VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

FAKULTA INFORMATIKY A STATISTIKY

DIPLOMOVÁ PRÁCE

2010 Karel Ouzký

VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

FAKULTA INFORMATIKY A STATISTIKY

Název diplomové práce:

Zlomkový simplexový algoritmus ve VBA

Autor:	**Karel Ouzký**
Katedra:	**Katedra ekonometrie**
Obor:	**Matematické metody v ekonomii**
Vedoucí práce:	**prof. RNDr. Jan Pelikán, CSc.**

Prohlášení:

Prohlašuji, že jsem diplomovou práci na téma „Zlomkový simplexový algoritmus ve VBA“ zpracoval samostatně. Veškerou použitou literaturu a další podkladové materiály uvádím v seznamu použité literatury.

V Praze dne 5. května 2010 .....

Karel Ouzký

Poděkování:
Rád bych poděkoval za vedení a odborné rady panu prof. RNDr. Janu Pelikánovi, CSc.
Za připomínky k textu děkuji svému bratrovi Ing. Lubomíru Ouzkému ml. Dále děkuji za podporu a zázemí mým rodičům Ing. Lubomíru Ouzkému st. a Marii Ouzké. Veliký dík patří mému nebeskému Otci, za jeho stále pokračující milosrdenství a milost.

Abstrakt

Název práce: Zlomkový simplexový algoritmus ve VBA
Autor: Karel Ouzký
Katedra: Katedra ekonometrie
Vedoucí práce: prof. RNDr. Jan Pelikán, CSc.

Základní myšlenka zlomkového simplexového algoritmu spočívá ve spojení teorie maticového počtu a znalosti maticového vyjádření simplexové tabulky z revidované simplexové metody. Mou snahou je vysvětlit teoretické základy, na kterých algoritmus staví, a nabídnout zpracování problému v jazyku Visual Basic for Applications v prostředí MS Excel 2007. Hlavní přínos spatřuji v tom, že algoritmus je schopen vyřešit předem specifikovanou skupinu úloh zcela exaktně bez nutnosti počítání v desetinných číslech.

Klíčová slova: simplex metoda, zlomky, maticový počet

Abstract

Title: Fractal simplex algorithm in VBA
Author: Karel Ouzký
Department: Department of Econometrics
Supervisor: prof. RNDr. Jan Pelikán, CSc.

Basic idea of fractal simplex algorithm is based in the theory of matrix counting and knowledge of matrix representation of simplex tabuleao from revised simplex method. My desire is to explain theoretical basics on which this algorithm works and provide solution in language Visual Basic for Applications in application MS Excel 2007. Main benefit I see in the fact, that algorithm can solved specific class of mathematical problems in a way of exactness counting whithout necessity of using decimal numbers.

Keywords: simplex method, fractals, matrix algebra

Obsah

Úvod

Matematická přesnost výpočtu je stále aktuální téma. Při výpočtu různého typu optimalizačních úloh se setkáváme s možností vzniku zaokrouhlovacích chyb. Základní snahou celé této práce je poskytnout zcela exaktní výpočetní nástroj, který povede k řešení předem specifikovaných úloh.

Snahou této práce bude odvodit a algoritmizovat algoritmus simplexové metody, který po celou dobu výpočtu pracuje ve zlomkové aritmetice. Navržený postup bude implementován do aplikace v prostředí Microsoft Excel 2007, přičemž se pokusím poskytnout uživatelsky přívětivé prostředí. Excel podporuje programovací jazyk Visual Basic for Applications, který pro psaní využiji. Správná funkcionalita bude ověřena na náhodně vygenerovaných úlohách, které budou vykazovat smysluplnou řešitelnost. Aplikace bude navržena pro učební účely katedry ekonometrie Vysoké školy ekonomické v Praze a pokusím se zodpovědět i na možnosti jejího praktického využití.

Vzhledem k tomu, že nabízený postup do jisté míry staví na poznatcích z oblasti celočíselného programování, chtěl bych prozkoumat, možnost využití nástrojů lineárního programování v této oblasti, a proto se v jedné kapitole zaměřím na problém totální unimodularity matic, která úzce souvisí s možností řešitelnosti těchto úloh jako úloh lineárního programování.

1 Teoretické základy a odvození zlomkového simplexového algoritmu

1.1 Původ lineárního programování a simplexového algoritmu

Disciplína, kterou nazýváme lineární programování, má přes své široké uplatnění v reálném životě poměrně mladou historii. Pomáhá s řešením různorodých problémů a poskytuje nástroj vhodný pro matematické modelování možných situací, aby bylo dosaženo optimální alokace zdrojů, dostatečně uváženy různorodé informace pro kompromisní rozhodnutí a aplikace tohoto postupu se využívá v mnoha oblastech lidské činnosti.

Řadu podobných situací přinesla také 2. světová válka, ve které nezávisle na sobě bylo třeba vyřešit řadu optimalizačních problémů. Za počátek lineárního programování je možné označit však teprve rok 1947. Zde byly pomocí matematického rámce sjednoceny zdánlivě různorodé problémy a byla poskytnuta efektivní výpočetní metoda. Dantzigova simplexová metoda posloužila k možnosti explicitní formulace a řešení celé škály úloh lineárního programování. Další rozvoj souvisí s příchodem počítačů, které se brzy staly nezbytnou součástí pro aplikaci lineárního programování do oblastí, ve kterých nestačí ruční výpočet (viz Dantzig, 1998, s. 1-28).

Lineární programování spadá do oblasti deterministické optimalizace. Přestože bylo ukázáno, že simplexová metoda je v případě analýzy nejhorších případů exponenciální povahy (Klee-Minty [1972]), její praktická použitelnost je vysvětlována analýzou „průměrné situace“, kterou je poměrně složité specifikovat. Problém exponenciální složitosti simplexového algoritmu vedl k formulaci polynomiálních algoritmů sloužících k výpočtu úloh lineárního programování. Navzdory úspěchu a nalezení algoritmů, které i v nejhorších případech vykazují polynomiální řešitelnost, se ukázalo, že implementace těchto, metod ve srovnání s efektivitou simplexového algoritmu, více či méně selhává (Zimmermann, 1988, s. 1-2). Simplexový algoritmus se při řešení praktických úloh chová jako polynomiální algoritmus (Fiala, 2003, s.43).

V současnosti rozlišujeme řadu různých postupů, které na simplexové metodě staví. Podle typu úloh byly navrženy postupy, které jsou založeny na řádkových či sloupcových úpravách výchozí simplexové tabulky reprezentující úlohu lineárního programování. V následujícím odstavci zmíníme obecné vyjádření simplexové tabulky a postup

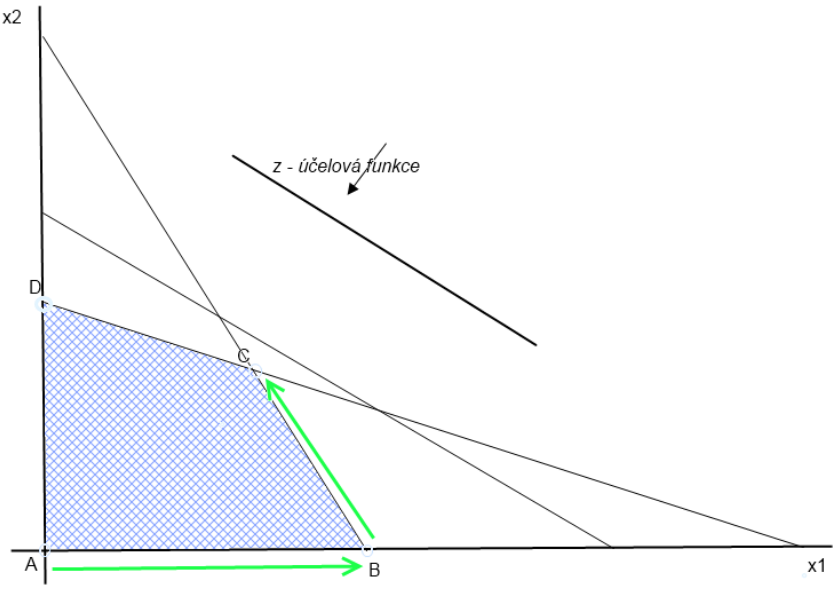
revidovaného simplexového algoritmu, které jsou součástí teoretického základu, na němž bude založen algoritmus simplexové metody, který pracuje se zlomky dále zlomkový simplexový algoritmus.

Maximalizační úlohu lineárního programování, která má všechna omezení typu $\leq$ můžeme obecně vyjádřit takto:

$$\begin{aligned} &\max c^T x, \\ &Ax \leq b, x \geq 0 \end{aligned} \qquad (1.1)$$

Kde
- A je matice strukturních koeficientů typu m x n
- b je vektor pravých stran omezení typu 1 x n
- c^T je transponovaný vektor účelové funkce typu 1x m
- x je vektor proměnných typu 1 x n

Simplexová metoda je metoda iterační, která na množině přípustných řešení postupně hledá optimální řešení podle základní věty lineárního programování. Výpočet je možné začít, jakmile máme alespoň jedno základní přípustné řešení (viz Lagová, Jablonský, 2009, s. 82). Zjednodušený postup metody je ilustrován na následujícím obrázku.



Obrázek 1.1 *Ilustrace simplexové metody*

Na obr. 1.1 vidíme jak simplexová metoda postupuje. Postupně prohledává množinu přípustných řešení (vyšrafovaná oblast) za tím účelem, aby v našem případě (1.1) vzrostla hodnota účelové funkce. V našem obrázku je výchozím základním přípustným řešením bod

bod A. V první iteraci se dostaneme do bodu B, kde je lepší hodnota účelové funkce a opět se jedná o základní přípustné řešení. Maximum účelové funkce je v bodě C, kde se výpočet zastaví, protože jsme dosáhli optima. Pokud bychom postupovali až do bodu D, snížili bychom hodnotu účelové funkce.

1.2 Obecné vyjádření simplexové tabulky

Výpočet simplexovou metodou je vhodné uspořádat do simplexové tabulky, která se dá rozdělit do šesti částí. Výchozí simplexová tabulka vypadá následovně (Jablonský, 2007, s. 35):

A	I	b
$-c^T$	0^T	0

Tabulka 1.1: *Maticové vyjádření výchozí simplexové tabulky*

kde:

A je matice strukturních koeficientů rozměru m x n

I je jednotková matice vyjadřující přídatné proměnné, které vznikly doplněním modelu

b je vektor pravých stran rozměru 1 x n

$-c^T$ vyjadřuje anulované koeficienty řádku účelové funkce rozměru 1 x m

0^T nulový vektor, který je pod přídatnými proměnnými rozměru 1 x n

0 výchozí hodnota účelové funkce rozměru 1

Když vyjádříme s-tou iteraci výpočtu, dostaneme následující tabulku (Jablonský, 2007, s. 36):

$B_S^{-1}A$	B_S^{-1}	$B_S^{-1}b$
$c_B{}^T B_S^{-1}A - c^T$	$u^T = B_S^{-1}c_B{}^T$	$c_B{}^T B_S^{-1}b$

Tabulka 1.2: *Maticové vyjádření simplexové tabulky v s-tém kroku výpočtu*

1.3 Revidovaná simplexová metoda

Revidovaná simplexová metoda vychází ze znalosti obecného maticového vyjádření simplexové tabulky a skutečnosti, že pro výpočet s-té iterace je stěžejní znalost inverzní matice báze B_s^{-1}, a proto se na výpočet inverzní matice redukuje. Při známosti výchozí tabulky a inverzní matice báze je pak možné dopočítat zbytek. Oproti běžné simplexové metodě vypadá obecné schéma výchozí tabulky následovně (Jablonský, 2007, s. 51):

I	b
0^T	0

B_S^{-1}	$B_S^{-1}b$
$u^T = B_S^{-1}c_B{}^T$	$c_B{}^T B_S^{-1}b$

Tabulka 1.3: *Tabulka revidované metody - výchozí a s-tý krok*

Matice B^{-1} vyjadřuje inverzní matici báze, která je inverzní k matici B vytvořené z těch sloupců matice A (výchozí tabulky), které odpovídají základním proměnným simplexové tabulky (Pelikán, 2001, s. 76).

1.4 Maticový počet

Pro výpočet využijeme větu o inverzní matici.

Věta *o inverzní matici*[1] (Coufal, Klůfa, 2000, s. 312)
Je-li $A = [a_{ij}]$ regulární matice řádu n, potom inverzní matice k matici A se dá zapsat ve tvaru

$$A^{-1} = \frac{1}{\det A} \times \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix}^T \qquad (1.2)$$

[1] Někdy je v literatuře je odkaz na úzce spojené Cramerovo pravidlo např. (Salkin, Mathur, 1989, s. 74).

Kde A_{ij} je algebraický doplněk prvku a_{ij}.

Algebraický doplněk je tvořen subdeterminanty řádu n-1 a platí pro něj:

$$A_{ij} = \begin{bmatrix} (-1)^{1+1} M_{11} & (-1)^{1+2} M_{11} & \dots & (-1)^{1+n} M_{1n} \\ (-1)^{2+1} M_{11} & (-1)^{2+2} M_{11} & \dots & (-1)^{2+n} M_{2n} \\ \dots & \dots & \dots & \dots \\ (-1)^{1+1} M_{n1} & (-1)^{1+1} M_{n2} & \dots & (-1)^{n+n} M_{nn} \end{bmatrix} \tag{1.3}$$

kde M_{ij} je subdeterminant, který vznikne z det$A$ po vynechání i-tého řádku a j-tého sloupce.

$$M_{11} = \begin{vmatrix} A_{12} & \dots & A_{1n} \\ \dots & \dots & \dots \\ A_{n2} & \dots & A_{nn} \end{vmatrix} \tag{1.4}$$

Někdy se můžeme také setkat s tímto zápisem: (1.5)
(Salkin, Mathur, 1989, s. 74)

$$A^{-1} = \frac{A^{adj}}{\det A}$$

Výpočet A^{adj} není z matematického hlediska příliš výhodný, protože např. výpočet inverzní matice k regulární matici pátého řádu vyžaduje určit 1 determinant pátého řádu a 25 subdeterminantů čtvrtého řádu (Coufal, Klůfa, 2000, s. 313). Z tohoto důvodu při výpočtu inverzní matice raději zvolíme jiný způsob než pomocí adjungované matice. Znalost věty o inverzní matici báze je však klíčová pro pochopení možnosti vyjádřit celou simplexovou tabulku ve zlomcích.

1.5 Užití zlomkového simplexu

1.5.1 Vyjádření vektoru základních proměnných

Z obecného maticového zápisu simplexové tabulky víme, že hodnota vektoru pravých stran je rovna B_s^{-1} b. Pro bazické proměnné můžeme tedy psát:

$$x_B = B^{-1}b \tag{1.6}$$

Pokud inverzní matici báze vyjádříme dle věty o inverzní matici, dostáváme pro x_B následující rovnici:

$$x_B = \left\{ \frac{1}{\det B} \times \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{n1} & B_{n2} & \dots & B_{nn} \end{bmatrix}^T \right\} \times b = \frac{1}{\det B} \times \left\{ \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{n1} & B_{n2} & \dots & B_{nn} \end{bmatrix}^T \times b \right\}, \tag{1.7}$$

kde

$$B_{ij} = (-1)^{i+j} \times M_{ij} \tag{1.8}$$

resp.

$$x_B = \frac{1}{\det B} \times B^{adj} \times b \tag{1.9}$$

Daný vzorec je platný pro čtvercovou matici – tj. typu *n* x *n*, což v případě výpočtu simplexovou metodou platí právě pro matici báze, jejíž řádky odpovídají počtu omezení (s výjimkou podmínek nezápornosti) a sloupce odpovídají stejnému počtu přídatných proměnných, které je nutno doplnit k vyrovnání nerovnic na rovnice.

1.5.2 Přesnost výpočtu

Pokud spojíme výše uvedené obecné vyjádření simplexového algoritmu revidované metody a větu o inverzní matici, dostáváme teoretický základ pro zlomkový simplexový algoritmus. Ještě než tak učiníme, je třeba si ujasnit, pro jaké úlohy má tento postup smysl.

Hlavní předností zlomkového simplexu, je počítání ve zlomkové aritmetice. Zlomkovou aritmetikou zde rozumíme podíl dvou celých čísel, který je zřejmě přesnější, než snaha o vyjádření čísla v desetinném rozvoji. Tak např. číslo $0,\overline{33}$ můžeme vyjádřit se vznikem zaokrouhlovací chyby jako 0,33333 nebo přesně jako 1/3. V případě velkého množství desetinných míst nebo při „příliš přísném“ zaokrouhlování si můžeme představit situaci, ve které by vliv nepřesného vyjádření mohl vést ke zkreslení výpočtu.

Obecně se dá říct, že výpočet v celých číslech je přesnější než výpočet v číslech desetinných, což je také hlavní myšlenka zlomkového simplexového algoritmu.

1.5.3 Vyjádření celé tabulky ve zlomcích

Z věty o inverzní matici (1.2) je třeba si uvědomit, že celá simplexová tabulka, jakožto výsledek operací s inverzní maticí báze, je vyjádřitelná jako zlomky se společným jmenovatelem, kterým je hodnota determinantu matice báze. Pro s-tou iteraci vypadá simplexová tabulka následovně:

$\frac{B_s^{adj} A}{\det B_s}$	$\frac{B_s^{adj}}{\det B_s}$	$\frac{B_s^{adj} b}{\det B_s}$
$\frac{-c_B^T B_s^{adj} A(-c^T)}{\det B_s}$	$u^T = \frac{c_B^T B_s^{adj}}{\det B_s}$	$c_B^T \frac{B_s^{adj} b}{\det B_s}$

Tabulka 1.4: *Maticové vyjádření simplexové tabulky v s-tém kroku výpočtu dle věty o inverzní matici*

Pokud chceme eliminovat vliv zaokrouhlovacích chyb, musíme zajistit, aby jak hodnoty v čitateli, tak hodnota ve jmenovateli byly celočíselné. Klíčový je opět pohled na inverzní matici báze. Výchozí inverzní matice báze je jednotková. Všechny další inverzní matice báze jsou odvozeny ze sloupců, které odpovídají základním proměnných ve výchozí simplexové tabulce, je tedy nutné, aby byla výchozí matice strukturních koeficientů celočíselná.

Determinant matice báze je možné vyjádřit také jako součet všech klíčových prvků od začátku výpočtu simplexovou metodou, resp. hodnoty determinantu v (s-1) iteraci s hodnotou klíčového prvku v s-té iteraci. (Pelikán, 2001, s. 76):

$$\det B_s = \det B_{s-1} \times a_{lk} \tag{1.10}$$

$$\det B_s = a_{lk(1)} \times a_{lk(2)} \times \ldots \times a_{lk(s)} \tag{1.11}$$

Pokud výchozí matice strukturních koeficientů *A* je celočíselná, také determinant báze bude celočíselný, protože matice báze *B* je odvozena ze sloupců výchozí matice *A* a zrovna tak determinant matice *B*.

Celočíselná výchozí matice *A* je dostačující požadavek, aby celá tabulka 1.4 měla společného celočíselného dělitele, je však ještě třeba zajistit, aby i hodnoty jmenovatele byly v celých číslech.

Pokud je také celočíselný výchozí vektor hodnot proměnných, je zřejmé, že výraz $B_s^{adj} b$ bude celočíselný. K tomu, aby také spodní řádek tab. 1.4 obsahoval v čitateli celá čísla, je třeba, aby výchozí řádek účelové funkce byl také v celých číslech.

Celá simplexová tabulka bude při splnění uvedených požadavků vyjádřitelná ve zlomcích se společným celočíselným jmenovatelem, který odpovídá v první iteraci prvnímu klíčovému prvku, ve druhé iteraci součinu prvního a druhého klíčového prvku atd.

Pokud tedy zaručíme, že všechny vstupní údaje ve výchozí simplexové tabulce budou v celých číslech a při každém kroku výpočtu budeme počítat s maticí rozšířenou o $\det B_s$ (resp. o klíčový prvek z předchozího kroku – viz (1.11)) je zřejmé, že celou dobu počítáme s celými čísly. Ilustrujme navržený postup na následující úloze lineárního programování, která má matici strukturních koeficientů, vektor hodnot pravých stran i řádek účelové funkce vyjádřeny v celých číslech.

Př. 1.1:

$$4x_1 + 2x_2 + 3x_3 + 6x_4 \rightarrow \max$$

$$\begin{aligned} 2x_2 + 4x_3 + 5x_4 &\leq 50 \\ 2x_1 + x_2 + 4x_4 &\leq 60 \\ 10x_1 + 5x_2 + 4x_3 + 10x_4 &\leq 180 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

Po převodu nerovnic na rovnice bude výchozí simplexová tabulka vypadat takto:

det=	1				Výchozí matice báze			
	x1	x2	x3	x4	x5	x6	x7	b
x5	0	2	4	5	1	0	0	50
x6	2	2	0	4	0	1	0	60
x7	10	5	4	10	0	0	1	180
Z	-4	-2	-3	-6	0	0	0	

Tabulka 1.5: *Výchozí simplexová tabuka k příkladu. 1.1*

Výchozí matice báze je jednotková proto i determinant této matice je roven 1.

Podle jednofázového simplexového algoritmu zvolíme klíčový sloupec, jako nejmenší nekladnou hodnotu z řádky účelové funkce. Vstupující proměnná je x_4. Vystupující proměnnou určíme podle nejmenší hodnoty podílu pravých stran vůči příslušnému prvku v klíčovém sloupci. Vystupující proměnná je x_5 a klíčový prvek je a_{14} s hodnotou 5. Podle standardních vzorců přepočteme celou tabulku tak, aby hodnota klíčového sloupce byla v pro všechny hodnoty kromě klíčového řádku rovna nule. Přepočet jednofázovou simplexovou metodou je vyjádřen v následující tabulce.

det=	5							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0	2/5	4/5	1	1/5	0	0	10
x6	2	2/5	-3 1/5	0	-0,8	1	0	20
x7	10	1	-4	0	-2	0	1	80
z	-4	2/5	1 4/5	0	1 1/5	0	0	60

Tabulka 1.6: *1. iterace dle běžné simplexové metody*

Do báze vstoupila proměnná x_4, a tak z výchozí simplexové tabulky můžeme vyjádřit příslušnou matici báze v první iteraci.

	x4	x6	x7
x4	5	0	0
x6	4	1	0
x7	10	0	1

Tabulka 1.7: *Matice báze dle výchozí simplexové tabulky v 1. iteraci*

Determinant matice báze bychom mohli vypočítat podle standardních vzorců k určení determinantu, ale využijeme souvislosti hodnoty determinantu v současné a následující iteraci a určíme ho podle vztahu (1.10) $\det B_2 = \det B_1 . \alpha_{lk} = 1 \times 5 = 5$

Pokud rozšíříme celou tabulku o tento výraz, máme zaručeno, že opět budeme počítat s celými čísly. Rozšíření jednotlivých řádků simplexové tabulky neporuší optimalitu řešení, protože násobíme pravou i levou část rovnic. Rozšířená tabulka vypadá následovně:

det=	5							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0	2	4	5	1	0	0	50
x6	10	2	-16	0	-4	5	0	100
x7	50	5	-20	0	-10	0	5	400
z	-20	2	9	0	6	0	0	300

Tabulka 1.8: 1. iterace rozšířená o hodnotu klíčového prvku

Nyní porušuje optimalitu výpočtu pouze první sloupec, proto vstupující proměnnou bude x_1. Vstupující proměnnou určíme z minima podílu pravých stran vůči druhému a třetímu řádku. Vystupující proměnnou je x_7 a klíčovým prvkem je nyní a_{31}. Všimněme si, že klíčový prvek má v „běžné“ simplexové tabulce hodnotu 10, zatímco v rozšířené tabulce hodnotu 50. Hodnotu determinantu matice báze je možné opět dopočítat z běžné tabulky podle vztahu (1.11): $\det B_3 = .\alpha_{lk(1)} \times \alpha_{lk(2)} = 5 \times 10 = 50$

V následující iteraci není porušena optimalita v řádku účelové funkce, dostáváme optimální řešení soustavy:

det=	50							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0	2/5	4/5	1	1/5	0	0	10
x6	0	1/5	-2 2/5	0	- 2/5	1	- 1/5	4
x1	1	1/10	- 2/5	0	- 1/5	0	1/10	8
z	0	4/5	1/5	0	2/5	0	2/5	92

Tabulka 1.9: *2. iterace dle běžné simplexové metody - optimální řešení soustavy*

-

Výsledné řešení je: $x = (8,0,0,10,0,4), z = 92$

Sice jsme dosáhli celočíselného řešení, ale obecně nemáme celočíselné řešení zaručené ani při výchozím zadání všech parametrů celočíselně.

1.6 Odvození vzorce pro přepočet tabulky

Skutečnost, že klíčový prvek v rozšířené tabulce odpovídá hodnotě determinantu matice báze v následujícím kroku, využijeme pro stanovení pravidla pro přepočet simplexové tabulky zlomkovým simplexovým algoritmem, které zaručí, že výpočetní tabulka bude ve všech krocích výpočtu obsahovat pouze celá čísla.

Úpravy zlomkového simplexového algoritmu jsou podobné postupu jednofázové simplexové metody, ale ne zcela identické. Neupravujeme klíčový řádek. Klíčový řádek s-té iterace zlomkového simplexu totiž vyjadřuje hodnotu běžného simplexového algoritmu rozšířenou o hodnotu det$B_{s+1,}$ což je přesně ta hodnota, kterou zde chceme mít, proto pro klíčový řádek beze změny opíšeme.

Řádky, které mají v klíčovém sloupci nenulové prvky upravíme tak, aby na místě klíčového sloupce byly nuly, ale celý řádek ještě rozšíříme o hodnotu klíčového prvku v běžném simplexu.

Řádky, které obsahují v klíčovém sloupci nuly je třeba rozšířit o hodnotu klíčového prvku v běžné simplexové tabulce.V našem případě se jedná o podíl $\det_{Bs+1}/ \det_{Bs.}$

Všechny poznatky shrneme do vzorce, který poslouží jako pravidlo pro přepočet zlomkovým simplexem:

pro $i \neq l$

$$a_{ij(s)} = \frac{a_{ij(s)} \times a_{lk(s)} - a_{lj(s)} \times a_{ik(s)}}{\det_{(s-1)}} \qquad (1.12)$$

pro $i = l$

$$a_{ij(s)} = a_{ij(s-1)}$$

kde $a_{ij(s)}$ představuje počítaný prvek v s-té iteraci

$a_{lk(s)}$ je klíčový prvek v s-té iteraci

i = 1,2..., n

j = 1,2,..., m

Uvažujme opět zmíněnou úlohu (Př. 1.1), ale počítejme pouze v rozšířené tabulce. Výchozí iterace vypadá stejně. Klíčovým prvkem bylo zvoleno a_{14} s hodnotou 5. Víme tedy, že determinant matice báze v první iteraci bude roven 5. Klíčový řádek opíšeme a zbytek tabulky dopočítáme dle (1.12).

det=	5							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0	2	4	5	1	0	0	50
x6	10	2	-16	0	-4	5	0	100
x7	50	5	-20	0	-10	0	5	400
z	-20	2	9	0	6	0	0	300

Tabulka 1.10: *1. iterace vypočtená dle vzorce (1.12)*

Klíčový řádek určíme podle minima ze záporných hodnot v řádku účelové funkce. Vstupující proměnná je x_1. Klíčovým prvkem je na základě minima z podílu {(100/10, (400/50)}zvolen prvek a_{31} s hodnotou 50. Víme tedy, že determinant matice báze ve druhé iteraci bude roven 50. Klíčový řádek opíšeme a zbytek tabulky opět upravíme dle zmíněného vzorce. Zde je vidět, že v prvním řádku simplexové tabulky je již dopředu hodnota nula, a kdyby se jednalo o běžnou jednofázovou simplexovou metodu, řádek bychom neupravovali. Zde je třeba jej vynásobit podílem současného a předchozího klíčového prvku, 50/5 = 10. Tento výpočet je také zahrnut ve vzorci (1.12).

Dostáváme:

det=	50							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0	20	40	50	10	0	0	500
x6	0	10	-120	0	-20	50	-10	200
x1	50	5	-20	0	-10	0	5	400
Z	0	40	10	0	20	0	20	4600

Tabulka 1.11: *2. iterace vypočtená dle vzorce (1.12) – optimum*

V řádce z žádná hodnota neporušuje optimalitu řešení a tak další úpravy neprovádíme. Proto abychom si zajistili počítání po celou dobu ve zlomkové aritmetice, byla celá tabulka rozšířena o hodnotu 50. Teď touto hodnotou musíme vydělit celou tabulku, abychom dostali výsledné řešení původní úlohy.

det=	50							
	x1	x2	x3	x4	x5	x6	x7	b
x4	0/50	20/50	40/50	50/50	10/50	0/50	0/50	500/50
x6	0/50	10/50	- 120/50	0/50	- 20/50	50/50	- 10/50	200/50
x1	50/50	10/100	- 20/50	0/50	- 10/50	0/50	5/50	400/50
z	0/50	40/50	10/50	0/50	20/50	0/50	20/50	4600/50

Tabulka 1.12: *Optimální řešení soustavy po vydělení hodnotou, o kterou byla tabulka rozšířena*

Při výpočtu tedy nakonec pracujeme pouze s jednou tabulkou. K interpretaci skutečných hodnot využijeme klíčový prvek z předchozí tabulky. Po zkrácení dostáváme stejný výsledek jako při počítání běžným způsobem – viz tab. 1.9.

Zlomkový simplexový algoritmus je založen na znalosti vztahů z revidované simplexové metody a maticového počtu. Pokud zajistíme, že celá výchozí simplexová tabulka bude obsahovat celá čísla, celý výpočet bude probíhat ve zlomcích.

Výchozí matice A je stěžejní pro určení celočíselných determinantů a klíčových prvků a celočíselný zbytek tabulky zaručuje, že budeme pracovat vždy s podílem dvou celých čísel. K tomu, abychom dopředu zaručili, že úloha lineárního programování bude v případě optimálního řešení mít celočíselné koeficienty, musíme se podrobněji zaměřit na povahu matice strukturních koeficientů.

2 Unimodularita

Velkou podskupinou optimalizačních úloh jsou úlohy, které vyžadují, aby řešení vyšlo v celých číslech. Pro daný typ úloh neexistuje optimalizační algoritmus, který by řešil celou třídu úloh v polynomiálním čase. Při řešení těchto úloh tak nemáme zaručeno, že se v rozumném čase dobereme optimálního řešení, i když máme k dispozici nejmodernější počítače. Základními metodami výpočtu praktických úloh jsou metody větvení a hranic, existují však také metody řezných nadrovin popřípadě metody, které vzniknou kombinací obou postupů (viz Pelikán, 2001, s.73). Řada publikací se zaobírá problémem unimodularity matic.

Uvažujme následující typ úloh:

$$\begin{aligned} &\max c^T x, \\ &Ax \le b, x \ge 0 \\ &x \in R^m \end{aligned} \qquad (2.1)$$

Kde:

A je matice strukturních koeficientů typu m x n

b je vektor pravých stran omezení typu 1 x n

c^T je transponovaný vektor účelové funkce typu 1x m

x je vektor proměnných typu 1 x n

Můžeme si položit otázku, kdy máme zaručeno při řešení lineární relaxace úlohy (2.1) celočíselné řešení, nebo jinak řečeno, kdy je možné k řešení úlohy typu (2.1) využít simplexovou metodu a mít zaručeno, že se dobereme požadovaného optima. V předchozí kapitole bylo naznačeno, že pokud chceme vyjádřit jednotlivé prvky simplexové tabulky jako podíl dvou celých čísel, je na základě revidovaného simplexového algoritmu klíčové postavení inverzní matice báze (odvozené z matice strukturních koeficientů A) a při přepočtu tabulky také celočíselný vektor b a řádka účelové funkce. Pokud na chvíli opustíme požadavek tohoto typu výpočtu a položíme si otázku, kdy máme zaručen celočíselný výsledek

bez ohledu na přepočet řádku účelové funkce je zřejmé, že klíčové postavení zaujímá výchozí matice strukturních koeficientů *A* a důležitá je také hodnota vektoru *b*.

Navzdory tomu, že ve zmíněných ukázkových úlohách předchozí kapitoly jsme dosáhli řešení, které požadavky celočíselného výsledku splnilo, jedná se spíše o náhodu a obecně takové řešení zajištěné nemáme.

Když se vrátíme ke vztahu (1.9) $x_B = \frac{1}{\det B} \times B^{adj} \times b$ vidíme, že celočíselné řešení dostáváme v případě, když se podaří součinem adjungované matice báze a vektoru pravých stran zkrátit hodnotou determinantu matice báze. Toho jsme dosáhli v příkladu (1.1). Pokud však chceme zajistit, aby vůbec neexistovalo nebezpečí, že náš výpočet skončí neceločíselným výsledkem, je třeba, aby determinant báze byl v každém kroku výpočtu +/-1, přičemž předpokládáme, že vektor hodnot pravých stran je v celých číslech. Abychom toho dosáhli, je třeba, aby matice *A* byla totálně unimodulární.

Unimodularita matice znamená, že všechny prvky dané matice jsou rovny: +1, -1, 0, ta nám však ještě nezaručí, že hodnota determinantu nedopadne jinak než potřebujeme.

Př. 2.1 *Určete hodnotu determinantu matice A*

$$A = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \det A = -1 \times 1 - 1 \times 1 = -2$$

Jak vidíme i přesto, že matice A je unimodulární, determinant je roven -2. Zvláštním případem unimodulární matice je tzv. totální unimodularita matice, která je popsána v následující definici.

Definice *totálně unimodulární matice* (Turzík, 1999, s. 61): (2.1)

Řekneme, že matice $A = a_{ij}$ je totálně unimodulární, je-li determinant každé její čtvercové podmatice roven číslům +1, -1, 0.

Pokud vyjádříme inverzní matici báze, při totálně unimodulární matici A ve tvaru $B^{-1} = B^{adj} / |B|$ vidíme, že v čitateli bude vždy hodnota 0, +1 nebo -1 (všechny subdeterminanty jsou unimodulární), zatímco ve jmenovateli bude hodnota determinantu – tzn. +/- 1 (do báze mohou vstoupit pouze nenulové prvky). Tato úvaha nám ukazuje, že pokud vyřešíme úlohu lineárního programování, pro kterou platí, že matice *A* je totálně unimodulární a výchozí

vektor hodnot pravých stran je v celých číslech, máme zaručeno, – viz (1.10), že v případě řešitelnosti soustavy, dostaneme celočíselné řešení.

Zmíněné poznatky jsou obsaženy v následující větě:

Věta – *Význam totální unimodulární matice pro celočíselné programování* (Turzík, 1999, s. 61): (2.2)

Je-li A totálně unimodulární matice typu (m,n) a $b \in \mathrm{R}^m$ celočíselný vektor, pak každé bazické řešení soustavy $Ax = b$ je rovněž celočíselné.

Danou skutečnost budeme ilustrovat na následujícím příkladu.

Př. 2.2 *Ilustrace věty o významu unimodulární matice pro celočíselné programování*

Uvažujme následující přiřazovací problém. Máme tři pracovníky přiřadit ke třem projektům, abychom dosáhli nejefektivnějšího přiřazení, máme zadáno ocenění i-tého pracovníka k j-tému projektu:

$$8x_{11} + 9x_{12} + 7x_{13} + 6x_{21} + 5x_{22} + 12x_{23} + 4x_{31} + 7x_{32} + 5x_{33} \to \max$$

$$\sum_{i=1}^{3} x_{ij} = 1 \quad j = 1,2,3$$

$$\sum_{i=1}^{3} x_{ji} = 1 \quad j = 1,2,3$$

$$x_{ij} \geq 0, \textit{celé}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \qquad b = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$$

Pokud úlohu řešíme simplexovou metodou jako úlohu lineárního programování, dostaneme v 6-té iteraci následující řešení: $x_{opt} = (1,0,0,0,0,1,0,1,0), \quad z = 27$

Zmíněné řešení znamená, že první pracovník bude přiřazen k prvnímu projektu s přínosem 8-mi jednotek. Druhý pracovník bude přiřazen ke třetímu projektu s přínosem 12-ti jednotek a třetí pracovník bude přiřazen k druhému projektu s přínosem 7-mi jednotek.

Mezi třídu celočíselných úloh, které mají totálně unimodulární matici strukturních koeficientů a mohou být řešeny nástroji lineárního programování, patří vedle zmíněného přiřazovacího problému také klasický dopravní problém a úlohy o hledání maximálního a minimálního toku na síti (Salkin, 1989, s. 74). Poslední zmíněné jsou důležitou součástí klasifikace celé škály úloh s totálně unimodulárními maticemi.

Navzdory tomu, že zmíněné úlohy jsou řešitelné jako úlohy lineárního programování, je také důležitá jejich formulace jakožto úloh celočíselného programování, protože později mohou obsahovat další omezení, které odpovídají reálným situacím a původní znění úlohy je narušeno a tak je třeba zvolit jiný postup výpočtu (Pelikán, 2001, s. 131).

2.1 Vlastnosti totálně unimodularní matice

V této podkapitole zmíníme větu, která popisuje vlastnosti unimodulárních matic, které shrnují a rozšiřují vlastnosti totálně unimodulárních matic.

Věta: *O vlastnostech totálně unimodulární matice* (2.3)
Nechť A je matice s hodnotami 0, +1 nebo -1. Potom následující tvrzení jsou ekvivalentní (Schrijver, 1999, s. 269):

(i) A je totálně unimodulární, tzn. každá čtvercová submatice z A má determinant 0, +1, -1. $\{x \mid x \geq 0, Ax \leq b\}$

(ii) Pro každý celočíselný vektor b má polyedr pouze celočíselné vrcholy.

(iii) Pro všechny celočíselné vektory a,b,c,d, polyedr $\{x \mid c \leq x \leq d, a \leq Ax \leq b\}$ má pouze celočíselné vrcholy.

(iv) Každý soubor sloupců z A může být rozdělen do dvou částí tak, aby suma sloupců z jedné části mínus suma sloupců z druhé části byl vektor, který obsahuje pouze hodnoty 0, +1 a -1.

(v) Každá nesingulární submatice z A má řádku s lichým počtem nenulových členů.

(vi) Součet členů každé čtvercové submatice se sudým počtem řádků a sloupců je dělitelný čtyřmi.

(vii) Žádná čtvercová submatice z A nemá determinant +2 nebo -2.

2.2 Síťové matice

Síťové matice jsou předním typem totálně unimodulárních matic. V literatuře (Schrijver, 1999, s. 272) je uvedeno sedm typů totálně unimodulárních matic, které jsou zvláštním případem síťových matic, známých z teorie grafů. Totálně unimodulární síťové matice jsou základními stavebními kameny všech totálně unimodulárních matic, jak bude naznačeno v následujícím odstavci. Testování, zdali matice *M* je síťová matice, je možné provézt v polynomiálním čase. Pro hlubší specifikaci daných matic doporučuji uvedenou literaturu.

2.3 Dekompozice totálně unimodulárních matic

Abychom naznačili, jak probíhá zkoumání, jestli matice *M* je totálně unimodularní, uvedeme zde znění Seymurovi věty. Seymur (1980) ukázal, že každou totálně unimodulární matici je možno složit ze síťových matic a těchto dvou matic (Schrijver, 1999, s. 280) :

$$\begin{bmatrix} 1 & -1 & 0 & 0 & -1 \\ -1 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Nejprve zmíníme operace, které neporušují totální unimodularitu matic:

(i) Permutace řádků a sloupců.

(ii) Uvažovaní transponované matice místo výchozí.

(iii) Rozšíření řádku nebo sloupce -1.

(iv) Otočení, tj. nahrazení: $\begin{bmatrix} \varepsilon & c \\ b & D \end{bmatrix} za \begin{bmatrix} -\varepsilon & \varepsilon c \\ \varepsilon b & D - \varepsilon bc \end{bmatrix}$,

$kde: \varepsilon \in \{\pm 1\}$, b je sloupcový vektor, c je řádkový vektor a D je matice.

(v) Přidání řádku nebo sloupce, který obsahuje samé nuly, nebo řádku či sloupce, který obsahuje jeden nenulový prvek, který je ±1.

(vi) Zopakováním řádku či sloupce.

(2.5)

Navíc je možné provést ještě následující sestavy:

i) $$A \oplus_1 B := \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$ (1-sum);

ii) $$\begin{bmatrix} A & a \end{bmatrix} \oplus_2 \begin{bmatrix} b \\ B \end{bmatrix} := \begin{bmatrix} A & ab \\ 0 & B \end{bmatrix}$$ (2-sum);

iii) $$\begin{bmatrix} A & a & a \\ c & 0 & 1 \end{bmatrix} \oplus_3 \begin{bmatrix} 1 & 0 & d \\ d & d & B \end{bmatrix} := \begin{bmatrix} A & ab \\ dc & B \end{bmatrix}$$ (3-sum). (2.6)

kde A a B jsou matice, *a* a *d* jsou sloupcové vektory, a *b* a *c* jsou řádkové vektory odpovídajících rozměrů.

Věta: *Seymourova dekompoziční věta o totálně unimodulárních maticích* (Schrijver, 1999, s. 279 – 280).

Matice *A* je totálně unimodulární právě tehdy a jen tehdy, když *A* je složena ze síťových matic a matic (2.4) aplikací pravidel (2.5) a (2.6). Kde operace (2.6) jsou proveditelné, pouze pokud pro matici *A* i *B* máme: počet řádků plus počet sloupců roven alespoň 4.

2.4 Testování totální unimodularity matice

Jedním z důsledků Seymourovy věty je, že je možné sestavit test na totální unimodularitu v polynomiálním čase. Nyní naznačíme, jak tento test vypadá, pro detailnější popis opět odkazuji na literaturu, ze které je postup převzat (Schrijver, 1999, s. 282-293).

Zkoumání totální unimodularity matice *M* znamená postup, který se skládá ze čtyř kroků dosažitelných v polynomiálním čase:

1. Prozkoumání jestli celá matice obsahuje pouze hodnoty +1,-1,0, vyřazení všech lineárně závislých řádků a sloupců, a vyřazení všech řádků či sloupců, které obsahují nanejvýš jednu nenulovou hodnotu – výsledná matice *M* sestává ze všech lineárně nezávislých řádků a sloupců, které v každém řádku i sloupci mají minimálně dvě nenulové hodnoty.

2. Na základě Seymourovy věty zkoumáme, jestli je možné matici M nebo M^T převézt na některou síťovou matici či jednu z matic (2.2). Pokud ano → jedná se o totálně unimodulární matici. Jinak → krok 3.

3. Dekompoziční test – zkontrolovat, jestli se daná matice M dá rozložit jako :

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

kde hodnost (B) + hodnost (C) ≤ 2 a jak pro matici A tak i pro matici D platí:
počet sloupců + počet řádků ≥ 4.

Pokud ne → nejedná o totálně unimodulární matici.
Jinak → krok 4.

4. Může nastat šest případů, ve kterých rozhoduje hodnost matice A a B – na základě toho se po další analýze těchto relativně malých matic rozhodne, jestli M je totálně unimodulární matice. Ve všech případech je prověřování totální unimodularity M redukováno na zkoumání totální unimodularity matic menších rozměrů, které je možné provézt v polynomiálním čase.

V této kapitole jsme upozornili na zvláštní případ úloh celočíselného programování, které mají totálně unimodulární matici. Zkoumání totální unimodularity matice je možné provézt v polynomiálním čase. Pokud bychom tedy provedly tento test a uspěli, můžeme se vyhnout nutnosti užití postupů, pro které není znám polynomiální algoritmus.

Při celočíselném vektoru *b* odpovídá optimální řešení těchto úloh (2.1) řešení úloh (1.1), k jejich řešení tedy můžeme použít nástroje řešící úlohy lineárního programování.

3 Zlomkový simplexový algoritmus

Zlomkový simplexový algoritmus využívá znalosti, že v každém kroku výpočtu je možné vyjádřit simplexovou tabulku ve tvaru zlomků se společným jmenovatelem, který je roven 1/det B_s. Postup předpokládá, že výchozí matice báze obsahuje přídatné proměnné, tudíž je jednotková. V této kapitole uvádíme znění algoritmu spolu s jedním příkladem, na kterém postup demonstrujeme.

3.1 Zlomkový simplexový algoritmus.

Krok 1. Inicializace

Ověření, že matice strukturních koeficientů a vektoru pravých stran jsou zadány v celých číslech.

s = 1 (index iterace)

$det_{B(s-1)}$ = 1 (výchozí inverzní matice báze je jednotková matice)

m = počet omezení

n = počet proměnných (včetně přídatných)

Krok 2. Test optima

z_{min} = 0

k = 0

Volba vystupující proměnné

pro j = 1,2,..,n:

pokud: $z_j < z_{min}$, pak $z_j = z_{min}$ a $k = j$, $j = j + 1$

jinak: $j = j + 1$

pokud $k = 0$ a $s > 1$ → jdi na krok 5

pokud $k = 0$ a $s = 1$ → konec

jinak jdi na krok 3.

Krok 3. Volba vstupující proměnné

t = 10000000

l = 0

pro i = *1,2,..,m:*

pokud $a_{ik} > 0$ pak podíl(i) = $b_{(i)}/a_{(i,k)}$

pokud podíl < t, pak $t = podil\ (i)$ a $l = i$, $i = i + 1$

jinak $i = i + 1$

pokud $a_{ik} < 0$, $i = i + 1$

pokud $l = 0$, jdi na krok 5.

jinak jdi na krok 4

Krok 4. Přepočet tabulky

pro i = 1,2,.., m+1

pokud $i \neq l$

pro j = 1,2,...,n + 1

$$a_{ij(s)} = \frac{a_{ij(s)} \times a_{lk(s)} - a_{lj(s)} \times a_{ik(s)}}{a_{lk(s-1)}}$$

$j = j + 1$

$i = i + 1$

jinak $i = i + 1$

$s = s + 1$

$\det_{B(s)} = a_{lk}$

jdi na krok 2

Krok 5. Převod na zlomky

pro $v = s$:

pro i = 1,2,.., m + 1

pro j = 1,2,…,n + 1

$$a_{ij(v)} = \frac{a_{ij(v)}}{\det_{lk(s-1)}}$$

$j = j + 1$

$i = i + 1$

pokud $i = m+1$ → konec

3.2 Ukázkový příklad řešený Zlomkovým simplexovým algoritmem:

Uvažujme následující úlohu lineárního programování (Jablonský, Lagová, 2009, s.135):

Př. 3.1:

$$6x_1 + 4x_2 + 6x3 \rightarrow \max$$
$$2x_1 + 3x_2 + 2x_3 \leq 180$$
$$2x_1 + x_2 + x_3 \leq 100$$
$$x_1 + x_2 + x_3 \leq 110$$
$$x_1, x_2, x_3 \geq 0$$

Výchozí simplexová tabulka vypadá následovně:

	x1	x2	x3	x4	x5	x6	b
x4	2	3	2	1	0	0	180
x5	2	1	1	0	1	0	100
x6	1	1	1	0	0	1	110
Z	-6	-4	-6	0	0	0	0

Tabulka 3.1: *Výchozí tabulka Př..1*

Krok 1. Inicializace

Matice *A* i vektor *b* jsou zadány v celých číslech, vstupní požadavky jsou tedy splněny a přecházíme k výpočtu.

$s = 1$

$\det_B = 1$

$m = 3$

$n = 6$

Krok 2. Test optima

zmin = 0 k = 0

Volba vystupující proměnné:

$j = 1$

$z_1 = -6$

$z_1 > \text{zmin} \rightarrow \text{zmin} = -6,\ k = 1$

$j = 2$

$z_2 = -4$

$z_2 < \text{zmin} \rightarrow j = 3$

$z_3 = -6$

$z_3 = \text{zmin} \rightarrow j = 4$

$z_4 = 0 > \text{zmin} \rightarrow j = 5$

$z_5 = 0 > \text{zmin} \rightarrow j = 6$

$z_6 = 0 > \text{zmin},\ k \neq 0 \rightarrow$ Krok 3.

Jako vstupující proměnná byla zvolena nezákladní proměnná x_1, protože spolu s proměnnou x_3 představuje největší ztrátu, ale přišla na řadu jako první, proto ji algoritmus vybral.

Krok 3. Volba vystupující proměnné

$t = 10000000$

$l = 0$

$i = 1$

$a_{11} > 0 \rightarrow$ podil(1) = 180/2 = 90, podil(1) < *t*, podil(1) = *t*, $l = 1$, $i = 2$

$a_{12} > 0 \rightarrow$ podil(2) = 100/2 = 50, podil(2) < *t*, podil(2) = *t*, $l = 2$, $i = 3$

$a_{13} > 0 \rightarrow$ podil(3) = 110/1 = 110, podil(2) > *t*,

$l \neq 0 \rightarrow$ Krok 4.

Volba vystupující proměnné probíhá standardním způsobem známým z jednofázové simplexové metody, tzn. podílem pravých stran a příšlušných kladných hodnot klíčového sloupce. V našem případě je vystupující proměnná x_5. Po určení klíčového prvku, kterým je x_{12} bude přepočítána celá tabulka.

Krok 4. Přepočet tabulky

$i = 1$

$i \neq l \rightarrow j = 1, a_{11} = (2*2 - 2*2)/1 = 0/1 = 0$

$\rightarrow j = 2, a_{12} = (3*2 - 1*2)/1 = 4/1 = 4$

$\rightarrow j = 3, a_{13} = (2*2 - 2*2)/1 = 0/1 = 0$

$\rightarrow j = 4, a_{14} = (1*2 - 0*2)/1 = 2/1 = 2$

$\rightarrow j = 5, a_{15} = (0*2 - 1*2)/1 = -2/1 = -2$

$\rightarrow j = 6, a_{16} = (0*2 - 0*2)/1 = 0/1 = 0$

$\rightarrow j = 7, a_{17} = (180*2 - 100*2)/1 = 160/1 = 160$

$i = 2, i = l \rightarrow$ pro $j = 1,2,3,4,5,6,7$: $a_{2j} = a_{2j}, i = 3$

$i \neq l \rightarrow j = 1, a_{31} = (1*2 - 2*1)/1 = 0/1 = 0$

$\rightarrow j = 2, a_{32} = (1*2 - 1*1)/1 = 1/1 = 1$

$\rightarrow j = 3, a_{33} = (1*2 - 1*1)/1 = 1/1 = 1$

$\rightarrow j = 4, a_{34} = (0*2 - 0*1)/1 = 0/1 = 2$

$\rightarrow j = 5, a_{35} = (0*2 - 1*1)/1 = -1/1 = -1$

$\rightarrow j = 6, a_{36} = (1*2 - 0*1)/1 = 2/1 = 2$

$\rightarrow j = 7, a_{37} = (110*2 - 100*2)/1 = 20/1 = 20$

$i = 4, i \neq l \rightarrow j = 1, a_{31} = [(-6)*2 - 2*(-6)]/1 = 0/1 = 0$

$\rightarrow j = 2, a_{32} = [(-4)*2 - 1*(-6)]/1 = -2/1 = -2$

$\rightarrow j = 3, a_{33} = [(-6)*2 - 1*(-6)]/1 = -6/1 = -6$

$\rightarrow j = 4, a_{34} = [0*2 - 0*(-6)]/1 = 0/1 = 0$

$\rightarrow j = 5, a_{35} = [0*2 - 1*(-6)]/1 = 6/1 = 6$

$\rightarrow j = 6, a_{36} = [0*2 - 0*(-6)]/1 = 2/1 = 2$

$\rightarrow j = 7, a_{37} = [0*2 - 100*(-6)]/1 = 600/1 = 600$

$s = 2$

Det$B(2) = 2$

jdi na krok 2

Přepočet proběhl tak, že jsme klíčový řádek opsali a zbytek dopočítali podle vzorce. Celá tabulka je nyní rozšířena o determinant matice báze následujícího kroku resp. hodnotu klíčového prvku. Vracíme se zpět na druhý krok a testujeme optimalitu řešení.

V tabulce to vypadá takto:

det=	2						
	x1	x2	x3	x4	x5	x6	b
x4	0	4	2	2	-2	0	160
x1	2	1	1	0	1	0	100
x6	0	1	1	0	-1	2	120
z	0	-2	-6	0	6	0	600

Tabulka 3.2: *Optimální řešení soustavy*

Krok 2. Test optima

zmin = 0

k = 0

Volba vystupující proměnné:

$j = 1$

$z_1 = 0$

z_1 = zmin → $j = 2$

$z_2 = -2$

z_2 < zmin → zmin = -2, l = 2, j = 3

$z_3 = -6$

z_3 < zmin → zmin = -6, l = 3, j = 4

$z_4 = 0 >$ zmin → $j = 5$

$z_5 = 6 >$ zmin → $j = 6$

$z_6 = 0 >$ zmin, $k \neq 0$ → Krok 3.

Optimalita je porušena u proměnných x_2 a x_3, přičemž klíčový sloupec je x_3, protože zde je hodnota účelové funkce minimální.

Krok 3. Volba vystupující proměnné

t = 10000000

l = 0

i = 1

a_{11} > 0 → podil(1) = 160/2 = 80, podil(1) < t, podil(1) = t, $l = 1$, $i =$ 2

a_{12} > 0 → podil(2) = 100/1 = 100, podil(2) > t, podil(2), i = 3

a_{13} > 0 → podil(3) = 120/1 = 120, podil(2) > *t*, $l \neq 0$ → Krok 4.

Vystupující proměnnou je tentokrát x_4, protože podíl pravé strany vůči příslušné hodnotě v simplexové tabulce je vůči ostatním minimální.

Krok 4. Přepočet tabulky

i = 1 *i* = *l* → pro *j* = 1,2,3,4,5,6,7: $a_{1j} = a_{1j}$, *i* = 2

$i \neq l$ → *j* = 1, a_{21} = (2*2 – 0*1)/2 = 4/2 = 2

→ j = 2, a_{22} = (1*2 – 4*1)/2 = -2/2 = -1

→ j = 3, a_{23} = (1*2 – 2*1)/2 = 0/2 = 0

→ j = 4, a_{24} = (0*2 – 2*1)/2 = -2/2 = -1

→ j = 5, a_{25} = [1*2 – (-2)*1]/2 = 4/2 = 2

→ j = 6, a_{26} = (0*2 – 0*1)/2 = 0/2 = 0

→ j = 7, a_{27} = (100*2 – 160*1)/2 = 40/2 = 20

i = 3

$i \neq l$ → *j* = 1, a_{21} = (0*2 – 0*1)/2 = 0/2 = 0

→ *j* = 2, a_{22} = (1*2 – 4*1)/2 = -2/2 = -1

→ *j* = 3, a_{23} = (1*2 – 2*1)/2 = 0/2 = 0

→ *j* = 4, a_{24} = (0*2 – 2*1)/2 = -2/2 = -1

→ *j* = 5, a_{25} = [(-1)*2 – (-2)*1]/2 = 0/2 = 0

→ *j* = 6, a_{26} = (2*2 – 0*1)/2 = 4/2 = 2

→ *j* = 7, a_{27} = (120*2 – 160*1)/2 = 80/2 = 40

s = 3

DetB(3) = 2

jdi na krok 2

Ve čtvrtém kroku je opět přepočítaná tabulka s výjimkou klíčového řádku. Výsledné řešeni je v následující tabulce.

det=	2						
	x1	x2	x3	x4	x5	x6	b
x3	0	4	2	2	-2	0	160
x1	2	-1	0	-1	2	0	20
x6	0	-1	0	-1	0	2	40
Z	0	10	0	6	0	0	1080

Tabulka 3.3: *Optimální řešení soustavy*

Krok 2. Test optima

$z_{min} = 0$

$k = 0$

Volba vystupující proměnné:

$j = 1$

$z_1 = 0$

$z_1 = \text{zmin} \rightarrow j = 2$

$z_2 = 10$

$z_2 > \text{zmin} \rightarrow j = 3$

$z_3 = 0$

$z_3 = \text{zmin} \rightarrow j = 4$

$z_4 = 6$

$z_4 > \text{zmin} \rightarrow j = 5$

$z_5 = 0$

$z_5 = \text{zmin} \rightarrow j = 6$

$z_6 = 0$

$z_6 = \text{zmin}, k = 0 \rightarrow$ jdi na krok 5

Vidíme, že v tomto bodě již žádná proměnná neporušuje optimalitu řešení. Abychom dostali skutečné hodnoty proměnných a hodnotu kriteriální funkce, je třeba celou tabulku převézt na zlomky tj. vydělit hodnotu celé tabulky hodnotou determinantu matice báze – resp. klíčovým prvkem v předchozím kroku. Pokud bychom požadovali, např. pro kontrolu výpočtu jednofázového simplexového algoritmu, abychom měli k dispozici všechny iterace výpočtu, bylo by třeba vydělit také první iteraci příslušnou hodnotou determinantu.

Krok 5. Převod na zlomky

$v = 3, a_{lj(2)} = 2,$

$i = 1,$ $j = 1$

$a_{11(3)} = 0/2 = 0,$ $j = 2$

$a_{12(3)} = 4/2 = 2,$ $j = 3$

$a_{13(3)} = 2/2 = 1,$ $j = 4$

$a_{14(3)} = 2/2 = 1,$ $j = 5$

$a_{15(3)} = -2/2 = -1$, $j = 6$

$a_{16(3)} = 0/2 = 0$, $j = 7$

$a_{17(3)} = 160/2 = 80$

$i = 2$, $j = 1$

$a_{21(3)} = 2/2 = 1$, $j = 2$

$a_{22(3)} = -1/2$, $j = 3$

$a_{23(3)} = 0/2 = 0$, $j = 4$

$a_{24(3)} = -1/2$, $j = 5$

$a_{25(3)} = 2/2 = 1$, $j = 6$

$a_{26(3)} = 0/2 = 0$, $j = 7$

$a_{27(3)} = 20/2 = 10$

$i = 3$, $j = 1$

$a_{31(3)} = 0/2 = 0$, $j = 2$

$a_{32(3)} = -1/2$, $j = 3$

$a_{33(3)} = 0/2 = 0$, $j = 4$

$a_{34(3)} = -1/2$, $j = 5$

$a_{35(3)} = 0/2 = 0$, $j = 6$

$a_{36(3)} = 2/2 = 1$, $j = 7$

$a_{37(3)} = 40/2 = 20$

$i = 4$, $j = 1$

$a_{41(3)} = 0/2 = 0$, $j = 2$
$a_{42(3)} = 10/2 = 5$, $j = 3$
$a_{43(3)} = 0/2 = 0$, $j = 4$
$a_{44(3)} = 6/2 = 3$, $j = 5$
$a_{45(3)} = 0/2 = 0$, $j = 6$
$a_{46(3)} = 0/2 = 0$, $j = 7$
$a_{47(3)} = 1080/2 = 540$

	x1	x2	x3	x4	x5	x6	b
x3	0	2	1	1	-1	0	80
x1	1	- 1/2	0	- 1/2	1	0	10
x6	0	- 1/2	0	- 1/2	0	1	20
z	0	5	0	3	0	0	540

Tabulka 3.4: *Optimální řešení soustavy ve formě zlomků*

Optimálního řešení jsme dosáhli ve druhé iteraci. Nejprve do báze vstoupila nezákladní proměnná x_1, která nahradila přídatnou proměnnou x_5 a v dalším kroku opět nezákladní proměnná x_3, která nahradila přídatnou proměnnou x_4. Hodnota účelové funkce je $z_{opt} = 540$ a optimální vektor proměnných je roven $x_{opt} = (10, 0, 80, 0, 0, 20)$.

Popsali jsme zlomkový simplexový algoritmus. Oproti běžnému postupu se nemusí upravovat klíčový řádek, ale za to se musí přepočítávat zbytek tabulky, včetně řádků, které mají v klíčovém sloupci nuly.

4 Uživatelské prostředí aplikace „Zlomkový simplex“

V této části popíšeme základní funkcionalitu aplikace „Zlomkový simplex“ vytvořenou v prostředí Microsoft Excel 2007 a pro správnou funkcionalitu je třeba mít tuto verzi Excelu. Postupně se seznámíme se vstupním uživatelským prostředím a dále podrobněji rozvedeme požadavky na zadávání úloh. Stěžejní bude popis uživatelského prostředí poskytované aplikace.

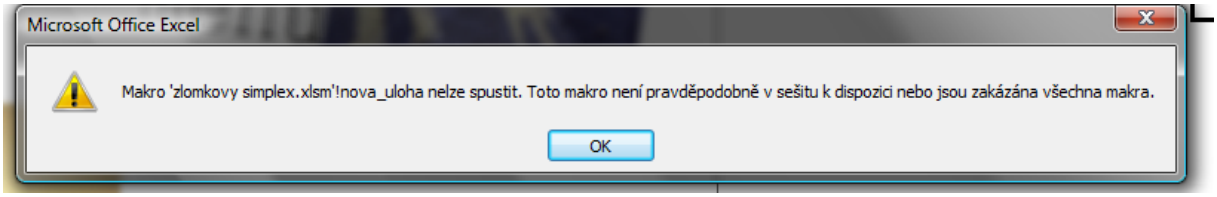
Implementován je pouze algoritmus jednofázové zlomkové simplexové metody, tzn. pomocí aplikace „Zlomkový simplex“ je možné řešit pouze úlohy lineárního programování, které mají všechna omezení typu „<=”. Celá aplikace je určena jako možný nástroj pro řešení především jednodušších úloh lineárního programování, které jsou jinak řešitelné pomocí algoritmu jednofázové simplexové metody.

Tabulový kalkulátor Excel byl vybrán, protože je hojně využíván a navíc podporuje programovací jazyk *Visual Basic for Applications*, který je vhodný ke tvorbě aplikace v uživatelsky příznivé podobě. Přáním autora je, aby aplikace posloužila zejména pro účely katedry ekonometrie VŠE, ale je samozřejmě možné ji využít i pro samostatné menší výpočty.

4.1 Problémy se správným fungováním

Správnou funkcionalitu aplikace může ohrozit nesprávné nastavení maker. Makra mohou v některých případech představovat nebezpečí, proto na některých počítačích nejsou povolena. Podle vašeho nastavení je možné setkat se v zásadě se dvěma problémy. Následující postup je platný pro verzi 2007, i když aplikace také bezproblémově fungovala na betaverzi Exelu 2010.

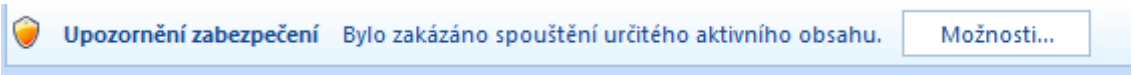
Pokud máte makra zcela zakázána, objeví se při stisknutí tlačítka v uživatelské nabídce následující hláška:



Obrázek 4.1: *Zakázaná makra v MS Excel 2007*

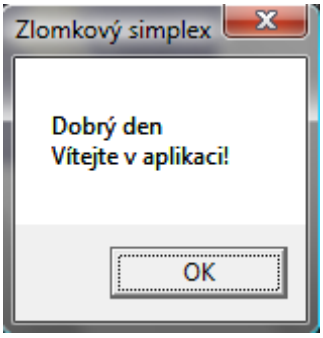
Je třeba kliknout na tlačítko MS Office, dále zvolte „Možnosti aplikace Excel“, „Centrum zabezpečení“, klikněte na tlačítko „Nastavení Centra zabezpečení…“ a zde vyberte možnost „Povolit všechna makra“. Pro správný chod aplikace je ještě třeba Excel restartovat.

Druhý problém nastává v případě, že máte makra zakázána s oznámením. Po spuštění aplikace se to projeví tak, že se zobrazí varovná hláška pod horní lištou v následující podobě:



Obrázek 4.2: *Zakázáná makra s oznámením v MS Excel 2007*

Klikněte na „Možnosti…“ a dále „Povolit tento obsah“.
Klikněte na „OK”.



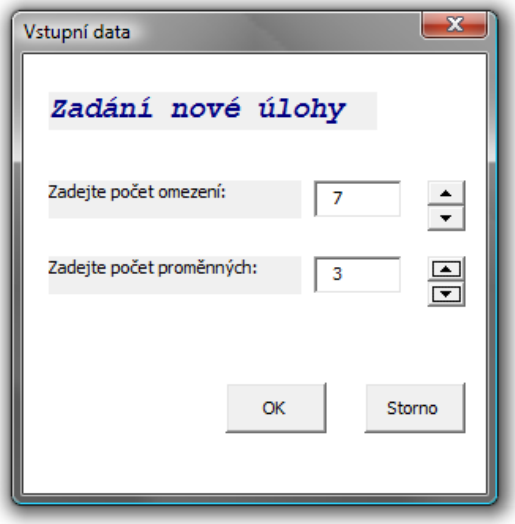
Obrázek 4.3: *Potvrzení správného nastavení maker a spuštění aplikace*

4.2 Vstupní prostředí

Obrázek 4.4: *Vstupní uživatelské prostředí aplikace "Zlomkový simplex"*

4.2.1 Nová úloha

Volbu „nová úloha“ zvolte v případě, že chcete přejít rovnou k zadávání nového příkladu. Pokud na něj kliknete, objeví se nabídka, ve které je třeba zadat počet omezení a proměnných. Počet proměnných je třeba zadat včetně přídatných proměnných. K výběru použijte posuvníky a potvrďte tlačítkem „OK“.



Obrázek 4.5: *Dialogové okno pro zadání nové úlohy*

Dostáváme se na vstupní kartu, ve které máme vygenerovanou tabulku podle zadání. Další postup je popsán v podkapitole výpočet.

det=	1							
	x1	x2	x3	x4	x5	x6	x7	b
x5								
x6								
x7								
z								

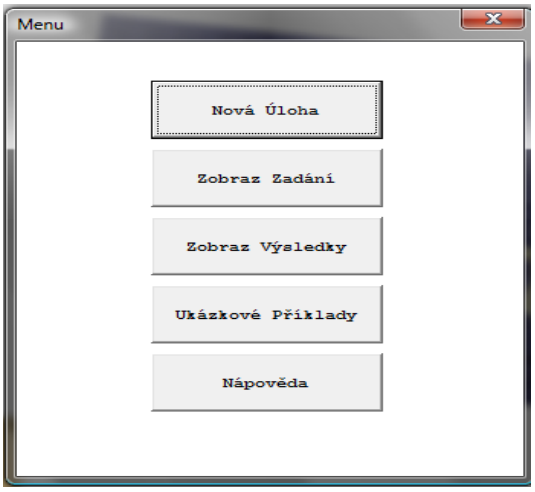
Tabulka 4.1: *Prázdná vygenerovaná tabulka podle počtu omezení a proměnných*

Zadání počtu omezení je nastaveno na maximální hodnotu 150. V případě, že byste potřebovali zadat úlohu větších rozměrů, je možné zadat na kartě hodnoty do buněk, které

jsou nadepsány „počet proměnných a počet omezení“ předtím než vybere možnost „nová úloha.“

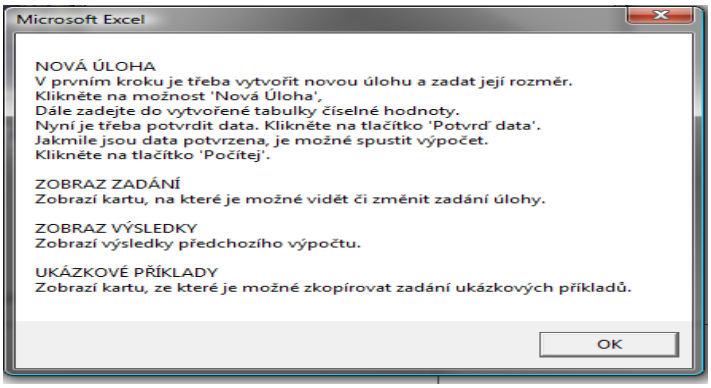
4.2.2 Menu

Tlačítko „menu“ vyvolá následující nabídku:



Obrázek 4.6: *Zobrazení základní nabídky aplikace "Zlomkový simplex"*

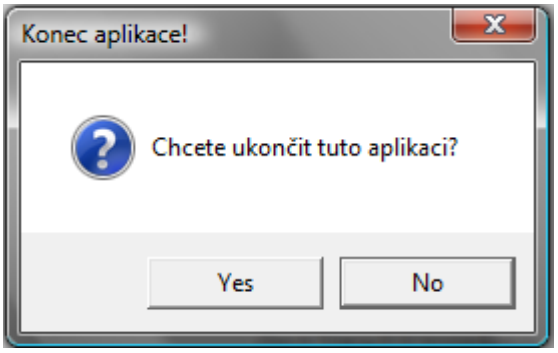
Tlačítko „Nová Úloha“ má stejný efekt jako stejnojmenné tlačítko z uživatelské nabídky. Pokud zvolíme možnost „Zobraz Zadání“, dostaneme se na kartu se zadáním, tímto postupem se např. dostaneme k minulé uložené úloze. Stisknutím tlačítka „Zobraz Výsledky“ se dostaneme na kartu, ve které je provedený výpočet, pokud jsme již dříve spustili výpočet. „Ukázkové Příklady“ nabízí několik příkladů, na kterých je možné vyzkoušet chod aplikace. Pokud se rozhodneme využít tyto příklady, je nejprve nutné zadat novou úlohu a potom zkopírovat znění z ukázkových příkladů do zadání. Pokud zvolíme možnost „Nápověda“, zobrazí se nám následující okno:



Obrázek 4.7: *Zobrazení nápovědy v nabídce menu*

4.2.3 Exit

Poslední tlačítko „exit“ slouží k ukončení aplikace.



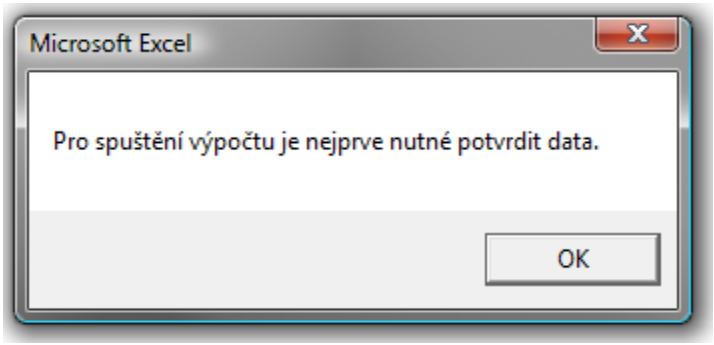
Obrázek 4.8: *Tlačítko "exit" - ukončení aplikace "Zlomkový simplex"*

4.3 Výpočet

V úvodní nabídce klikneme na tlačítko ‚nová úloha’, nebo v Menu zvolíme první možnost. Zadáme rozměr úlohy a potvrdíme tlačítkem OK. Dostáváme se na kartu výpočtu. Nahoře vidíme několik tlačítek:

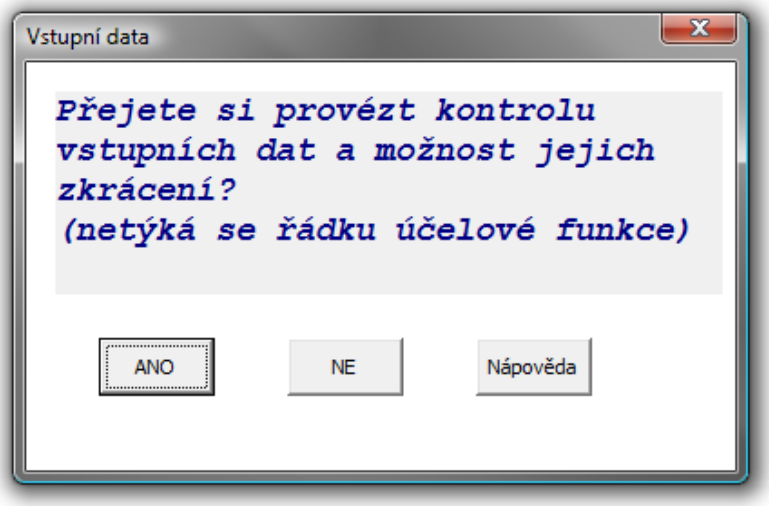
Obrázek 4.9: *Uživatelská nabídka na kartě vstupních dat*

Pokud jsme zadali špatný rozměr úlohy, můžeme zvolit možnost Nová úloha a vygenerovat novou tabulku. Jinak vyplníme data do tabulky a zvolíme možnost „Potvrď data“. V případě, že zapomeneme potvrdit data úlohy a pokusíme se rovnou spustit výpočet, aplikace nás upozorní následujícím oknem:



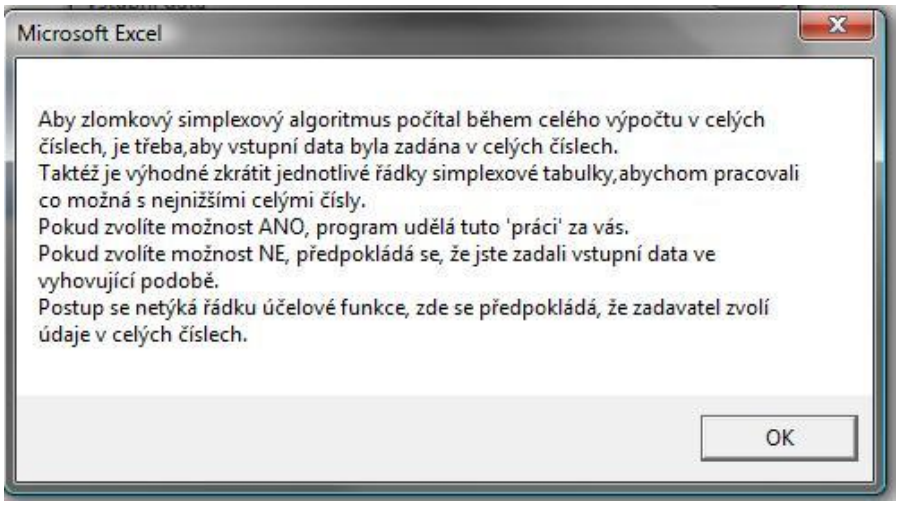
Obrázek 4.10: *Kontrola potvrzení vstupních dat*

Když klikneme na tlačítko „Potvrď data“, ukáže se následující okno:



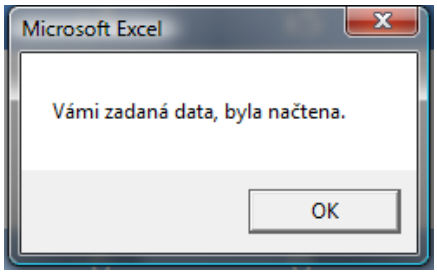
Obrázek 4.11: *Zvolení kontroly vstupních dat*

Pokud zobrazíme nápovědu, vidíme, proč se kontrola provádí:



Obrázek 4.12: *Nápověda ke vstupním datům*

Pokud chceme pracovat s námi zadanými daty, přeskočíme kontrolu možností „NE“.



Obrázek 4.13: *Potvrzení načtení dat bez kontroly*

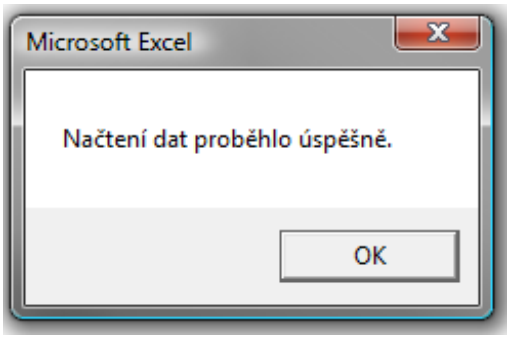
V aplikaci je implementován automatický převod na celá čísla. Řekněme, že uvažujeme úlohu, která má následující výchozí simplexovou tabulku:

det=	1				
	x1	x2	x3	x4	b
x3	0,25	1,5	1	0	10
x4	1,12	0,6	0	1	13
z	-4	-9	0	0	0

Tabulka 4.2: *Zadání dat v desetinných číslech*

První a druhý řádek porušují předpoklady celočíselných prvků v matici *A*. Při potvrzení dat (viz obr. 4.10) vybereme možnost „ANO“. Program postupně vyhodnotí každý řádek zvlášť. V prvním řádku najde číslo s nejdelším desetinným rozvojem a na základě toho vynásobí celý řádek (s výjimkou přídatných proměnných) hodnotou 10^n, kde *n* odpovídá délce desetinného rozvoje. Poté se pomocí implicitní funkce tabulkového kalkulátoru GCD (funkce vrací nejvyšší společný dělitel) pokusí nalézt nejvyšší společný dělitel rozšířeného řádku a tímto tento řádek vydělí. Proč se pokoušíme řádky zkrátit? Zlomkový simplexový algoritmus pracuje s akumulovanou hodnotou klíčových prvků a tato hodnota velmi rychle roste. Pokud se nám podaří zkrátit již úvodní tabulku, při celém výpočtu budeme pracovat s nižšími čísly. Zmíněný postup se provede také pro druhý řádek.

Když je kontrola vstupních dat hotova, dostaneme následující potvrzení:



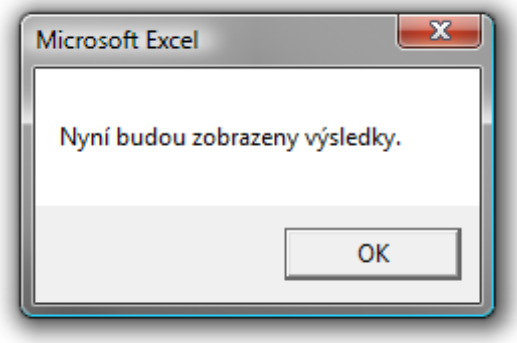
Obrázek 4.14: *Potvrzení kontroly a načtení vstupních dat*

Upravená tabulka bude vypadat následovně:

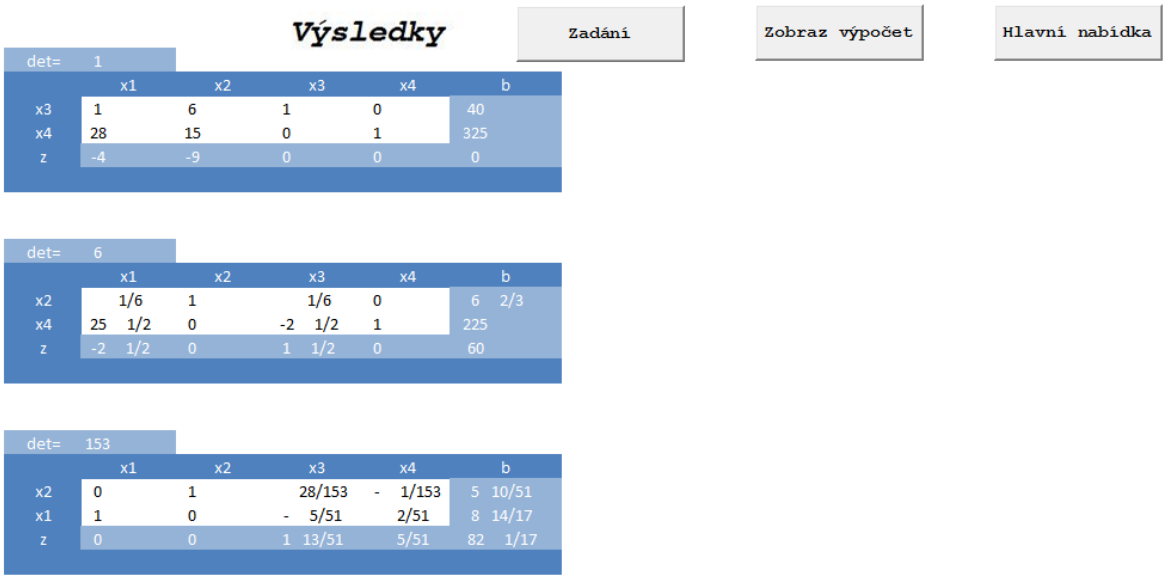
det=	1				
	x1	x2	x3	x4	b
x3	1	6	1	0	40
x4	28	15	0	1	325
z	-4	-9	0	0	0

Tabulka 4.3: *Úloha z tab. 4.2 po převedení na celá čísla a zkrácení*

Když máme potvrzená data, nic nebrání tomu spustit výpočet. Na vstupní kartě klikneme na tlačítko „Počítej“ (viz obr. 4.11). Ukončení výpočtu je potvrzeno následujícím oknem:



Obrázek 4.15: *Potvrzení provedení výpočtu*



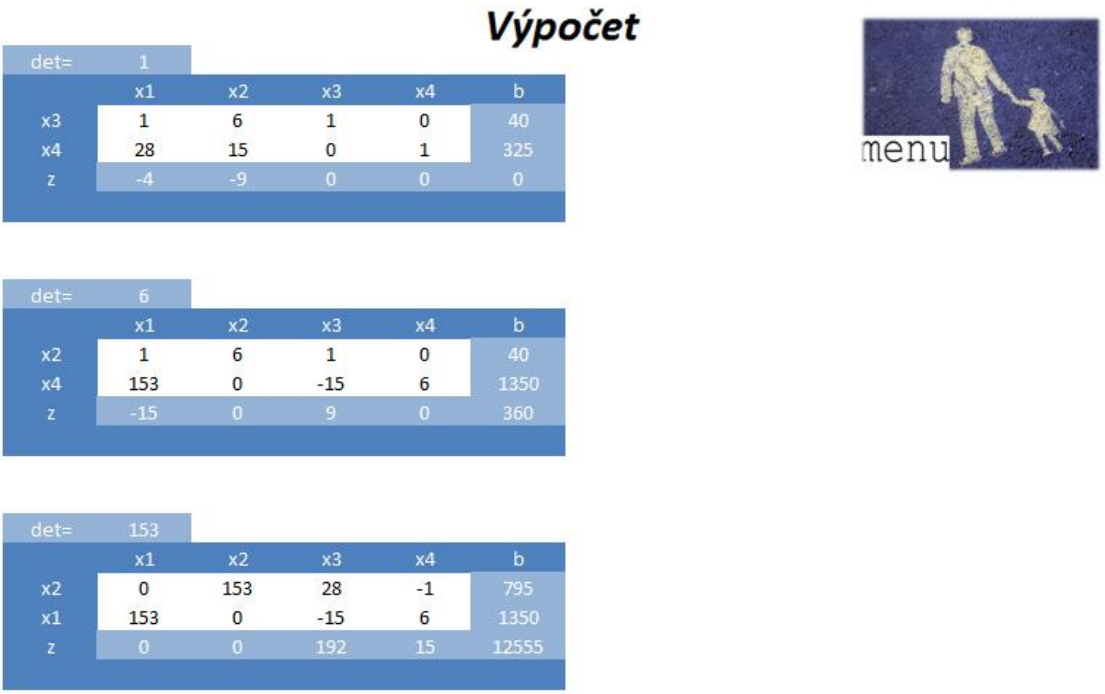
Obrázek 4.16: *Zobrazení karty s výsledky ve zlomcích*

Po spuštění výpočtu dostaneme potvrzení a zobrazí se výsledky ve zlomcích. Nejprve do báze vstoupila nezákladní proměnná x_2, která nahradila přídatnou proměnnou x_3 a v dalším kroku nezákladní proměnná x_1, která nahradila přídatnou proměnnou x_4. Optimálního řešení jsme tedy dosáhli ve druhé iteraci s hodnotou účelové funkce 8 a 7/34.

Vektor x = (8 a 14/17, 5 a 10/51, 0, 0).

Zajímavý pro nás může být také průběh výpočtu. K tomu, abychom jej zobrazili, zvolíme možnost „Zobraz výpočet“.

.



Obrázek 4.17 *Zobrazení karty s průběhem výpočtu*

Pokud chceme námi získaný výpočet uložit, je vhodné si tabulky zkopírovat do jiného souboru, protože při dalším výpočtu jsou dřívější data přepsána.

Poslední možnost, kterou jsme nezmínili, je zobrazení ukázkových příkladů. Na tuto kartu se dostaneme z nabídky „Menu“ kliknutím na možnost „Ukázkové příklady“. Nejprve však na kartě „Vstup“ zadejte příslušný rozměr úlohy. Na kartě ukázkových příkladů jsou k dispozici zadání úloh, na kterých byla aplikace testována, viz následující kapitola.

5 Testování aplikace „Zlomkový simplex“

Aplikace „Zlomkový simplex“ byla testována na mnoha učebnicových příkladech. Kromě toho jsem se však rozhodl vygenerovat několik desítek úloh lineárního programování, které splňují požadavky užití algoritmu a vybraná řešení porovnat s řešením, které poskytne profesionální systém na podporu modelování společnosti Lindo Systems - Lingo 11. Počítač, který byl k testování použit je vybaven dvoujádrovým procesorem Athlon 64 QL65, 3 gb operační paměti a pracuje na 32. bitovém operačním systému Windows Vista Home Premium.

Úlohy vznikly za pomocí generátoru náhodných čísel v Excelu funkce (RANDBETWEEN), která vrací náhodnou hodnotu čísla ze zadaného intervalu hodnot. Znění testovaných úloh spolu s několika ukázkovými příklady, je možné nalézt přímo v aplikaci na kartě „příklady“.

V následující tabulce je uvedena specifikace úloh, na kterých se algoritmus testoval a potřebná doba výpočtu.

Počet nezákladních proměnných	Počet omezení	Počet řešených úloh	Průměrná doba výpočtu	Rozmezí pro hodnoty matice A	Rozmezí pro hodnoty vektoru b	Rozmezí pro hodnoty v řádce z	Označení úloh	Ověření optimality
4	4	10	4,8 sekund	$0\text{-}10^4$	$0\text{-}10^9$	$0\text{-}10^2$	1. - 10.	ano
8	4	10	8,6 sekund	$0\text{-}10^4$	$0\text{-}10^9$	$0\text{-}10^3$	11.-20.	ano
8	8	5	10,4 sekund	$0\text{-}10^4$	$0\text{-}10^9$	$0\text{-}10^3$	21.-25.	ano
10	10	5	18 sekund	$0\text{-}10^4$	$0\text{-}10^{10}$	$0\text{-}10^3$	26.-30.	ano
20	20	5	93,2 sekund	$0\text{-}10^4$	$0\text{-}10^9$	$0\text{-}10^3$	31.-35.	ano
150	50	1	2682 sekund	$0\text{-}10^4$	$0\text{-}10^9$	$0\text{-}10^3$	36.	ano

Tabulka 5.1 *Specifikace testovaných úloh*

Algoritmus byl testován celkem na 31. úlohách, které byly všechny úspěšně vyřešeny. Ukázalo se, že zvolená implementace není příliš vhodná pro úlohy většího rozsahu a vůči profesionálnímu řešiteli, který je obsažen v Lingu, nemůže, co se týče rychlosti výpočtu, konkurovat. Největší řešená úloha obsahovala 150 základních proměnných a 50 omezení a její řešení trvalo přibližně 45 minut. Vzhledem k této skutečnosti, se zdá být prostředí, které tabulkový kalkulátor nabízí dostatečné, protože jak bylo zmíněno ve první kapitole simplexový algoritmus má tendenci vykazovat polynomiální řešitelnost a tak automaticky vytvářené tabulky nepřesáhnou možnosti tabulkového kalkulátoru. Ukázalo se, že aplikace je

vhodná pouze pro úlohy menšího rozsahu. V programu je implementován nástroj pro řešení úloh, které mají všechna omezení typu <= postup by tedy bylo možné rozšířit i o další postupy, které vycházejí ze simplexové metody.

Během testování se ukázalo, že je třeba změnit typ proměnných z „integer“ na „long“ a také pozměnit parametr *t*, který slouží jako strop pro podíl pravých stran. Za hlavní úspěch považuji, že všechny řešené úlohy se podařilo úspěšně vyřešit a optimální hodnoty odpovídaly hodnotám, které poskytl řešitel z Linga.

Závěr

Na základě teorie lineárního programování a maticového počtu se podařilo odvodit a algoritmizovat postup simplexové metody, který po celou dobu výpočtu pracuje ve zlomkové aritmetice. Navržený postup je schopen řešit všechny úlohy lineárního programování, které jsou zadány v celých číslech. Hlavní předností je, že eliminujeme riziko vzniku zaokrouhlovacích chyb.

Pokud navíc splníme požadavek totální unimodularity matice strukturních koeficientů, můžeme takto počítat i úlohy celočíselného programování. V práci bylo naznačeno, že tento požadavek se dá ověřit v polynomiálním čase.

Slabinou navrženého postupu je, že v případě velkého počtu iterací roste rychle hodnota jmenovatele, kterým je celá tabulka násobena. Jedním z možných řešení je pokusit se zkrátit jednotlivé řádky simplexové tabulky.

Podařilo se naprogramovat aplikaci, která využívá obecně známého prostředí tabulkového kalkulátoru MS Excel 2007 a nabízí možnost řešení zmíněných úloh lineárního programování, včetně automatické tvorby tabulek a možnosti sledování průběhu výpočtu. Aplikace je navržena především k demonstrativním účelům, proto je celý výpočet poměrně pracným způsobem vyjádřen v grafickém prostředí. Autor si nekladl za cíl, aby konkuroval profesionálním řešitelským systémům, proto aplikace „Zlomkový simplex“ byla navržena tak, aby bezproblémově vyřešila menší úlohy lineárního programování (přibližně okolo 20-ti nezákladních proměnných a 20-ti omezení) a sloužila především pro studijní účely.

Uvědomění si souvislostí maticového počtu, konkrétně věty o inverzní matici (1.2), a teorie lineárního programování prohlubuje porozumění způsobu práce simplexové metody. Touhou autora je, aby k těmto účelům práce posloužila.

Literatura

[1] BENÁČANOVÁ, H.: *Tvorba aplikací v MS EXCEL: materiály ke cvičení.* Praha : Oeconomica, 2005. ISBN 80-245-0953-9

[2] Coufal, J., Klůfa, J.: *Matematika pro ekonomické fakulty*. Praha: Ekopress, 2000. ISBN 80-86119-30-0.

[3] ČERNÝ, J. *Programování v Excelu 2000, 2002, 2003.* Praha : Grada, 2005. ISBN 80-247-0922-8.

[5] FIALA, P.: *Řízení projektů. Praha: Oeconomica.* 2007. ISBN 80-245-0448-0.

[4] Dantzig, G. B.: *Linear Programming and Extensions.* United Kingdom: Princeton University Press: 1998. ISBN 978-0-691-05913-6. [cit. 2010-03-02].

[6] JABLONSKÝ, J.: *Programy pro matematické modelování.* Praha: Oeconomica, 2007. ISBN 978-80-245-1178-8.

[7] LAGOVÁ, M., JABLONSKÝ, J.: *Lineární modely.* Praha : Oeconomica, 2009. ISBN 978-80-245-1511-3.

[8] PELIKÁN, J.: *Diskrétní modely v operacním výzkumu.* Praha: Professional Publishing, 2001. ISBN 80-86419-17-7.

[9] SALKIN, H., MATHUR K.: *Foundations of integer programming.* New York; Amsterdam; London : North-Holland, 1989. ISBN 0-444-01231-1.

[11] STINSON, C., DODGE M.: *Mistrovství v Microsoft Office Excel 2003.* Brno: CP Books, 2005. ISBN 80-251-0669-1.

[10] SCHRIJVER, A.: *Theory of linear integer programming.* London, 1999. ISBN 0-471-98232-6. [cit. 2010-03-04].

[13] TURZÍK D-. *Matematika III: základy optimalizace [online].* Version 1.0. Praha : VŠCHT Praha, 1999 (2006 dotisk) [cit. 2010-03-02]. P. 061. Dostupné z www: <http://vydavatelstvi.vscht.cz/knihy/uid_isbn-80-7080-363-0/pages-img/061.html>. ISBN 80-7080-363-019-4.

[12] ZIMMERMAN, U.: *On Recent Development in Linear Programming.* V HOFFMANN, K., HIRIART-URRUTY, J., ZOWE, J., LEMARECGAL, C. (ed.). *Trends in Mathematical Optimization.* 1988. ISBN 0-8176-1919-4.

6 Přílohy

6.1 Zlomkový simplexový algoritmus naprogramovaný ve VBA

```
Sub simplex_pocitej()

Dim m, n, zac, det, zmin, k, l, zj, aik, alk, aij, alj, As Long

det = 1 'vyjadruje hodnotu determinantu v prvnim kroku

zac = 4 'pomocna promena, ktera slouzi pro spravne nacteni vstupnich
dat

m = Sheets("Vypocet").Range("a1:m6").Cells(3, 3) 'promenna
vyjadrujici pocet omezeni

n = Sheets("Vypocet").Range("a1:m6").Cells(3, 7) 'promenna
vyjadrujici pocet promennych


iterace = 1

Sheets("Vypocet").Range("a1:m6").Cells(2, 7) = iterace


Do

'test optima

    Sheets("Vypocet").Range("a1:m6").Cells(zac, 1) = "det="

    Sheets("Vypocet").Range("a1:m6").Cells(zac, 2) = det

    k = 0

    zmin = 0

        For j = 1 To n

        zj = Sheets("Vypocet").Range("a1:m6").Cells(zac + m + 1, j)
'algoritmus v prvni fazi prochazi hodnotu ucelove funkce a zkouma,
jestli neni v optimu

        If zj < zmin Then zmin = zj: k = j ' hledame klicovy sloupec
pro pokracovani vypoctu

    Next j


If zmin = 0 Then Exit Do  'pokud je minimalni hodnota ucelove funkce
nula, algoritmus konci
```

```
' volba klíč. řádku

l = 0

t = 1000000 'vstupni hodnota podilu pravych stran - slouzi jako
strop

    For i = 1 To m

        bi = Sheets("Vypocet").Range("a1:m6").Cells(zac + i, n + 1)

        aik = Sheets("Vypocet").Range("a1:m6").Cells(zac + i, k)

        If aik > 0 Then 'jednofazova simplexova metoda pocita pouze
s kladnymi pravymi stranami

        podil = bi / aik  'vypocet klicoveho prvku na zaklade pomeru
pravych stran

        If podil < t Then t = podil: l = i 'vystupem je minimalni
hodnota pomeru prave strany - tj. nasli jsme klicovy prvek

        End If

    Next i

If l = 0 Then Exit Do 'pokud zadny prvek nevyhovuje reseni - vypocet
konci

Debug.Print "k=" & k & "l=" & l

 Sheets("Vypocet").Range("a1:m6").Cells(zac + m + 5, n + 2) = k

 Sheets("Vypocet").Range("a1:m6").Cells(zac + m + 5, n + 3) = l

'eliminace

alk = Sheets("Vypocet").Range("a1:m6").Cells(zac + l, k)

For i = 1 To m + 1 'prepocet se tyka vsech radku simplexove tabulky
vcetne radku ucelove fce

    aik = Sheets("Vypocet").Range("a1:m6").Cells(zac + i, k)

    If i <> l Then 'And aik <> 0 Then 'podminka pro prepocet

         For j = 1 To n + 1 'prepocet je treba opakovat pro vsechny
sloupce vcetne pravych stran

            aij = Sheets("Vypocet").Range("a1:m6").Cells(zac + i, j)

            alj = Sheets("Vypocet").Range("a1:m6").Cells(zac + l, j)
```

```
            aij = (aij * alk - alj * aik) / det ' vzorec pro
prepocet
            Sheets("Vypocet").Range("a1:m6").Cells(zac + m + 5 + i,
j) = aij
        Next j
        End If
    Next i

    For j = 1 To n + 1
        Sheets("Vypocet").Range("a1:m6").Cells(zac + m + 5 + l, j) =
Sheets("Vypocet").Range("a1:m6").Cells(zac + l, j)
    Next j
det = alk
zac = zac + m + 5
iterace = iterace + 1
Sheets("Vypocet").Range("a1:m6").Cells(2, 7) = iterace
Loop
Debug.Print "konec"
End Sub
```

6.2 Příloha cd s aplikací „Zlomkový simplex“ pro MS Excel 2007