciálně velmi pomalá. Pokud však velmi záleží na přesnosti nebo úloha není příliš rozsáhlá, lze o něm uvažovat. Dá se například najít řešení pomocí postupu popsaném v předchozím odstavci, a pokud není

dostatečně spolehlivé, můžeme aplikovat hledání po jednotlivých úlohách. Další možností je spolehlivost odhadnout a podle toho se rozhodnout, kterou metodu využít.

## 5.2 Nelineární úloha

Jinou z možností, jak zobecnit původní úlohu (2.5), je ustoupit od lineární účelové funkce a lineárních omezení k nelineárním. Úlohu pak zformuluji takto:

$$\begin{aligned} &\min f(x) \\ &\mathrm{P}(g(x) \geq \xi) \geq p \\ &x \in X, \end{aligned} \tag{5.2}$$

Otázkou samozřejmě je, pro jaké funkce $f(x), g(x)$ umíme úlohu řešit a jak je to se spolehlivostí takového řešení. V této kapitole vycházím z práce [2], kde za podmínek konvexnosti $f(x)$ a $g(x)$ navrhli aproximativní algoritmy pro řešení této úlohy.

Pokud je $f(x)$ konvexní funkce, $g(x)$ je konkávní ve složkách a $X$ je konvexní množina, můžeme aplikovat stejný postup jako v předchozích kapitolách. I zde můžeme převést úlohu na tvar analogický s úlohou (3.5). PLEP se počítají stejně jako u lineární úlohy, protože při jejich výpočtu záleží jen na pravděpodobnostním rozdělení vektoru pravých stran. Stejně jako v předchozích případech tak získáme úlohu

$$\begin{aligned} &\min f(x) \\ g(x) - \textstyle\sum_{s\in S} \lambda_s z_s &\geq 0 \\ \textstyle\sum_{s\in S} \lambda_s &= 1 \\ x &\in X \\ \lambda_s &\in \{0,1\}, \end{aligned} \tag{5.3}$$

kde $z_s,\ s \in S$ jsou $p$-level eficientní body. Tuto úlohu lze opět řešit pomocí souboru úloh, kde v $s$-té je nenulové právě $\lambda_s$, a pak najít minimum $f(x)$ mezi řešeními těchto subproblémů. Jediným rozdílem oproti lineární úloze je použití algoritmů a solverů pro nelineární programování. Na faktu, že tento postup může být příliš pomalý, se nic nemění.

I zde je samozřejmě možné uvolnit úlohu tak, aby byla konvexní tím, že uvolníme podmínku na $\lambda$ tak, že požadujeme $\lambda \in [0,1]$. Všechna omezení pak vytyčují konvexní množiny, jejichž průnik (množina přípustných řešení) je

tedy také konvexní. S konvexní účelovou funkcí pak máme běžnou konvexní nelineární deterministickou úlohu, která se dá opět řešit běžnými algoritmy pro nelineární programování (i tyto jsou implementovány v solverech z [5]). Podobně jako u předchozích úloh nám hrozí, že optimální řešení nebude splňovat podmínku $g(x) \geq \xi$ s pravděpodobností větší nebo rovnou $p$, ale s pravděpodobností menší. Skutečnou pravděpodobnost $p^*$ pro optimální řešení $x^*$ uvolněné úlohy spočteme ze vztahu $p^* = \mathrm{P}(g(x^*) \geq \xi) = \mathrm{F}(g(x^*))$.

Je zajímavé podívat se také na to, co toto uvolnění udělá s množinou přípustných řešení. Stejně jako v lineárním případě děláme konvexní kombinace $p$-level eficientních bodů a ze stejných důvodů jako v lineárním případě je množina přípustných řešení původní úlohy podmnožinou množiny přípustných řešení uvolněné úlohy. Protože lineární funkce jsou konkávní i konvexní, dá se použít příklad (3.6) stejně jako v kapitole o lineárním programování, který ukazuje, že uvolněná množina přípustných řešení není konvexním obalem množiny řešení původních.

V [2] jsou popsány dva numerické algoritmy na výpočet nelineárních úloh. Oba algoritmy sice nepředpokládají apriorní znalost $p$-level eficientních bodů a počítají při svém průběhu jen ty, které potřebují, ale oba zároveň pracují s množinou, která je jimi vymezená, tudíž jejich znalost je pro řešení úlohy ve skutečnosti velmi vhodná. Oba algoritmy využívají Lagrangeovy duality, proto se nejprve stručně zmíním o ní. Více o Lagrangeově dualitě se lze dočíst v [1].

Mějme primární úlohu

$$\begin{array}{c} \min f(x) \\ g(x) \leq 0 \\ h(x) = 0 \\ x \quad \geq X \end{array} \tag{5.4}$$

Jako Lagrangeovu duální úlohu k ní označíme úlohu

$$\begin{array}{c} \max \theta(u, v) \\ u \geq 0, \end{array} \tag{5.5}$$

kde $\theta(u, v) = \inf\{f(x) + u^T g(x) + v^T h(x) | x \in X\}$, $u, v$ jsou Lagrangeovy multiplikátory a tedy $\theta(u, v)$ je infimum přes $x \in X$ z Lagrangeovy funkce. Provázanost úlohy (5.5) s úlohou (5.4) ukazují následující dvě věty, které jsou převzaté z [1], kde jsou i dokázané.

**Věta 4 (Slabá dualita)** *Nechť $x$ je přípustné řešení úlohy (5.4) a $(u, v)$ přípustné řešení úlohy (5.5). Pak platí $f(x) \geq \theta(u, v)$.*

**Věta 5 (Silná dualita)** *Nechť $X$ je neprázdná konvexní množina v $E_n$ a $f : E_n \to E_1$, $g : E_n \to E_m$ nechť jsou konvexní. Nechť $h : E_n \to E_l$ je afinní, tedy je tvaru $h(x) = Ax - b$. Nechť existuje $\hat{x}$ takové, že platí $g(\hat{x}) \leq 0$, $h(\hat{x}) = 0$ a nechť $0 \in \text{int}(h(X))$, kde $h(X) = \{h(x)|x \in X\}$. Pak platí*

$$\inf\{f(x)|x \in X, g(x) \leq 0, h(x) = 0\} = \sup\{\theta(u, v)|v \geq 0\}.$$

*Navíc, je-li levá strana konečná a infima se nabývá v bodě $\hat{x}$, pak se suprema na pravé straně nabývá pro $\bar{u} \geq 0$ a platí $\bar{u}^T g(\hat{x}) = 0$.*

První věta tedy říká, že hodnoty účelové funkce Lagrangeovo duální úlohy jsou vždy menší než hodnoty účelové funkce primární úlohy. Tedy i optimální hodnota duální úlohy je menší než optimální hodnota primární úlohy, tedy duální úloha dává spodní mez pro úlohu primární.

Druhá věta pak dává postačující podmínky, za kterých dochází k rovnosti optimálních hodnot obou úloh.

Alternativní pohled na Lagrangeovu dualitu je v [1] nabízen přes sedlový bod:

**Definice 4** *Označme $\phi(x, u, v) = f(x)+u^T g(x)+v^T h(x)$. Pak řešení $(\bar{x}, \bar{u}, \bar{v})$ se nazývá* sedlový bod *Lagrangeovy funkce, když $\bar{x} \in X$, $\bar{u} \geq 0$ a pro všechna $x \in X$, $(u, v) : u \geq 0$ platí:*

$$\phi(\bar{x}, u, v) \leq \phi(\bar{x}, \bar{u}, \bar{v}) \leq \phi(x, \bar{u}, \bar{v})$$

Sedlový bod funkce $\phi(x, u, v)$ je tedy takový bod, který ji při pevném $x$ maximalizuje přes $(u, v)$ a při pevném $(u, v)$ minimalizuje přes $x$. Jak ukazuje následující věta, sedlový bod je také bodem, ve kterém mají primární a duální úloha stejná optima.

**Věta 6** *Řešení $(\bar{x}, \bar{u}, \bar{v})$, $\bar{x} \in X$, $\bar{u} \geq 0$ je sedlový bod Lagrangeovy funkce $\phi(x, u, v) = f(x) + u^T g(x) + v^T h(x)$ právě tehdy když:*

1. $\phi(\bar{x}, \bar{u}, \bar{v}) = \min\{\phi(x, \bar{u}, \bar{v})|x \in X\}$,
2. $g(\bar{x}) \leq 0$, $h(\bar{x}) = 0$ *a*
3. $\bar{u}^T g(\bar{x}) = 0$

*Navíc $(\bar{x}, \bar{u}, \bar{v})$ je sedlový bod právě tehdy, když $\bar{x}$ je optimální řešení primární úlohy (5.4) a $(\bar{u}, \bar{v})$ je optimální řešení duální úlohy (5.5) a pro tyto úlohy platí $f(\bar{x}) = \theta(\bar{u}, \bar{v})$.*

Důkaz věty je v [1].

Přeformulujme si nyní úlohu (5.3) do tvaru, se kterým dále budou pracovat konkrétní algoritmy.

$$\begin{aligned} &\min f(x) \\ g(x) &\geq z \\ x &\in X \\ z &\in \operatorname{cloconv}(C), \end{aligned} \tag{5.6}$$

kde $C = \bigcup_{s \in S} \{z | z \geq z_s\}$, $z_s$ jsou $p$-level eficientní stejně jako v předchozích kapitolách. Lagrangeova duální funkce má pak tvar

$$\phi(u) = \inf_{(x,z) \in X \times \operatorname{cloconv}(C)} (f(x) + u^T(z - g(x))). \tag{5.7}$$

Zaveďme následující značení:

$$h(u) := \inf\{f(x) - u^T g(x) | x \in X\} \tag{5.8}$$

$$d(u) := \inf\{u^T z | z \in \operatorname{cloconv}(C)\} \tag{5.9}$$

Rovnice (5.7) jde tedy zapsat jako

$$\phi(u) = h(u) + d(u) \tag{5.10}$$

Následující algoritmus na hledání optimálního řešení Lagrangeovy duální úlohy k úloze (5.6) je s drobnými úpravami převzat z [2]. Zastavovací podmínka je taková, že se optimální hodnota s novým krokem zvětší o méně než $\varepsilon$ a v [2] je dokázáno, že pro $\varepsilon = 0$ konverguje posloupnost optimálních hodnot $\phi_k(u^{k+1})$ v jednotlivých krocích $k$ k optimálnímu řešení duální úlohy. Podle věty 4 pak víme, že tento odhad omezuje optimální hodnotu primární úlohy zdola. Věta 5 a věta 6 pak dávají podmínky, za kterých jsou si optimální hodnoty primární a duální úlohy rovny, a tedy optimální hodnoty z jednotlivých kroků konvergují přímo k optimální hodnotě primární funkce. Povšimněme si toho, že v naší úloze je $h(x) \equiv 0$, tedy všechny podmínky na $h(x)$ z věty 5 o silné dualitě jsou splněny. Navíc pro $\varepsilon = 0$ platí, že hromadné body posloupnosti dílčích optimálních řešení $\{u^k\}$ jsou optimálními

řešeními Lagrangeovy duální úlohy, tedy i znalost dílčích optimálních řešení může být užitečná.

Z numerických důvodů zavádějí autoři omezení $b$ a množinu duálně přípustných řešení pak definují takto: $U := \{u \in \mathbb{R}^m,\ 0 \leq u_i \leq b\ i = 1, \ldots, m\}$. Na začátku je nutné zvolit hladinu $\varepsilon$, která určuje zastavovací podmínku algoritmu.

**Algoritmus (duální):**

1. krok: Vezměme vektor $u^1 \in U$. Definujme $\phi_0(u^1) = \infty$ a dosaďme $k = 1$.

2. krok: Spočťeme

$$h(u^k) = \min\{f(x) - (u^k)^T g(x) | x \in X\} \tag{5.11}$$

$$d(u^k) = \min\{(u^k)^T z | z \in \text{cloconv}(C)\} \tag{5.12}$$

   Označme $x^k$ řešení (5.11) a $v^k$ řešení (5.12).

3. krok: Spočťeme $\phi(u^k) = h(u^k) + d(u^k)$. Pokud $\phi(u^k) \geq \phi_{k-1}(u^k) - \varepsilon$ tak algoritmus končí, jinak pokračuje.

4. krok: Definujme

$$\phi_k(u) = \min_{1 \leq j \leq k} (\phi(u^j) + (v^j - g(x^j))^T (u - u^j))$$

   a spočtěme řešení $u^{k+1}$ úlohy

$$\max_{u \in U} (\phi_k(u))$$

5. krok: $k = k + 1$, přejděme na 2. krok

Několik poznámek k tomuto algoritmu. Povšimněme si, že $v^j$, tedy optimální řešení úlohy (5.12), jdou volit jako $p$-level eficientní body (vzhledem k tvaru množiny cloconv$(C)$, která je polyhedrická, její krajní body jsou právě PLEP a účelová funkce $(u^k)^T z$ je lineární). Tento algoritmus je možné implementovat pro strojový výpočet, ale je zapotřebí si uvědomit, že bez předem spočítaných PLEP je úloha (5.12) netriviální v tom smyslu, že je zapotřebí nějakým způsobem nejprve určit množinu cloconv$(C)$. Pokud však máme PLEP spočítané (například algoritmem z kapitoly 3.4), není v realizaci algoritmu žádný problém.

Druhý algoritmus uvedený v [2] vychází z postupného generování $p$-level eficientních bodů a jejich přidávání do omezení úlohy. Předpokládá znalost množiny $B$ takové, že obsahuje všechny $p$-level eficientní body $v$, pro které existuje $x \in X,\ g(x) \geq v$. Takovou množinu lze najít třeba tak, že vezmeme $\bigotimes_{j=1}^{m}[\hat{z}_j, g_j(\hat{x}^j)]$, kde $\hat{z}_j = \min_{s\in S}\{z_j^s\}\ \forall j,\ \hat{x}^j = \operatorname{argmax}_{x\in X}\{g_j(x)\}$.

Označme $S_k := \{\lambda \in \mathbb{R}^k | \lambda \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1\}$.

**Algoritmus (primárně-duální):**

1. krok: Vyberme PLEP $v^1 \in B$ takový, že existuje $\tilde{x} \in X$ pro které $g(\tilde{x}) \geq v^1$. $J_1 = \{1\}$, $k = 1$. Takové $v^1$ lze najít například jako $\operatorname{argmin}\{u^T z | z \in C \cap B\}$ pro nějaké $u \geq 0$.

2. krok: Vyřešme

$$\begin{array}{c} \min f(x) \\ g(x) \geq \sum_{j\in J_k} \lambda_j v^j \\ x \in X \\ \lambda \in S_k \end{array} \tag{5.13}$$

   Označme $u^k$ vektor Lagrangeovo multiplikátorů v této úloze.

3. krok: Spočtěme $d_k(u^k) = \min_{j\in J_k}((u^k)^T v^j)$

4. krok: Spočtěme řešení $v^{k+1}$ (bude $p$-level eficientní) úlohy

$$\min_{z\in C\cap B}((u^k)^T z) \tag{5.14}$$

   a spočtěme $d(u^k) = (v^{k+1})^T u^k$.

5. krok: Když $d(u^k) \geq d_k(u^k) - \varepsilon$ pak algoritmus končí, jinak $J_{k+1} = J_k \cup \{k+1\}$, $k = k+1$ a přejděme do 2. kroku.

Pro tento algoritmus je v [2] ukázáno, že pro $\varepsilon = 0$ konverguje posloupnost optimálních hodnot úlohy (5.13) k optimální hodnotě řešené úlohy (5.6). Také je tam dokázáno, že každý hromadný bod $\hat{x}$ posloupnosti dílčích optimálních řešení $\{x^k\}$ je optimálním řešení původní úlohy (5.6) s tím, že platí $g(\hat{x}) = z$. Tedy algoritmus je opět efektivní v tom smyslu, že jím spočítaná řešení a optimální hodnoty pro $\varepsilon > 0$ se blíží řešením a optimálním hodnotám původní úlohy. Stejně jako v případě předchozího algoritmu je i tento aplikovatelný, pokud se podaří popsat množinu $C$ (pro výpočet v podúloze

(5.14)), což je v případě konečného počtu $p$-level eficietních bodů možné jejich spočením před spuštěním tohoto algoritmu. Množina $B$ pak umožňuje omezit úlohu tak, aby se nepočítalo na zbytečně velké množině.

Za povšimnutí stojí princip, na kterém tento algoritmus stojí. V podstatě jde o řešení úlohy (5.3) s tím, že se začíná pouze s jedním $p$-level eficientním bodem a postupně se přidávají do omezení další relevantní PLEP tak, abychom se blížili původní úloze. Algoritmus se zastavuje, pokud se již nedosáhne dostatečně výrazného zlepšení (což je kontrolováno parametrem $\varepsilon$).

**Poznámka:** Oba algoritmy jsou použitelné nejen pro diskrétní rozdělení, ale i obecná, pokud se pro danou úlohu podaří popsat množinu $C$ respektive cloconv($C$). Definice $p$-level eficientních bodů zůstává v platnosti i mimo prostor diskrétních rozdělení, ale v případě, že jejich počet je nekonečný, tak nejde množina $C$ popsat pomocí jejich úplného vyjmenování a je zapotřebí hledat efektivnější popis.

# Kapitola 6

# Příklad

## 6.1 Popis úlohy

Pro numerickou ilustraci použití $p$-level eficientních bodů při výpočtu lineární úlohy pravděpodobnostního programování jsem se rozhodl vyjít z příkladu řešeného v [4], kde se předpokládá spojité rozdělení (normální). Já uvažuji diskrétní simulace z něj a své výsledky srovnávám s výsledky původními.

Úloha popisuje vodní systém na řece Bodrog na jihu východního Slovenska. Na řece jsou zbudovány rezervoáry a je zapotřebí zásobovat vodou v průběhu roku průmysl (především tepelnou elektrárnu), poskytovat vodu na zavlažování, chránit oblast před povodněmi a zachovávat životní prostředí v nádržích a jejich okolí.

Úloha se počítá pro hydrologický rok, který rozděluje do čtyř období. První je zima a jaro (listopad až duben), druhé je květen a červen, třetí je období letních prázdnin a dovolených (červenec, srpen) a čtvrté je září a říjen. Během prvního období se počítá s naplněním rezervoárů, které se pak budou po zbytek roku využívat a upouštět. Po zbylá tři období se předpokládá s výdaji na průmysl a zavlažování. Po celý rok je zapotřebí zachovávat dostatečnou hladinu vody pro zachování ekosystému a zároveň mít dostatečnou rezervu pro případ povodní.

Předmětem výpočtu pak je, jak velká nádrž bude zapotřebí, aby celý systém fungoval.

## 6.2 Matematická formulace úlohy

V úloze se snažíme minimalizovat náklady při splnění požadavků (průmysl, ochrana před povodněmi, zavlažování,...). Náklady jsou představované cenou za zbudování rezervoáru a rostou s jeho velikostí. Proto stačí minimalizovat velikost nádrže $x_0$.

Z hlediska této práce je nejzajímavějším omezením to, které požaduje dostatečné zásobení průmyslu a zavlažování. Požadavky průmyslu jsou pevné, zatímco požadavky na zásobení vodou zemědělců závisí na náhodě, přičemž je zapotřebí, aby byla spolehlivost zajištěna přes všechna období dohromady. Jde tedy o sdružená pravděpodobnostní omezení. Označme si množství vypuštěné vody v obdobích $1, 2, 3, 4$ proměnnými $x_1, x_2, x_3, x_4$, požadavky průmyslu a požadavky na zavlažování pak v obdobích $2, 3, 4$ po řadě $d_2, d_3, d_4$ a $\beta_2, \beta_3, \beta_4$. V prvním období je totiž zapotřebí splnit požadavky (na které v tom období má hlavní vliv tepelná elektrárna) s takovou spolehlivostí, že je použita pevná mez $x_1 \geq d_1$. V ostatních obdobích pak musí platit

$$\mathrm{P}\{x_i \geq d_i + \beta_i,\ i = 2, 3, 4\} \geq p, \tag{6.1}$$

kde $p$ je požadovaná hladina spolehlivosti.

Dále je zapotřebí zahrnout omezení na minimální hladinu vody v každém období ($m_i$) a minimální množství volného prostoru $v_i$ v nádrži v $i$-tém období, což jsou podmínky dané ochranou ekosystému a ochranou před povodněmi. Označme si $s_i$ množství vody v nádrži v $i$-tém období (období uvažujeme jako bodové stavy). Předpokládejme, že veškerou ztátu vody z rezervoáru popisují proměnné $x_i$ (tedy zanedbáváme například odpařování vody). Přítoky do rezervoáru v čase $i$ označme $r_i$ a celkové množství vody, které do nádrže přiteklo do času $i$ včetně, označme $\zeta_i$, tedy platí $\zeta_i = \sum_{j=1}^{i} r_j,\ i = 1, 2, 3, 4$.

Předpokládejme, že na začátku hydrologického roku je nádrž na minimální možné hladině, tedy $s_0 = m_4$, což je považováno za realistický předpoklad. Množství vody v $i$-tém obodobí pak můžeme vyjádřit následujícím vztahem:

$$s_i = m_4 + \zeta_i - \sum_{j=1}^{i} x_i,\ i = 1, 2, 3, 4$$

Vzhledem k tomu, že $\zeta_i$ jsou náhodné veličiny, je zapotřebí omezeními na minimální a maximální množství vody v nádrži pracovat jako s omezeními

s náhodnou pravou stranou. Nicméně jsou uvažována jako individuální omezení, tedy není zapotřebí na zvolené hladině udržet všechna najednou. Navíc je možné v každém období požadovat jinou spolehlivost. Získáme tak následující pravděpodobností omezení:

$$\begin{aligned} \mathrm{P}\{s_i \geq m_i\} &\geq \alpha_i,\ i = 1,2,3,4 \\ \mathrm{P}\{s_i + v_i \leq x_0\} &\geq \gamma_i,\ i = 1,2,3,4 \end{aligned}$$

Po dosazení za $s_i$ a nahrazení $\alpha_i$ a $\gamma_i$ příslušnými kvantily $z_i$ rozdělení veličin $\zeta_i$, dostáváme tento tvar:

$$\sum_{i=1}^{k} x_i \leq z_k(1-\alpha_k) + m_4 - m_k,\ k = 1,2,3,4 \tag{6.2}$$

$$\sum_{i=0}^{k} x_i \geq z_k(\gamma_k) + m_4 + v_k,\ k = 1,2,3,4 \tag{6.3}$$

Poslední omezení jsou dána převážně přirozenou hydrologickou a morfologickou situací (například že není možné vypustit příliš mnoho vody najednou, nebo vypouštět záporná množství vody):

$$l_0 \leq x_0 \leq u_0 \tag{6.4}$$

$$d_1 \leq x_1 \leq u_1 \tag{6.5}$$

$$0 \leq x_i \leq u_i,\ i = 2,3,4 \tag{6.6}$$

Po dosazení konkrétních čísel do modelu dostáváme následující úlohu pravěpodobnostního programování:

$$
\begin{array}{c}
\min x_0 \\
\mathrm{P}\{x_i \geq \beta_i + 12.7, i = 2,3,4\} \geq p \\
\begin{array}{rcrcrcrcl}
 & & x_1 & + & x_2 & & & & \leq 156.4 \\
 & & x_1 & + & x_2 & + & x_3 & & \leq 201.9 \\
 & & x_1 & + & x_2 & + & x_3 & + & x_4 \leq 225.3 \\
x_0 & + & x_1 & & & & & & \geq 512.9 \\
x_0 & + & x_1 & + & x_2 & & & & \geq 595.9 \\
x_0 & + & x_1 & + & x_2 & + & x_3 & & \geq 654.2 \\
x_0 & + & x_1 & + & x_2 & + & x_3 & + & x_4 \geq 720.2
\end{array} \\
100.0 \leq x_0 \leq 500.0 \\
38.1 \leq x_1 \leq 102.3 \\
0 \leq x_2 \leq 252.0 \\
0 \leq x_3 \leq 252.0 \\
0 \leq x_4 \leq 252.0
\end{array}
\tag{6.7}
$$

Vektor $\beta$ byl v původní úloze (viz [4]) považován za normálně rozdělený se střední hodnotou

$$\mathrm{E}(\beta) = (20.2, 27.37, 10.65), \tag{6.8}$$

směrodatnou odchylkou

$$\sigma(\beta) = (8.61, 10.65, 6.00) \tag{6.9}$$

a korelační maticí

$$R(\beta) = \begin{pmatrix} 1.0 & 0.360 & 0.125 \\ 0.360 & 1.0 & 0.571 \\ 0.125 & 0.571 & 1.0 \end{pmatrix}. \tag{6.10}$$

Protože jsem neměl naměřené historické hodnoty, které by byly pro použití ideální, přistoupil jsem ke generování dat, jakožto výběru z tohoto normálního rozdělení. Nagenerovaným scénářům (trojicím realizací $(\beta_1, \beta_2, \beta_3)$) jsem pak přiřadil stejné pravděpodobnosti a takto získané empirické rozdělení považoval za teoretické diskrétní rozdělení, kterým je popisován vektor $\beta$.

## 6.3 Výpočet

Při výpočtu jsem aplikoval algoritmus na výpočet PLEP popsaný v kapitole 3.4 a implementovaný v kapitole 4. Bohužel se ukázalo, že podprogram na

| $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $p$ |
|---|---|---|---|---|---|
| 494.89 | 51.22 | 63.07 | 77.35 | 33.69 | 0.973 |
| 494.89 | 50.00 | 63.07 | 77.40 | 34.92 | 0.983 |
| 494.89 | 49.91 | 63.07 | 77.35 | 35.01 | 0.984 |
| 494.89 | 46.17 | 63.07 | 77.40 | 38.75 | 0.997 |
| 494.89 | 43.41 | 63.04 | 77.35 | 41.50 | 0.999 |

Tabulka 6.1: Výsledky z [4]

výpočet distribuční funkce je pro data takového rozsahu (pro $n$ scénářů má distribuční funkce $n^3$ bodů) zcela nepoužitelný (byl schopen upočítat jen malý počet simulací), tak jsem na tuto část programu použil algoritmus naprogramovaný v Pythonu, který je sice méně obecný, ale psaný přímo na míru této úloze a tedy podstatně rychlejší.

Python, ve kterém se generují scénáře a počítá distribuční funkce, má výstup do binárních souborů, ze kterých si R načte data a spočte $p$-level eficientní body tak, jak je popsáno v kapitole 4. R pak pomocí textových souborů předá programu GAMS $p$-level eficientní body a GAMS pak použije model z úlohy (3.5). Úlohu řeším solverem Cplex, který používá duální simplexovou metodu.

Použité zdrojové kódy pro R a GAMS jsou přiloženy v CD příloze této práce, stejně jako kód v programu Python, jehož autorem je Jan Dvořák.

## 6.4 Výsledky

Při srovnávání s výsledky z [4] je klíčové především srovnání závislosti $x_0$ (rozhodnutí, jak velká nádrž je zapotřebí, které se provede na základě úlohy) na $p$ (spolehlivosti, s jakou chceme uspokojit poptávku po vodě na zavlažování a průmysl). Pro srovnání přikládám také hodnoty $x_1, x_2, x_3$ a $x_4$, které jsou však silněji ovlivněné konkrétní realizací vektoru $\beta$. Srovnávat čas výpočtu je irelevantní vzhledem k téměř 20 letům rozdílu ve výpočetní technice a jejím výkonu, respektive nemožnosti spustit výpočet na stejném stroji. Stejně tak se nedá očekávat, že počet iterací v nelineárním solveru MINOS, který je použit v [4], bude korespondovat s iteracemi v lineárním Cplexu, který používám já.

V talbulkách 6.1 a 6.2 srovnávám výsledky z [4] s výsledky, které jsem získal já. Tabulka 6.1 počítá se spojitým normálním rozdělením, zatímco

| $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $p$ | PLEP |
|---|---|---|---|---|---|---|
| 500.00 | 38.10 | 57.80 | 98.13 | 31.27 | 0.973 | 91 |
| 500.00 | 38.10 | 57.80 | 86.90 | 42.50 | 0.983 | 37 |
| 500.00 | 38.10 | 57.80 | 86.90 | 42.50 | 0.984 | 37 |
| 554.00 | -16.40 | 58.30 | 120.40 | 63.00 | 0.997 | 2 |
| 558.70 | -21.10 | 58.30 | 124.20 | 63.90 | 0.999 | 1 |

Tabulka 6.2: Moje výsledky

tabulka 6.2 je postavena na 500 výběrech z tohoto rozdělení, které jsou uvažované jako teoretické diskrétní rozdělení. Vzhledem k testované přesnosti by bylo vhodnější mít simulací ještě více, ale protože množství bodů v simulovaném rozdělení roste s třetí mocninou počtu simulací, byl ideální počet několika tisíc simulací mimo můj dosah.

Na toto omezení, vyplývající z dat, je třeba myslet při analýze výsledků. Například při hladině spolehlivosti 0.999 ve skutečnosti požadujeme ochranu před všemi možnostmi, protože takto jemné síto data neumožňují. Není tedy divu, že výsledná hodnota účelové funkce je o tolik zvýšená oproti původním datům. Ze stejného důvodu příliš hrubých dat není žádný rozdíl mezi spolehlivostí 0.983 a 0.984.

Pokud vynecháme nejhorší případy, které jsou brány v úvahu při nejvyšších spolehlivostech a které jsou dost vzdálené od běžných hodnot, pak dostáváme rozumné výsledky. Hodnota účelové funkce je oproti původním datům zvýšená jen trochu, a tedy pro spolehlivosti 0.973 a 0.983, kde máme dostatek dat, jsou výsledky použitelné. Dá se tedy očekávat, že pokud by se podařilo zajistit dost velký rozsah, bude metoda účinná i pro vyšší spolehlivosti.

Sloupec nadepsaný "PLEP" značí, kolik $p$-level eficientních bodů bylo pro danou hladinu spolehlivosti získáno. Vzhledem k rovnoměrnosti teoretického rozdělení, se kterým jsem pracoval, není překvapující, že jejich počet klesá s rostoucím $p$.

Za povšimnutí také stojí, že i při několika stech generovaných scénářích byla výsledná optimalizační úloha malá - $p$-level eficientních bodů bylo relativně málo (díky malému počtu dimenzí náhodného vektoru pravých stran) a tedy bylo málo omezení i rozhodujících proměnných $\lambda_i$. Díky tomu probíhal výpočet rychle a dal by se aplikovat i přístup, kdy se úloha vyřeší pro každý z PLEP a jako celkové řešení se použije nejlepší řešení z těchto podúloh. To by odstranilo riziko nedodržení spolehlivosti, nicméně pro názornost

výsledků obecné metody jsem se rozhodl k tomuto vylepšení nepřistoupit a úlohu řešit jako uvolněnou úlohu.

Během výpočtu se také ukázalo, že výpočet PLEP je rychlý i pro velká vstupní data a hlavním problémem tak zůstává výpočet sdružené distribuční funkce, který je hlavním limitujícím faktorem pro počet uvažovaných scénářů. I samotné načítání už spočítané distribuční funkce programem R je nepoměrně delší než samotné výpočty $p$-level eficientních bodů i řešení úlohy v GAMSu.

Hlavní obtíže při používání této metody v prezentovaném příkladu souvisí se simulacemi, kdy vzniká rozdělení na mnoha bodech, přičemž drtivá většina z nich se nabývá s nulovou pravděpodobností, tedy je informace rozprostřena přes zbytečně velký prostor. Pokud by se podařilo tuto nevýhodu odstaranit, umožnili bychom tak řešit úlohu pro řádově větší počty simulací.

# Kapitola 7

# Závěr

Po formulaci základní úlohy lineárního pravděpodobnostního programování s diskrétním rozdělením pravé strany jsem ukázal strukturu množiny přípustných řešení a popsal ji pomocí $p$-level eficientních bodů. Prezentoval jsem uvolnění množiny přípustných řešení, které využívá konvexního obalu PLEP. Tato metoda v lineárním programování, kde nejsou další deterministická omezení, dosahuje dobrých výsledků, ale ukazuje se, že nemusí vždy dodržet požadovanou hladinu spolehlivosti.

Dále jsem se zabýval zobecněním této úlohy, a to v lineárním případě, kdy na množinu přípustných řešení máme další lineární omezení, a především pak v případě, kdy máme celočíselná omezení na rozhodující proměnné. Ukázal jsem, k jakým komplikacím může dojít, a nabídl apriorní odhad nejhorší možné dosažitelné spolehlivosti při aplikaci popisovaného uvolnění.

V další části jsem popsal rozšíření teorie $p$-level eficientních bodů do oblasti nelineárního programování. Ukázal jsem, že za určitých podmínek jdou použít stejné postupy jako v lineárním programování, ale opět není zajištěna požadovaná hladina spolehlivosti. Ukázal jsem, jak se projeví uvolnění na množinu přípustných řešení a prezentoval poznatky o Lagrangeově dualitě a její aplikaci na praktické řešení nelineární úlohy pravděpodobnostního programování.

Představil jsem algoritmus na výpočet $p$-level eficientních bodů a dokázal jeho totální korektnost. Dále jsem prezentoval a komentoval dva algoritmy na řešení nelineárních úloh. Algoritmus pro výpočet PLEP jsem naprogramoval v prostředí R a jeho výstupy přizpůsobil aplikaci pro výpočet v programu GAMS. Představil jsem zdrojový kód programu a popsal jeho běh, aby byl snadno použitelný.

Na příkladu jsem ukázal, že při dostatečně velkém počtu simulací dávají PLEP výsledky srovnatelné se spojitým rozdělením. Ale ukázalo se, že simulace jsou výpočetně velmi náročné, obzvláště pro vyšší počet dimenzí náhodného vektoru. Metody popsané v této práci je tedy vhodné aplikovat spíše na jiný typ úloh nebo najít lepší způsob reprezentace a výpočtu distribuční funkce.

Pro další výzkum bych doporučil vývoj algoritmů, které nepotřebují znát všechny $p$-level eficientní body předem, či je dokáží počítat úsporněji než přes výpočet sdružené distribuční funkce, protože tento výpočet se v určitých podmínkách ukázal jako značně limitující. Také by bylo zajímavé vyvinout metody, které přesněji určí, jakou (nejhorší) spolehlivost bude mít výsledek a to ještě předtím, než je spočítán, aby se daly včas aplikovat spolehlivější, byť pomalejší metody, než je zde používané uvolnění množiny přípustných řešení, neboť jak jsem ukázal v této práci, tak splnění předepsané spolehlivosti lze požadovat přesně, pokud je k dispozici dost času nebo výpočetního výkonu na počítání většího množství úloh. Za pozornost také stojí vývoj metod, které dokáží uvolnit množinu přípustných řešení pouze na konvexní obal, aby se zvýšila spolehlivost řešení.

# Literatura


[1] Mokhtar S. Bazaraa, Hanif D. Sherali, C.M. Shetty: *Lagrangian Duality and Saddle Point Optimality Conditions*, kapitola 6 z *Nonlinear Programming: Theory and Algorithms* (2006), str. 199-242

[2] Darinka Dentcheva, Bogumila Lai, Andrzej Ruszczyński: *Dual methods for probabilistic optimization problems*, Mathematical Methods of Operations Research (2004), str. 1-16

[3] Darinka Dentcheva, András Prékopa, Andrzej Ruszczyński: *Concavity and efficient points of descrete distributions in probabilistic programming*, Mathematical Programming **Ser. A 89** (2000), str. 55-77

[4] Jitka Dupačová, Alexei Gaivoronski, Zdeněk Kos, Tamás Szántai: *Stochastic programming in water management: A case study and a comparison of solution techniques*, European Journal of Operational Research **52** (1991). str. 28-44

[5] GAMS Development Corporation: *GAMS - The Solver Manuals*, 2008

[6] GAMS Development Corporation: *GAMS - A User's Guide*, 2008

[7] Petr Lachout: *Matematické programování*, pracovní text k přednášce *EKN011 Optimalizace I*
(http://www.karlin.mff.cuni.cz/ lachout/Vyuka/Optima1/081026-Opt-text.pdf)

[8] András Prékopa: *Probabilistic Programming*, kapitola 5 z Andrzej Ruszczynski, Alexander Shapiro, (eds.) *Handbook on Stochastic Programming*, Handbooks in Operations Research and Management Science **10** (2002), str. 267-352

[9] R Development Core Team: *The R Manuals* (2010), www.r-project.org

[10] Suvrajeet Sen: *Relaxations for probabilistically constrained programs with discrete variables*, Operations Research Letters **11** (1992), str. 81-86

# Dodatek A

# Zdrojový kód výpočtu PLEP v programu R

## A.1 Soubor Program.R

```
##################
## Priprava dat ##
##################

## Nacteni funkci:
source("PLEPS.R");

## Hodnoty, kterych mohou nabyvat jednotlivé slozky vektoru (vyplnte):
slozky <- list(c(...),c(...),...);

## Spocte pocet moznych hodnot pro kazdou slozku vektoru (nemente)
max <- dimenze(slozky);

## "Matice" pravdepodobnosti dosazeni bodu nahodným vektorem (nemente)
psti <- array(0,max);

## Zde vyplnte pravdepodobnosti, ze nahodny vektor nabude danych hodnot
## Souradnice jsou dany poradim hodnoty ve slozce, tj. psti[1,3] znamena
## pravdepodobnost ze vektor nabude v prvni slozce prvni mozne hodnoty a
## ve druhe slozce treti mozne hodnoty
## (defaultní hodnota ve vsech bodech je 0)
## Pokud znate distribucni funkci, vyplnte misto pravdepodobnosti jeji
## hodnoty
psti[c(...)] <- ...;
```

```
sum(psti); ## kontrolni soucet, ma byt = 1, pokud
        ## pokud jste vyplnovali pravdepodobnosti

## Vypocet sdruzene distribucni funkce
F <- distribuuj(psti,max); ## spocte distribucni funkci

## Nebo pokud jste do pole "psti" vyplnovali hodnoty distribucni funkce:
## F <- as.array(psti);

## array(F,max); ## zobrazi F jako "vicerozmernou matici"

########################################
## Vypocet p-level eficientnich bodu: ##
########################################

## Hladina p z (0,1) na které chceme hledat p-eficientní body (zvolte):
p <- ...;

## Spusteni vypoctu:

PLEPs <- plepsuj(F,max,p,slozky); ## spocte p-level eficientni body

## Pro spolupraci s GAMSem (vyplnte cestu k adresari, kam chcete ukladat
## misto "ADRESA")

write(c("s PLEPsy /1*",length(PLEPs[1,]),"/"), file = "ADRESA/mnozina.inc",
ncolumns = 3, sep = "");
write("z(k,s) plepsy /",file = "ADRESA/PLEPsy.inc", ncolumns = 1);
for (j in 1:length(PLEPs[,1])){
 for (i in 1:length(PLEPs[1,])){
  write(c(j+1,".",i," ",PLEPs[j,i]), file= "ADRESA/PLEPsy.inc",
  ncolumns = 5, append = TRUE, sep = "");
 };
};
write("/", file = "ADRESA/PLEPsy.inc", append = TRUE);

PLEPs; ## vypise hodnoty PLEPsu na obrazovku
```

## A.2 Soubor PLEPS.R

```
distribuuj <- function(psti,max){ ## spocte distribucní funkci

## nejdriv si definuje funkce potrebne
## pro svuj chod
```

```
 souradnice <- function(i){ ## spocte souradnice bodu na pozici "i"
## souradnice si preindexuji, aby zacínaly
## od nuly a na konci je opet posunu
## na jednicku (lépe se s tím pocítá)
  i <- i - 1;
  s <- rep(NA,r); ## inicializace
  for (j in r:1){
   soucin <- prod(max[1:j-1]); ## k_1*...*k_{j-1}
   s[j] <- i%/%soucin; ## spocte j-tou souradnici; %/% je
## celocislene deleni
   i <- i - s[j]*soucin; ## upraví císlo pro další výpocet
  };
  s <- s + rep(1,r);
  return(s);
 }; ## konec souradnice()

 spocti_souradnice <- function(){ ## spoce souradnice pro pole "ps"
## aby dimenze byly o rozmerech
## "max"
  s <- c();
  for (i in 1:delka){s <- cbind(s,souradnice(i))};
  return(s);
 }; ## konec spocti_souradnice()

 secti_ps <- function(i){ ## secte hodnoty na souradnicích menších
## než souradnice "i"
  soucet <- 0;
  tr <- rep(TRUE,r) ## vektor samých TRUE
  for (j in 1:i){ ## projde všechny prvky, které jsou
## v matici dríve
   vztah <- s[,i] >= s[,j]; ## jsou-li menší souradnice
   ifelse(identical(vztah,tr), soucet <- soucet + ps[j],
    soucet <- soucet);
  };
  return(soucet);
 }; ## konec secti_ps()

## vlastni kod funkce distribuuj

 r <- length(max);
 ps <- as.vector(psti); ## uloží matici jako vektor
 delka <- length(ps);
 s <- spocti_souradnice(); ## uloží souradnice bodu v poli
## pro prvky vektoru
 F <- rep(NA,delka); ## inicializace promenných
 for (i in 1:delka){
```

```
  F[i] <- secti_ps(i); ## spocte hodnotu na pozici "i"
 };
 return(F);
}; ## konec distribuuj()



plepsuj <- function(F,max,p,slozky){## nastartuje pocítání PLEPs

## nejprve definuje funkce potrebne
## pro svuj chod

 souradnicim_cislo <- function(souradnice){ ## priradí index v matici
## zapsané jako seznam
## poli o souradnicích
## "souradnice"
  souradnice <- souradnice - 1; ## pro snažší výpocet
  i <- 0;
  for (j in 1:r){
   i <- i + souradnice[j]*prod(max[1:j-1]);
  };
  return(i+1); ## protože se indexuje od jedné
 }; ## konec souradnicim_cislo()

 minimalizuj <- function(vek,i){ ## minimalizuje distr. fci
                                 ## v "i"-té složce
## a ostatní jsou fixovány jako "vek"
  while(F[souradnicim_cislo(vek)] < p){
   vek[i] <- vek[i] + 1;
  }; ## konec while cyklu
  return(vek[i]); ## prvni hodnota pro kterou prekroci "p"
 }; ## konec minimalizuj()

 najdi_plep <- function(zacatek){ ## najde p-eficientní bod,
## pokud tam je; pocátecní body
## už jsou zadány v "zacatek"
  pocet <- length(zacatek) + 1;
  z <- zacatek; ## z je budoucí PLEP
  while (pocet < r){ ## cyklus pres pocet rozmeru
   konec <- max[(pocet+1):r] ## konec vektoru pro optimalizaci
   z <- c(z,minimalizuj(c(z,1,konec),pocet));
## spocte další složku PLEP
   pocet <- pocet + 1;
  }; ## konec cyklu pres rozmery
  z <- c(z,minimalizuj(c(z,1),pocet)); ## pro poslední složku
  z1 <- z[length(zacatek)+1]; ## promenná pro konec hledání PLEPs
```

```
  return(c(z,z1));
 }; ## konec najdi_plep()

 pridej <- function(E,z){ ## pridá do množiny "E" bod "bod", pokud
## ten nedominuje žádný bod z "E"
  OK <- TRUE; ## prvek je v porádku
  tr <- rep(TRUE,r);
  for (i in 1:length(E[1,])){ ## cyklus pres prvky "E"
   if (identical((E[,i] <= z),tr)){ ## bod dominuje
    OK <- FALSE; ## prvek nevyhovuje
    break; ## není treba dále overovat
   }; ## konec podmínky
  }; ## konec cyklu
  ifelse (OK,return(cbind(E,z)),return(E));
 }; ## konec pridej()

 sluc <- function(E,H){ ## sloucí 2 množiny, aby nic z "H"
## nedominovalo nic v "E"
  for (i in 1:length(H[1,])){ ## cyklus pres prvky "H"
   E <- pridej(E,H[,i]); ## pridá "i"-tý prvek do "E"
  };
  return(E);
 }; ## konec sluc()

 spocti_pleps <- function(zacatek){ ## funkce na spocteni p-level
## eficientních bodu
  k <- 1; ## na cyklus (1. krok)
  plep <- najdi_plep(zacatek); ## spocte PLEP (2. krok)
  z <- plep[1:r]; ## souradnice PLEP
  j <- plep[r+1]; ## index pro zastavení pocítání PLEPs
  E <- cbind(z); ## pridá PLEP do množiny (3. krok)
  kk <- max[length(zacatek)+1]; ## puvodní "k1"
  do <- ((j + k) <= kk); ## zda zacne cyklus (4. krok)
  H <- c(); ## inicializace
  while(do){ ## dokud platí podmínka
   zacatek2 <- c(zacatek,j+k);
   ifelse(length(zacatek2) < r,{ ## zastavovací podmínka rekurze nesplena
    H <- spocti_pleps(zacatek2); ## pocte PLPEs pro o 1 více složek
                                 ## pevných
    if (!identical(H,NULL)){E <- sluc(E,H)};## sloucí "E" a "H" tak, aby
                                 ## žádný prvek z "H" nedominoval
                                 ## žádný v E
   },{ ## zastavovací podmínka splnena
    break; ## vyskocí z while cyklu
   }); ## konec podmínky
   k <- k + 1;
```

```
   do <- (j + k) <= kk; ## podmínka pro pokracování ve výpoctu
  }; ## konec cyklu pro výpocet PLEPs
  return(E); ## vrací množinu PLEPs
 }; ## konec spocti_pleps()

 prepocitej <- function(PLEPs){ ## prepocte poradi zpet na hodnoty
  for (j in 1:length(PLEPs[1,])){ ## cyklus pres pocet PLEPs
   for (i in 1:r){ ## cyklus pres slozky PLEPs
    PLEPs[i,j] <- slozky[[i]][PLEPs[i,j]];## priradi spravnou hodnotu
   };
  };
  return(PLEPs); ## vraci nove hodnoty
 }; ## konec prepocitej

## vlastni kod funkce plepsuj

 r <- length(max); ## pocet dimenzí
 PLEPs <- spocti_pleps(c()); ## spustí s prázdným "zacátkem"
 PLEPs <- prepocitej(PLEPs);
 return(PLEPs);
}; ## konec plepsuj()

dimenze <- function(slozky){ ## spocte delky vektoru v listu
 r <- length(slozky);
 max <- rep(NA,r); ## inicializace
 for (i in 1:r){ ## cyklus pres pocet vektoru
  max[i] <- length(slozky[[i]]); ## ulozeni poctu
 };
 return(max);
};
```