VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING

INSTITUTE OF MATHEMATICS

MODELOVACÍ JAZYKY V OPTIMALIZACI

OPTIMIZATION MODELING LANGUAGES

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE EVA MOLLIKOVÁ
AUTHOR

VEDOUCÍ PRÁCE RNDR. PAVEL POPELA, PH.D.
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce je koncipována jako úvod do problematiky používání programovacího jazyka GAMS, který je nástrojem pro tvorbu optimalizačních modelů.

Úvodem práce je uveden přehled matematických pojmů a jejich vlastností, které jsou využívány v programových nástrojích a při řešení později popsaných modelů..

Dále jsou rozebrány a vyřešeny celkem čtyři příklady. Na prvním jsou podrobně vysvětleny základní pojmy, příkazy a postupy jazyka GAMS, včetně vyhodnocení výsledků výpočtu. Druhý příklad pak ilustruje vývoj zápisu modelu od zcela konkrétního a intuitivního až po obecný, v němž lze snadno zaměnit vstupní data.

V závěru práce jsou pak řešeny dva příklady, které ukazují další možné využití jazyka GAMS a které mohou motivovat čtenáře k dalšímu studiu a následně k aplikacím optimalizace v jeho oboru.

Klíčová slova:

Optimalizace, matematické programování, úloha lineárního programování, optimalizační software, GAMS

Abstract

The goal of the thesis is to introduce readers into the use of the GAMS modelling language that is an important tool for optimization model building. The overview of useful mathematical concepts introduces the text. The main part of thesis contains four examples presented in detail. Firstly, the basic GAMS related concepts, statements and techniques involving analysis of results are described. The second example illustrates development of model description beginning from quite specific instance towards general and flexible one. Two examples solved at the end show further use of the GAMS and can motivate reader to enhance their konwledge and apply optimization in his areas of specialization.

Keywords:

Optimization, mathematical programing, linear program, optimization software, GAMS

Bibliografická citace:

MOLLIKOVÁ, E. *Modelovací jazyky v optimalizaci*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010. 54 s. Vedoucí bakalářské práce RNDr. Pavel Popela, Ph.D.

Prohlášení

Prohlašuji, že jsem tuto práci vypracovala samostatně, pouze za odborného vedení RNDr. Pavla Popely, Ph.D.. Dále prohlašuji, že veškeré podklady, které jsem k vypracování použila a ze kterých jsem čerpala, jsou uvedeny vseznamu literatury.

V Brně dne 27. května 2010

Eva Molliková

Poděkování

Ráda bych poděkovala RNDr. Pavlu Popelovi, Ph.D. za příkladné vedení při psaní práce, pomoc při shánění materiálů a podkladů a za důslednou kontrolu celého textu.

Obsah

1 Úvod

Lidstvo od nepaměti hledá cesty, jak si ulehčit život, jak s vynaložením co nejmenší práce získat co největší výnosy v zemědělství, těžbě dřeva, ve výrobě kovů, směně zboží se vzdálenými kulturními a obchodními centry, ... V současnosti pak jde většinou o co největší zisk s vynaložením co nejmenších nákladů a za co nejkratší dobu.

Všechny tyto procesy směřují k jedinému – jak optimalizovat svou činnost s ohledem na požadované výsledky?

Když pralidé byli ještě lovci a sběrači, většinou někam přišli, snědli co našli a když byly zásoby potravy vyčerpané, šli dál. Postupně ale zjistili, že aby se mohli více rozvíjet a zakládat větší společenstva, musí se usadit. A s tím přišel problém – jak vystačit s omezenými zásobami potravin v přírodě? Člověka tehdy napadlo, že si přírodu může přizpůsobit – naučil se pěstovat rostliny a chovat zvířata a tím výrazně zvýšil nejen své šance na přežití tuhé zimy, ale také na rozvoj svého společenství. Dá se říct, že člověk optimalizoval své životní podmínky s cílem zvýšit množství dostupné potravy.

Později člověk vymyslel různé nástroje a zjistil, jak se připravuje který kov a že železo a ocel je lepší než kámen a bronz a postupně se naučil, jak z rudy získat kvalitnější a kujnější ocel. Optimalizoval výrobu oceli s cílem zvýšit kvalitu výrobku.

Ve starověku se rozproudil čilý obchod mezi národy, které dělily velké vzdálenosti, a přesto si měly vzájemně co nabídnout. Číňané posílali do Evropy hedvábí a koření a odváželi si drahé kovy a jiné cenné suroviny. Stejně tak pobřežní oblasti měly dostatek ryb a měnily je za dřevo či kožešiny. Severané dokázali vyrábět ze zlata či jiných kovů propracované umělecké předměty a jižané zase měli na dosah exotické druhy dřev a zvířat. Každý obchodník proto neustále počítal, jestli se mu vyplatí dovézt jednu surovinu odtamtud nebo jinou odjinud a kolik za ni na místním trhu získá zlata - optimalizoval nákup a prodej zboží s cílem maximalizovat zisk.

Úplně stejný systém funguje i dnes – firmy neustále hledají dodavatele, kteří nabídnou nižší ceny vstupních surovin, a trhy, které budou ochotny za výsledný výrobek zaplatit víc.

A nejde jen o peníze – cílů optimalizačních úloh může být mnoho, například „Z jakých rud vyrobit výslednou slitinu s požadovanými vlastnostmi, když se rudy liší složením a cenou?“ nebo „Kdy a jak zásobit sklad a kdy a kolik zboží prodat, abychom maximalizovali zisk?“ [1] (tato úloha bude podrobněji probrána v kapitole 6.3).

Toto je jen několik příkladů, kde všude najde optimalizace uplatnění. Základní roli zde hrají optimalizační matematické modely, o kterých uvedu základní informace v kapitole 2.

Ve své bakalářské práci se tedy zabývám problematikou počítačové podpory optimalizačního modelování, která je v oboru matematického inženýrství probírána až v magisterském studiu. S problematikou jsem se seznámila ve volitelném předmětu Optimalizační modely na podzim roku 2009.

Tato práce nemá být další z řady odborných publikací s využitím počítačové podpory optimalizace s pomocí modelovacího jazyka GAMS, ale má být jednoduchým a čtivým textem, který uživateli bez předešlých zkušeností s tímto programovacím jazykem nejprve nabídne snadné a zejména rychlé zorientování se při zahájení jeho používání.

Cílem mé práce bylo vytvořit text, který by pomohl modelovat a řešit optimalizační

úlohy v prostředí modelovacího jazyka. Zaměřen je podle zadání práce na nematematiky, kteří chtějí optimalizaci aplikovat, nebo ji již používají, ale nemají žádné zkušenosti s optimalizačními modelovacími jazyky. Cílem tedy je umožnit zájemci co nejrychlejší využívání základních možností jazyka GAMS pomocí jednotlivých vybraných příkladů.

2 Optimalizační modely

Matematická optimalizace navazuje na problematiku řešení úloh volných a vázaných extrémů v matematické analýze a využívá poznatky lineární algebry. V této krátké úvodní části shrnu některé pojmy z uvedených oblastí, které mohou čtenáři tohoto textu pomoci při chápání významu jednotlivých prvků jazyka. Při sestavování přehledu jsem vycházela z [5].

2.1 Úloha matematického programování

Nejprve uvedeme definici obecné úlohy matematického programování (matematické optimalizace):

Nechť $S \subset \mathbb{R}^n$ a $f : \mathbb{R}^n \to \mathbb{R}$. Pak je úloha matematického programování definována takto: $\min_x \{f(x) | x \in S\}$.

2.2 Úloha nelineárního programování

Jedním z typů úloh matematického programování, které se řeší pomocí modelovacích jazyků je úloha nelineárného programování:

Nechť $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ a $X \subset \mathbb{R}^n$. Dále pak nechť $\circ \in \{\leq, \geq, =\}^m$. Pak $\min_x \{f(x) | g(x) \circ 0, x \in X\}$ nazveme úlohou nelineárního programování.

2.3 Uspořádání na množině $\mathbb{R}$ a jeho vlastnosti

Zavedli jsme syntakticky korektní zápis modelů optimalizačních úloh. Nyní se budeme zabývat jejich sémantikou tj. významem použité symboliky. Proto připomeneme definici uspořádání na množině $\mathbb{R}$ a jeho vlastnosti:

Dvojice $(\mathbb{R}, \leq)$ představuje (úplné) uspořádání reálných čísel, protože existuje binární relace $\leq \subset \mathbb{R} \times \mathbb{R}$, která splňuje následující tři požadavky (čtvrtý zaručuje úplnost):

- $\forall a \in \mathbb{R} : a \leq a$
- $\forall a, b \in \mathbb{R} : (a \leq b) \wedge (b \leq a) \Rightarrow a = b$
- $\forall a, b, c \in \mathbb{R} : (a \leq b) \wedge (b \leq c) \Rightarrow a \leq c$
- Navíc musí platit $\forall a, b, c \in \mathbb{R} : (a \leq b) \vee (b \leq a)$

2.4 Maximum, minimum, infimum a supremum množiny

Nyní využijeme připomenuté pojmy k definicím minima, maxima, infima a suprema podmnožiny množiny reálných čísel:

Nechť $B \subset \mathbb{R}$, $\mathbb{R}^* = \mathbb{R} \cup \{-\infty, +\infty\}$, kde $-\infty$ a ∞ jsou prvky, pro něž platí $\forall a \in \mathbb{R}: -\infty \leq a \leq +\infty$. Pak definujeme

- $B_{\min} = \min B \Leftrightarrow (b_{\min} \in B) \wedge (\forall b \in B : b_{\min} \leq b)$,
- $B_{\max} = \max B \Leftrightarrow (b_{\max} \in B) \wedge (\forall b \in B : b_{\max} \geq b)$,
- $\inf B = \max\{c | (c \in \mathbb{R}^*) \wedge (\forall b \in B : c \leq b)\}$,
- $\sup B = \min\{c | (c \in \mathbb{R}^*) \wedge (\forall b \in B : c \geq b)\}$

Definiční obor funkce f označme Dom f a obor hodnot funkce f označme Im f. Pak sémantický význam zápisu úlohy matematického programování v Definici 1 je následující: $\min_x \{f(x) | x \in S\} = \min \mathrm{Im}\, f$

2.5 Minimum funkce

Nyní doplňme hledání minimální hodnoty účelové funkce o zavedení pojmu minima tj. bodu, ve kterém funkce nabývá minimální hodnoty.

Nechť $S \subset \mathbb{R}^n$ a $f : S \to \mathbb{R}$. Pak definujme $x_{\min} \in S$ jako:

- lokální minimum, pokud platí $\exists N_\varepsilon(x_{\min}) : \forall x \in S \cap N_\varepsilon(x_{\min}) : f(x_{\min}) \leq f(x)$
- globální minimum, pokud platí $\forall x \in S : f(x_{\min}) \leq f(x)$
- doplňujeme, že se jedná o ostré minimum, pokud platí $f(x_{\min}) < f(x)$
- a neostré minimum, pokud platí $f(x_{\min}) \leq f(x)$

2.6 Weierstrassova věta

V aplikačních úlohách je důležité ověřit, zda jsou splněny podmínky, za kterých minimum existuje. Lze využít následující větu, protože v aplikačních úlohách lze zaručit její předpoklady, tj. uzavřenost množiny S použitím neostrých nerovnic a rovnic, omezenost, a tedy celkově kompaktnost, množiny S použitím mezí pro proměnné a spojitost funkce vhodným modelováním:

Nechť $S \subset \mathbb{R}^n$, $S \neq \emptyset$, S je kompaktní množina a funkce $f : S \to \mathbb{R}$ je na S spojitá. Pak úloha matematického programování $\min_x \{f(x) | x \in S\}$ má na množině S globální minimum, značené $x_{\min}$.

2.7 Konvexní množina

Dále je vhodné umět identifikovat případy, kdy lokální minimum je rovněž minimem globálním. Nejprve zavedeme konvexní množiny a konvexní funkce, pak uvedeme potřebnou větu:

Nechť $S \subset \mathbb{R}^n$. Pak řekneme, že S je konvexní množina právě tehdy, když platí $\forall x_1, x_2 \in S, \forall \lambda \in [0;1]: \lambda x_1 + (1-\lambda)x_2 \in S$.

2.8 Konvexní funkce

Nechť $S \subset \mathbb{R}^n$, $S \neq \emptyset$, S je konvexní množina a f je zobrazení $f : S \to \mathbb{R}$.

Funkci f nazveme konvexní na $S \Leftrightarrow \forall x_1, x_2 \in S, \forall \lambda \in (0,1) : f(\lambda x_1 + (1-\lambda) f(x_2)) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$

Funkci f nazveme ryze konvexní na $S \Leftrightarrow$

$\forall x_1, x_2 \in S, x_1 \neq x_2, \forall \lambda \in (0,1) : f(\lambda x_1 + (1-\lambda) f(x_2)) < \lambda f(x_1) + (1-\lambda) f(x_2)$

2.9 Konvexnost a globální minimum

Nechť $S \subset \mathbb{R}^n$, $S \neq \emptyset$, S je konvexní množina a $f : S \to \mathbb{R}$ je na S konvexní funkce. Nechť $\bar{x} \in \arg loc \min_x \{f(x) | x \in S\}$, tedy předpokládáme, že. je lokální minimum, pak platí, že $\bar{x} \in \arg glob \min_x \{f(x) | x \in S\}$, a tedy je globálním minimem.

2.10: Úloha lineárního programování

Důležitou roli v našich příkladech hrají úlohy lineárního programování.

Speciálním případem úlohy matematického programování je úloha lineárního programování (ve standardním tvaru) $\min_x \{c^T x | Ax = b, x \geq 0\}$, kde *c* a *b* jsou vektory a *A* je matice.

2.11 Základní věta lineárního programování

(viz [4], kde lze najít informace o dále použitých pojmech)

Mějme úlohu lineárního programování ve standardním tvaru (viz Pozn. 10) $h(A) = m$, $S \neq \{\}$. Nechť $x_1, ..., x_k$ jsou krajní body S a $d_1, ..., d_l$ jsou všechny krajní směry v S.

Pak existuje optimální řešení $x_{\min} \in S \Leftrightarrow \forall j = 1, ..., k : c^T d_j \geq 0$. Navíc, jestliže existují optimální řešení, pak mezi nimi existuje alespoň jeden krajní bod.

Věta říká, že úloha LP, která má přípustná řešení, má optimální řešení (dokonce

v krajním bodě), právě tehdy, když neexistuje krajní směr, který je směrem poklesu (spádovým směrem) hodnot účelové funkce.

Uvedená věta umožňuje efektivně hledat řešení úlohy lineárního programování, protože krajních bodů a krajních směrů je konečný počet. Protože množina S je u úlohy lineárního programování polyedrická (v našich aplikacích mnohostěn), a tedy konvexní a lineární, kriteriální funkce je konvexní, následující lokálně prohledávající algoritmus podle věty v bodu 2.9 najde globální minimum:

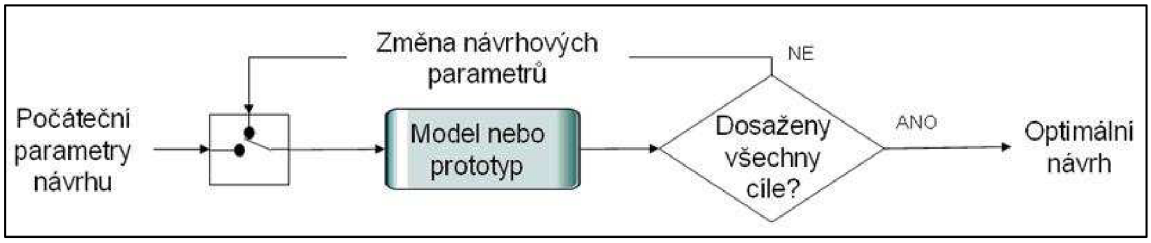
2.12 Geometrická idea simplexové metody - algoritmus

Nalezneme počáteční řešení – první krajní bod. Najdeme hranu vycházející z tohoto bodu, která je určena směrem poklesu (spádovým směrem). Postupujeme po dané hraně, až dosáhneme sousedního krajního bodu. Postup opakujeme. Procedura končí, když nastane některá z možností:

1. Nepodaří se najít počáteční krajní bod. Pak úloha nemá řešení.
2. Neexistuje hrana, podél které by bylo možné dále snížit hodnotu účelové funkce. Pak jsme řešení právě nalezli.
3. Hrana je určena krajním směrem, který je zároveň směrem poklesu (spádovým směrem) a hodnotu účelové funkce lze v daném směru libovolně snižovat.

3 Optimalizační software

Mnohdy se setkáme s úlohami, které jsou velmi komplexní a složité (obchodní vztahy několika velkých korporací a jejich subdodavatelů, celosvětový letecký provoz, hromadná doprava ve velkoměstech, ...), viz např. [6]. Řešení takových optimalizačních problémů vyžaduje použití speciálních programů - optimalizačních algoritmů, využívajích speciální poznatky matematické analýzy a numerické matematiky (viz kapitola 2)., a které se v celém vývojovém cyklu optimalizačního modelu řídí následujícím schématem:



Obr. 1: *Proces optimalizace*

Komplexnost reálných úloh nás vede k závěru, že vytváření vlastních optimalizačních programů nelze běžnému uživateli optimalizačních modelů pro velkou časovou náročnost a finanční nákladnost vždy doporučit. Naštěstí existuje široká škála nejrůznějších programovacích nástrojů, ze kterých si mohou uživatelé vybrat, viz např [3].

Tyto nástroje lze rozdělit do několik kategorií:

- Tabulkové procesory
- Maticově orientované systémy
- Symbolicky orientované systémy
- Inženýrské systémy

Tabulkové procesory (např. MS Excel pod MS Windows) jsou populární v různých oborech, ale jejich vybavení pro optimalizaci nemůže konkurovat specializovaným programům. Používají vyzkoušené procedury profesionálních autorů, takže nejsou problémy s jejich spolehlivostí, ale velikost řešitelných úloh bývá značně omezena a tabulkový vstup dat je nevhodný pro rutinní využití v inženýrské praxi, charakterizované častými změnami dat v rozsáhlých úlohách s řídkými maticemi. Stále častěji se používají jako rozhraní pro zadávání vstupů a zobrazování výsledků.

Typickým **maticově orientovaným programovacím jazykem** je MATLAB se svými rozsáhlými knihovnami numerických procedur. Díky své kompaktní syntaxi je vhodným nástrojem pro testování a vývoj algoritmů, který může pomoci při studiu jejich vlastností. MATLAB Optimization toolbox rovněž obsahuje, kromě standardních optimalizačních procedur, také procedury vícekriteriální optimalizace. Při jeho běžném používání v aplikacích bývá nutné doplnit vlastní procedury pro vstup a výstup a testovat algoritmy při řešení rozsáhlých úloh s řídkou strukturou dat.

Mezi **symbolicky orientované systémy** patří např. MAPLE či MATHEMATICA, které zahrnují symbolické operace, včetně derivování a integrování. Jejich optimalizační procedury jsou určeny pro řešení menších úloh se složenými funkcemi, vyskytujícími se v omezeních i samotné účelové funkci. Mohou také poskytovat výkonné symbolické derivační nástroje pro algoritmy, které dříve počítaly derivace numericky.

Řada **inženýrských úloh** může být řešena pomocí optimalizačních procedur. Ty jsou obvykle integrovány ve speciálním tvaru do složitých systémů (např. ANSYS). Tyto systémy představují kvalitativní přínos pro řešení optimalizačních úloh, ale vyžadují důkladné matematické znalosti, aby byly získané výsledky právně interpretovány.

Dalším krokem ve vývoji optimalizačních programů je využití částečně či úplně předkompilovaných procedur – řešičů. Uživatel je pak využívá, aniž by byl nucen znát implementační detaily použitého programovacího jazyka a může se soustředit na jejich vstup a výstup. Mezi nejznámější řešiče patří:

- LINDO, což je řešič lineárního programování a bývá napojován zejména na tabulkové procesory
- BDMLP, který používá simplexovou metodu (viz kap. 2, def. 12) a je základním řešičem lineárního programování v jazyce GAMS
- LOQO je alternativou k výše zmíněnému BDMPL, protože k řešení úloh lineárního programování využívá metodu vnitřního bodu.
- CPLEX, který účinně řeší rozsáhlé úlohy lineárního a lineárního celočíselného programování, včetně síťové struktury problému.

K řešení optimalizačních úloh jsou v praxi nezbytné nástroje, umožňující provádět snadné změny vstupních dat optimalizačních programů. Tento problém byl v minulosti řešen několika možnými přístupy:

Napřed byly vyvinuty speciální maticové generátory. Program pak generoval datové struktury podle použitých příkazů jazyka ze dvourozměrných datových tabulek. Tyto generátory ale vyžadovaly značnou programátorskou disciplínu a byly zaměřeny na operace se sloupci matic.

Další generaci jazyků pro popis optimalizačních úloh představují blokově orientované jazyky pro úlohy lineárního programování, kde je problém popsán pomocí blokových diagramů, které kombinují řádky (omezení) a sloupce (proměnné) do bloků označených vhodnou ikonou.

Další pokrok přinesl vývoj algebraických modelovacích jazyků. Jedná se deklarativní jazyky speciálního určení, které optimalizační problém popisují pomocí množiny omezení tak, jak je zvykem v matematice a jejich překladače generují vstupy pro řešiče. První modelovací jazyk MGG vznikl již v roce 1969. Následující jazyky však stále neumožňovaly obecný popis problému, ale jen konkrétního případu. Teprve další jazyky, jako např. GAMS či MODLER, přinesly potřebná rozšíření.

Modelovací jazyk GAMS se od té doby stal světovým standardem ve výzkumu, aplikacích i výuce. Byl navržen zejména pro úlohy lineárního, nelineárního a míšeného celočíselného programování, pro které nabízí výběr ze široké škály řešičů.

Modelovací jazyky vzniklé v 90. letech minulého století, jako např. AIMMS, umožňují snadné oddělení obecné specifikace modelu od konkrétních dat. Zahrnují navíc speciální prvky pro síťové úlohy, a také nabízejí interaktivní prostředí příkazového řádku. Při jejich vývoji se zdokonalila práce se složitými indexovými množinami pomocí objektů.

Výzkum se dále zabýval otázkami rozšíření vlastností modelovacích jazyků pro řešení složitých optimalizačních úloh (stochastické programování, vícestupňové úlohy, ...), vzájemné kompatibility kompilátorů (AIMMS čte kód GAMSu) a vývojem různých doplňujících nástrojů.

Současné optimalizační systémy generují kromě textového výstupu, také grafické přehledy. Grafické uživatelské rozhraní tak umožňuje uživateli být zcela oddělen od vlastního modelu a pouze editovat tabulky vstupních dat, používat předprogramovaná tlačítka a studovat výstupní okna, viz [3].

Z výše zmíněných důvodů jsem si pro tvorbu této práce vybrala programovací jazyk a prostředí GAMS. Osobně si myslím, že k pochopení celkového přístupu k optimalizaci a jejímu modelování napomohou požadavky na přesný a srozumitelný zápis v GAMSu, které bych shrnula „co si nenapíšeme, to nemáme“ (samozřejmě, pokud nevyužijeme rozsáhlé knihovny předprogramovaných modelů).

Navíc, GAMS je na VUT v Brně (a i ve světě) často používaným optimalizačním nástrojem, takže studenti VUT mají široké možnosti uplatnit nabyté znalosti a především prohlubovat své vědomosti v takové oblasti využití optimalizace, která je jim oborově blízká.

3.1 O GAMSu

GAMS (General Algebraic Modeling System) je velmi pokročilý modelovací systém, kterým lze formulovat matematické modely, zejména optimalizační, pomocí výstižných algebraických výrazů a programových příkazů srozumitelných jak programátorům, tak lidem zabývajícím se návrhem matematických modelů. Nebude však činit potíže ani praktikům s pouze částečnou znalostí obojího. Skládá se z kompilátoru jazyka a sady integrovaných výkonných řešičů. GAMS je určen pro komplexní a rozsáhlé modelovací aplikace a umožňuje vytvářet obsáhlé modely, které mohou být rychle přizpůsobeny novým podmínkám a situacím. Uživatel může při úpravách měnit formulace, vybírat různé řešiče a dokonce přecházet z modelu lineárního programování do modelu nelineárního programování a to jen s velmi malými souvisejícími změnami.

Jednoduché prostředí umožňuje uživateli se soustředit na samotný problém modelování tím, že si systém sám hlídá detaily v nastavení konkrétního počítače a implementace softwaru, které by mohly zvýšit časovou náročnost výpočtů. Je uzpůsoben k řešení úloh převážně lineárního, nelineárního a celočíselného programování. Podle rozsahu a složitosti modelu je možné GAMS spustit na osobních počítačích, pracovních stanicích, sálových počítačích či super-počítačích.

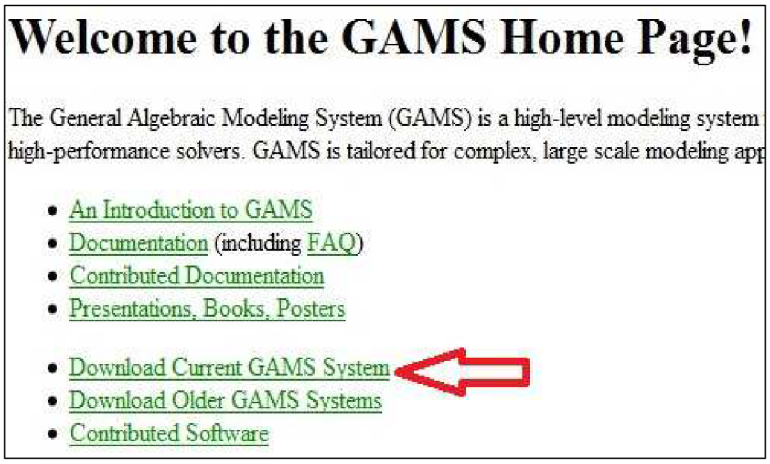
Jazyk GAMS se podobá mnoha známým programovacím jazykům. Vyžaduje stručnou a exaktní specifikaci prvků a vztahů, což napomáhá k vytváření správných modelovacích návyků. Modely jsou plně přenosné z jednoho počítače na druhý, viz [2]

3.2 Kde program získat?

Úplným základem práce je obstarat si samotný GAMS. Ten lze stáhnout zcela legálně ze stránek **www.gams.com**. Zde pak pro získání nejnovější verze programu stačí kliknout na „Download Current GAMS System".

Dále se pak dostaneme k výběru konkrétní verze, podle toho, jaký operační systém na konkrétním počítači máme.

GAMS je možné instalovat na Windows (32 bit, 64 bit – 7, Vista, XP, Server 2008, Server 2003) nebo Unix (AIX, Digital UNIX, HP-UNIX, Linux 32 a 64 bit, IRIX, Mac OS X Intel 32 a 64 bit, Mac OS X PPC, Solaris SPARC 32 a 64 bit, Solaris x64 64 bit), viz [2]

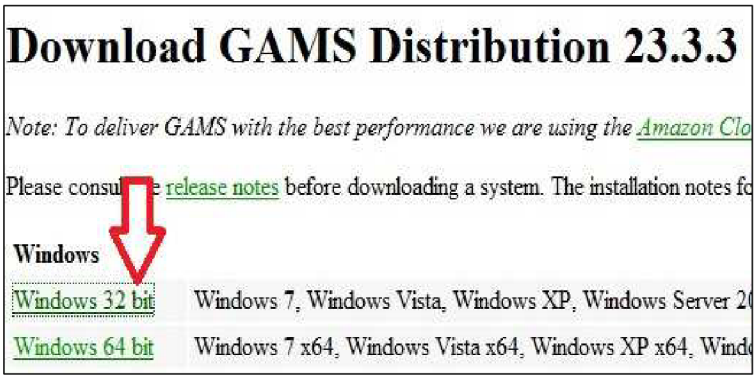


Obr. 2: *Jak stáhnout GAMS*

3.3 Jak program nainstalovat?

Po kliknutí na „Download Current GAMS System" si v zobrazené nabídce vybereme právě tu verzi, která se hodí pro náš OS.

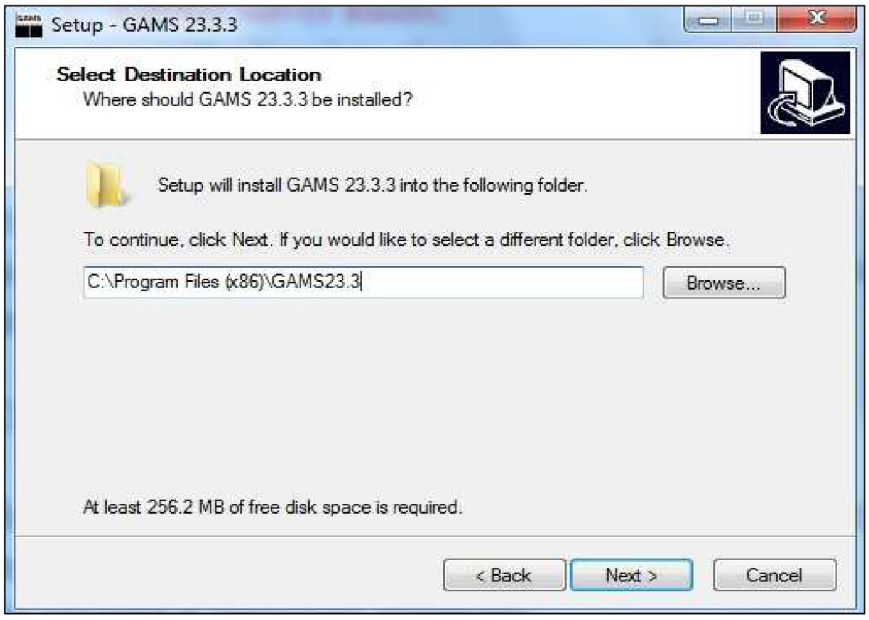
Já budu celou práci zpracovávat na OS Windows 7 (32 bit). Proto jsem zvolila hned první možnost v nabídce:



Obr. 3: *Začátek instalace – stažení souboru*

Poté se zobrazí dotaz, jestli chceme uložit instalační soubor, jehož velikost byla ve zmíněné verzi 68,9MB.

Po stažení a spuštění souboru **windows_x86_32.exe** se objeví GAMS Setup Wizard, který ve druhém kroku umožňuje si vybrat, kam bude program nainstalován.



Obr. 4: *Kam chcete GAMS nainstalovat?*

V pátém kroku pak spustíme samotnou instalaci, na jejímž závěru můžeme přidat nebo nepřidat licenční soubory. Pokud zvolíme, že je přidat nechceme, ukončíme instalaci a poprvé spustíme GAMS.

Bez licenčních souborů můžeme GAMS využívat legálně a zdarma. Tato varianta však s sebou nese jistá omezení výsledného modelu:

- počet omezení a proměnných: 300
- počet nenulových prvků: 2000, z toho nelineárních: 1000
- počet celočíselných proměnných: 50

4 K čemu mi optimalizace a GAMS budou?

V úvodu jsme si uvedli, že pomocí optimalizace a GAMSu můžeme řešit různé matematické problémy. Uveďme nyní konkrétní příklady.

Dá se říci, že optimalizovat se dá všechno, kde není nějaký výsledek závislý na zcela striktně daných hodnotách, které nemůžeme ovlivnit – například neovlivníme dobu nutnou k růstu stromu. Nemůžeme proto říct, že na dané ploše lesa získáme dvakrát více dřeva za stejný čas tím, že urychlíme růst stromů o 100 %. Stejně tak nemůžeme nutit stroje pracovat rychleji a s vyššími výkony, než jsou jejich konstrukční limity.

Můžeme ale ovlivnit například způsoby přepravy zboží mezi výchozí a cílovou stanicí a tím ovlivníme i časovou náročnost přepravy, náklady na ni, množství zboží, které se dá přepravit najednou, ... Což se už započítává do celkového zisku společnosti a pod.

Nyní si ukážeme konkrétní zadání příkladů, které budeme řešit v GAMSu – obsahují vždy seznámení se situací a položenou otázku. Příklady jsou dále řešeny v kapitole 5.3.

a) Splátkový kalendář [1]

Banka nabízí hypotéku se základní úrokovou sazbou 6% ročně plus dodatečný úrok od 0% do 2% s krokem 0,25% jako rozlišení rizikovosti půjčky - čím rizikovější, tím vyšší úroky. Při výši hypotéky 1 000 000 Kč je možné splácení rozložit na 10, 20 nebo 30 let. Po celou dobu splácení je pak garantována konstantní výše splátek při měsíčním splácení.

Jak vysoká bude splátka pro kterou skupinu podle rizika a při jaké době splácení?

b) Sklad [1]

Majitel skladu chce zvýšit své zisky. Kapacita skladu je omezená a majitel obchoduje se zbožím, jehož cena se mění v závislosti na ročním období (čtvrtletí). Na uskladnění zboží jsou navíc v průběhu roku různé náklady.

Kolik má majitel kdy nakoupit a kdy kolik prodat, aby byly jeho náklady minimální a zisk tedy maximální?

5 Práce v GAMSu

Nyní si na jednoduchém zadání ukážeme, jak se s GAMSem obvykle pracuje – již od začátku budeme využívat sumačně indexovaný zápis, který se může zdát složitější, ale je stručnější a umožňuje pohodlně měnit vstupní data modelu.

Po podrobném vysvětlení základních pojmů a příkazů na takto formulovaném programu následuje na dalším příkladě popis vývoje podobného programu od zcela konkrétní, intuitivní a na první pohled jednoduché formy zápisu až po sofistikovaný sumačně indexovaný zápis, včetně předvedení jeho výhod. V devíti krocích si tak ukážeme, jak se vlastně vyvíjel a proč vznikl právě ten styl zápisu programu, který budeme využívat v prvním příkladu.

Ukážeme si také, jak vyhodnotit výpis výsledků a jak v něm najít právě ty údaje, které hledáme.

5.1 První (ne)smělé krůčky...

V první řadě si musíme otevřít nový projekt (nabídka File – New). Po tom si tento projekt uložíme (File – Save As) do adresáře gamsdir a podadresáře projdir. Ve Window 7 se tento adresář nachází ve složce Dokumenty.

Řekněme, že první program pojmenujeme Test. Ve složce Projdir se tedy objeví soubor Test.gms. Přípona .gms znamená, že daný soubor je zdrojovým souborem pro výpočet v GAMSu.

Napíšeme program (viz níže) a spustíme výpočet (klávesou F9 nebo v nabídce File – Run). Poté, co GAMS zadání spočítá, otevře se nám vedle okýnka s nápisem Test.gms, které pod tlačítky Otevřít a Uložit, nová záložka – Test.lst. V toto souboru jsou zapsány výsledky výpočtu a další údaje (viz níže).

Když bychom se teď podívali do složky Projdir, najdeme tam, kromě těchto dvou souborů, také Test.log a Test.lxi.

Test.log je textový soubor, který v sobě nese informace o modelu, který byl právě spočítán (název modelu, datum, čas, operační systém, verze GAMS, údaje o licenci, výpočtu a úspěšnosti hledání optimálního řešení a výslednou hodnotu účelové funkce). Tyto údaje jsou dostupné (díky formě textového souboru) i po zavření celého programu GAMS. Soubor s příponou .log lze navíc otevřít i na počítači, kde GAMS není vůbec nainstalován.

Test.lxi je doprovodným souborem (více informací najdete na www.gams.com).

Nyní si ukážeme, ja budeme postupovat při řešení konkrétního příkladu.

5.1.1 Zadání úlohy

Následuje zadání úlohy, kterou budeme podrobně rozebírat, postupně programovat a později vyhodnocovat.

Slovní zadání je následující:

Firma vyrábějící nábytek prodává křesla a židle. Na výrobu jedné židle potřebuje dva kusy dřeva a jeden kus polstrování. Na výrobu jednoho křesla potřebuje jeden kus dřeva, dva kusy polstrování a jeden ocelový rám. Každou jednu židli prodá se ziskem 4 tisíc korun, na křesle vydělá 5 tisíc korun (dále jen 4 a 5 jednotek peněz). Dodavatel surovin však zkrachoval a firmě ve skladu zůstalo pouze 8 kusů dřeva, 7 kusů polstrování a 3 ocelové rámy.

Kolik židlí a křesel má firma z těchto surovin vyrobit, aby vydělala co nejvíce peněz a majitel tak měl čas zajistit si nové dodávky?

Jedná se o úlohu lineárního programování (viz kapitola 2, Def. 10)

Nyní sestavme **matematický model**.

Použijeme některé prvky obvyklé matematické symboliky, ale budeme ji modifikovat již s ohledem na syntaktická pravidla GAMSu. Model tedy obecně zapíšeme následovně:

Nechť $P = \{p_1, p_2, ..., p_k\}$ je množina produktů, které firma vyrábí, $S = \{s_1, s_2, ..., s_l\}$ je množina surovin, z nichž jsou produkty $p \in P$ vyráběny a A(s,p) je matice spotřeby suroviny $s \in S$ na výrobu produktu $p \in P$. Dále vektor se složkami C(p) značí zisk z prodeje jednotkového množství produktu $p \in P$, vektor se složkami O(s) uvádí omezení na množství suroviny $s \in S$ a proměnná X(p) objem výroby produktu $p \in P$.

Pak $U = \max \sum_P C(p)X(p)$ nazveme účelovou funkcí s omezeními $X(p) \geq 0$ a $\sum_P A(s, p)X(p) \leq O(s)$.

Přičemž pro náš konkrétní příklad platí $k = 2$ a $l = 3$.

Uvedený zápis jsme přizpůsobili nejen zápisu v GAMSu ale i zvyklostem uživatelů z praxe, kteří nemusí být zvyklí na symboliku kapitoly 2.

Při vysvětlování programu a jeho vlastností jsem se inspirovala malou učebnicí Modelovací systém GAMS, vydanou na MFF UK v roce 1993, viz [1].

Nechť S={dřevo, polstrování, rám} označuje množinu surovin, které jsou k dispozici pro výrobu produktů z množiny P={židle, křeslo}. Nechť O(s) hodnoty určují omezení na množství surovin, které jsou k dispozici pro výrobu. Spotřebu suroviny s na výrobu jednotkového množství produktu p udává prvek matice SPOTREBA(s,pP). Při zisku C(p) z jednotkového množství produktu p je cílem určit objemy výroby X(p) tak, aby se maximalizoval součet zisku z výrob jednotlivých produktů. Nakonec všechno přepíšeme do programu.

5.1.2 Přepis do programu GAMS

Sám GAMS umí tloušťkou písma a jeho barvou rozlišovat mezi **příkazy** (modře tučně), názvy jednotlivých prvků v kódu (černě), parametry jednotlivých prvků (zeleně) a

vysvětlujícími popisy (modře, tence). Toto značení zachováme i při vysvětlování, jen názvy budeme značit tučně, aby se nepletly s ostatním textem práce. Pro další odlišení budou prvky kódu psány písmem Arial.

Postupně si vysvětlíme všechny součásti programu a aby se nám jednotlivé úseky kódu nepletly, vždy bude příslušný úsek očíslován. Tato čísla se však v samotném zápisu do jazyka GAMS nevyskytují.

Musíme začít se zápisem názvu programu. Proto použijeme symbol dolaru **$**, který je interpretován jako parametr překladu systémem GAMS. Existuje celá řada parametrů, z nichž my si vybereme **TITLE**, sloužící k nastavení titulního řádku pro výpisy programu. Svůj první program nazveme MODEL OPTIMALNI SCENARE VYROBY a máme tak vše potřebné pro zápis prvního řádku programu.

```
1  $TITLE MODEL OPTIMALNI SCENARE VYROBY
```

Když už máme program pojmenovaný, určíme, jestli se mají, nebo nemají vypisovat tzv. křížové reference, které mohou posloužit při ladění programu pro nalezení výskytu jednotlivých proměnných úlohy a jejich vzájemného propojení. My zatím program ladit nepotřebujeme, takže zvolíme možnost nevypisování křížových vazeb **OFFSYMXREF** a protože je to opět parametr překladu, stejně jako **TITLE** v předchozím odstavci, musíme použít i symbol dolaru **$**.

```
2  $OFFSYMXREF
```

Úvodní nastavení máme za sebou a teď přejdeme ke zpracování samotných vstupních dat, která máme v zadání. Víme, že máme dvě množiny s názvy **s** a **p**, kde množina **s** obsahuje dostupné suroviny a množina **p** žádané produkty. Každou tuto množinu musíme zavést a to se provádí pomocí příkazu **set**, který slouží k definici množin, určených většinou výčtem indexů, používaných v modelu. Pomocí tohoto příkazu můžeme zavést každou množinu zvlášť a nebo jedním příkazem **set** uvést obě množiny dohromady, což šetří čas a hlavně tvoří text čitelnějším a přehlednějším. Každý příkaz **set** musí být ukončen středníkem.

Možné formy zápisu jsou proto:

```
set s .....;
set p .....;
```

nebo

```
set s .....
    p .....;
```

Povšimněme si, že při použití jednoho příkazu **set** je středník až na konci druhého řádku, nikoliv na koncích obou, jak je tomu v prvním příkladě. Pro jednoduchost zápisu si vybereme druhou možnost.

Dalším důležitým údajem je vysvětlující popis množiny, který se přenáší společně s názvem množiny do všech výpisů výsledků vztahujících se na k dané množině, což výrazně zvyšuje přehlednost těchto výsledků.

V našem případě přiřadíme množině s názvem **s** popis mnozina surovin a množině **p** popis mnozina produktu.

Obě množiny musí mít i své prvky. Ty píšeme do lomítkových závorek a oddělujeme je čárkami, konci řádků a někdy také mezerami.

GAMS neumí vždy správně pracovat s diakritikou a proto veškerý text píšeme bez háčků a čárek.

Výsledný zápis tedy vypadá takto:

```
3   set s mnozina surovin / Drevo, Polstrovani, Ram/
        p mnozina produktu /Zidle, Kreslo /;
```

V dalším kroku musíme zavést kolik a jakého materálu potřebujeme na výrobu židle a křesla. Tedy musíme zavést vztah mezi množinami **s** a **p**. Vzájemné vztahy nejlépe zobrazíme pomocí tabulky, kdy do řádků uvedeme suroviny a do sloupečků produkty. Tabulku jako takovou zavedeme pomocí příkazu **table**, který definuje hodnoty dvou a vícerozměrných parametrů. Tabulku si nazveme **Spotreba** a do závorky za název uvedeme množiny, kterými je tabulka indexována, tedy **s** a **p**. Za příkaz **table** tedy uvedeme **Spotreba(s,p)**. Stejně jako při zadávání množin, i tady můžeme využít komentáře – mnozstvi surovin na vyrobu produktu, který usnadní následnou orientaci v textu. Teď se nám to může zdát zbytečné, když se nám celý program vejde na jednu stránku, ale později, při řešení složitých úloh, se každý zpřehledňující údaj hodí.

Máme tedy zavedenou tabulku a nyní ji naplníme daty. V prvním řádku uvedeme seznam prvků množiny **p**, jejichž hodnoty jsou po jednotlivých řádcích příslušejících prvkům množiny **s** uvedeny dále. Je nutné dodržovat základní pravidlo, že prvky tabulky musí být uváděny přesně pod hodnoty indexů (musí být s indexy v jednom sloupečku), jinak by nebyly GAMSem vnímány jako tabulka. Prvky, které by v této tabulce nebyly uvedeny, mají implicitně nulovou hodnotu. Jako oddělovače prvků v jednotlivých řádcích slouží mezery. Automaticky se pak provádí kontrola na příslušnost prvků k uvedeným množinám. Tabulky, stejně jako množiny, mohou být až desetirozměrné.

Výsledný zápis vypadá takto:

```
4   table Spotreba(s,p) mnozstvi surovin na vyrobu produktu
                    Zidle   Kreslo
    Drevo             2       1
    Polstrovani       1       2
    Ram               0       1
```

Nyní víme, kolik surovin potřebujeme na výrobu produktů. Nikde však zatím nefigurují silně omezené skladové zásoby a zisky z prodeje jednotlivých produktů. Toto nyní napravíme.

Použijeme příkaz **parameter**, který slouží k zadávání hodnot polí (nejčastěji jednorozměrných). Použijeme dva parametry – s názvy **c** a **o**, neboli cena a omezení, přičemž cena má vztah k množině produktů **p** a omezení k množině surovin **s**. Tuto skutečnost zavedeme v kulatých závorkách za název parametru: **c(p)** a **o(s)**.

Opět využijeme možnosti si každý parametr popsat: k **c(p)** přiřadíme komentář cena produktu na trhu a k **o(s)** přiřadíme komentář omezeni na suroviny.

Přiřazení jednotlivých hodnot je určeno výčty v závorkách. Víme, že za židli získá výrobce 4 jednotky peněz, za křeslo 5 jednotek peněz. Toto přepíšeme do tvaru /Zidle 4,

Kreslo 5/ jakožto výčet hodnot parametru **c(p)**. Dále víme, že k dispozici má 8 kusů dřeva, 7 kusů polstrování a 3 ocelové rámy, což tvoří omezení na suroviny a tedy to patří k parametru **o(s)** a zapíšeme to takto: /Drevo 8, Polstrovani 7, Ram 3/.

Na konci posledního řádku musí opět být středník ukončující celý příkaz **parameter**. Výsledek vypadá takto:

```
5   parameter c(p) cena produktu na trhu
                   /Zidle 4, Kreslo 5/
          o(s) omezeni na suroviny
                   /Drevo 8, Polstrovani 7, Ram 3/;
```

Dále zavedeme potřebné proměnné. Deklaraci proměnných vždy uvozuje klíčové slovo **variables**. Za ním následuje seznam proměnných s případným slovním doprovodem, který se opět objevuje i při výstupech pro lepší čitelnost výsledků.

My potřebujeme zavést dvě proměnné: jedna - **x(p)** - se bude starat o objem výroby a druhá - **obj** - bude hodnota účelové funkce, nebo-li výpočtu zisku při výrobě. Víme, že objem výroby může být pouze kladný a proto u **x(p)** před **variables** přiřadíme **positive**. Klíčové slovo **positive** určuje obor hodnot deklarovaných proměnných na nezáporné ($\langle 0, \infty)$). Takto definovaný obor hodnot lze chápat jako omezení na nezápornost, které se projeví při vlastním výpočtu.

Před klíčové slovo **variables** můžeme použít kromě výše použité **positive**: tyto deklarace:

- NEGATIVE – značí nekladný obor hodnot = $(-\infty, 0\rangle$
- BINARY – umožňuje používat pouze hodnoty 0 a 1
- INTEGER – znamená celočíselné hodnoty od 0 do 100

Každou proměnnou doplníme opět zpřehledňujícím komentářem.

Opět musíme každý příkaz ukončit středníkem. Výsledný zápis tedy vypadá takto:

```
6   positive variables x(p) objemy vyroby;
          variable obj hodnota ucelove funkce;
```

Nyní musíme zavést seznam omezení definující model. K tomu využijeme klíčové slovo **equations** (toto klíčové slovo se používá při deklaraci rovnic i nerovnic).

Náš model musí obsahovat deklaraci účelové funkce a omezení na množství surovin. V minulém kroku jsme zavedli proměnnou **obj**, jakožto hodnotu účelové funkce. K výpočtu této hodnoty bude sloužit rovnice účelové funkce, kterou nazveme **obj1**.

Pro definici omezení zavedeme soubor nerovnic, který nazveme **omezeni** a který bude indexován množinou **s**, která má 3 prvky. Tomu bude odpovídat i počet tří nerovnic, tvořících celý soubor omezení. Obě deklarace (rovnice i souboru omezení) doplníme upřesňujícím popisem a ukončíme středníkem.

```
7   equations omezeni(s) omezeni na spotrebu surovin;
    equations obj1 ucelova funkce;
```

Nyní následuje samotné zavedení rovnice a souboru nerovnic. Pro přehlednost budu v obecných komentářích dále používat původní klíčové slovo **equation** či **equations**, které se dá v rámci práce s GAMSem brát jako označení rovnice, nerovnice či jejich souboru. Pokud budu popisovat náš konkrétní příklad, uvedu adekvátní český ekvivalent.

Definice **equation** probíhá následovně: Nejprve na řádek uvedeme název a pak definici. Jako oddělovač názvu a definice se používají dvě tečky a mezera.

Pro oddělení levé a pravé strany **equation** můžeme použít jednu z následujících tří možností:

- =L= ve smyslu menší nebo rovno
- =G= ve smyslu větší nebo rovno
- =E= ve smyslu rovno.

V definici **equation** lze používat běžné aritmetické operace (+, -, *, /, mocnění pomocí **) a standardní funkce (s výjimkou generátorů náhodných čísel). Lze používat i indexové funkce, nejčastěji SUM, jakožto součet přes prvky množiny (dále pak PROD, SMAX a SMIN – viz [1]).

Proměnné se mohou vyskytovat na obou stranách **equations**, protože GAMS sám provede jejich převedení na levou stranu, která je potřebná pro zpracování řešičem. Tvar upravených **equations** se objeví ve výstupním souboru (s příponou .lst), pokud není nastaveno jinak. K tomuto tématu se vrátíme při popisu výpisu.

Účelová funkce **obj1**, kterou budeme definovat jako první, určuje vlastní zisk, tedy cíl našeho výpočtu. Zisk je určen tím, kolik kusů jakého produktu se prodá a kolik se za kus daného produktu utrží peněz. Zopakujme si (viz text nad rámečkem 5), že máme množinu produktů **p**, která obsahuje prvky kreslo a zidle. Za kreslo dostaneme 5 jednotek peněz a za zidli dostaneme 4 jednotky peněz.

Abychom se dobrali výsledku, musíme zjistit, kolik utržíme za židle a kolik za křesla a tyto zisky sečíst. Proto použijeme funkci **sum**. Sčítat budeme ceny za všechny vyrobené kusy daného produktu, tedy **c(p)*x(p)** přes prvky množiny **p**, kde **c(p)** je cena produktu a **x(p)** je objem výroby daného produktu z množiny **p**. Zisk se musí tomuto součtu rovnat, proto použijeme **=E=** a získáme konečný tvar rovnice: **obj =E= sum**(p, c(p)*x(p)), který ukončíme opět středníkem.

Nyní si nadefinujeme omezení na spotřebu surovin, neboli **omezeni(s)**. Zde je nutné si vzpomenout na omezené množství surovin (viz text nad rámečkem 5) na skladě, tedy 8 kusů dřeva, 7 kusů polstrování a 3 rámy, a jejich spotřebu na jednotlivé výrobky (zidle = 2 kusy dřeva + 1 kus polstrování, kreslo = 1 kus dřeva + 2 kusy polstrování + 1 rám). Stejně jako u definice účelové funkce i tady použijeme funkci **sum**. Sečítat budeme tentokrát množství spotřebovaných surovin na všechny vyrobené kusy daného produktu a to opět přes prvky množiny **p**, tedy **spotreba(s,p)*x(p)**, kde **spotreba(s,p)** určuje množsví surovin na výrobu produktu a **x(p)** je objem výroby daného produktu z množiny **p**. Výsledný zápis levé strany rovnice vypadá takto: **sum** (p, spotreba(s,p)*x(p)).

Z logiky věci plyne, že nemůžeme použít více surovin, než kolik máme a proto budeme požadovat, aby součet spotřebovaných surovin byl menší nebo roven jak jejich omezení **o(s)**. Proto použijeme **=L=**. Řádek opět ukončíme středníkem.

```
8  obj1.. obj =E= sum(p, c(p)*x(p));
   omezeni(s).. sum (p, spotreba(s,p)*x(p)) =L= o(s);
```

Nyní máme vše potřebné k definici samotného modelu.

Začneme klíčovým slovem **model**, poté vypíšeme název modelu, tedy **optim**, a následně popis modelu rozdeleni vyroby mezi dva produkty. V lomítkových závorkách je uveden seznam **equations**, které model definují, tedy **obj1** a **omezeni(s)**. Příkaz klasicky ukončíme středníkem.

```
9  model optim rozdeleni vyroby mezi dva produkty /obj1, omezeni/;
```

Dalším, předposledním krokem je spuštění vlastního řešení úlohy. To se provádí pomocí příkazu **solve**. Následuje název modelu, který chceme řešit, v našem případě **optim**, a to, jestli se jedná o minimalizační, nebo maximalizační úlohu. Nám se jedná o maximalizaci zisku, tudíž použijeme klíčové slovo **maximizing**. Kdybychom chtěli minimalizovat např. vzdálenost, čas nebo účelové funkce jiných úloh, použili bychom klíčové slovo minimizing. Déle musíme uvést, co chceme maximalizovat – v našem případě je to účelová funkce **obj**.

GAMS umí řešit modely různých typů, například:

- LP – úloha lineárního programování
- NLP – úloha nelineárního programování
- MIP – celočíselné programování – hodnoty diskrétních proměnných musí nabývat diskrétních hodnot.

Při řešení našeho problému nebudeme uvažovat celočíselnost proměnných, tedy použijeme model lineárního programování a proto použijeme zkratku **LP**, kterou uvedeme klíčovým slovem **using**. Řádek opět ukončíme středníkem.

```
10  solve optim maximizing obj using LP;
```

Poslední částí programu je příkaz **display**, kterým získáme ve výstupním souboru informaci o všech složkách proměnné **x**, tedy objemy vyroby.

Každý identifikátor proměnné představuje záznam, který zahrnuje čtyři hodnoty popsané příponou s tečkou za identifikátorem:

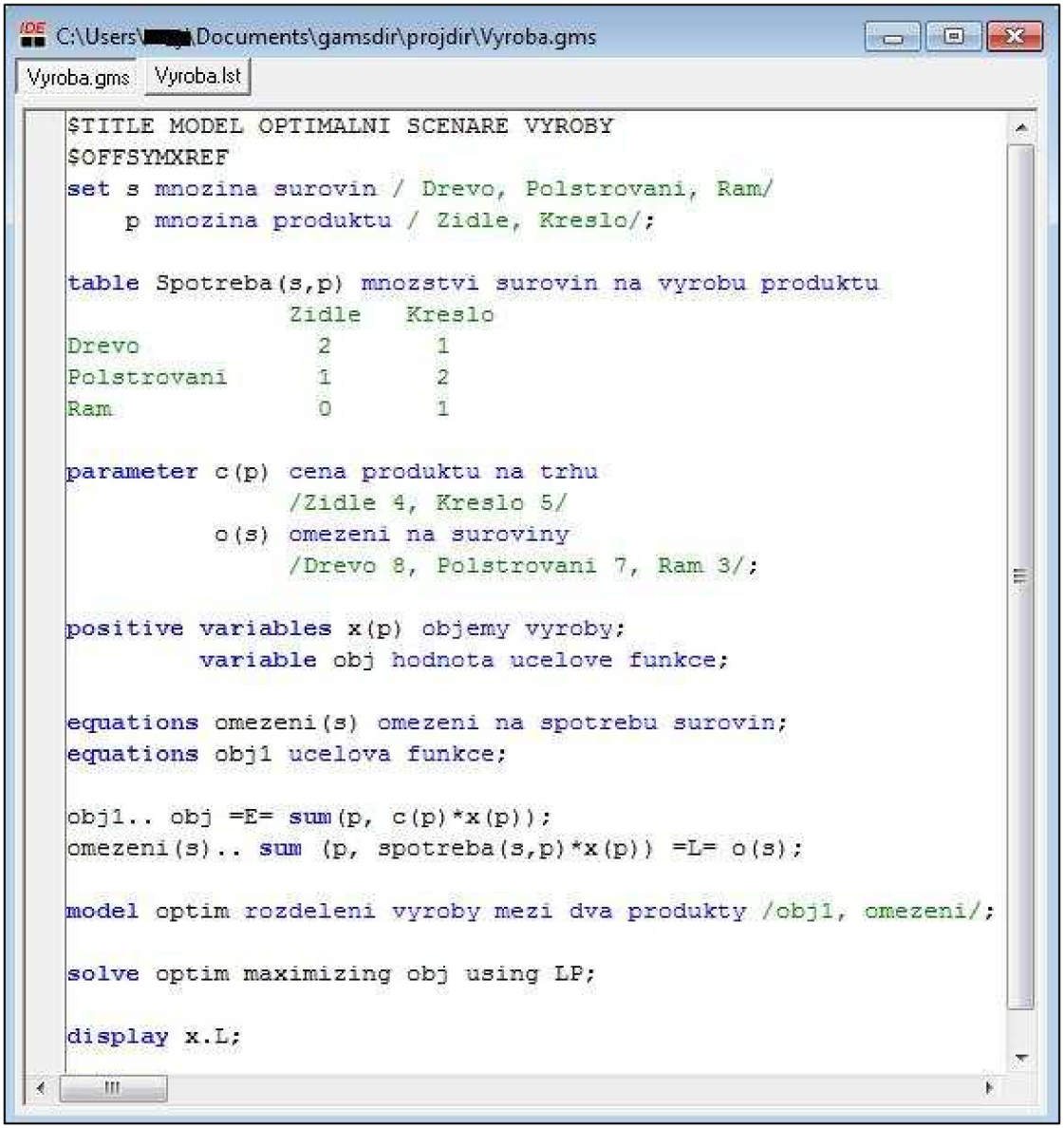
- .LO – určuje dolní mez pro hodnoty proměnné
- .UP – určuje horní mez hodnot proměnné
- .L – určuje vlastní hodnotu proměnné
- .M – určuje marginální, duální hodnotu proměnné odpovídající omezení kladenému na uvedenou proměnnou. Udává tedy rychlost změny účelové funkce při změně daného omezení viz [1]

Pro zjištění vlastní hodnoty proměnné **x** stačí použít tečkovou notaci a zapsat **display x.L** a příkaz již tradičně ukončit středníkem.

11	**display x.L**;

Tímto jsme ukončili zápis programu do systému GAMS. Pro ověření jsem uvedla i originál, jak to vypadá přímo GAMSu.

Soubor uložíme do adresáře projektu a nazveme jej VYROBA.GMS. Přípona .GMS zaručuje použití dříve zmíněného barevného rozlišení zdrojového kódu.



Obr. 5: *Zápis problému v programovacím jazyce GAMS*

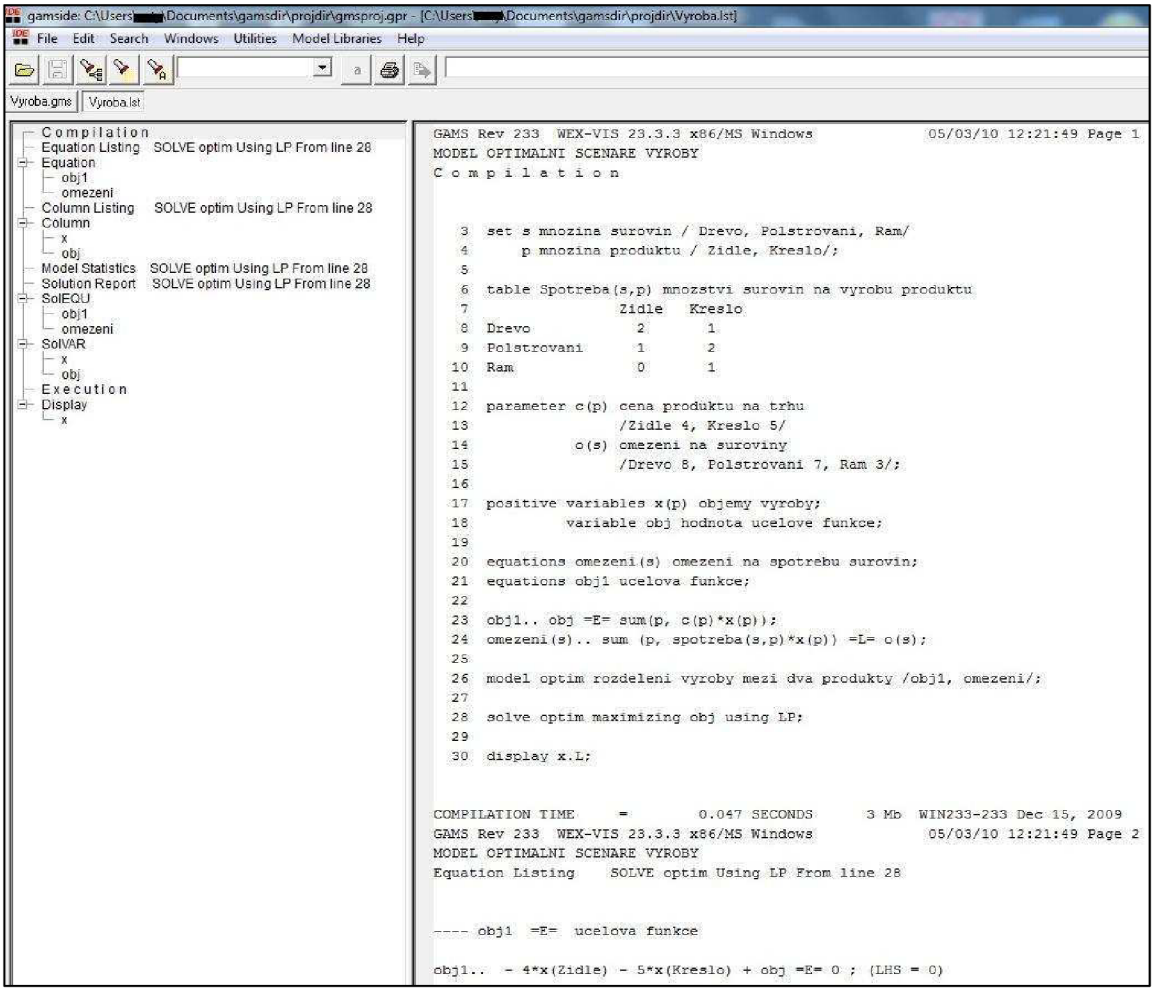
5.1.3 Spuštění programu

Dalším krokem je spuštění programu, které se provádí pomocí tlačítka F9, případně v záložce File, příkazem Run.

Jestliže jsme soubor zapsali bez chyb, proběhne vlastní výpočet. Kompletní informace o překladu a průběhu výpočtu se ukládá do souboru s příponou .LST. V našem případě tedy do souboru VYROBA.LST (podrobně bude rozebrán dále). Pokud při překladu došlo k chybám, jsou v tomto souboru uloženy informace o těchto chybách spolu s jejich označením a stručným popisem. Označení chyby je provedeno znakem $ pod místem, kde překladač chybu nalezl. Za znakem $ je uvedeno číslo chyby, jehož význam je uveden v závěru výstupního souboru. [1]

5.1.4 Vyhodnocení výstupu výpočtu

Po ukončení výpočtu se zobrazí vedle záložky Vyroba.gms nová záložka Vyroba.lst. Tato stránka je rozdělena na dva sloupce, přičemž levý sloupec zobrazuje souhrnné informace o vyřešeném modelu (seznam **equations**, proměnných, zvolený způsob řešení a informace o tom, co má být zobrazeno) a v pravém sloupci je kompletní výpis.



Obr. 6: *GAMS po vyřešení modelu – nová záložka Vyroba.lst, vlevo souhrné informace a vpravo začátek samotného výpisu*

Nyní si popíšeme výstupní soubor VYROBA.LST, jehož originální výpis v GAMS je natolik dlouhý, že ho sem nebudu kopírovat v celku ale po částech, které budu rovnou komentovat. Využiji stejný systém, jako u zápisu do GAMSu, tzn. Orámované a očíslované části výpisu s doplněným komentářem, tentokrát pod rámečkem.

Tak tedy začneme:

```
1
□GAMS Rev 233  WEX-VIS 23.3.3 x86/MS
Windows 04/03/10 12:21:49 Page 1
MODEL OPTIMALNI SCENARE VYROBY
C o m p i l a t i o n
```

První speciální znak se přímo ve výpisu neobjeví, ale pokud výpis zkopírujeme, už se ukáže. Jedná se o znak, který při zaslání výstupního souboru na tiskárnu způsobí přechod na novou stránku tisku. První řádek výpisu obsahuje také informace o verzi GAMS, která byla použita, pod jakým operačním systémem výpočet probíhal, datum a čas výpočtu a číslo stránky. Na druhém řádku je název modelu a na třetím řádku prvního oddílu je co na dané stránce nalezneme – tady kompilovaný model.

```
2
 3  set s mnozina surovin / Drevo, Polstrovani, Ram/
 4      p mnozina produktu / Zidle, Kreslo/;
 5
 6  table Spotreba(s,p) mnozstvi surovin na vyrobu produktu
 7                  Zidle   Kreslo
 8  Drevo             2       1
 9  Polstrovani       1       2
10  Ram               0       1
11
12  parameter c(p) cena produktu na trhu
13                 /Zidle 4, Kreslo 5/
14             o(s) omezeni na suroviny
15                  /Drevo 8, Polstrovani 7, Ram 3/;
16
17  positive variables x(p) objemy vyroby;
18           variable obj hodnota ucelove funkce;
19
20  equations omezeni(s) omezeni na spotrebu surovin;
21  equations obj1 ucelova funkce;
22
23  obj1.. obj =E= sum(p, c(p)*x(p));
24  omezeni(s).. sum (p, spotreba(s,p)*x(p)) =L= o(s);
25
26  model optim rozdeleni vyroby mezi dva produkty /obj1, omezeni/;
27
28  solve optim maximizing obj using LP;
29
30  display x.L;

COMPILATION TIME     =        0.047 SECONDS      3 Mb  WIN233-233
Dec 15, 2009
```

Následuje kopie vstupního souboru s očíslovanými řádky. Na tato čísla jsou dále uváděny odkazy umožňující snadnější orientaci. Příkazy, které ve vstupním souboru začínají symbolem $, se do výpisu neuvádějí. Na konci je informace o spotřebě času na překlad.

```
3  □GAMS Rev 233  WEX-VIS 23.3.3 x86/MS Windows
   04/03/10 12:21:49 Page 2
   MODEL OPTIMALNI SCENARE VYROBY
   Equation Listing    SOLVE optim Using LP From line 28
```

Začátek nové stránky, která bude obsahovat seznam **equations** určených k řešení podle řádku 28, tedy v našem případě podle příkazu **solve optim maximizing obj using LP**;. Kdybychom v programu řešili více modelů, byla by to informace o tom, které výsledky následují.

```
4  ---- obj1  =E=  ucelova funkce

   obj1..  - 4*x(Zidle) - 5*x(Kreslo) + obj =E= 0 ; (LHS = 0)

   ---- omezeni  =L=  omezeni na spotrebu surovin

   omezeni(Drevo)..  2*x(Zidle) + x(Kreslo) =L= 8 ; (LHS = 0)

   omezeni(Polstrovani)..  x(Zidle) + 2*x(Kreslo) =L= 7 ; (LHS = 0)

   omezeni(Ram)..  x(Kreslo) =L= 3 ; (LHS = 0)
```

Definice rovnice a nerovnic ve vstupním souboru se odrazí ve výstupním souboru. Zde jsou tyto **equations** zapsány ve formě, kterou GAMS používá pro vlastní výpočet – neznámé na levé straně, konstanty na pravé. Informace uvedená v závorce za středníkem slouží k představě o hodnotě levé strany v okamžiku spuštění výpočtu. V našem případě nebyla inicializována ani jedna z hodnot **obj** ani **x(p).L**, tedy implicitně jsou tyto hodnoty nulové a tím je nulová i celá levá strana rovnice **obj1**. Pokud by v závorce byly tři hvězdičky (např.: LHS = 0***), znamená to, že počáteční řešení není přípustné.

Dále se vypisují nerovnice **omezeni(s)**. Vypíší se všechny nerovnice určené indexy množiny **s**. Kdyby ale množina **s** měla více prvků než náš triviální případ, stal by se výpis programu zbytečně zdlouhavý a nepřehledný. GAMS proto umožňuje ovlivňovat počet konkrétních omezení vypisovaných vždy k dané skupině omezení, určené indexující množinou. Standardně je tato hodnota nastavena na 3. Počet omezení *n*, které se vypíší, se dá určit pomocí příkazu **option limrow = n** ve vstupním souboru. Pokud je $n = 0$, **equations** se nevypisují vůbec. Je-li **equations** více než je nastavená hladina **limrow**, objeví se ve výstupním souboru hlášení `remaining entry skipped`. Příkazem **option** lze případně měnit i další parametry výstupu, výpočtu apod.

```
5  □GAMS Rev 233  WEX-VIS 23.3.3 x86/MS Windows
   04/03/10 12:21:49 Page 3
   MODEL OPTIMALNI SCENARE VYROBY
   Column Listing      SOLVE optim Using LP From line 28
```

Následuje hlavička uvádějící třetí stránku výpisu, na které budou uvedeny informace o použitých proměnných. Opět je doplněna o datum, čas a informaci, podle kterého modelu je úloha řešena.

```
6
---- x  objemy vyroby

x(Zidle)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
       -4       obj1
        2       omezeni(Drevo)
        1       omezeni(Polstrovani)

x(Kreslo)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
       -5       obj1
        1       omezeni(Drevo)
        2       omezeni(Polstrovani)
        1       omezeni(Ram)

---- obj  hodnota ucelove funkce

obj
                (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
        1       obj1
```

Nejprve je uveden název proměnné s doplňujícím komentářem, který jsme k jejímu názvu připsali ve vstupním souboru.

Pak jsou standardně uvedeny hodnoty této proměnné pro maximálně první tři prvky z množiny, kterou byla tato proměnná indexována. V případě proměnné **x** tedy pro prvky množiny produktů **p**, tedy zidle a kreslo. Počet indexů *n*, pro který se vypisuje informace, lze změnit pomocí příkazu **option limcol = n**. Pokud by bylo n = 0, nevypisovaly by se žádné hodnoty.

Následuje výpis všech čtyř prvků dané proměnné, tedy dolní mez (.LO), vlastní hodnota (.L). horní mez (.UP) a marginální, nebo-li duální hodnota (.M).

Dále je uveden seznam omezení v nichž se proměnná **x(p)** vyskytuje. Hodnota před každou rovnicí udává koeficient, se kterým se daná proměnná v rovnici vykytuje.

Při informacích o hodnotách proměnných se někdy používají speciální symboly (+INF pro +∞, -INF pro - ∞, UNDF pro nedefinovanou hodnotu a EPS pro nekonečně malou ale nenulovou hodnotu).

```
7
□GAMS Rev 233  WEX-VIS 23.3.3 x86/MS Windows
04/03/10 12:21:49 Page 4
MODEL OPTIMALNI SCENARE VYROBY
Model Statistics    SOLVE optim Using LP From line 28
```

Další stránka výpisu obsahuje hlavičku s technickou zprávou o modelu.

```
8
MODEL STATISTICS
BLOCKS OF EQUATIONS           2     SINGLE EQUATIONS            4
BLOCKS OF VARIABLES           2     SINGLE VARIABLES            3
NON ZERO ELEMENTS             8
GENERATION TIME      =        0.016 SECONDS      4 Mb  WIN233-233
Dec 15, 2009
EXECUTION TIME       =        0.016 SECONDS      4 Mb  WIN233-233
Dec 15, 2009
```

Zde jsou uvedeny informace o počtu omezení, proměnných a nenulových prvků, o časové náročnosti, nárocích na paměť počítače v průběhu výpočtu a verzi řešiče.

```
9  □GAMS Rev 233  WEX-VIS 23.3.3 x86/MS Windows
   04/03/10 12:21:49 Page 5
   MODEL OPTIMALNI SCENARE VYROBY
   Solution Report     SOLVE optim Using LP From line 28
```

Pátá stránka obsahuje zprávu o provedeném výpočtu.

```
10                    S O L V E      S U M M A R Y

     MODEL   optim               OBJECTIVE  obj
     TYPE    LP                  DIRECTION  MAXIMIZE
     SOLVER  CPLEX               FROM LINE  28

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      1 Optimal
**** OBJECTIVE VALUE                22.0000

 RESOURCE USAGE, LIMIT          0.149      1000.000
 ITERATION COUNT, LIMIT         2    2000000000

ILOG CPLEX       Nov  1, 2009 23.3.3 WIN 13908.15043 VIS x86/MS
Windows
Cplex 12.1.0, GAMS Link 34

LP status(1): optimal
Optimal solution found.
Objective :          22.000000
```

Informace o provedeném výpočtu obsažené v první části:

- `MODEL optim` – jde o optimalizační úlohu
- `OBJECTIVE obj` – cíl výpočtu, tedy hodnota účelové funkce
- `TYPE LP` – úloha lineárního programování
- `DIRECTION MAXIMIZE` – požadujeme maximální hodnotu účelové funkce
- `SOLVER CPLEX` – název řešiče (součást GAMS)
- `FROM LINE 28` – umístění modelu, který se počítal
- `SOLVER STATUS 1 Normal Completion` – stav řešiče – úspěšně ukončil jedno řešení
- `MODEL STATUS 1 Optimal` – stav modelu – jedno optimální řešení
- `OBJECTIVE VALUE 22.0000` – cílová hodnota, hodnota účelové funkce
- `RESOURCE USAGE, LIMIT 0.149 1000.000` – použité zdroje a jejich limit
- `ITERATION COUNT, LIMIT 2 2000000000` – počet provedených iterací a jejich limit
- `ILOG CPLEX Nov 1, 2009 23.3.3 WIN 13908.15043 VIS x86/MS Windows Cplex 12.1.0, GAMS Link 34` – informace o verzi GAMS, OS a dalších technických detailech.
- `LP status(1): optimal` – stav úlohy lineárního programování – v pořádku
- `Optimal solution found.` – optimální řešení bylo nalezeno
- `Objective : 22.000000` – hodnota tohoto optimálního řešení je 22. Tedy maximální zisk při daných omezeních činí 22 jednotek peněz, nebo-li 22 000 Kč.

```
                           LOWER      LEVEL      UPPER     MARGINAL

---- EQU obj1                .          .          .         1.000
  obj1  ucelova funkce

---- EQU omezeni  omezeni na spotrebu surovin
                LOWER      LEVEL      UPPER     MARGINAL

Drevo            -INF      8.000      8.000      1.000
Polstrovani      -INF      7.000      7.000      2.000
Ram              -INF      2.000      3.000        .

---- VAR x  objemy vyroby
           LOWER      LEVEL      UPPER     MARGINAL

Zidle        .         3.000      +INF         .
Kreslo       .         2.000      +INF         .

                           LOWER      LEVEL      UPPER     MARGINAL
---- VAR obj                -INF      22.000      +INF         .

  obj  hodnota ucelove funkce

**** REPORT SUMMARY :        0     NONOPT
                             0  INFEASIBLE
                             0  UNBOUNDED
```

11

Následuje přehled 4 prvků výsledné hodnoty **equations** a proměnných. Vidíme, že rovnice **obj1**, tedy účelová funkce, má minimální, vlastní i maximální hodnotu nulovou (nulová hodnota se značí tečkou). Hodnota MARGINAL (.M) určuje duální hodnotu proměnné, která se vztahuje k příslušnému omezení. Určuje změnu hodnoty účelové funkce při jednotkové změně omezení.

Omezeni, na rozdíl od **obj1,** nabývá všech hodnot a to pro každý prvek množiny, kterou byla indexována, zvlášť. Ve výsledné tabulce vidíme, že pro všechny prvky z indexující množiny **s** je určena dolní mez (.LO, LOWER) na úrovni 0, což znamená, že je nezáporná. Vlastní hodnota (.L, LEVEL) ukazuje, kolik které suroviny spotřebujeme na vypočtené optimální řešení. Horní mezí (.UP, UPPER) je dostupné množství surovin zadané na začátku. Vidíme, že v případě dreva a polstrovani jsme dosáhli rovnosti mezi použitým a dostupným množstvím a v případě ramu jsme použili méně kusů, než jsme měli na skladě. Takže jsme splnili podmínku **=L=** (menší nebo rovno). Sloupeček MARGINAL vnímáme jako informaci o možném navýšení zisku při dokoupení jednotkového množství dané suroviny. Pokud bychom tedy dokoupili 1 kus dřeva, vzrostl by zisk o jednotku peněz (tedy o 1 000 Kč). Stejně tak, kdybychom dokoupili jeden kus polstrování, zvedl by se zisk o dvě jednotky (2 000 Kč). V případě rámu se jedná o neaktivní omezení (= podle principu komplementarity [1, 3] musí být duální hodnota proměnné nulová, není-li daná nerovnice splněna jako rovnost), protože při výrobě nedošlo k vyčerpání limitního množství rámů.

V popisu proměnných se orientujeme úplně stejně. Vidíme, že u proměnné **x** je dolní mez nulová – nemůžeme vyrobit záporné množství produktů. Horní mez je nastavena na $+\infty$, tedy není omezena. A vlastní hodnota udává počet vyrobených kusů daného produktu. Marginální hodnota je opět nulová, protože žádná z proměnných nesplňuje aktivně omezení, které je na ni kladeno. V našem případě omezení na nezáporost a obě jsou

nezáporné, nikoliv nulové. U popisu proměnné **obj** najdeme opět dolní i horní mez bez omezení a vlastní hodnotu 22.

Výsledkem je, že ze zadaných surovin, které máme na skladě vyrobíme 3 židle a 2 křesla a celkem za ně utržíme 22 000Kč. Tento výpočet si ověříme: Jednu židli prodáme za 4 000 Kč. Křeslo prodáme za 5 000. Když vyrobíme 3 židle, získáme za ně 12 000, za 2 křesla 10 000, což nám dává dohromady oněch 22 000. GAMS měl opět pravdu.

Poslední údaje v tomto úseku se týkají samotného reportu a případných problémů s jeho vytvořením. K těm nedošlo.

```
12
```

```
□GAMS Rev 233  WEX-VIS 23.3.3 x86/MS Windows
04/03/10 12:21:49 Page 6
MODEL OPTIMALNI SCENARE VYROBY
E x e c u t i o n
```

Šestá stránka se závěrečným shrnutím údajů.

```
13
```

```
----     30 VARIABLE x.L  objemy vyroby

Zidle  3.000,    Kreslo 2.000

EXECUTION TIME       =        0.000 SECONDS      3 Mb  WIN233-233
Dec 15, 2009

USER: GAMS Development Corporation, Washington, DC
G871201/0000CA-ANY
      Free Demo,  202-342-0180,  sales@gams.com,  www.gams.com
DC0000

**** FILE SUMMARY
Input      C:\Users\xxxxxx\Documents\gamsdir\projdir\Vyroba.gms
Output     C:\Users\xxxxxx\Documents\gamsdir\projdir\Vyroba.lst
```

Na řádku 30 zdrojového kódu je uveden příkaz **display x.L**; a proto se zde, spolu s vysvětlujícím popisem, objevuje výpis vlastní hodnoty proměnné **x** pro prvky množiny **p**.

Následuje informace o časové náročnosti, velikosti programu, verzi GAMS, vlastníkovi licence a o tom, že výpočet byl proveden na volně dostupné free verzi. Poslední údaje ukazují umístění vstupního a výstupního souboru

Tímto jsme si úspěšně naprogramovali, spustili a vyhodnotili první úlohu.

5.2 Od jednoduššího ke složitějšímu: Výrobní plán v devíti krocích

V této kapitole si ukážeme, jak se pomocí zdánlivě čím dál složitějšího kódu dostaneme k jednoduššímu řešení mnohem obecnější a rozsáhlejší úlohy (jak jsme si řekli na začátku kap. 5).

Celý problém si budeme ilustrovat na popisu výrobního plánu, ze začátku velmi podobného výše uvedenému příkladu. Zápis už nebudu rozebírat tak podrobně, jako u předešlého vzorového příkladu, ale zaměřím se na rozdíly a popis změn mezi jednotlivými postupnými kroky. Proto se již nebudu příliš vracet k vysvětlování základních pojmů a nástrojů, spíš je jen připomenu.

5.2.1 Krok 1 - zadání

Slovní zadání nové úlohy:

Řekněme, že si před Velikonocemi otevřeme malou pekárnu a začneme péct vánočky a mazance.

K dispozici máme zásobu těsta, rozinek a mandlí. Na vánočku spotřebujeme 4 jednotky těsta, 2 jednotky rozinek a 2 jednotky mandlí. Na mazanec spotřebujeme jen 2 jednotky těsta, 1 jednotku rozinek a 2 jednotky mandlí. Celkem máme na skladě 700 jednotek těsta, 300 jednotek rozinek a 450 jednotek mandlí.

Za vánočku utržíme 50 Kč a za mazanec 30 Kč.

Otázky zní: Kolik máme z daného množství surovin upéct vánoček a kolik mazanců, abychom vydělali co možná nejvíce? A kolik to bude?

Jedná se, stejně jako v předešlém případě, o úlohu lineárního programování (viz kap. 2, def. 10). Postup je proto shodný s předcházejícím příkladem, takže budeme postupovat poněkud rychleji v popisu zdrojového kódu.

Projekt nazveme Výrobní plán. Následuje deklarace proměnných **X**, **Y** a **Z** s jejich vysvětlujícím popisem. **X** představuje počet vánoček a **Y** počet mazanců.

V další části deklarujeme **equations** (viz text nad rámečkem 7, kap. 5.1.2). Máme opět tři suroviny, jejichž množství je v zadání omezené. Proto zavedeme celkem 3 nerovnice, které budou odrážet všechna omezení a požadavky na suroviny a rovnici, která bude určovat účelovou funkci, tedy zisk.

Ze zadání vyplývá, že první suroviny máme na skladě 700 kusů, druhé 300 a třetí 450. Proto zavedeme nerovnice **OMEZ1** pro těsto, **OMEZ2** pro rozinky a **OMEZ3** pro mandle. Do první nerovnice musíme zavést požadavek na 4 jednotky těsta na každou jednu vánočku při počtu vánoček **X**, tedy 4***X**, a 2 jednotek těsta na každý jeden mazanec při počtu mazanců **Y**, tedy 2***Y**.

Ve výsledku tedy bude vypadat nerovnice **OMEZ1** takto: **4*X + 2*Y =L= 700** (=L= značí menší nebo rovno). Do druhé nerovnice musíme stejným způsobem zapsat spotřebu rozinek (**2*X + 1*Y =L= 300**) a do třetí spotřebu mandlí (**2*X + 2*Y =L= 450**).

Celkový zisk Z je dán cenou jednotlivého produktu a počtem kusů daného produktu. Rovnici nazveme **ZISK** a definujeme ji takto: **Z =E= 50*X + 30*Y**, kde =E= značí rovnost.

Následně musíme zadat definici modelu, který nazveme **VYROBA**, definovaný **equations** **OMEZ1**, **OMEZ2**, **OMEZ3** a **ZISK**.

Předposlení krok je spuštění samotného řešení úlohy **VYROBA** za předpokladu maximalizace zisku, tedy **MAXIMIZING Z**, a za použití lineárního programování (**USING LP**).

Posledním krokem je příkaz pro vypsání vlastních hodnot proměnných **X**, **Y** a **Z**.

Zde je kompletní zápis v GAMSu:

```
$TITLE VYROBNI PLAN

variables X   mnozstvi vanocek
          Y   mnozstvi mazancu
          Z   vysledny zisk;

equations OMEZ1  omezeni dane zasobami testa
          OMEZ2  omezeni dane zasobami rozinek
          OMEZ3  omezeni dane zasobami mandli
          ZISK   udava vysledny zisk;

OMEZ1..  4 * X + 2 * Y =L= 700;
OMEZ2..  2 * X + 1 * Y =L= 300;
OMEZ3..  2 * X + 2 * Y =L= 450;
ZISK..   Z =E= 50 * X + 30 * Y;

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;

solve VYROBA MAXIMIZING Z USING LP;

display X.L, Y.L, Z.L;
```

Obr. 7: *Výroba – krok 1*

Takto zadaný program je velmi přehledný, stručný a konkrétní. Umí nám velmi rychle říct, kolik peněz vyděláme a kolik produktů prodáme.

Problém nastane ve chvíli, když si představíme, že najednou máme z daných surovin vyrábět místo vánoček třeba loupáky. V takovém případě bychom museli projít celý program a vyhledat všechna data vztahující se k vánočkám (spotřebované suroviny na jeden kus, cenu za kus, ...) a ručně je přepsat na údaje o loupácích. Nehledě na přepisování všech komentářů.

To se při rozsahu programu, jaký máme před očima, nezdá být tak obtížné. Představme si však, že nepracujeme se třemi surovinami a dvěma výrobky, ale že surovin a výrobků je daleko více (to uvidíme v kroku 8 a 9) a že bychom museli procházet program ne na pár řádků, ale na pár desítek (nebo i více) řádků a hledat všude konkrétní písmenka a čísla, která k nim patří. To už by bylo hodně složité a zdlouhavé a tím pádem by to byl velmi silný zdroj lidských chyb.

Pro pozdější kontrolu uvedu výsledky, ke kterým jsme pomocí tohoto programu dospěli:

```
VARIABLE X.L                =       75.000  mnozstvi vanocek
VARIABLE Y.L                =      150.000  mnozstvi mazancu
VARIABLE Z.L                =     8250.000  vysledny zisk
```

Obr. 8: *Výroba – krok 1 - výsledek*

Vyrobili jsme tedy 75 vánoček a 150 mazanců a dohromady za ně utžili 8 250 Kč.

5.2.2 Krok 2 – názorné identifikátory

V osmi zbývajících bodech této kapitoly si ukážeme, že není složité upravit program tak, aby se daly pohodlně, rychle a bez chyb měnit vstupní údaje bez zasahování do samotné definice modelu a že pak už je jen krůček k rozšíření modelu na daleko obsáhlejší soubory vstupních dat.

První, co si zobecníme, je zadávání výrobků. Už nebudeme počítat s konkrétními výrobky (vánočka, mazanec, ...) a jejich počty (**X**, **Y**, ...), ale s obecným zadáním druhu výrobku 1 a 2 a určení jejich množství pomocí proměnných **VYROBEK1** a **VYROBEK2**, které nám ulehčí pozdější případnou záměnu výrobků (vánočky za loupáky a pod.).

Tato změna se v zápisu do GAMSu dotkne deklarace proměnných (místo **X** a **Y** deklarujeme **VYROBEK1** a **VYROBEK2**), definice **equations**, kde proběhne stejná záměna a požadavku na zobrazení vlastních hodnot proměnných.

Ve stejném duchu přejmenujeme výsledný zisk ze **Z** na **OPTCENA**. Tato změna se projeví stejným způsobem, jako výše uvedené přejmenování proměnných u množství výrobků.

V deklaraci **equations** došlo ke změně „pouze“ ve vysvětlujících komentářích, kde se objevily suroviny S1, S2 a S3, které „nahradily“ těsto, rozinky a mandle. Je to krok k dalšímu zobecnění, které teprve přijde.

Pro kontrolu jsem do vysvětlujících popisů uvedla i význam, který měla daná proměnná či rovnice podle původního značení. Kompletní zápis v GAMS je zde:

```
$TITLE Vyrobni plan
variables VYROBEK1   mnozstvi vyrobku prvniho typu - napr. vanocek
          VYROBEK2   mnozstvi vyrobku druheho typu - napr. mazancu
          OPTCENA    vysledny zisk;

equations OMEZ1  dane zasobami suroviny S1 - napr. testa
          OMEZ2  dane zasobami suroviny S2 - napr. rozinek
          OMEZ3  dane zasobami suroviny S3 - napr. mandli
          ZISK   udava vysledny zisk ;

OMEZ1..   4 * VYROBEK1 + 2 * VYROBEK2 =L= 700;
OMEZ2..   2 * VYROBEK1 + 1 * VYROBEK2 =L= 300;
OMEZ3..   2 * VYROBEK1 + 2 * VYROBEK2 =L= 450;
ZISK..    OPTCENA =E= 50 * VYROBEK1 + 30 * VYROBEK2;

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;

solve VYROBA MAXIMIZING OPTCENA USING LP;

display VYROBEK1.L, VYROBEK2.L, OPTCENA.L;
```

Obr. 9: *Výroba – krok 2*

Pro kontrolu si opět uvedeme výsledky:

```
VARIABLE VYROBEK1.L             =       75.000  mnozstvi vyrobku prvn
                                                iho typu - napr. vano
                                                cek
VARIABLE VYROBEK2.L             =      150.000  mnozstvi vyrobku druh
                                                eho typu - napr. maza
                                                ncu
VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 10: *Výroba – krok 2 - výsledek*

Vidíme, že jsme vyrobili 75 kusů výrrobku prvního typu, tedy třeba vánoček, 150 kusů výrobku druhého typu, tedy třeba mazanců a utržili jsme 8 250 Kč. Dospěli jsme tedy ke stejným výsledkům, jako v kroku 1 – změna označení neměla vliv na výsledek.

5.2.3 Krok 3 – indexování proměnných

Ve druhém zjednodušovacím kroku dojde na sloučení **VYROBEK1** a **VYROBEK2** do **VYROBEK(CISLO)**, což je logická a do budoucna velmi prospěšná změna. Potřebujeme k tomu však ještě nadeklarovat ono **CISLO**, tedy indexující množinu proměnné **VYROBEK**, která musí obsahovat čísla (prvky) 1 a 2. To se provede pomocí příkazu **sets** (viz text nad rámečkem 3, kap. 5.1.2) (set nebo sets – obě varianty GAMS přijme).

Tato změna ovlivní zápis deklarace proměnné i **equations**, kde místo **VYROBEK1** bude vystupovat **VYROBEK("1")**. Také se zjednodušší zápis příkazu **display** pro zobrazení vlastní hodnoty proměnné.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /;

variables VYROBEK(CISLO)  mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;

equations OMEZ1  dane zasobami suroviny S1
          OMEZ2  dane zasobami suroviny S2
          OMEZ3  dane zasobami suroviny S3
          ZISK   udava vysledny zisk;

OMEZ1..  4 * VYROBEK("1") + 2 * VYROBEK("2") =L= 700;
OMEZ2..  2 * VYROBEK("1") + 1 * VYROBEK("2") =L= 300;
OMEZ3..  2 * VYROBEK("1") + 2 * VYROBEK("2") =L= 450;
ZISK..   OPTCENA =E= 50 * VYROBEK("1") + 30 * VYROBEK("2");

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 11: *Výroba – krok 3*

Výsledky výpočtu ukazují, že jsme vyrobili 75 kusů výrobku "1" a 150 kusů výrobku "2" a utržili jsme 8 250 Kč. Opět jsme dosáhli shodných výsledků – program funguje správně.

```
----     20 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     20 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 12: *Výroba – krok 3- výsledek*

5.2.4 Krok 4 – pojmenování ceny

Ve čtvrtém kroku využijeme změn, které se odehrály v kroku 3 a zobecníme zápis ceny za každý jeden výrobek, tedy zavedeme parametr **CENA**, který bude udávat ceny výrobků a bude záviset na množině **CISLO**. Do lomítkových závorek vypíšeme hodnoty parametru **CENA(CISLO)**. V úvodním příkladu jsme si ukázali jeden způsob zápisu hodnot (viz text nad rámečkem 5, kap. 5.1.2):

```
parameter c(p) cena produktu na trhu
                /Zidle 4, Kreslo 5/;
```

Tento styl zápisu by bylo možné použít i zde, protože pro dvě hodnoty by byl stále ještě přehledný. Vypadal by následovně:

```
parameter CENA(CISLO) ceny jednotlivych vyrobku
                      / 1 50, 2 30/;
```

Pro ilustraci, zápis hodnot pro pět výrobků by vypadal takto:

```
/ 1 50, 2 30, 3 25, 4 45, 5 36 /
```

Což je už dosti nepřehledné a hlavně v tomto zápise upravovat jednotlivé hodnoty by bylo docela riskantní vzhledem ke vzniku překlepu, záměny čísel či jiné chyby. My si proto ukážeme zápis, který bude více vyhovovat pozdějšímu rozšíření počtu produktů.

Jak jsme si už dříve řekli, k oddělování hodnot používáme čárky nebo také konce řádků. Toho využijeme tentokrát a zápis pak bude vypadat takto:

```
parameter CENA(CISLO) ceny jednotlivych vyrobku
                      / 1 50
                        2 30/;
```

Zápis umožňuje rychlou orientaci v pořadovém čísle výrobku i jeho ceně a stejně tak umožňuje rychlou a bezpečnou úpravu dat.

Tato změna zasáhne pouze do definice rovnice **OPTCENA** pro výpočet zisku, kde místo konkrétních čísel budou vystupovat **CENA("1")** a **CENA("2")**.

Zápis s uvedenými změnami v GAMS:

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /;

parameters CENA(CISLO)    ceny jednotlivych vyrobku
           / 1  50
             2  30 /;

variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk ;

equations OMEZ1  dane zasobami suroviny S1
          OMEZ2  dane zasobami suroviny S2
          OMEZ3  dane zasobami suroviny S3
          ZISK   udava vysledny zisk  ;

OMEZ1..  4 * VYROBEK("1") + 2 * VYROBEK("2") =L= 700;
OMEZ2..  2 * VYROBEK("1") + 1 * VYROBEK("2") =L= 300;
OMEZ3..  2 * VYROBEK("1") + 2 * VYROBEK("2") =L= 450;
ZISK..   OPTCENA =E= CENA("1") * VYROBEK("1") + CENA("2") * VYROBEK("2");

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;

solve VYROBA MAXIMIZING OPTCENA USING LP;

display VYROBEK.L, OPTCENA.L;
```

Obr. 13: *Výroba – krok 4*

Kontrola:

```
----     26 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     26 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 14: *Výroba – krok 4 - výsledek*

Dospěli jsme opět ke shodným číslům – model je správně.

5.2.5 Krok 5 – pojmenování kapacity

Obdobně, jako jsme v předchozím kroku zavedli parametr **CENA** pro zobecnění cen výrobků, tak nyní zavedeme parametr **SKLAD** pro zobecněný zápis limitů surovin na skladě. Opět musíme zavést indexující množinu, kterou v tomto případě nazvene **SUROVINA** a bude obsahovat prvky S1, S2 a S3.

Hodnoty parametru **SKLAD(SUROVINA)** budeme zapisovat stejně, jako hodnoty parametru **CENA(CISLO)**, tedy oddělené konci řádků.

Tato změna ve značení se dotkne pouze definice nerovnic, kde na pravé straně bude místo konkrétního čísla vystupovat **SKLAD("S1")** , **SKLAD ("S2")** a **SKLAD ("S3")**.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /
     SUROVINA typ suroviny / S1, S2, S3 /;

parameters CENA(CISLO)   ceny jednotlivych vyrobku
           / 1  50
             2  30 /

           SKLAD(SUROVINA) limit surovin na sklade
           / S1 700
             S2 300
             S3 450 /;

variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;

equations OMEZ1  dane zasobami suroviny S1
          OMEZ2  dane zasobami suroviny S2
          OMEZ3  dane zasobami suroviny S3
          ZISK   udava vysledny zisk;

OMEZ1.. 4 * VYROBEK("1") + 2 * VYROBEK("2") =L= SKLAD("S1");
OMEZ2.. 2 * VYROBEK("1") + 1 * VYROBEK("2") =L= SKLAD("S2");
OMEZ3.. 2 * VYROBEK("1") + 2 * VYROBEK("2") =L= SKLAD("S3");
ZISK..  OPTCENA=E=CENA("1")*VYROBEK("1")+CENA("2")*VYROBEK("2");

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;

solve VYROBA MAXIMIZING OPTCENA USING LP;

display VYROBEK.L, OPTCENA.L;
```

Obr. 15: V*ýroba – krok 5*

Pro ověření, že ani tato změna ve značení a zápisu neměla vliv na hodnoty výledků uvádím část výpisu výstupního souboru:

```
----     32 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     32 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 16: *Výroba – krok 5 - výsledek*

5.2.6 Krok 6 – použití matice

Na oba výrobky používáme stejné suroviny. Proto je současné uspořádání, kde informace o spotřebě surovin zjistitíme pouze z definice nerovnic pro jejich omezení, poněkud nepřehledné a vzhledem k dalšímu zobecnění a rozšíření modelu, neefektivní.

Pokud použijeme tabulku, bude zápis daleko jednodušší, přehlednější a otevřenější pro zavádění případných změn, doplnění či rozšíření.

Tabulka se zavádí příkazem **table** (viz text nad rámečkem 4, kap. 6.1.2) pak následuje název tabulky, v našem případě **SPOTREBA**, v kulatých závorkách se uvedou indexující množiny, tedy **SUROVINA** a **CISLO**, a nakonec přidáme upřesňující komentář. Následuje samotná tabulka. Opět nezapomeňme uvádět prvky množiny přesně pod indexy v prvním řádku.

Pro zmenšení obrázku s výpisem zápisu v GAMS jsem kód rozdělila na několik částí a ty, kterých se netýkají změny v tomto kroku, jsem zmenšila. Tento postup použiji i později.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /
     SUROVINA typ suroviny / S1, S2, S3 /;

parameters CENA(CISLO) ceny jednotlivych vyrobku
           / 1  50
             2  30 /
           SKLAD(SUROVINA) limit surovin na sklade
           / S1 700
             S2 300
             S3 450 /;

table SPOTREBA(SUROVINA,CISLO)  spotreba suroviny na vyrobek
      1      2
S1    4      2
S2    2      1
S3    2      2 ;

variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;

equations OMEZ1  dane zasobami suroviny S1
          OMEZ2  dane zasobami suroviny S2
          OMEZ3  dane zasobami suroviny S3
          ZISK   udava vysledny zisk;

OMEZ1..  SPOTREBA("S1","1") * VYROBEK("1") +
         SPOTREBA("S1","2") * VYROBEK("2")=L= SKLAD("S1");

OMEZ2..  SPOTREBA("S2","1") * VYROBEK("1") +
         SPOTREBA("S2","2") * VYROBEK("2")=L= SKLAD("S2");

OMEZ3..  SPOTREBA("S3","1") * VYROBEK("1") +
         SPOTREBA("S3","2") * VYROBEK("2")=L= SKLAD("S3");

ZISK..   OPTCENA=E=CENA("1")*VYROBEK("1")+CENA("2")*VYROBEK("2");

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 17: *Výroba – krok 6*

Jak je vidět, změny se projevily, krom samotného vzniku deklarace tabulky, pouze v oblasti definice nerovnic, kdy konkrétní čísla nahradily výrazy **SPOTREBA(“Si“, “j“)**, kde **Si** je **S1**, **S2** nebo **S3** – tedy prvek množiny **SUROVINA** a **j** je **1** nebo **2** – tedy prvek

množiny **CISLO**.

Pro kontrolu, že právě provedené změny opět neovlivnily číselné výsledky je zde část výpisu výstupního souboru.

```
----     41 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     41 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 18: *Výroba – krok 6 - výsledek*

5.2.7 Krok 7 – využití sumace

Když se nyní podíváme na definici nerovnic, vidíme, že se tam vždy opakují dva téměř shodné sčítance. Jediný rozdíl je v **"1"** a **"2"**, což jsou vlastně prvky množiny **CISLO**. Toho využijeme a zavedeme sčítání pomocí funkce **sum** (viz text nad rámečkem 8, kap. 5.1.2), která využije indexování právě přes množinu **CISLO**.

Tento postup aplikujeme na všechny 3 nerovnice omezení. Funkci **sum** a in ování přes množinu **CISLO** využijeme i v případě definice rovnice **ZISK**.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /
     SUROVINA typ suroviny / S1, S2, S3 /;

parameters CENA(CISLO) ceny jednotlivych vyrobku
           / 1  50
             2  30 /
           SKLAD(SUROVINA) limit surovin na sklade
           / S1 700
             S2 300
             S3 450 /;

table SPOTREBA(SUROVINA,CISLO)  spotreba suroviny na vyrobek
      1       2
S1    4       2
S2    2       1
S3    2       2 ;

variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;

equations OMEZ1  dane zasobami suroviny S1
          OMEZ2  dane zasobami suroviny S2
          OMEZ3  dane zasobami suroviny S3
          ZISK   udava vysledny zisk;

OMEZ1..  SUM(CISLO, SPOTREBA("S1",CISLO) * VYROBEK(CISLO))
                  =L= SKLAD("S1");
OMEZ2..  SUM(CISLO, SPOTREBA("S2",CISLO) * VYROBEK(CISLO))
                  =L= SKLAD("S2");
OMEZ3..  SUM(CISLO, SPOTREBA("S3",CISLO) * VYROBEK(CISLO))
                  =L= SKLAD("S3");
ZISK..   OPTCENA=E=SUM(CISLO, CENA(CISLO)*VYROBEK(CISLO)) ;

model VYROBA / OMEZ1, OMEZ2, OMEZ3, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 19: *Výroba – krok 7*

Abychom si opět ověřili, že nedošlo ke změně ve výsledcích, zde je výpis:

```
----     39 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     39 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 20: *Výroba – krok 7 - výsledek*

5.2.8 Krok 8 – indexování omezení

Po použití indexování přes množinu **CISLO** se ukázalo, že se vlastně nerovnice **OMEZ1**, **OMEZ2** a **OMEZ3** liší již jen v druhu suroviny. Když si uvědomíme, že všechny suroviny máme definované jako prvky množiny **SUROVINA**, nabízí se použít tuto množinu jako indexující a sloučit tak tři nerovnice omezení na suroviny do jedné. Užitečnost této změny se ukáže především při nárůstu počtu surovin – museli bychom pak pro každou nově přidanou surovinu psát novou nerovnici. Takhle přidáme jen další prvek do množiny **SPOTREBA**, do parametru **SKLAD** přidáme její limitní množství a do tabulky **SPOTREBA** uvedeme její spotřebu na výrobu jednotlivých produktů. A tím práce končí, zbytek spočítá GAMS.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1, 2 /
     SUROVINA typ suroviny / S1, S2, S3 /;

parameters CENA(CISLO)    ceny jednotlivych vyrobku
           / 1  50
             2  30 /
           SKLAD(SUROVINA) limit surovin na sklade
           / S1 700
             S2 300
             S3 450 /;

table SPOTREBA(SUROVINA,CISLO)   spotreba suroviny na vyrobek
      1       2
S1    4       2
S2    2       1
S3    2       2   ;

variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;
equations OMEZ(SUROVINA)  dane zasobami suroviny
          ZISK   udava vysledny zisk;

OMEZ(SUROVINA)..  SUM(CISLO, SPOTREBA(SUROVINA,CISLO) * VYROBEK(CISLO))
                  =L= SKLAD(SUROVINA);
ZISK..   OPTCENA =E= SUM(CISLO, CENA(CISLO) * VYROBEK(CISLO)) ;

model VYROBA / OMEZ, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 21: *Výroba – krok 8*

Opět pro kontrolu – výpis. Stále pečeme stejné množství vánoček a mazanců a vydělváme stejné množství peněz. Na číslech řádků, na nichž je umístěn příkaz pro výpočet, vidíme, že se při použití zobecněného zápisu zmenší celková délka programu - tím se zvýší přehlednost a sníží riziko chyby.

```
----     32 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  75.000,    2 150.000


----     32 VARIABLE OPTCENA.L              =     8250.000  vysledny zisk
```

Obr. 22: *Výroba – krok 8 - výsledek*

5.2.9 Krok 9 – změna vstupních dat

Nyní máme obecně zapsaný model a ukážeme si, jak jednoduché je rozšířit tento model o nová data. Přidáme třeba 8 výrobků a 3 suroviny.

Při zápisu množin **CISLO** a **SUROVINA** by chom mohli využít stávající systém, tedy oddělování jednotlivých prvků čárkami. Výsledek by vypadal takto:

```
set CISLO poradove cislo vyrobku / 1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
    SUROVINA typ suroviny / S1, 2, S3, S4, S5, S6/ ;
```

Tento styl zápisu je však mírně zdlouhavý a pro řádově vyšší počty prvků množiny by už byl nepřehledný.

Naštěstí existuje pravidlo tzv. hvězdičkové notace, kdy GAMS hledá rozdíly mezi dvěma návěstími (prvky množiny) a pokud se tato dvě návěstí liší pouze v číslicích a jestliže číslo v předchízejícím návěstí je menší, než v následujícím, potom jsou prvky množiny určeny společným textem a čísly v uvedených mezích. [1]

Prvky množiny **CISLO** zdánlivě žádný společný text nemají, ale GAMS chápe i žádný text jako společný. U prvků množiny **SUROVINA** je pak jasné, že společný text je S a meze jsou 1 až 6.

Výledný zápis s použitím hvězdičkové notace pak vypadá takto:

```
set CISLO poradove cislo vyrobku / 1 * 10/
    SUROVINA typ suroviny / S1* S6/ ;
```

Zavedení nových výrobků a surovin vyžaduje ještě několik úprav programu – pro každý nový výrobek musíme do parametru **CENA(CISLO)** zavést jeho cenu a pro každou novou surovinu musíme do parametru **SKLAD(SUROVINA)** zavést její množství, které máme na skladě. Poslední nutnou úpravou je rozšíření tabulky **SPOTREBA(SUROVINA, CISLO)**. Tím změny v programu končí. Definici proměnných, rovnice, nerovnic ani modelu samotného toto rozšíření neovlivní – právě díky vhodně zvoleným zobecňovacím úpravám.

Na další stránce je uvedena první část kódu, kde došlo k právě popsaným změnám.

```
$TITLE Vyrobni plan
sets CISLO  poradove cislo vyrobku / 1 * 10 /
     SUROVINA typ suroviny / S1 * S6 /;
parameters CENA(CISLO) ceny jednotlivych vyrobku
           /  1  50
              2  30
              3   5
              4  17
              5   9
              6  48
              7  23
              8  12
              9  31
             10  54 /
           SKLAD(SUROVINA) limit surovin na sklade
           / S1 650
             S2 320
             S3 400
             S4 500
             S5 600
             S6 300/;

table SPOTREBA(SUROVINA,CISLO)  spotreba suroviny na vyrobek
      1    2   3   4    5   6   7   8   9   10
S1    4    2   1   1
S2    2    1                    1
S3    2    2       3
S4                 2    1   4   5   2   1   5
S5                 1    2   8   7   8
S6                                     12      ;
```

Obr. 23: *Výroba – krok 9 – část A*

A pro ověření, že se definice modelu nezměnila, uvádím druhou část kódu z kroku 8 a 9.

```
variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu
          OPTCENA vysledny zisk;

equations OMEZ(SUROVINA)  dane zasobami suroviny
          ZISK   udava vysledny zisk;

OMEZ(SUROVINA)..  SUM(CISLO, SPOTREBA(SUROVINA,CISLO) * VYROBEK(CISLO))
                  =L= SKLAD(SUROVINA);
ZISK..   OPTCENA =E= SUM(CISLO, CENA(CISLO) * VYROBEK(CISLO)) ;

model VYROBA / OMEZ, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 24: *Výroba – krok 8 – definice proměnných, rovnice, souboru nerovnic a modelu*

```
positive variables VYROBEK(CISLO) mnozstvi vyrobku jednotlivych typu;
         variables OPTCENA vysledny zisk;
equations OMEZ(SUROVINA)  dane zasobami suroviny
          ZISK   udava vysledny zisk;

OMEZ(SUROVINA)..  SUM(CISLO, SPOTREBA(SUROVINA,CISLO) * VYROBEK(CISLO))
                  =L= SKLAD(SUROVINA);
ZISK..   OPTCENA =E= SUM(CISLO, CENA(CISLO) * VYROBEK(CISLO)) ;
model VYROBA / OMEZ, ZISK /;
solve VYROBA MAXIMIZING OPTCENA USING LP;
display VYROBEK.L, OPTCENA.L;
```

Obr. 25: *Výroba – krok 9 – část B*

GAMS je natolik schopný optimalizační nástroj, že zjistí, které výrobky se oplatí vyrábět a které ne. Takže ve výpisu výsledků nejsou uvedeny všechny výrobky, jejichž pořadová čísla jsme uvedli v množině **CISLO**, ale jen ty, které jsou z hlediska maximalizace zisku přínosné:

```
----     43 VARIABLE VYROBEK.L  mnozstvi vyrobku jednotlivych typu

1  120.000,    2   80.000,    3   10.000,    6   75.000,    9   25.000
10  35.000


----     43 VARIABLE OPTCENA.L             =    14715.000  vysledny zisk
```

Obr. 26: *Výroba – krok 9 - výsledek*

Vidíme, že jsme vyrobili 120 vánoček, 80 mazanců a pak 10 výrobků 3. typu, 75 výrobků 6. typu, 25 výrobků 9. typu a 35 výrobků 10. typu. Výrobky 4., 5., 7. a 8. typu jsme nevyráběli vůbec. Za celou zakázku jsme utržili 14 715Kč.

5.3 Řešení motivačních příkladů

V kapitole 5 jsme uvedli zadání dvou příkladů, společně se dvěma otázkami. Nyní na tyto otázky odpovíme.

5.3.1 Splátkový kalendář

Tento příklad jsem vybrala jako příklady využití funkce **ord** (viz níže). Navíc je zde vidět, že GAMS se dá použít i na výpočty jiných, než jen optimalizačních úloh.

Zadání:

Banka nabízí hypotéku se základní úrokovou sazbou 6% ročně plus dodatečný úrok od 0% do 2% s krokem 0,25% jako rozlišení rizikovosti půčky - čím rizikovější, tím vyšší úroky. Při výši hypotéky 1 000 000 Kč je možné splácení rozložit na 10, 20 nebo 30 let. Po celou dobu splácení je pak garantována konstantní výše splátek při měsíčním splácení.

Otázka:

Jak vysoká bude splátka pro kterou skupinu podle rizika a při jaké době splácení?

Matematický zápis:

Označme **P(i)** velikost dlužné částky, kterou zbývá ještě uhradit po zaplacení i-té splátky. Předpokládejme, že k vyrovnání dluhu dojde za **R** let a během jednoho roku proběhne **K** splátek. Velikost splátek označíme **S** a její hodnotu určíme z následujících rovnic: [1]

$$P(0) = 100$$

$$P(1) = P(0) * (1 + \frac{UROK}{K}) - S$$

:

$$P(T) = P(T-1) * (1 + \frac{UROK}{K}) - S = 0$$

Odtud vyjádříme hodnotu splátek **S** jako:

$$S = 1000000 * \frac{\frac{UROK}{K}}{[1 - (1 + \frac{UROK}{K})^{(-K*R)}]}$$

Zápis do jazyka GAMS:

Při zadávání tohoto příkladu do GAMSu využijeme několik nových postupů.

První z těchto novinek jsou příkazy **option limrow = 0** a **option limcol = 0**. První z příkazů určuje, že ve výpisu výsledků nebudou uváděny rovnice či nerovnice, které jsme pro výpočet zadali. Druhý příkaz pak určuje, že se nebudou vypisovat hodnoty proměnné (dokud sami nenapíšeme, že se nějaká hodnota vypsat má).

Druhou novinkou je použití funkce **ord(U)**. Ta určuje pořadí prvku v množině **U**. Je-li tedy v našem případě prvkem možiny **U** číslo 1, je také prvním členem množiny a **ord(U) = 1**. Předepsaná rovnost probíhá všechny prvky množiny **U** a hodnoty **ord(U)** jou tedy v našem případě všechny celé hodnoty od 1 do 9 (pro množinu **U**, která má 9 prvků). [1]

Poslední novinka je slovo **power** ve významu mocniny. Za toto klíčové slovo se pak do kulatých závorek vkládají definice členu, který má být umocněn, a jeho mocniny, oddělené čárkou.

Dále využijeme poznatků z předešlých příkladů. Na začátku programu zavedeme množiny **U** (velikost dodatečného úroku) a **R** (doba splatnosti dluhu). Množina **U** má prvky 1*9, množina R má prvky 10-let, 20-let a 30-let. Následuje zavedení a definice parametrů **doba(R)** (prvkům z množiny **R** přiřadíme jejich hodnoty, tzn. 10, 20 a 30), **urok(U)** (zavede výpočet ročního úroku v závislosti na prvcích množiny **U**), **K** (počet splátek) a **S** (velikost splátky).

Hodotu parametru **urok(U)** spočteme jako $urok(U) = 0{,}06 + 0{,}0025 * (ord(U) - 1)$. **K** položíme rovno 12 (splátky měsíčně). Velikost splátky **S** spočítáme podle vzorečku uvedeného výše. Na závěr necháme vypsat hodnoty **S**. Výsledný zápis vypadá takto:

```
$TITLE Stanovani splatek
$OFFSYMXREF
option limrow = 0
option limcol = 0

sets U velikost dodatecneho uroku / 1*9 /
     R Doba splatnosti dluhu / 10-let, 20-let, 30-let /;


parameters doba(R) doba splatnosti / 10-let 10
                                     20-let 20
                                     30-let 30 /
           urok(U) rocni uroky
           K pocet splatek za rok
           S velikost splatky ;

urok(U) = 0.06 + 0.0025*(ord(U)-1);
K = 12;

S(U,R) = 1000000*urok(U)/K/(1-power(1+urok(U)/K, -K*doba(R)));

display S;
```

Obr. 27: *Splátkový kalendář – zápis v GAMS*

Po uložení, spuštění a vyřešení programu najdeme ve výpisu výsledků (díky zvoleným oemezním) pouze rekapitulaci zadaného programu s očíslovanými řádky, výpis hodnoty parametru **S** a technické údaje o velikosti programu, výpočetním čase, verzi GAMSu a umístění vstupního a výstupního souboru v počítači. Výpis hodnoty **S** bude vypadat takto:

```
----     22 PARAMETER S  velikost splatky

        10-let      20-let      30-let        urok

1    11102.050    7164.311    5995.505       6.000
2    11228.010    7309.282    6157.172       6.250
3    11354.798    7455.731    6320.680       6.500
4    11482.411    7603.640    6485.981       6.750
5    11610.848    7752.989    6653.025       7.000
6    11740.104    7903.760    6821.763       7.250
7    11870.177    8055.932    6992.145       7.500
8    12001.063    8209.486    7164.122       7.750
9    12132.759    8364.401    7337.646       8.000
```

Obr. 28: *Splátkový kalendář - výsledek*

V prvním sloupci je uvedeno číslo řádku, které odpovídá hodnotě **ord(U)**. Ve druhém, třetím a čtvrtém sloupci jsou uvedeny výše splátek pro délku splácení 10, 20 a 30 let a v řádcích pro jednotlivé rizikové skupiny. V posledním sloupci je pak uveden výsledný roční úrok pro jednotlivé kategorie rizikovosti. Tato tabulka je odpovědí na úvodní otázku.

5.3.2 Sklad

Na tomto příkladě si ukážeme nové klíčové slovo **scalars** (viz níže), práci s větším počtem proměnných a minimalizaci účelové funkce.

Zadání: [1]

Majitel skladu chce zvýšit své zisky. Kapacita skladu je omezená a majitel obchoduje se zbožím, jehož cena se mění v závislosti na ročním období (čtvrtletí). Na uskladnění zboží jsou navíc v průběhu roku různé náklady.

Otázka:

Kolik zboží má majitel kdy nakoupit a kdy kolik prodat, aby byly jeho náklady minimální a zisk maximální?

Matematický zápis:

Předpokládejme, že závislosti jsou lineárními funkcemi. Model budeme tedy řešit metodou lineárního programování.

Označme si:

- **z(t)** je zásoba ve skladu v čase t
- **pz(t)** je počáteční zásoba ve skladu v čase t ($pz(t) = 0; t = 2,3,...$)
- **p(t)** je zboží prodávané v čase t
- **n(t)** značí zboží nakoupené v čase t
- **c(t)** značí cenu zboží v čase t
- **s** jsou skladovací náklady na jendotku zboží
- **c** je kapacita skladu v jednotkách

Pak hledáme minimum výrazu $\sum_t c(t) * [n(t) - p(t)] + s * z(t)$, který značí součet hodnoty skladovaného zboží a nákladů na uskladnění daného zboží za čas t. Pokud tento výraz minimalizujeme, získáme prázdnější sklad, menší náklady na skladování a větší množství prodaného zboží, tím pádem větší příjmy a tedy i celkový zisk bude větší, dokonce maximální. Musíme však ještě uvést podmínky, nutné ke konstrukci modelu:

$p(t) \geq 0$, $n(t) \geq 0$, $pz(t) \geq 0$ – množství zboží prodaného, nakoupeného a skladovaného nesmí být záporné.

$z(t) \leq c$ – skladované množství zboží nesmí přesáhnout kapacitu skladu

$z(0) = 0$ – v čase 0 nenavážíme žádné zásoby

$z(t) = z(t-1) + n(t) - p(t) + pz(t); t \geq 1$ – určuje množství zásob v závislosti na předešlém objemu, nakoupených a prodaných surovinách a na počátečním množství suroviny ve skladu.

Zápis do jazyka GAMS:

Řekněme, že sklad má kapacitu 200 jednotek zboží. 30 jednotek je ve skladě již umístěných.

Majitel nakupuje a prodává za tržní cenu, takže se v daném období nákupní a prodejní cena neliší. V prvním čtvrtletí má jednotka zboží cenu 100 Kč, ve druhém 300 Kč, ve třetím 220 Kč a ve čtvrtém 155 Kč. Náklady na uskladnění jsou po celý rok stejné – na jednotku zboží 100 Kč za čtvrtletí.

Při zapisování do GAMSu použijeme novou formulaci: Pro zadávání konstant využijeme klíčové slovo **scalars**. Pracuje se s ním stejně, jako s jinými klíčovými slovy – za ním následuje název konstanty, její doplňující popis a v lomítkových závorkách hodnota, kterou nabývá. Zadávání se ukončuje opět středníkem za poslední definicí konstanty. Při použití obvyklého postupu vypadá výsledný zápis následovně.

```
$TITLE Sklad
options limrow = 0
options limcol = 0

set t ctvrtleti / Q-1 * Q-4 /;
parameters cena(t) prodejni cena (za jednotku) / Q-1 100
                                                 Q-2 300
                                                 Q-3 220
                                                 Q-4 155 /
           pzasoba(t) pocatecni zasoba v jednotkach / Q-1 30 /;
scalars skladne cena za uskladneni (korun za ctvrtleti) / 100 /
        kapacita kapacita skladu v jednotkach / 200 /;
variables zasoba(t) zasoba v case t v jednotkach
          prodej(t) prodane zbozi v case t v jednotkach
          nakup(t) nakoupene zbozi v case t v jednotkach
          naklady celkove naklady v korunach;
positive variables zasoba
                   nakup
                   prodej;
equations sb(t) rovnovaha zbozi v case t v jednotkach
          at celkova cena;
sb(t).. zasoba(t) =E= zasoba(t-1) + nakup(t) - prodej(t) + pzasoba(t);
at.. naklady =E= sum(t, cena(t)*(nakup(t)-prodej(t)) + skladne*zasoba(t));
zasoba.up(t) = kapacita;
model sklad / all /;
solve sklad minimizing naklady using LP;
```

Obr. 29: *Sklad – zápis v GAMS*

Výpis výsledků jako obrázek zde výjimečně uvádět nebudu, neb je velmi dlouhý. Zjištění tedy shrnu slovně. Zjistili jsme, že:

- Hodnota účelové funkce, tedy nákladů, je -23 000 Kč (záporné náklady značí zisk).
- Rovnice **sb**, tedy rovnováha zboží v čase t, nabývá hodnoty 30 v prvním čtvrtletí, v dalších čtvrtletích je nulová.
- Rovnice **at**, tedy celková cena, nabývá nulové vlastní hodnoty, ale marginální hodnota

je 1, protože pokud se hladina této rovnice změní o 1, změní se o 1 také duální proměnná.

- Proměnná **zasoba** nabývá hodnoty 200 jednotek uskladněného zboží v prvním čtvrtletí a v dalších třech čtvrtletích nabývá hodnoty 0, tedy neoplatí se sklad využívat.
- Proměnná **prodej** tak nabývá hodnoty 200 ve druhém čtvrtletí, kdy suroviny, které jsme celé první čtvrtletí skladovali, prodáme, neboť je právě ve druhém čtvrtletí prodejní cena nejvyšší. Jinak nabývá hodnoty 0.
- Poslední proměnná **nakup** ukazuje, že v prvním čtvrtletí nakoupíme 170 jednotek zboží, čímž doplníme počátečních 30 jednotek do plné kapacity skladu.

Odpověď na úvodní otázku zní: Majitel má v prvním čtvrtletí k již uskladněným 30 kusům dokoupit 170 kusů zboží a všech 200 kusů pak ve druhém čtvrtletí prodat. Dále už by obchodovat neměl. Pak budou jeho náklady minimální a tím zisky maximální.

6 Závěr

Svou práci jsem podle zadání tématu bakalářské práce koncipovala jako příručku pro začátečníky v používání optimalizačních modelovacích jazyků. Předpokládám, že čtenář buď zjistil nebo již ví, že optimalizace je zajímavý nástroj a že se s ní dají dělat různá kouzla. Neví však zatím nic o tom, jak se tato kouzla provádí pomocí modelovacích jazyků. A právě tato tajemství měla má práce poodhalit.

Po přečtení by zájemce měl ovládat základy práce v prostředí GAMS, podrobně popsaném na jednoduchých příkladech, aby nedošlo k opomenutí nějakého úkonu v průběhu práce na modelu.

Z vlastní zkušenosti vím, jak demotivující je prohlášení „Práce s touto knihou předpokládá základní znalosti programování v jazyce...“ či jiné podobné věty, uváděné na začátcích odborných knih o různých programovacích jazycích či jiných vývojových prostředích. Takováto prohlášení se vyskytují dokonce i v příručkách určených přímo začátečníkům. Jenže když pak při čtení textu člověk narazí na věty typu „stejně jako v (tom či onom) jazyce, i tady se použije (takový a onaký) prostředek...“ nebo „ze zkušeností s (tím a tím) programem víme, že....“ a nemá přitom žádnou zkušenost s „tím či oním“, „takovým a onakým prostředkem“ či „tím a tím programem“, většinou je v koncích a pokud v sobě nenajde opravdu silnou motivaci, knihu zavře, odloží a uchýlí se k hledání informací na internetu. Tedy na ne vždy spolehlivém zdroji, co se pravdivosti a správnosti informací týče. Pokud takto zklamaný člověk narazí na popis postupu či návod, který s vypětím sil použije v pro něj zatím zcela neznámém prostředí a tento postup ho zklame, bude již jen těžko zkoušet další a další příklady a jen opravdu odhodlaný člověk se tak sám svépomocí naučí pracovat s oním zoufale tajemným programem..

Právě tomuto zklamání z vazby na předpokládané, avšak neexistující, zkušenosti a znalosti jsem se chtěla vyhnout a proto jsem zvolila poněkud pomalejší, pro někoho možná zbytečně vysvětlující postup přiblížení práce s GAMSem.

Tento optimalizační nástroj jsem si zvolila pro jeho jednoduchost, názornost a přehlednost při psaní modelu. GAMS totiž nevyužívá tlačítek, záložek, skrytých nabídek a nástrojů, které mizí do rohů či do seznamů na bočním panelu, když je zrovna nepoužíváme. Kód v GAMSu vidíme stále celý před sebou. To umožňuje kontrolu zadaných čísel, proměnných a pod. bez překlikávání do jiných podnabídek a pod., což může být někdy zdrojem potíží, když změníme čísla ve většině záložek a např. v jedné tu změnu provést zapomeneme.

Doufám, že člověk, který se dostal při čtení této práce až sem si z ní něco odnese, začne se zajímat o to, kde by se daly sehnat další informace, jak své znalosti prohloubit, jak se naučit řešit složitější modely a jak zapracovat optimalizaci do své praxe.

7 Literatura

[1] FRYŠOVÁ, D., HANTYCH, P., CHARAMAZA, P., KADLČÁK, R., KLICNAR, M., MYDLO, M., PAVLICOVÁ, M., POPELA, P. TLUSTÝ, P., VEČEŘ, J. *Modelovací systém GAMS*. 1. vyd. Praha: MFF UK, 1993. 127 s.

[2] *GAMS home page* [online]. 2009, [cit. 2010-05-10]. Dostupné z: <http://www.gams.com/>.

[3] KLAPKA, J., DVOŘÁK, J., POPELA, P. *Medoty operačního výzkumu*. 2. vyd. Brno: VUTIUM, 2001

[4] POPELA, P. *Lineární programování v kostce* [CD-ROM]. 2000 [cit. 2010-05-20].

[5] POPELA, P. *Nonlinear Programing*. [CD-ROM]. 2002 [cit. 2010-05-20].

[6] WINSTON, W. L. *Operations Research, Application and Algorithm*. 2nd. printing, 1991. Belmont (California): Duxbury Press,. 350 s.