GAMS

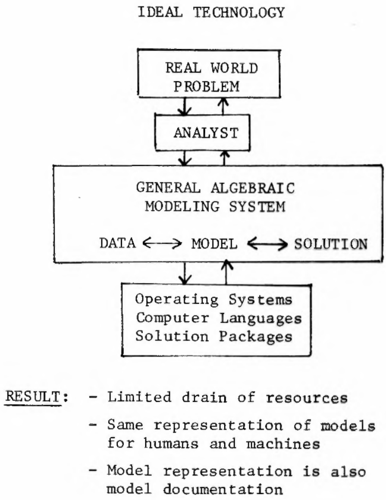
GAMS

Outline I

GAMS

- The General Algebraic Modeling System
- Roots: World Bank, 1976
- Went commercial in 1987
- Application Areas:
  - Agricultural Economics
  - Chemical Engineering
  - Econometrics
  - Environmental Economics
  - Finance
  - International Trade
  - Macro Economics
  - Management Science/OR
  - ...
  - Applied General Equilibrium
  - Economic Development
  - Energy
  - Engineering
  - Forestry
  - Logistics
  - Military
  - Mathematics

The Vision: World Bank Slide, 1976



Language

Declarative Language:

- Similar to mathematical notation
- Few basic language elements: sets, parameters, variables, equations, models
- Model is executable (algebraic) description of the problem

Imperative Elements:

- Control flow statements: loops, for, if, ...
- build algorithms within GAMS
- exchange data with other systems

Independence of Model and Solver

GAMS is not a Solver!

GAMS: Model building and interaction with solvers and environment.

Solver: Solve an instance (instantiation of a model with data) using mathematical optimization.

- Major commercial and academic solvers integrated:
  31 solvers, half of them actively developed/updated
- Switch between solvers with one statement:
  `option solver = scip;`

Solvers ↔ Problemtypes (GAMS 24.5)

	LP	MIP	NLP	MCP	MPEC	CNS	DNLP	MINLP	QCP	MIQCP	Stoch.	Global
ALPHAECP								x		x		
ANTIGONE 1.1			x			x	x	x	x	x		x
BARON 15.8	x	x	x			x	x	x	x	x		x
BDMLP	x	x										
BONMIN 1.8								x		x		
CBC 2.9	x	x										
CONOPT 3	x		x			x	x		x			
COUENNE 0.5			x			x	x	x	x	x		x
CPLEX 12.6	x	x							x	x		
DECIS	x										x	
DICOPT								x		x		
GUROBI 6.0	x	x							x	x		
IPOPT 3.12	x		x			x	x		x			
KNITRO 9.1	x		x			x	x	x	x	x		
LGO	x		x				x		x			(x)
LINDO 9.0	x	x	x				x	x	x	x	x	x
LOCALSOLVER 5.5	x	x	x			x	x	x	x	x		
MILES				x								
MINOS	x		x			x	x		x			
MOSEK 7	x	x	x				x		x	x		
MSNLP			x				x		x			(x)
NLPEC				x	x							
OQNLP			x				x	x	x	x		(x)
PATH				x		x						
SBB								x		x		
SCIP 3.2		x	x			x	x	x	x	x		x
SNOPT	x		x			x	x		x			
SOPLEX 2.2	x											
SULUM 4.3	x	x										
XA	x	x										
XPRESS 28.01	x	x							x	x		

Independence of Model and Platform

Supported Platforms:

Solver/Platform availability - 24.5	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit MacOS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✔	✔	✔	✔	✔	✔	✔
ANTIGONE 1.1	✔	✔	✔	✔			
BARON 15.8	✔	✔	✔	✔			
BDMLP	✔	✔	✔	✔	✔	✔	✔
BONMIN 1.8	✔	✔	✔	✔	✔		
CBC 2.9	✔	✔	✔	✔	✔		
CONOPT 3	✔	✔	✔	✔	✔	✔	✔
COUENNE 0.5	✔	✔	✔	✔	✔		
CPLEX 12.6	✔	✔	✔	✔	✔	✔	✔
DECIS	✔	✔	✔			✔	
DICOPT	✔	✔	✔	✔	✔	✔	✔
GLOMIQO 2.3	✔	✔	✔	✔			
GUROBI 6.0	✔	✔	✔	✔			✔
GUSS	✔	✔	✔	✔	✔	✔	✔
IPOPT 3.12	✔	✔	✔	✔	✔		
KESTREL	✔	✔	✔	✔	✔	✔	✔
KNITRO 9.1	✔	✔	✔	✔			
LGO	✔	✔	✔	✔	✔	✔	
LINDO 9.0	✔	✔	✔	✔			
LINDOGLOBAL 9.0	✔	✔	✔	✔			
LOCALSOLVER 5.5	✔	✔	✔	✔			
MILES	✔	✔	✔	✔	✔	✔	✔
MINOS	✔	✔	✔	✔	✔	✔	✔
MOSEK 7	✔	✔	✔	✔			
MSNLP	✔	✔	✔	✔		✔	
NLPEC	✔	✔	✔	✔	✔	✔	✔
OQNLP	✔	32bit					
PATH	✔	✔	✔	✔	✔	✔	✔
SBB	✔	✔	✔	✔	✔	✔	✔
SCIP 3.2	✔	✔	✔	✔	✔		
SNOPT	✔	✔	✔	✔	✔	✔	✔
SOPLEX 2.2	✔	✔	✔	✔	✔		
SULUM 4.3	✔	✔	✔				
XA	✔	✔	✔				
XPRESS 28.01	✔	✔	✔	✔	✔	✔	✔

Tools/Platform availability - 24.5	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit MacOS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ASK	✔	32bit					
BIB2GMS	✔	✔	✔	✔	✔	✔	✔
CHK4UPD	✔	✔	✔	✔	✔	✔	✔
CHOLESKY	✔	✔	✔	✔	✔	✔	✔
CSDP	✔	✔	✔	✔	✔		
CSV2GDX	✔	✔	✔	✔	✔	✔	✔
EIGENVALUE	✔	✔	✔	✔	✔	✔	✔
EIGENVECTOR	✔	✔	✔	✔	✔	✔	✔
ENDECRYPT	✔	✔	✔	✔	✔	✔	✔
GAMSIDE	✔	32bit					
GAMS POSIX Utilities[1)]	✔	✔	✔	✔	✔	✔	✔
GDX2ACCESS	✔	32bit					
GDX2HAR	✔	32bit					
GDX2SQLITE	✔	✔	✔	✔	✔	✔	✔
GDX2VEDA	✔	✔	✔	✔	✔	✔	✔
GDX2XLS	✔	32bit					
GDXCOPY	✔	✔	✔	✔	✔	✔	✔
GDXDIFF	✔	✔	✔	✔	✔	✔	✔
GDXDUMP	✔	✔	✔	✔	✔	✔	✔
GDXMERGE	✔	✔	✔	✔	✔	✔	✔
GDXMRW	✔	✔	✔	✔			
GDXRANK	✔	✔	✔	✔	✔	✔	✔
GDXRENAME	✔	✔	✔	✔	✔	✔	✔
GDXRRW	✔	✔	✔	✔	src only	src only	src only
GDXTROLL	✔	✔	✔	✔	✔	✔	✔
GDXVIEWER	✔	32bit					
GDXXRW	✔	32bit					
GMSUNZIP	✔	✔	✔	✔	✔	✔	✔
HAR2GDX	✔	32bit					
IDECMDS	✔	32bit					
INVERT	✔	✔	✔	✔	✔	✔	✔
MCFILTER	✔	✔	✔	✔	✔	✔	✔
MDB2GMS	✔	32bit					
MODEL2TEX	✔	✔	✔	✔	✔	✔	✔
MPS2GMS	✔	✔	✔	✔	✔	✔	✔
MSAPPAVAIL	✔	32bit					
SCENRED	✔	✔	✔	✔	✔	✔	✔
SCENRED2	✔	✔	✔	✔	✔	✔	✔
SHELLEXECUTE	✔	32bit					
SQL2GMS	✔	32bit					

Documentation and Help

Online: `http://www.gams.com/help` (with search)

Offline: `<GAMS system directory>/docs/index.html` (no search, use grep!)

- GAMS – A User's Guide: Tutorial, Basics, Advanced Topics
- McCarl (Expanded) GAMS User Guide
- Solver Manuals
- Tools Manuals
- APIs: Tutorials and Reference Manuals
- Release Notes

Tutorial Videos: `http://www.youtube.com/user/GAMSLessons`

Support wiki: `http://support.gams.com/doku.php`

Discussion group: `http://www.gamsworld.org/`

Model Libraries

Online: `http://www.gams.com/modlibs`

Offline: `gamslib`, `apilib`, `datalib`, `emplib`, `testlib` tools

- GAMS Model Library
  - representing interesting and sometimes classic problems
  - illustrating GAMS modeling capabilities

Model Libraries

Online: `http://www.gams.com/modlibs`

Offline: `gamslib`, `apilib`, `datalib`, `emplib`, `testlib` tools

- GAMS Model Library
  - representing interesting and sometimes classic problems
  - illustrating GAMS modeling capabilities
- GAMS API Library
  - scripts to compile and execute GAMS API examples
- GAMS Data Utilities Library
  - demonstrate utilities to interface GAMS with other applications
- GAMS EMP Library
  - illustrate and test capabilities of extended mathematical programming facility
- Contributed Libraries:
  - FINLIB – financial optimization models (by Consiglio, Nielsen and Zenios)
  - NOALIB – nonlinear optimization applications models (by Neculai Andrei)
- GAMS Testlib Library
  - testing and quality control

Outline I

Cows & Pigs Example

Variables:

x_1 the number of cows to purchase

x_2 the number of pigs to purchase

Objective:

$$\text{maximize} \quad z = 3x_1 + 2x_2$$

Constraints:

$$\begin{aligned} x_1 &\in \{0, 1, 2\} \\ x_2 &\in \{0, 1, 2\} \\ x_1 + x_2 &\leq 3 \end{aligned}$$

File: `cowspigs.gms`

Structure of GAMS models

Instructions:

- a GAMS model is a sequence of instructions in the GAMS language
- multiline instructions, empty lines, and several instructions per line are allowed
- instructions should be closed with a semicolon ';'
- case insensitive (!!)

Structure of GAMS models

Instructions:

- a GAMS model is a sequence of instructions in the GAMS language
- multiline instructions, empty lines, and several instructions per line are allowed
- instructions should be closed with a semicolon ';'
- case insensitive (!!)

Comments and Documentation:

- lines that start with a '*' are comments and thus ignored
- documenting text can be contained inside instructions (we will see)

Structure of GAMS models

Instructions:

- a GAMS model is a sequence of instructions in the GAMS language
- multiline instructions, empty lines, and several instructions per line are allowed
- instructions should be closed with a semicolon ';'
- case insensitive (!!)

Comments and Documentation:

- lines that start with a '*' are comments and thus ignored
- documenting text can be contained inside instructions (we will see)

Entities (Sets, Data, Variables, Equations, ...):

- 2 steps to build an entity: Declaration and Definition
- Declaration: state the existence of an entity
- Definition: assign a value or a "form" to an entity
- no entity can be referenced before it was declared
- names start with letters and can contain up to 62 further characters (excluding -, %, $)

Variables

Syntax: `[var-type] variable[s] varname [text] {, varname [text]}`

- `var-type` allows to pre-determine the Range of a variable:

Variable type	Range
`free` (default)	$\mathbb{R}$
`positive`	$\mathbb{R}_{\geq 0}$
`negative`	$\mathbb{R}_{\leq 0}$
`binary`	$\{0, 1\}$
`integer`	$\{0, 1, \ldots, 100\}$ (!!)
`semicont`	$\{0\} \cup [\ell, u]$ (default: $\ell = 1$, $u = \infty$)
`semiint`	$\{0\} \cup \{\ell, \ell + 1, \ldots, u\}$ (default: $\ell = 1$, $u = 100$)
`sos1`, `sos2`	special ordered sets of type 1 and 2

- Examples:

```
variables x, y, objvar;
positive variable x;
integer  variable z;
```

Variable Attributes

Attributes of a variable:

Attribute	Meaning
.lo	lower bound on variable range
.up	upper bound on variable range
.fx	fixed value for a variable
.l	current primal value (updated by solver)
.m	current dual value (updated by solver)
.scale	scaling factor
.prior	branching priority

- Examples:

```
x.up = 10;
y.fx = 5.5;
display z.l;
```

Equations

- `Equations` serve to define restrictions (constraints) and an objective function

Declaration:

- Syntax: `Equation[s] eqnname [text] {, eqnname [text]} ;`
- Example:

```
Equation objective   "Objective Function";
```

Equations

- Equations serve to define restrictions (constraints) and an objective function

Declaration:

- Syntax: `Equation[s] eqnname [text] {, eqnname [text]} ;`
- Example:

```
Equation objective    "Objective Function";
```

Definition:

- Syntax: `eqnname(domainlist).. expression eqn_type expression;`

eqn_type	meaning
=e=	$=$
=g=	$\geq$
=l=	$\leq$
=n=	no relation between left and right side
=x=	for external functions
=c=	for conic constraints
=b=	for logic constraints

- Example:

```
objective.. objvar =e= 2 * x + 3 * y * y - y + 5 * z ;
e1..        x + y  =l= z;
```

Model statement

- Model = a collection of equations
- Syntax:

```
model[s] model_name [text] [/ all | eqn_name {, eqn_name}]
       {,model_name [text] [/ all | eqn_name {, eqn_name}] };
```

- Example:

```
model m / all /;
model m / objective, e1 /;
```

Model statement

- Model = a collection of equations
- Syntax:

```
model[s] model_name [text] [/ all | eqn_name {, eqn_name}]
       {,model_name [text] [/ all | eqn_name {, eqn_name}] };
```

- Example:

```
model m / all /;
model m / objective, e1 /;
```

Attributes:

set by user		set by solver	
iterlim	iteration limit	iterusd	number of iterations
reslim	time limit in seconds	resusd	solving time
optcr	relative gap tolerance	modelstat	model status
optfile	number of solver options file	solvestat	solver status
...		...	

Example:

```
m.reslim = 60;      m.optcr = 0;
solve m minimize objvar using MINLP;
display m.resusd;
```

Solving a model

- Passing a model to a solver and evaluation of results
- Specification of one free variable to be minimized or maximized
- Syntax:

```
solve modelname using modeltyp maximizing|minimizing varname|;
solve modelname maximizing|minimizing varname using modeltyp ;
```

- the model type defines the problem class to be used for the model:

LP	a linear problem
QCP	quadratically constraint problem (only linear and quadratic terms)
NLP	a nonlinear problem with continuous functions
DNLP	a nonlinear problem with discontinuous functions
MIP	a mixed-integer linear problem
MIQCP	a mixed-integer quadratically constraint problem
MINLP	a mixed-integer nonlinear problem
CNS	a nonlinear constraint satisfaction problem (no objective function)
RMIP,...	a mixed-integer problem with relaxed integrality restrictions

- Example:

```
solve m using  MINLP minimizing objvar;
solve m using RMINLP min        objvar;
```

GAMS command line call

Calling GAMS from the command line:

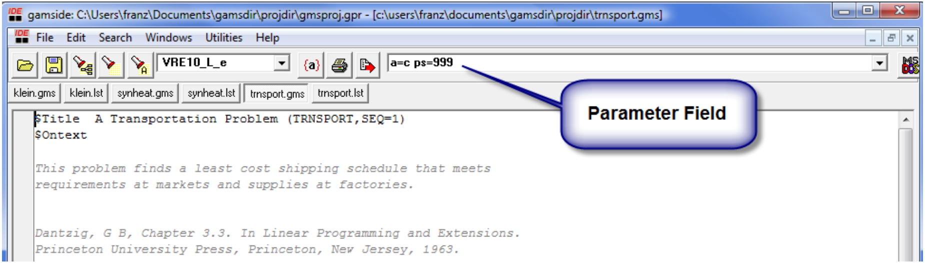
```
$ gams <modelfile> { [-]key=value | key value }
                   { --ControlVariable=string }
```

Examples:

- `gams trnsport.gms`
- `gams trnsport`
- `gams trnsport.gms LP=CBC`
- `gams trnsport LP CBC`
- `gams trnsport -LP CBC`
- `gams trnsport -LP CBC --Output=result.txt`

Passing command line parameters in the GAMS IDE

- in the IDE, command line parameters can be passed to GAMS via the parameter field



GAMS Log

The GAMS log can be written to the console, to standard output and/or to a file. This is controlled by the command line parameter logOption (or lo).

The following items (and more) are part of the log:

- GAMS version

```
*** ************* BETA release
*** GAMS Base Module 24.5.0 r53642 BETA Released 25Aug15 WEI x86 64bit/MS Windows
*** ************* BETA release
```

- Licensee

```
Licensee: Lutz Westermann                                 G141124/0001AW-GEN
          GAMS Software GmbH                                          DC8674
```

- Problem statistics

```
--   2 rows  3 columns  5 non-zeroes
--   2 discrete-columns
```

- Solver log

```
Cplex 12.6.2.0

Reading data...
Starting Cplex...
...
```

- Results

```
MIP Solution:      8.000000    (0 iterations, 0 nodes)
Final Solve:       8.000000    (0 iterations)
Best possible:     8.000000
Absolute gap:      0.000000
Relative gap:      0.000000
```

Listing File

Running a GAMS model generates a listing file (`.lst` file).

Compilation Errors:

- are indicated by ****
- contain a '$' directly below the point at which the compiler thinks the error occurred
- are explained near the end of the line-numbered listing part
- in the IDE, they are also indicated by red lines in the process (log) window (can be double-clicked)
- check carefully for the cause of the first error, fix it, and try again
- usual causes: undefined / undeclared symbols (parameters, variables, equations), unmatched brackets, missing semi-colons

Listing File: Equation and Column Listing

Equation Listing:

- listing of generated equations with sets unrolled, parameters removed, ...
- useful for model debugging: is the intended model generated?
- for nonlinear equations, a linearization in the starting point is shown

```
   AcidDef..  AcidDilut*AcidErr  =e= 35.82-22.2*F4Perf;
-> AcidDef..  (1)*AcidDilut + 22.2*F4Perf + (3.6)*aciderr =E=
                         35.82 ; (LHS = 35.79, INFES = 0.03 ****)
```

- activity and violation of constraint in starting point also shown

Listing File: Equation and Column Listing

Equation Listing:

- listing of generated equations with sets unrolled, parameters removed, ...
- useful for model debugging: is the intended model generated?
- for nonlinear equations, a linearization in the starting point is shown

```
   AcidDef..  AcidDilut*AcidErr  =e= 35.82-22.2*F4Perf;
-> AcidDef..  (1)*AcidDilut + 22.2*F4Perf + (3.6)*aciderr =E=
                        35.82 ; (LHS = 35.79, INFES = 0.03 ****)
```

- activity and violation of constraint in starting point also shown

Column Listing:

- shows coefficients, bounds, starting values for generated variables

```
-- F4Perf  F4 Performance Number

F4Perf
                (.LO, .L, .UP, .M = 1.45, 1.45, 1.62, 0)
       22.2     AcidDef
       (1)      F4Def
```

Listing File: Solve Summary

- generated for each solve command
- reporting status and result of solve

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

     MODEL    m                   OBJECTIVE  F
     TYPE     NLP                 DIRECTION  MINIMIZE
     SOLVER   CONOPT              FROM LINE  85

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      2 Locally Optimal
**** OBJECTIVE VALUE               -1.7650

 RESOURCE USAGE, LIMIT          0.006      1000.000
 ITERATION COUNT, LIMIT        16    2000000000
 EVALUATION ERRORS              0             0
```

Listing File: Solution Listing

- equation and variable primal and dual values and bounds
- marking of infeasibilities, “non-optimalities”, and unboundedness
- ‘.’ = zero

	LOWER	LEVEL	UPPER	MARGINAL	
-- EQU Objective	.	.	.	1.0000	
-- EQU AlkylShrnk	.	.	.	-4.6116	
-- EQU AcidBal	.	-0.0020	.	11.8406	INFES
-- EQU IsobutBal	.	0.0952	.	0.0563	INFES
-- EQU AlkylDef	.	0.0127	.	-1.0763	INFES
-- EQU OctDef	0.5743	0.5747	0.5743	-25.9326	INFES
-- EQU AcidDef	35.8200	35.8533	35.8200	0.2131	INFES
-- EQU F4Def	-1.3300	-1.3300	-1.3300	-4.1992	

	LOWER	LEVEL	UPPER	MARGINAL	
-- VAR F	-INF	-1.4143	+INF	.	
-- VAR OlefinFeed	.	1.6198	2.0000	-0.1269	NOPT
-- VAR IsobutRec	.	1.3617	1.6000	-0.2133	NOPT
-- VAR AcidFeed	.	0.7185	1.2000	-0.0411	NOPT
-- VAR AlkylYld	.	2.8790	5.0000	-0.0076	NOPT
-- VAR IsobutMak	.	1.8926	2.0000	-0.4764	NOPT
-- VAR AcidStren	0.8500	0.8998	0.9300	0.5273	NOPT

Sets

- Basic elements of a model
- Syntax:

```
set set_name ["text"] [/element ["text"] {,element ["text"]} /]
  {,set_name ["text"] [/element ["text"] {,element ["text"]} /]} ;
```

Sets

- Basic elements of a model
- Syntax:

```
set set_name ["text"] [/element ["text"] {,element ["text"]} /]
  {,set_name ["text"] [/element ["text"] {,element ["text"]} /]} ;
```

- Name `set_name` is identifier
- Elements have up to 63 characters, start with letter or digit or are quoted:

```
A    Phos-Acid    September    1986    1952-53    Line-1
'*TOTAL*'    '10%incr'    '12"/foot'    "Line 1"
```

- Elements have no value (!), that is, '1986' does not have the numerical value 1986 and '01' $\neq$ '1'
- Text has up to 254 characters, all in one line

Sets

- Basic elements of a model
- Syntax:

```
set set_name ["text"] [/element ["text"] {,element ["text"]} /]
  {,set_name ["text"] [/element ["text"] {,element ["text"]} /]} ;
```

- Name `set_name` is identifier
- Elements have up to 63 characters, start with letter or digit or are quoted:

```
A   Phos-Acid   September   1986   1952-53   Line-1
'*TOTAL*'   '10%incr'   '12"/foot'   "Line 1"
```

- Elements have no value (!), that is, '1986' does not have the numerical value 1986 and '01' $\neq$ '1'
- Text has up to 254 characters, all in one line
- Example:

```
Set n   Nutrients
/ Prot  "Protein (mg)"
  VitA  "Vitamine A",    VitC   'Vitamine C',
  Calc  Calcium
/;
```

Data

- data in GAMS consists always of real numbers (no strings)
- uninitialized data has the default value 0
- 3 forms to declare data:
  - Scalar: a single (scalar) date
  - Parameter: list oriented data
  - Table: table oriented data (at least 2 dimensions)

Data

- data in GAMS consists always of real numbers (no strings)
- uninitialized data has the default value 0
- 3 forms to declare data:
  - Scalar: a single (scalar) date
  - Parameter: list oriented data
  - Table: table oriented data (at least 2 dimensions)

Scalar Data:

- Syntax:

```
scalar[s] scalar_name [text] [/signed_num/]
        { scalar_name [text] [/signed_num/] };
```

- Example:

```
Scalars rho Discountfactor / .15 /
        izf internal rate of return;
```

Data: Parameters

Parameter:

- can be indexed over a one or several sets
- Syntax:

```
parameter[s] param_name [text] [/ element [=] signed_num
                                 {,element [=] signed_num} /]
          {,param_name [text] [/ element [=] signed_num
                                 {,element [=] signed_num} /] } ;
```

- Example:

```
set ice icecreams / chocolate, strawberry, cherry, vanilla /;
parameter demand(ice) / chocolate 50, strawberry = 30
                        vanilla    20 /
```

- Example:

```
 set c  'countries'  / jamaica, haiti, guyana, brazil /;
 parameter demand(c,ice)  "Demand of icecream per country (t)"
  / Jamaica.Chocolate 300, Jamaica.Strawberry 50, Jamaica.Cherry 5
    Haiti.(Chocolate,Vanilla,Strawberry)  =   30,
    (Guyana,Brazil).Chocolate                100 /
```

Data: Tables

Tables:

- Syntax:

```
table table_name [text] EOL
            element    { element }
    element signed_num { signed_num } EOL
   {element signed_num { signed_num } EOL} ;
```

- Example:

```
table demand(c,ice)  "Demand of icecream per country (t)"
                    Chocolate  Strawberry  Cherry  Vanilla
Jamaica                   300          50       5
Haiti                      30          30                30
(Guyana,Brazil)           100                              ;
```

- no “free form”: position of elements is of importance
- tables with more than 2 dimensions are also possible

Exercise: Cows & Pigs Continued

Define:

- a set of animals (`i`)
- a parameter with profit for each animal (`p(i)`)
- a parameter with maximal number for each animal (`xmax(i)`)
- a parameter for the max number of all animals (`maxanimal`)
- an integer variable to count how many of each animal to buy (`x(i)`)
- a real variable to hold the profit (`profit`)
- an equation to define the objective
- an equation to limit the total number of purchased animals by `maxanimal`

Fill sets and parameters with the data from the original example:

- 2 animals: cow and pig
- profit for cow: 3 profit for pig: 2
- maximal number of cows: 2 maximal number of pigs: 2
- maximal number of animals: 3

Exercise: Cows & Pigs Continued

Define:

- a set of animals (`i`)
- a parameter with profit for each animal (`p(i)`)
- a parameter with maximal number for each animal (`xmax(i)`)
- a parameter for the max number of all animals (`maxanimal`)
- an integer variable to count how many of each animal to buy (`x(i)`)
- a real variable to hold the profit (`profit`)
- an equation to define the objective
- an equation to limit the total number of purchased animals by `maxanimal`

Fill sets and parameters with the data from the original example:

- 2 animals: cow and pig
- profit for cow: 3 profit for pig: 2
- maximal number of cows: 2 maximal number of pigs: 2
- maximal number of animals: 3

File: `cowspigs2.gms`

Sequences, Alias

Sequences in Sets: ∗-Notation

- `set t "time" / 2000*2008 /;`
  corresponds to

```
set t "time" / 2000,2001,2002,2003,2004,2005,2006,2007,2008 /;
```

- `a1bc*a20bc` is different from `a01bc*a20bc`
- the following are wrong: `a1x1*a9x9`, `a1*b9`

Sequences, Alias

Sequences in Sets: $*$-Notation

- `set t "time" / 2000*2008 /;`
  corresponds to

```
set t "time" / 2000,2001,2002,2003,2004,2005,2006,2007,2008 /;
```

- `a1bc*a20bc` is different from `a01bc*a20bc`
- the following are wrong: `a1x1*a9x9`, `a1*b9`

Several names for one set: `alias` command

- Syntax: `alias(set_name, set_name \{, set_name\})`
- Example:

```
  set ice / chocolate, strawberry, cherry, vanilla /;
  alias(ice, icecreme, mantecado, sorvete);
```

Data: Assignments

- Scalar Assignment:

```
scalar x / 1.5 /;
x = 1.2;
x = x + 2;
```

- Indexed Assignment:

```
Set row          / r1*r10 /
    col          / c1*c10 /
    subrow(row)  / r7*r10 /;
Parameter a(row,col), r(row), c(col);
a(row,col)       = 13.2 + r(row)*c(col);
a('r7','c4')     = -2.36;

a(subrow,'c10')  = 2.44 - 33*r(subrow);

a(row,row)       = 7.7 - r(row);
alias(row,rowp);
a(row,rowp)      = 7.7 - r(row) + r(rowp);
```

Data: Expressions

Expression: an arbitrarily complex calculation instruction

Arithmetic operators: ** (exponentiate), +, −, *, /

```
x = 5 + 4*3**2;
x = 5 + 4*(3**2);
```

- `x**n` corresponds to `exp(n*log(x))` ⇒ only allowed for $x > 0$ (`power(x,n)` can be used instead if $n \in \mathbb{I}$)
- `population(t) = 56*(1.015**(ord(t)-1))`

Data: Expressions

Expression: an arbitrarily complex calculation instruction

Arithmetic operators: ** (exponentiate), +, −, *, /

- ```
  x = 5 + 4*3**2;
  x = 5 + 4*(3**2);
  ```
- x**n corresponds to exp(n*log(x)) $\Rightarrow$ only allowed for $x > 0$ (power(x,n) can be used instead if $n \in \mathbb{I}$)
- population(t) = 56*(1.015**(ord(t)-1))

Indexed Operations:

- Syntax: indexed_op((controlling_indices), expressions)
- indexed_op can be: sum, prod, smin, smax

```
parameter demand(c,ice)   "demand (t)"
          totaldemand(c)  "totaler demand per country (t)"
          completedemand  "totaler demand for all countries (t)"
          mindemand(ice)  "minimal demand per icecream";
totaldemand(c) = sum(ice,      demand(c,ice));
completedemand = sum((c,ice), demand(c,ice));
mindemand(ice) = smin(c,       demand(c,ice));
```
```

Data: Expressions

Expression: an arbitrarily complex calculation instruction

Arithmetic operators: `**` (exponentiate), +, −, `*`, /

-
```
x = 5 + 4*3**2;
x = 5 + 4*(3**2);
```

- `x**n` corresponds to `exp(n*log(x))` ⇒ only allowed for $x > 0$ (`power(x,n)` can be used instead if $n \in \mathbb{I}$)
- `population(t) = 56*(1.015**(ord(t)-1))`

Indexed Operations:

- Syntax: `indexed_op( (controlling_indices), expressions)`
- `indexed_op` can be: `sum`, `prod`, `smin`, `smax`

```
parameter demand(c,ice)   "demand (t)"
          totaldemand(c) "totaler demand per country (t)"
          completedemand "totaler demand for all countries (t)"
          mindemand(ice) "minimal demand per icecream";
totaldemand(c) = sum(ice,     demand(c,ice));
completedemand = sum((c,ice), demand(c,ice));
mindemand(ice) = smin(c,      demand(c,ice));
```

Functions: `errorf(x)`, `exp(x)`, `power(x,n)`, `sqr(x)`, `uniform(a,b)`, `normal(mean,sdev)`, ...

Option command

- specification of systemwide parameters to control output, solving process, ...
- Syntax:
  `option keyword1 [ = value1 ] { ,|EOL keyword2 = [ value2 ] }`
- some important parameters:

keyword	meaning	default
iterlim	iteration limit	2000000000
reslim	time limit	1000 (!!)
optca	absolute gap tolerance	0.0
optcr	relative gap tolerance	0.1 (!!)
LP	choice of LP solver	CPLEX
NLP	choice of NLP solver	CONOPT
...		

- Example:

```
option iterlim = 100, optcr = 0;
solve icesale using mip min cost;
option mip = cbc;
solve icesale using mip min cost;
```

- options can also be set on the command line:
  `> gams icesale.gms mip=cbc optcr=0`

Display command

- display sets, data, variable/equation/model attributes in the listing file
- Examples:

```
display ice, sorbet;
display x.l, x.m;
display demand;
display satdemand.m;
```

- only non-zero values are displayed
- control number of digits after the decimal point for all displayed values

```
option decimals = 1 ;
```

- number of digits after the decimal point for variable x:

```
option x:6 ;
```

Outline I

Compilation vs. Execution

GAMS processed models in 2 phases:

Compilation Phase:

- reads the complete GAMS model and translate into GAMS specific byte code
- processes declarations (variables, equations, sets, parameters)
- processes labels (set elements) and data statements
- execute all compile-time commands (next slides)
- Listing file:

```
GAMS 24.1.2  r40979 Released Jun 16, 2013 LEX-LEG x86_64/Linux

3 16:03:38 Page 1
A Transportation Problem (TRNSPORT,SEQ=1)
C o m p i l a t i o n

<echo of all processed lines>

COMPILATION TIME     =        0.002 SECONDS      3 MB  24.1.2 r40979 LEX-LEG
...
```

Execution Phase:

- executes commands in program: assignments, solve, loop, while, for, if, ...

Compile Time Commands

- introduced by $ sign in the first column (!)
- Syntax: `$commandname argumentlist {commandname argumentlist}`
- modify behavior of GAMS compiler
- allow program flow control on compilation level: call external programs, include other files, if-else, goto, ...
- modifying and reading compile-time variables

Examples:

- define a title for your GAMS program

```
$Title  A Transportation Problem
```

- define a section that contains only comments

```
$onText
This problem finds a least cost shipping schedule that meets
requirements at markets and supplies at factories.
$offText
```

- disable echoing of input lines in listing file

```
$offListing
```

Compile Time Variables

- compile-time variables hold strings
- their value is accessed to via the %variablename% notation
- values are assigned via $set or $eval commands or on the GAMS command line via double-dash-options: `gams --variablename variablevalue`
- for `$eval`, the variable value string is interpreted as numerical expression
- Example:

```
$set  N    10
$eval Nsqr %N% * %N%
set i / 1 * %N% /;
```

- variants `$setLocal`, `$setGlobal`, `$evalLocal`, `$evalGlobal` allow to control the (file)scope of a variable

Compile Time Program Control: $If

- $If allows to do conditional processing
- Syntax: $If [not] <conditional expression> new_input_line
- only one-line clauses allowed (new_input_line can be on next line, though)
- Examples:

```
$if exist myfile.dat $log "myfile.dat exists, yeah!"
$if not set scenario $set scenario basic

scalar a;
$if     %difficulty% == easy a = 5;
$if not %difficulty% == easy a = 10;
```

Compile Time Program Control: $If

- $If allows to do conditional processing
- Syntax: $If [not] <conditional expression> new_input_line
- only one-line clauses allowed (new_input_line can be on next line, though)
- Examples:

```
$if exist myfile.dat $log "myfile.dat exists, yeah!"
$if not set scenario $set scenario basic

scalar a;
$if     %difficulty% == easy a = 5;
$if not %difficulty% == easy a = 10;
```

- $IfThen-$ElseIf-$Else-$Endif allows to control activity for a set of statements
- Example:

```
scalar a;
$ifthen %difficulty% == easy
a = 5;
$else
a = 10;
$endif
```

Compile Time Program Control: $Goto

- $Goto-$Label allows to skip over or repeat sections of the input
- Example:

```
scalar a / 5 /;
$if %difficulty% == easy $goto easy
a = 10;
$label easy
```

Executing Shell Commands

- `$Call` passes a following string to the current shell and waits for the command to be completed
- if the string starts with a '=', the operating system is called directly, i.e., no shell is invoked
- Example:

```
$call "gamslib trnsport"
$call "=gams trnsport"

$if exist myfile.dat $call cp myfile.dat mycopy.dat
```

- the `errorLevel` functions allows to check whether a previous command (e.g., `$call`) executed without error:

```
$call "gamslib trnsport"
$call "gams trnsport"
$if errorlevel 1 $abort "ouch! — solving trnsport failed"
```

Outline I

Writing text files

- $Echo and $onEcho-$offEcho allows to write to text files
- Example:

```
$Echo "hello, world!"  > myfile.txt
$OnEcho >> myfile.txt
ahoy—hoy!
$OffEcho
```

- `> myfile.txt` creates a new file `myfile.txt`, thereby overwriting a possibly existing one of the same name
- `>> myfile.txt` appends to an existing file `myfile.txt`
- Recall: These are compilation-time commands! Not usable to write solve outcomes or similar; use display command or put-facility (later) for this.

Including text files

- $Include allows to include ASCII files into a GAMS program
- compilation is then continued for the included file
- Example:

```
Parameter d(i,j) distance in thousands of miles;
$include dist.inc
```

where dist.inc contains

```
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  ;
```

Including csv files

- `$OnDelim` enables comma separated value (csv) format for data in table or parameter statements

Examples:

```
Table d(i,j)  distance
$ondelim
$include dist.csv
$offdelim
;
```

where `dist.csv` is

```
,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

```
Parameter d(i,j)  distance /
$ondelim
$include dist.txt
$offdelim
/;
```

where `dist.txt` is

```
SEATTLE,NEW-YORK,2.5
SAN-DIEGO,NEW-YORK,2.5
SEATTLE,CHICAGO,1.7
SAN-DIEGO,CHICAGO,1.8
SEATTLE,TOPEKA,1.8
SAN-DIEGO,TOPEKA,1.4
```

Put Command

- writing text files at execution time
- associating an identifier `fileid` with a file: `file fileid / myfile.txt /;`
- select a stream (and thus a file) to write to: `put fileid;`
- writing some items (text, labels, numbers): `put item {, item};`
- write a linebreak: `put /;`
- close a stream: `putclose;`

Put Command

- writing text files at execution time
- associating an identifier `fileid` with a file: `file fileid / myfile.txt /;`
- select a stream (and thus a file) to write to: `put fileid;`
- writing some items (text, labels, numbers): `put item {, item};`
- write a linebreak: `put /;`
- close a stream: `putclose;`
- Example:

```
file fx /result.txt/;
put fx 'Shipped quantities between plants and markets' /;
loop((i,j)$x.l(i,j),
  put 'Shipment from ', i.te(i):10, ' to ', j.te(j):10,
      ' in cases:', x.l(i,j) /;
); putclose;
```

gives

```
Shipped quantities between plants and markets
Shipment from seattle    to new-york  in cases:   50.00
Shipment from seattle    to chicago   in cases:  300.00
Shipment from san-diego  to new-york  in cases:  275.00
Shipment from san-diego  to topeka    in cases:  275.00
```

Text Items, Formatted Output

Label names and explanatory texts can be accessed via attributes:

- `ident.ts`: text associated with identifier
- `element.tl`: label associated with set element
- `set.te(element)`: text associated with element of set

Text Items, Formatted Output

Label names and explanatory texts can be accessed via attributes:

- `ident.ts`: text associated with identifier
- `element.tl`: label associated with set element
- `set.te(element)`: text associated with element of set

Local Item Formatting:

- Syntax for formatting item output: `item:{<>}width:decimals`
- `{<>}` specifies whether justified left (<), right (>), or centered (<>)
- `width` is the field width
- `decimals` is the number of decimals for numeric output
- each can be omitted, e.g., `x.l::5`

Text Items, Formatted Output

Label names and explanatory texts can be accessed via attributes:

- `ident.ts`: text associated with identifier
- `element.tl`: label associated with set element
- `set.te(element)`: text associated with element of set

Local Item Formatting:

- Syntax for formatting item output: `item:{<>}width:decimals`
- {<>} specifies whether justified left (<), right (>), or centered (<>)
- `width` is the field width
- `decimals` is the number of decimals for numeric output
- each can be omitted, e.g., `x.l::5`

Global Item Formatting:

- change field justification and width for all items of a type
- `.lj`, `.nj`, `.sj`, `.tj`, `.lw`, `.nw`, `.sw`, `.tw` attributes of stream identifier
- see GAMS User's Guide Section 15.10

Text Items, Formatted Output

Label names and explanatory texts can be accessed via attributes:

- `ident.ts`: text associated with identifier
- `element.tl`: label associated with set element
- `set.te(element)`: text associated with element of set

Local Item Formatting:

- Syntax for formatting item output: `item:{<>}width:decimals`
- `{<>}` specifies whether justified left (<), right (>), or centered (<>)
- `width` is the field width
- `decimals` is the number of decimals for numeric output
- each can be omitted, e.g., `x.l::5`

Global Item Formatting:

- change field justification and width for all items of a type
- `.lj`, `.nj`, `.sj`, `.tj`, `.lw`, `.nw`, `.sw`, `.tw` attributes of stream identifier
- see GAMS User's Guide Section 15.10

Cursor Positioning:

- `put @n;` moves cursor to column `n` of current line

Independence of Model and Data

GDX – GAMS Data Exchange:

- Binary data format for fast exchange of data with GAMS
- Stores sets, parameters, values of variables/equations with domain information, but no symbolic information (e.g., equation algebra)
- Consistency: no duplicates or contradictions
- Platform independent
- Can be compressed

Independence of Model and Data

GDX – GAMS Data Exchange:

- Binary data format for fast exchange of data with GAMS
- Stores sets, parameters, values of variables/equations with domain information, but no symbolic information (e.g., equation algebra)
- Consistency: no duplicates or contradictions
- Platform independent
- Can be compressed
- Read and write in GAMS:
  - on command line: parameter `gdx=<filename>`
  - during compilation: `$gdxin`, `$gdxout`, `$load`, `$unload`, ...
  - during execution: `execute_load`, `execute_unload`, ...
- Tools and APIs to read and write from other environments:
  - `gdxdump`, `csv2gdx`, `gdxviewer` (win only), ...
  - Matlab, MS Access, MS Excel, ODBC/SQL, R, SQLite
  - low-level APIs for C, C++, C#, Delphi, Fortran, Java, Python, VBA, VB.NET
  - high-level APIs for C#, Java, and Python

GDX – Gams Data eXchange Format

- Alternative format to read and write GAMS data (sets, parameters, variables, equations)
- Data file for multiple GAMS symbols
- Binary (no loss of precision)
- No Symbolic Equations
- Platform independent
- Contains domain information
- Validated data
  - no syntax errors on read
  - consistent: no duplicates, contradictions, etc.
- Can be compressed
- GDX Tools do not require a license
- Easily created by command line parameter `gdx=filename`

GDX – Gams Data eXchange Format

- Alternative format to read and write GAMS data (sets, parameters, variables, equations)
- Data file for multiple GAMS symbols
- Binary (no loss of precision)
- No Symbolic Equations
- Platform independent
- Contains domain information
- Validated data
  - no syntax errors on read
  - consistent: no duplicates, contradictions, etc.
- Can be compressed
- GDX Tools do not require a license
- Easily created by command line parameter `gdx=filename`
- Inspect GDX file with IDE
- Simple Exports from IDE
- Zoo of tools around GDX
  - IO tools (gdxxrw, sql2gms, ...)
  - Productivity tools (gdxdiff, gdxmerge)
- GDX API
  - C, C++, Java, C#, Python, Fortran, ...

GDX Tools

Tool	Description
ASK	The utility can be used to get input from an user interactively.
BIB2GMS	Analyses BibTeX files with file extension .bib and writes GAMS source files that can be used to create various author, reference and cross reference reports.
CHK4UPD	Checks whether the user can update to a more recent GAMS version.
CHOLESKY	Calculates the Choleksy decomposition of a symmetric positive definite matrix.
CSDP	The semidefinite programming CSDP solver from COIN-OR. The communication with CSDP requires the setup of matrix data structures in a CSDP input file. In a sense a GAMS model functions as a matrix generator.
CSV2GDX	Reads a CSV file (comma separated values) and writes to a GDX file.
EIGENVALUE	Calculates eigenvalues of a symmetric matrix.
EIGENVECTOR	Calculates eigenvector of a symmetric matrix.
ENDECRYPT	A tool to encrypt and decrypt text files.
GAMSIDE	GAMS Integrated Development Environment.

GDX Tools

Tool	Description
POSIX Utils	A collection of POSIX utilities which are usually available for Windows and the different Unix systems and therefore help to write platform independent scripts.
GDX2ACCESS	Converts GDX data to MS Access tables.
GDX2HAR	Translates files between GDX and HAR format.
GDX2SQLITE	Dumps the complete contents of a GDX file into a SQLite2 database. From Amsterdam Optimization Modeling Group.
GDX2VEDA	Translates a GDX file into the VEDA format.
GDX2XLS	Converts GDX data into a MS Excel spreadsheet.
GDXCOPY	Converts a GDX file into different GDX formats.
GDXDIFF	Compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file.
GDXDUMP	Writes scalars, sets and parameters (tables) to standard output formatted as a GAMS program with data statements.

GDX Tools

Tool	Description
GDXMERGE	Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.
GDXMRW	A suite of utilities to import/export data between GAMS and MATLAB and to call GAMS models from MATLAB and get results back into MATLAB.
GDXRANK	Reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.
GDXRENAME	Replaces UEL strings in GDX files.
GDXRRW	An interface between GAMS and R. It includes functions to transfer data between GDX and R and a function to call GAMS from R.
GDXTROLL	Translates a GDX file into the TROLL format.

GDX Tools

Tool	Description
GDXVIEWER	Views and converts data contained in GDX files.
GDXXRW	Preferred utility to read and write MS Excel spreadsheet data.
HAR2GDX	Translates files between GDX and HAR format.
IDECMDS	Sends commands to the GAMSIDE.
INVERT	Inverts a matrix.
MCFILTER	Removal of duplicate and dominated points in a multi-criteria solution set.
MDB2GMS	Converts data from an MS Access database into a GAMS readable format.
MODEL2TEX	Translates a GAMS model into LaTeX
MPS2GMS	Translates an MPS file into an equivalent short generic GAMS program using a GDX file to store data.
MSAPPAVAIL	Checks if a MS Office Application is available.
SCENRED	A tool for the reduction of scenarios that model random data processes of a stochastic program. From Humboldt-University Berlin.

GDX Tools

Tool	Description
SCENRED2	Scenred2 is a fundamental update of Scenred and offers a scenario tree construction algorithm. From Humboldt-University Berlin.
SHELLEXECUTE	Launches external programs from the command line.
SQL2GMS	Converts data from an SQL database into a GAMS readable format.
XLS2GMS	Converts spreadsheet data from a MS Excel spreadsheet into a GAMS readable format.
XLSDUMP	Writes all worksheets of a MS Excel workbook to a GDX file. Unlike gdxxrw, the program does not require that Excel is installed.
XLSTALK	Open/Close/Run macro in MS Excel.

Compile time vs. Execution time GDX

- Compile time GDX
  - `$gdxin filename` connects to GDX file for reading
  - `$gdxout filename` connects to GDX file for writing
  - `$gdxin` and
    texttt$gdxout closes connection to GDX file
  - `$load[DC][M,R] symb[=name,<name[.dimN]]`
  - `$unload symb=name`
- Execution time GDX
  - `Execute_load[DC] ‘filename’, symb[=name]`
  - `Execute_loadpoint[DC] ‘filename’ [, symb[=name]]`
  - `Execute_unload[DI] ‘filename’ [, symb[=name]]`

GDX Limitations

- Cannot add records or symbols
  - e.g.: combine two gdx files
  - GDX is immutable
- Not self contained wrt GAMS:
  - Needs declarations
- Zero vs non-existent
  - GAMS is a sparse system. It does not store 0 (Zero)

Detour: Special Values

GAMS represents data as double precision numbers (no integers) plus some special values:

- `+inf`, `-inf` (infinity)
- `NA` (not available)
- `UNDF` (undefined, cannot be part of input unless `$onundf`)
- `EPS` (numerical zero, logically present)
- Careful with numerical calculations (e.g. `0=EPS` is true)
  - What is `0*inf`, `0/inf`, `eps/inf`, `eps/inf + eps`
  - Model Library model crazy gives all answers
  - Mapval to check for a special value (`mapval(x)>0`, `mapval(x)=mapval(undf)`)

Outline I

Subsets, Cardinality

Subsets:

- Syntax for set_name1 $\subseteq$ set_name2: `set set_name1 (set_name2);`
- Example:

```
set ice          / chocolate, strawberry, cherry, vanilla /;
set sorbet(ice) /            strawberry, cherry          /;
```

- Domain checking:

```
set sorbet(ice) / strawberry, banana /;
```

$\Rightarrow$ error

Subsets, Cardinality

Subsets:

- Syntax for `set_name1` ⊆ `set_name2`: `set set_name1 (set_name2);`
- Example:

```
set ice          / chocolate, strawberry, cherry, vanilla /;
set sorbet(ice) /            strawberry, cherry          /;
```

- Domain checking:

```
set sorbet(ice) / strawberry, banana /;
```

⇒ error

`Card`(set)

- gives the number of elements in a set
- Example:

```
set c     'countries'  / jamaica, haiti, guyana, brazil /;
scalar nc 'number of countries;
nc = card(c);
```

Ordered Sets

Lag & Lead Operations:

- allow to access neighbors (next or further distant) of elements in a priori explicitly specified ordered set
- Syntax: `setelement` $\pm$ `n`
- Note: `x(setelement+n)` is zero if position of setelement $>$ `card(set)-n`
- Example:

```
Set t / 1*24 /;
Variables level(t), inflow(t), outflow(t);
Equation balance(t)  couple fill levels of reservoir over time;
* implicitly assume an initial fill level of zero
balance(t).. level(t) =e= level(t-1) + inflow(t) - outflow(t);
```

Ordered Sets

Lag & Lead Operations:

- allow to access neighbors (next or further distant) of elements in a priori explicitly specified ordered set
- Syntax: `setelement` $\pm$ `n`
- Note: `x(setelement+n)` is zero if position of setelement $>$ `card(set)-n`
- Example:

```
Set t / 1*24 /;
Variables level(t), inflow(t), outflow(t);
Equation balance(t)  couple fill levels of reservoir over time;
* implicitly assume an initial fill level of zero
balance(t).. level(t) =e= level(t-1) + inflow(t) - outflow(t);
```

`Ord`(setelement):

- gives position of an element in an a priori explicitly specified ordered sets
- Example:

```
Set t / 1*24 /;
Parameter hour(t);
hour(t) = ord(t);
```

Dynamic Sets

- dynamic sets allow elements to be added or removed
- dynamic sets are usually domain-checked, i.e., subsets
- Syntax: `setname(othersetelement) = yes | no` (add/remove single element)
- Syntax: `setname(subset) = yes | no` (add/remove another subset)
- Example:

```
set ice / chocolate, strawberry, cherry, vanilla /;
Parameter demand(ice) / chocolate 1000, strawberry 500,
                        cherry      10, vanilla    100 /;

set sorbet(ice);
sorbet(ice) = yes;
sorbet('chocolate') = no;   sorbet('vanilla') = no;

Set highdemand(ice)  ice creams with high demand;
highdemand(ice) = (demand(ice) >= 500);
```

- most often used as controlling index in an assignment or equation definition

```
Scalar sumhighdemand;
sumhighdemand = sum(highdemand, demand(highdemand));
```

Example: refer to first/last period of discrete-time models

```
Set t / 1*24 /;
Sets tb(t)  base period
     tn(t)  non-base periods
     tt(t)  terminal period;
tb(t) = (ord(t) = 1);
tn(t) = (ord(t) > 1);
tt(t) = (ord(t) = card(t));
Variables level(t), inflow(t), outflow(t);
Equations balance(t)     couple fill levels of reservoir over time
          basebalance(t) define fill level for base period;
* only for time periods > base period
balance(tn(t)).. level(t) =e= level(t-1) + inflow(t) - outflow(t);
* only for base period
basebalance(tb).. level(tb) =e= 100 + inflow(tb) - outflow(tb);
* lower bound on fill level in terminal period
level.lo(tt) = 100;
```

Alternatively (but less readable):

```
equation basebalance;  basebalance..
   sum(tb, level(tb)) =e= 100 + sum(tb, inflow(tb) - outflow(tb));
```

Multidimensional Sets

Multidimensional Sets:

- describing assignments (relations) between sets
- Example:

```
sets c  'countries'  / jamaica, haiti, guyana, brazil /
     h  'harbors'    / kingston, s-domingo, georgetown, belem /;
set  hc(h, c)  harbor to country relation
     / kingston.jamaica,   s-domingo.haiti
       georgetown.guyana,  belem.brazil /;
```

Singleton Sets

- A singleton set is a special set that is either empty or a singleton, i.e., has zero or one elements:

```
Set             i      / a, b, c /;
Singleton Set   j      / d       /
                k(i)   / b       /
                l(i,j) / c.d     /;
```

- Data statements with more than 1 element will create a compilation error:

```
   1  Singleton Set s / s1*s3 /;
****                          $844
   2  display s;

Error Messages
844  Singleton with more than one entry (see $onStrictSingleton)
```

- Useful to simplify access to parameters on a dynamically defined element:

```
Set t / 1*12 /;
Parameter a(t);
Singleton Set tb(t); tb(t) = (ord(t) = 1);
```

Can now use `a(tb)` instead of `sum(tb,a(tb))`.

sameas and diag

sameas(setelement, otherelement) and sameas(setelement, “text”)
diag(setelement, otherelement) and diag(setelement, “text”)

- sameas returns true if identifiers for given set elements are the same, or if identifier of one set element equals a given text
- sameas can also be used as a set
- diag is like sameas, but return 1 if true, and 0 otherwise
- Example:

```
sets ice1 / chocolate, strawberry, cherry, vanilla /
     ice2 / strawberry, cherry, banana /;
scalar ncommon;
ncommon = sum((ice1, ice2), diag(ice1,ice2));
ncommon = sum(sameas(ice1, ice2), 1);
```

Conditional expressions: $ Operator

Boolean Operators:

- numerical operators: `lt`, `<`, `le`, `<=`, `eq`, `=`, `ne`, `<>`, `ge`, `>=`, `gt`, `>`
- logical operators: `not`, `and`, `or`, `xor`
- set membership: `a(i)` evaluates to *true* if and only if `i` is contained in the (sub)set `a`, otherwise *false*
- *true* corresponds to 1, *false* to 0

Conditional expressions: $ Operator

Boolean Operators:

- numerical operators: `lt`, `<`, `le`, `<=`, `eq`, `=`, `ne`, `<>`, `ge`, `>=`, `gt`, `>`
- logical operators: `not`, `and`, `or`, `xor`
- set membership: `a(i)` evaluates to *true* if and only if `i` is contained in the (sub)set `a`, otherwise *false*
- *true* corresponds to 1, *false* to 0

$-Operator:

- allows to apply necessary conditions
- `$(condition)` can be read as "if `condition` is true"
- Example:
  "If `b`>1.5, then let `a`= 2." ⇒ `a$(b > 1.5) = 2;`
  "If `b`>1.5, then let `a`= 2, otherwise let `a`= 0." ⇒ `a = 2$(b > 1.5);`
- $ on left side: no assignment, if `condition` not satisfied
- $ on right side: always assignment, but term with $ evaluates to 0, if `condition` not satisfied
- the combination `$= term` is a short form of $ on the left side with condition `term` and assignment to `term`:
  - `a $= b;` ⇒ `a$(b>0) = b;`
- cannot be used in declarations

Applications for $ Operator

Filtering in indexed operations:

```
parameter sorbetbalance;
set ice; set sorbet(ice);
sorbetbalance = sum(ice$sorbet(ice), price(ice)*purchase.l(ice));
sorbetbalance = sum(sorbet(ice), price(ice) * purchase.l(ice));
```

Applications for $ Operator

Filtering in indexed operations:

```
parameter sorbetbalance;
set ice; set sorbet(ice);
sorbetbalance = sum(ice$sorbet(ice), price(ice)*purchase.l(ice));
sorbetbalance = sum(sorbet(ice), price(ice) * purchase.l(ice));
```

Conditioned indexed operations:

```
rho = sum(i$(sig(i) ne 0), 1/sig(i) - 1);
```

Applications for $ Operator

Filtering in indexed operations:

```
parameter sorbetbalance;
set ice; set sorbet(ice);
sorbetbalance = sum(ice$sorbet(ice), price(ice)*purchase.l(ice));
sorbetbalance = sum(sorbet(ice), price(ice) * purchase.l(ice));
```

Conditioned indexed operations:

```
rho = sum(i$(sig(i) ne 0), 1/sig(i) - 1);
```

Conditioned equations:

```
Equation satdemand(ice);
satdemand(ice)$sorbet(ice).. purchase(ice) =g= demand(ice);

balance(t)$(ord(t)>1)..
                 level(t) =e= level(t-1) + inflow(t) - outflow(t);
```

Existence of variables in constraints:

```
variables x, y;   parameter A;   equation e;
e.. x + y$(A>2) =e= A;
```

No Variables in $ condition

- The following DOES NOT WORK:

```
binary variable x;  variable y;   equation e1, e2;
e1$(x = 1).. y          =l= 100;
e2         .. y$(x = 1) =l= 100;
```

- The value of x is decided by the solver, not by GAMS.
- However, GAMS has to evaluate $-operators when assembling an instance in the `Solve` statement.
- Instead, you have to reformulate:

```
e.. y =l= 100 * x + y.up * (1-x);
```

That is: $x=1 \Rightarrow y \leq 100$; $x=0 \Rightarrow y \leq y.up$.

Outline I

Finding a good local optimum to a NLP: Multistart

- Starting an NLP solver from different starting points and pick the best solution.
- If we don't know how to pick a good point, let's pick one randomly.

Finding a good local optimum to a NLP: Multistart

- Starting an NLP solver from different starting points and pick the best solution.
- If we don't know how to pick a good point, let's pick one randomly.
- Multistart algorithm:
  1. $f^U = \infty$
  2. for $k = 1$ to N, do
     2.1 Generate starting point x uniformly at random over $[\underline{x}, \overline{x}]$.
     2.2 Run NLP solver from x and obtain solution x^*.
     2.3 if $f(x^*) < f^U$: $f^U = f(x^*)$ and $x^U = x^*$
  3. Return x^U and f^U.
- The GAMS solver MSNLP implements such an algorithm.

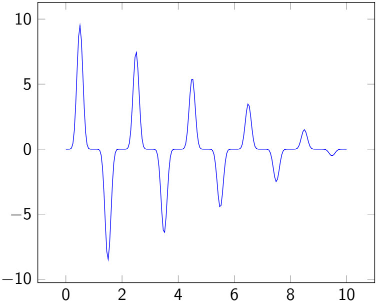
Finding a good local optimum to a NLP: Multistart

- Starting an NLP solver from different starting points and pick the best solution.
- If we don't know how to pick a good point, let's pick one randomly.
- Multistart algorithm:
  1. $f^U = \infty$
  2. for $k = 1$ to N, do
     2.1 Generate starting point x uniformly at random over $[\underline{x}, \overline{x}]$.
     2.2 Run NLP solver from x and obtain solution x^*.
     2.3 if $f(x^*) < f^U$: $f^U = f(x^*)$ and $x^U = x^*$
  3. Return x^U and f^U.
- The GAMS solver MSNLP implements such an algorithm.

- Example:

$$\max_{x \in [0,10]} \phi(x),$$

$$\text{where } \phi(x) = (10 - x)\sin^9(2\pi x)$$



- Optimal solution:
  $x = 0.24971128$, $\phi(x) = 9.75014433$
- File: `nlpsin.gms`

Program flow control, loop command

- controlling the execution of a GAMS program
- commands: `loop`, `if-else`, `while`, `for`
- declarations and definition of equations are not allowed inside these commands
- solve statements are allowed

Program flow control, loop command

- controlling the execution of a GAMS program
- commands: `loop`, `if-else`, `while`, `for`
- declarations and definition of equations are not allowed inside these commands
- solve statements are allowed

`loop` command:

- Syntax:

```
loop(controllingset[$(condition)],
     statement {; statement}
    );
```

- Example:

```
set t / 1985*1990 /;
parameter pop(t) / 1985  3456 /
          growth(t) / 1985  25.3,  1986  27.3,  1987  26.2,
                      1988  27.1,  1989  26.6,  1990  26.6 /;
loop(t, pop(t+1) = pop(t) + growth(t));
```

If-Elseif-Else command

- Syntax:

```
if( condition, statement {; statement};
{elseif condition, statement {; statement}; }
[else statement {; statement};]
);
```

- Example:

```
if( (ml.modelstat eq %ModelStat.Infeasible%),
*      model ml was infeasible,
*      relax bound and solve again
  x.up(j) = 2*x.up(j);
  solve ml using lp min z;
elseif ml.modelstat eq %ModelStat.Optimal%,
  display x.l;
else
  abort "Error solving the model";
);
```

While, Repeat commands

`While` command:

- Syntax: `while(condition, statement {; statement}; );`
- Example:

```
scalar count / 1 /;
scalar globmin / inf /;
while(count le 1000,
  x.l(j) = uniform(x.lo(j), x.up(j));
  solve ml using nlp min obj;
  if ( obj.l le globmin, globmin = obj.l; );
  count = count + 1;
);
```

While, Repeat commands

`While` command:

- Syntax: `while(condition, statement {; statement}; );`
- Example:

```
scalar count / 1 /;
scalar globmin / inf /;
while(count le 1000,
  x.l(j) = uniform(x.lo(j), x.up(j));
  solve ml using nlp min obj;
  if ( obj.l le globmin, globmin = obj.l; );
  count = count + 1;
);
```

`Repeat` command:

- Syntax: `repeat(statement {; statement}; until condition );`
- Example:

```
scalar count / 1 /;  scalar globmin / inf /;
repeat( x.l(j) = uniform(x.lo(j), x.up(j));
        solve ml using nlp min obj;
        if ( obj.l le globmin, globmin = obj.l; );
        count = count + 1;
until count eq 1000 );
```

For command

`for(i = start to|downto end [by incr], statement {; statement};);`

- Note: `i` is a scalar, not a set
- `start`, `end`, and `incr` can be real numbers, but `incr` needs to be positive
- Example:

```
scalar count;
scalar globmin / inf /;
for( count = 1 to 1000,
  x.l(j) = uniform(x.lo(j), x.up(j));
  solve ml using nlp min obj;
  if ( obj.l le globmin, globmin = obj.l; );
);
```

Exercise: Solve the Cows & Pigs example by Complete Enumeration in GAMS

x_1 the number of cows to purchase ($x_1 \in \{0, 1, 2\}$)

x_2 the number of pigs to purchase ($x_2 \in \{0, 1, 2\}$)

maximize $z = 3x_1 + 2x_2$

such that $x_1 + x_2 \leq 3$

Exercise: Solve the Cows & Pigs example by Complete Enumeration in GAMS

x_1 the number of cows to purchase ($x_1 \in \{0, 1, 2\}$)

x_2 the number of pigs to purchase ($x_2 \in \{0, 1, 2\}$)

maximize $z = 3x_1 + 2x_2$

such that $x_1 + x_2 \leq 3$

```
scalar x1, x2, obj;
scalar objbest, x1best, x2best;
objbest = 0;
for( x1 = 0 to 2,
  for( x2 = 0 to 2,
    if( x1 + x2 <= 3,
      obj = 3*x1 + 2*x2;
      if( obj > objbest,
        x1best = x1;
        x2best = x2;
        objbest = obj;
   ))))
display x1best, x2best, objbest;
```

File: `cowspigsenum.gms`