GAMS — pierwsze kroki

Marian Kostrzewski, WEiA, Politechnika Gdańska

Technika systemów – materiały pomocnicze

Publikacja jest w stanie jakim jest. Licząc, że mimo wszystko może okazać się pożyteczną udostępniam ją publiczności do użytku. Przyjmuję wszelkie uwagi co do treści i redakcji broszury.

Zamierzam zawrzeć w niej więcej informacji niż wynika to z aktualnego spisu treści. Póki co zawartość odpowiada popełnionemu przeze mnie brykowi sprzed 3 lat...

Spis treści

1 GAMS

GAMS (General Algebraic Modelling System) jest językiem wysokiego poziomu umożliwiającym zgrabne formułowanie problemów optymalizacji za pomocą zwięzłych formuł. Zapis jest czytelny dla osoby formułującej problem, nadto daje się łatwo modyfikować i przenosić pomiędzy różnymi środowiskami komputerowymi (sprzętowymi i operacyjnymi). Postać źródłowa jest niezależna od

stosowanego dalej we właściwych obliczeniach motoru (ang. *solver*). Główni autorzy języka GAMS to Anthony Brook, Paul van der Eijk i Alexander Meeraus. Opracowanie GAMS'a finansowane było przez Bank Światowy (Bank's Research Comittee, projekty RPO 671–58 i 673–06).

Dystrybujcą pakietu zajmuje się firma:

GAMS Development Corporation	
1217 Potomac Street N.W.	
Washington, DC 20007	
Phone:	(202)342–0180
Fax:	(202)342–0181
Email:	`gams@gams.com`
Web site:	`http://www.gams.com/`

```
Dalej w treści tego opracowania występować będą fragmenty
zapisane w języku GAMS. Dla ich wyróżnienia wtedy stosowana
będzie czcionka o stałym odstępie pomiędzy znakami
(jak w tym akapicie).
```

Treść tej broszury opracowano na podstawie książki "GAMS release 2.25, A User's Guide" autorstwa A.Brooke, D.Kendrick, A. Meeraus, The Scientific Press, rok wydania 1992. Przykłady modeli GAMS pochodzą ze strony w Internecie `http://www.gams.com/`. Ponadto wykorzystano strony "man" na temat GAMS'a pochodzące ze stron www Cornell Theory Center.

1.1 Główne założenia języka GAMS

Lata pięćdziesiąte i pojawienie się komputerów dały początek gwałtownemu rozwojowi algorytmów i programów komputerowych. Opracowano wiele metod rozwiązywania różnych problemów matematycznych. Gdy przeprowadzono analizę stosowania różnych programów okazało się, że lwią część czasu poświęcanego na rozwiązanie problemów zabiera odpowiednie przygotowanie danych i opracowanie wyników. Każdy model wymagał wielogodzinnych analiz i programowania; wszystko po to aby przygotować dane w formacie wymaganym przez program realizujący obliczenia. W takim stanie rzeczy odszukanie i wyszukanie błędów było znacznie utrudnione — zazwyczaj kto inny układał model, kto inny wprowadzał dane. Pojawiło się dodatkowe zadanie — sprawdzenie, że wprowadzone dane są poprawne.

Język GAMS był czymś na kształt próby uporządkowania tego stanu rzeczy. Dawał pewien sformalizowany język umożliwiający sformułowanie pewnej klasy problemów. Autorzy przygotowujący projekt postawili sobie następujące zadania:

- opracować język umożliwiający zwarty opis modeli wielkich systemów;
- język powinien umożliwiać w prosty sposób dokonywanie modyfikacji modelu;
- informacja o relacjach powinna być jednoznaczna;
- opis modelu nie powinien zależeć od programu rozwiązującego.

1.2 Model w języku GAMS

Zanim podejmiemy próbę formalizacji definicji języka GAMS spróbujemy przedstawić przykład jego użycia na prostym problemie optymalizacji. Przykład jest czystej postaci zagadnieniem transportowym programowania liniowego. Pokażemy sformułowanie zagadnienia w postaci zapisu matematycznego, dalej przedstawimy ten sam opis dokonany w języku GAMS w postaci bez zbędnych komentarzy i z drobiazgowym komentarzem ilustrującym poszczególne zapisy.

Matematycznie problem formułowany jest następująco:

Indeksy:

i = wytwórcy towaru
j = odbiorcy towaru

dane:

a_i = zdolność produkcyjna i–tego wytwórcy (w sztukach)
b_j = wymagana chłonność rynku j–tego odbiorcy (w sztukach)
c_{ij} = koszt transportu towaru od i–tego wytwórcy do j–tego odbiorcy (\$/sztukę)

zmienne decyzyjne

x_{ij} = ilość towaru do przesłania od i–tego wytwórcy do j–tego odbiorcy (w sztukach); gdzie $\forall i \, \forall j \, x_{ij} \geq 0$

ograniczenia

badaj zdolność produkcyjną i-tego wytwórcy (w sztukach):

$$\forall i \sum_j x_{ij} \leq a_i$$

zagwarantuj minimalne dostawy dla j-tego odbiorcy (w sztukach):

$$\forall j \sum_i x_{ij} \geq b_j$$

funkcja celu

minimalizuj (w k\$, i.e. w tysiącach \$):

$$Q = \sum_i \sum_j c_{ij} x_{ij}$$

Proszę zwrócić uwagę, że ten prosty przykład prezentuję pewną praktykę formułowania modelu, dodajmy — zgodną ze sposobem konstrukcji modeli w języku GAMS. Po pierwsze, wszystkie wielkości modelu ją nazwane, zdefiniowane i pogrupowane; określony jest ich typ. Po drugie, porządek wybrano tak, że żadna z wielkości nie została użyta zanim została zdefiniowana. Po trzecie, wszędzie nazwane są jednostki miary. Po czwarte, jednostki dobrano w taki sposób (k\$) aby program optymalizujący nie musiał zmagać się w jednym zadaniu z liczbami różniącymi się o kilka rzędów wielkości.

W języku GAMS wprowadzono następującą terminologię:

wskaźniki	=	`SETS`
dane	=	`PARAMETERS`
zmienne decyzyjne	=	`VARIABLES`
ograniczenia i funkcja celu	=	`EQUATIONS`

Model zagadnienia transportowego zapisany w języku GAMS mocno przypomina opisany powyżej model algebraiczny. Podstawowa różnica to fakt, że wersja GAMS może być odczytana i przetworzona przez komputer.

Sformułowanie zadania wymaga podania kilku liczb. Załóżmy, że nasz przykład dotyczy instytucji, która ma dwa zakłady produkujące konserwy (Seattle, San Diego) oraz trzech odbiorców określonych jako Nowy Jork, Chicago i Topeka. (Przykład pochodzi z książki: Dantzig G. B., *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963, Rozdział 3–3.)

Założono, że odległości podano w tysiącach mil. Koszt przesyłki jest stały (nie zależy od miejsca położenia producenta i odbiorcy) i wynosi 90 dolarów za transport jednej partii na odcinku długości 1000 mil. (Zachowano oznaczenia pochodzące z danych opublikowanych przez firmę GAMS).

	Markets			
	New York	Chicago	Topeka	
Plants	Distances			Supply
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
Demand	325	300	275	

Mały słowniczek (może się przydać):

demand	wielkość zamówiona
distance	odległość
market	odbiorca
plant	zakład produkcyjny
supply	moc produkcyjna

Poniżej przedstawiono modej sformułowany w języku GAMS.

```
SETS
 I  canning plants / SEATTLE, SAN-DIEGO /
 J  markets        / NEW-YORK, CHICAGO, TOPEKA / ;
PARAMETERS
 A(I)  capacity of plant i in cases
     /    SEATTLE     350
          SAN-DIEGO   600  /
 B(J)  demand at market j in cases
     /    NEW-YORK    325
          CHICAGO     300
          TOPEKA      275  / ;
 TABLE D(I,J)  distance in thousands of miles
                   NEW-YORK       CHICAGO      TOPEKA
     SEATTLE          2.5           1.7          1.8
     SAN-DIEGO        2.5           1.8          1.4;
```

```
SCALAR F  freight in dollars per case per thousand miles  /90/;
 PARAMETER C(I,J)  transport cost in thousands of dollars per case;
           C(I,J) = F * D(I,J) / 1000;
VARIABLES
      X(I,J)  shipment quantities in cases
      Z       total transportation costs in thousands of dollars;
 POSITIVE VARIABLE X ;
EQUATIONS
      COST        define objective function
      SUPPLY(I)   observe supply limit at plant i
      DEMAND(J)   satisfy demand at market j;
 COST ..        Z  =E=  SUM((I,J), C(I,J)*X(I,J));
 SUPPLY(I) ..   SUM(J, X(I,J))  =L=  A(I);
 DEMAND(J) ..   SUM(I, X(I,J))  =G=  B(J);
MODEL TRANSPORT /ALL/;
SOLVE TRANSPORT USING LP MINIMIZING Z;
```

Poniżej raz jeszcze rozpiszemy ten model tym razem z komentarzem ilustrującym znaczenie użytych w zapisie sformułowań. Zaczniemy jednak od rozważań natury ogólnej.

2 Struktura modelu

W odniesieniu do dalszej treści rozdziału omawiając podstawowe elementy modelu w języku GAMS odnosić się będziemy do przykładu przedstawionego wyżej. Podstawowe (istnieją także zaawansowane) składowe modelu to:

- na wejściu
  - zbiory (`SETS`)
    - deklaracja
    - określenie (wyliczenie) elementów
  - dane (`PARAMETERS`, `TABLES`, `SCALARS`)
    - deklaracja
    - przypisanie wartości
  - zmienne (`VARIABLES`)
    - deklaracja
    - przypisanie typu
    - (opcjonalnie) przypisanie ograniczeń i/lub wartości początkowych
  - równania (`EQUATIONS`)
    - deklaracja
    - definicja
  - instrukcjae `MODEL`
  - instrukcja `SOLVE`

  - (opcjonalnie) instrukcja `DISPLAY`

- na wyjściu
  - echo na drukarkę
  - mapa referencji
  - listing z równaniami
  - raport ze statusem
  - wyniki

Na początek kilka uwag generalnych

1. Model GAMS to zbiór zdań w języku GAMS. Jedyna reguła jaka reguluje porządek zdań to konieczność zadeklarowania obiektów zanim zostaną użyte.

2. Instrukcje GAMS mogą być ukszałtowane typograficznie zgodnie z upodobaniami użytkownika. Dopuszcza się instrukcje wieloliniowe, puste linie i wiele instrukcji w jednej linii.

3. Jeżeli jesteś początkującym użytkownikiem GAMS's powinieneś każdą instrukcję kończyć znakiem średnika (tak jak w pokazanym przykładzie).

4. Kompilator GAMS'a nie rozróżnia pomiędzy dużymi i małymi literami. Styl wdrożony w przykładach to używanie wielkich liter do elementów należących do języka GAMS i obiektów deklarowanych w konkretnym modelu GAMS. Proponuje się zarezerwować małe litery jedynie dla celów dokumentacyjnych.

5. Dokumentacja ma decydujące znaczenie dla użyteczności modeli matematycznych. Wydaje się bardziej pożytecznym wkomponować ją w model niż napisać oddzielnie. Istnieją przynajmniej dwie możliwości dołączania dokumentacji do modeli GAMS'a:

   - każda linia rozpoczynająca się od znaku "`*`" jest ignorowana przez kompilator GAMS'a (traktowana jak komentarz);
   - tekst dokumentujący może być wpleciony bezpośrednio w instrukcje GAMS'a; wszystkie opisy przedstawione w przykładzie wyżej małymi literami są taką formą dokumentacji.

6. Jeśli przyjrzeć się liscie składowych wejścia widać, że tworzenie obiektów GAMS'a składa się z dwóch kroków:

   - deklaracji, czyli stwierdzenie, że obiekt istnieje i nadanie mu nazwy;
   - przypisania lub definicji, czyli nadanie obiektowi określonej wartości lub formy.

   W przypadku równań należy w odrębnych instrukcjach zadeklarować obiekt i zdefiniować go. Jednakże dla wszelkich obiektów GAMS'a istnieje opcja dokonania deklaracji i przypisania jedną instrukcją lub w instrukcjach odrębnych.

7. W przypadku równań należy w odrębnych instrukcjach zadeklarować obiekt i zdefiniować go. Jednakże dla wszelkich obiektów GAMS'a istnieje opcja dokonania deklaracji i przypisania jedną instrukcją lub w instrukcjach odrębnych.

8. Nazwy nadawane obiektom modelu muszą zaczynać się od litery, dalej składać się z liter lub cyfr. Maksymalna długość nazwy to 10 znaków.

Wróćmy do komentowania przykładowego modelu nakładając na tekst konwencję struktury pokazaną wyżej.

2.1 ZBIORY (SETS)

W opisie modelu to następujący fragment:

```
SETS
 I   canning plants   / SEATTLE, SAN-DIEGO /
 J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;
```

GAMS pozwala na bezpośrednią specyfikację indeksów (tutaj `I` oraz `J`); po określeniu i nazwaniu zbiorów wyliczamy ich elementy składowe. Wyliczenia dokonujemy wewnątrz pary znaków "`/`" oddzielając kolejne elementy listy przecinkami. Należy zwrócić uwagę na zapis SAN-DIEGO i NEW-YORK. Znaki odstępu nie są dopuszczane w listach stąd obecność myślnika w konstrukcji wieloczłonowego idendyfikatora.

Nie było konieczności połączenia utworzenia zbiorów `I` oraz `J` w jednym zadaniu. Z równym skutkiem można było użyć następującego zapisu:

```
SET I canning plants / SEATTLE, SAN-DIEGO / ;
SET J markets        / NEW-YORK, CHICAGO, TOPEKA / ;
```

Użyte terminy `SET`/`SETS` mają wskazywać związek jeden/wiele. Jednak kompilator GAMS'a traktuje je dokładnie w taki sam sposób.

Istnieje wygodny sposób określania elementów zbioru gdy tworzą one ciąg. Oto przykład:

```
SET T okresy czasu / 1993 * 2000 / ;
SET M maszyny      / MASZ1 * MASZ8 / ;
```

W ten sposób można zapisać dane, które inaczej należałoby pracowicie wyliczać jak poniżej:

```
SET T okresy czasu / 1993, 1994, 1995, 1996,
                     1997, 1998, 1999, 2000 / ;
SET M maszyny      / MASZ1, MASZ2, MASZ3, MASZ4,
                     MASZ5, MASZ6, MASZ7, MASZ8 /;
```

Zwróćmy uwagę, że elementy zbioru T są ciągami znaków (character string) a nie liczbami. Inny wygodny element języka to możliwość utworzenia listy *aliasów* (przypisanie innej nazwy już zadeklarowanym zbiorom). Na przykład:

```
ALIAS (T, TP)
```

Zbiór `T` i `TP` to ten sam zbiór.

2.2 Dane

W modelu zadania transportowego pokazano trzy różne formaty wprowadznia danych:

- listy;
- tablice;
- bezpośrednie przypisanie.

2.2.1 Wprowadzenie poprzez listy (PARAMETERS)

Pierwszy format ilustruje pierwsza instrukcja `PARAMETERS`:

```
PARAMETERS
       A(I)  capacity of plant i in cases
          /    SEATTLE    350
               SAN-DIEGO   600 /
       B(J)  demand at market j in cases
          /    NEW-YORK    325
               CHICAGO     300
               TOPEKA      275 / ;
```

Instrukcja deklaruje istnienie dwóch parametrów, nadaje im nazwy `A` i `B`, i deklaruje ich dziedziny. Pojawia się również opis dokumentujący każdy parametr i przypisanie wartości dla `A(I)` i `B(J)` dla każdego elementu `I` oraz `J`. Równie dobrze można to zapisać jak dwa zdania:

```
PARAMETER A(I)  capacity of plant i in cases
       / SEATTLE   350
        SAN-DIEGO  600 / ;
PARAMETER B(J) demand at market j in cases
       / NEW-YORK  325
         CHICAGO   300
         TOPEKA    275 / ;
```

Oto kilka wskazówek związanych z tym formatem:

1. Lista z elementami i parametrami określającymi wartość może być wprowadzona praktycznie w dowolny sposób. Jedyne ograniczenie to należy pamiętać o znakach "`/`" na początku i końcu, a pary element–wartość powinny być oddzielone przecinkiem bądź należy je wpisać w osobych liniach.
2. Nie ma średnika oddzielającego listę element–wartość nazwy, dziedziny i opisu. Wynika to z faktu, że każda definicja musiałaby mieć postać taką jak ma obecnie.
3. Kompilator GAMS'a weryfikuje, czy podany *element* występuje w zbiorze.
4. Wartość domyślna dla wszystkich parametrów to zero. Z tego powodu wystarczy jedynie podać niezerowe dane. Kolejność par element–wartość może być dowolna.
5. Skalar traktowany jest jako parametr, który nie ma dziedziny. Może być zadeklarowany instrukcją SCALAR, zawierającą jedynie wartość. Np.
   `SCALAR F freight in dollars per case per thousand miles /90/ ;`

2.2.2 Wprowadzenie poprzez tablice (TABLE)

Oto odpowiedni fragment tekstu:

```
TABLE D(I,J) distance in thousands of miles
             NEW-YORK     CHICAGO   TOPEKA
 SEATTLE        2.5          1.7       1.8
 SAN-DIEGO      2.5          1.8       1.4 ;
```

Wynik tej instrukcji to deklaracja parametru `D` i określenie jego dziedziny jako zbiór par produktu kartezjańskiego `I`×`J`.

2.2.3 Wprowadzenie poprzez bezpośrednie przypisanie

```
PARAMETER C(I,J) transport cost in thousands of dollars per case ;
          C(I,J) = F * D(I,J) / 1000 ;
```

Pierwsze zdanie to deklaracja, drugie definicja. Konieczny jest znak średnika na końcu pierwszej instrukcji. Taka definicja jest możliwa tylko gdy uprzednio zdefiniowano `D` i `F`. Legalne jest również następujące przypisanie wartości:

```
C( "SEATTLE" , "NEW-YORK" ) = 0.40 ;
C( "SEATTLE" , "NEW-YORK" ) = 0.50 ;
```

Taki zapis pokazuje także niebezpieczeństwa. Deklaruje się jeden raz, przypisywać wartość można wielokrotnie. Przypisanie wdrożone jest natychmiast - zapamiętana jest ostatnia wartość. Wyrażenie po prawej stronie znaku przypisania może mieć wielce złożoną postać. Sposób notacji wywodzi się z języka FORTRAN. Tradycyjnie znaki $+ - */$ oznaczają cztery podstawowe operacje; "`**`" oznacza podniesienie do potęgi (t.j. zapis `x**y` oznacza x^y). Dostępne są różne funkcje (`ABS`, `ARCTAN`, `COS`, `EXP`, `LOG`, `LOG10`, `SIN`, `SQR`, `SQRT`, itd.). Po dalsze szczegóły odsyłam do dokumentacji.

2.3 Zmienne decyzyjne (`VARIABLES`)

Zmienne decyzyjne modelu GAMS'a muszą być zadeklarowane instrukcją `VARIABLES`. Każda zmienna opatrzona jest nazwą, dziedziną i (opcjonalnie) zawiera dodatkowo opis.

```
VARIABLES
  X(I,J)  shipment quantities in cases
  Z       total transportation costs in thousands of dollars ;
POSITIVE VARIABLE X ;
```

W wyniku takiej instrukacji zadeklarowana zostaje zmienna określająca ilość dostarczonego towaru od `I`-tego producenta do `J`-tego odbiorcy (dla wszystkich par `I,J`). Zmienna `Z` (funkcja celu) jest skalarem (brak dziedziny). Każdy model optymalizacji GAMS musi zawierać taką zmienną; jest ona wielkością, dla której poszukuje się wartości minimalnej bądź maksymalnej.

Zadeklarowana zmienna musi mieć przypisany typ. Dopuszcza się następujące możliwości:

Nazwa typu	zakres wartości	uwagi
FREE	$(-\infty, +\infty)$	$\Leftarrow$ typ domyślny
POSITIVE	$[0, +\infty)$	
NEGATIVE	$(-\infty, 0]$	
BINARY	$\{0, 1\}$	
INTEGER	$\{0, 1, 2, \ldots, 100\}$	

Zmienna, która służy jako wielkość optymalizowaną musi być skalarem i musi być typu FREE. W naszym zadaniu Z jest typu FREE (jako typ domyślny), natomiast na parametr X(I,J) musimy narzucić ograniczenie nieujemności; stąd obecność instrukcji

```
POSITIVE VARIABLE X;
```

Zwróćmy uwagę, że użyto tu jedynie zapisu X (a nie X(I,J)). Wszystkim zmiennym całej dziedziny nadany będzie wskazany typ.

2.4 Równania (EQUATIONS)

Siła języka modelowania takiego jak GAMS najmocniej uwidacznia się w trakcie tworzenia równiań i nierówności opisujących model. Wynika to z faktu, że gdy równiania (nierówności) mają tę samą strukturę algebraiczną wszystkie składowe tworzone są jednocześnie. Trochę przypomina to zapis macierzowy.

2.4.1 Deklaracja równiania

Równania muszą być zadeklarowane jedną instrukcją, następnie w inną zdefiniowane. Format deklaracji jest taki sam jak dla innych obiektów GAMS'a. Najpierw pojawia się słowo kluczowe (w tym przypadku EQUATIONS), dalej nazwa, dziedzina i ewentualnie opis:

```
EQUATIONS
       COST        define objective function
       SUPPLY(I)   observe supply limit at plant i
       DEMAND(J)   satisfy demand at market j ;
```

Zwróćmy uwagę na fakt, że słowo EQUATIONS ma szerokie znaczenie. Zawiera zarówno relacje równości, jak i nierówności; równanie GAMS'a może odnosić się zarówno do jednej jak i kilku z tych relacji. Dla przykładu, COST nie ma dziedziny jest więc pojedyńczym równaniem, ale już SUPPLY dotyczy zbioru nierówności określonych nad dziedziną I.

Notacja sumacyjna Zanim przejdziemy do właściwych definicji równań opiszemy sposób zapisu sumy. Założenie, że dane w języku GAMS można wprowadzać przy pomocy standardowej klawiatury uniemożliwia zastosowanie zapisu matematycznego.

Składnia wyrażenia jest następująca:

SUM (indeks_sumowania, składnik)

Oba argumenty oddzielone są przecinkiem. Drugi argument może być dowolnym wyrażeniem matematyczym włączając w to inną sumę. Oto sumy z naszego przykładu:

`SUM( J, X(I,J) )` czyli $\sum_j x_{ij}$

`SUM( (I,J), C(I,J)*X(I,J) )` czyli $\sum_i \sum_j c_{ij} x_{ij}$

Ostatnie wyrażenie można zapisać jako zagnieżdżone sumowanie w postaci:

```
SUM( I, SUM( J, C(I,J)*X(I,J) ) )
```

Podobnie jak sumę można również zapisywać iloczyn

`PROD( J, X(I,J) )` czyli $\prod_j x_{ij}$

Operatory sumowania i iloczynu mogą być użyte w instrukcji przypisania parametrów, np.

```
SCALAR TOTSUPPLY total supply over all plants ;
TOTSUPPLY = SUM ( I, B(I) ) ;
```

2.4.2 Definicja równania

Definicja równania jest najbardziej złożoną instrukcją języka GAMS. Oto, przytoczone we właściwej kolejności komponenty definicji:

1. Nazwa definiowanego równania
2. Dziedzina
3. (opcjonalnie) zakres dziedziny
4. Symbol "`..`"
5. Wyrażenie po lewej stronie.
6. Operator relacji (`=L=`, `=E=`, `=G=`).
7. Wyrażenie po prawej stronie.

Nasz przykład zawiera trzy takie równania:

```
COST ..      Z =E=  SUM((I,J), C(I,J)*X(I,J)) ;
SUPPLY(I) .. SUM(J, X(I,J)) =L= A(I) ;
DEMAND(J) .. SUM(I, X(I,J)) =G= B(J) ;
```

Operator relacji może przyjąć jedną z postaci:

`=E=` oznacza *równy*

`=L=` oznacza *mniejszy lub równy*

`=G=` oznacza *większy lub równy*

Zwróćmy jeszcze uwagę na drobny szczegół: "`=`" oznacza operator przypisania, zaś "`=E=`" oznacza operator relacji.

2.5 Funkcja celu

W języku GAMS nie występuje obiekt nazywany funkcja celu. Określenie funkcji, która będzie optymalizowana wymaga utworzenia zmiennej skalarnej typu `FREE` (bez ograniczenia na znak), która pojawia się w później w definicji równania i równa jest właściwej funkcji celu.

2.6 Instrukcja `MODEL`

Słowo `MODEL` ma ściśle określone znaczenie w języku GAMS. Jest to po prostu zbiór równań sekcji `EQUATIONS`. Podobnie jak inne obiekty również tu należy w deklaracji przypisać nazwę.

Format deklaracji jest następujący:

```
MODEL nazwa / lista_równań /
```

Jeżeli wszystkie wcześniej określone równania mają być włączone do listy można użyć specyfikacji `/ALL/`. W naszym przykładzie jest:

```
MODEL TRANSPORT /ALL/ ;
MODEL TRANSPORT / COST, SUPPLY, DEMAND / ;
```

Drugi zapis zawiera pełną listę równań; oba sposoby, w naszym przypadku, są równoważne.

2.7 Instrukcja `SOLVE`

Po określeniu modelu i napisaniu równiań jesteśmy przygotowani do wywołania motoru (ang. *solver*). To realizowane jest przy pomocy instrukcji `SOLVE`, np.

```
SOLVE TRANSPORT USING LP MINIMIZING Z ;
```

Składnia (format) instrukcji `SOLVE` jest następująca:

1. słowo `SOLVE`;
2. nazwa modelu (z instrukcji `MODEL`);
3. słowo `USING`;
4. dostępna procedura obliczeniowa:

   `LP` dla programowania liniowego;

   `NLP` dla programowanie nieliniowego;

   `MIX` dla mieszanego programowania całkowitoliczbowego;

5. słowo `MINIMIZING` lub `MAXIMIZING`;
6. nazwa zmiennej podlegająca optymalizacji.

Z analizy przykładów wynika, że możliwa jest również poniższa postać instrukcji `SOLVE`:

```
SOLVE TRANSPORT MINIMIZING Z USING LP;
```

Bazy .LO, .L, .UP, .M

Motor GAMS został zaprojektowany łącznie z niewielką bazą rekordów skojarzonych ze zmiennymi i równaniami. Rekord zawiera następujące pola:

pole	field name	nazwa pola
.LO	lower bound	kres dolny
.L	level or primal value	wartość prymalna
.UP	upper bound	kres górny
.M	marginal or dual value	wartość dualna

Referencje do tych pól wymagają podania nazwy zmiennej (lub równania), po której następuje bezpośrednio nazwa pola, dalej (jeśli jest wymagana) dziedzina lub element dziedziny. Użytkownik ma pełen dostęp (odczyt i zapis) do tych pól.

Pola .LO i .UP dla zmiennych ustalane są automatycznie; decyduje o tym typ zmiennej (FREE, POSITIVE, NEGATIVE, BINARY, lub INTEGER). Ale te wielkości mogą być zmienione odpowiednim zapisem przez operatora. Niektóre z dołączonych dalej przykładów zawierają takie konstrukcje.

2.8 Instrukcja DISPLAY

Po wywołaniu motoru (instrukcją SOLVE) wyniki zazwyczaj umieszcza się w pliku (jeżeli plik z modelem nazywa się xxx.GMS, to motor utworzy xxx.LST z wynikami). Jeżeli chcemy obejrzeć nieco więcej niż standardowe wyniki (np. wartości dualne), to musimy to nakazać instrukcją, np.

```
DISPLAY X.L, X.M ;
```

Taki zapis spowoduje dodatkowo wyprowadzenie finalnych wielkości X.L i X.M.

3 Przykłady

Katalog programu \GAMS386 zawiera podkatalog o nazwie MODLIB — ten zawiera wiele przykładów problemów sformułowanych w języku GAMS