GAMS – Hinweise für Fortgeschrittene

Matthes Koch

Institut für Verkehrswirtschaft

Universität Hamburg

02.11.2016

Universal Set - Konzept

- Alle im Skript definierten Mengenelemente werden zusätzlich automatisch in eine globale *universal set* genannte Menge gespeichert
- Die Reihenfolge der Mengenelemente im *universal set* entspricht der Definitionsreihenfolge im Gams-Skript
- Zugriff auf das *universal set*:

```
set v /v4*v6/;
set w /w1*w3,v1*v4/;

alias(*,pool);

display pool;
```

```
----      9 SET pool  Aliased with *
v4,    v5,    v6,    w1,    w2,    w3,    v1,    v2,    v3
```

Universal Set – Reihenfolgen und ord()

```
set y1 Jahre /y_2010*y_2012/;
set y2 Jahre /y_2009*y_2011/;

alias(*,pool);
display y1,y2,pool;
```

```
----      5 SET y2  Jahre

y_2010,    y_2011,    y_2009
```

Merkwürdig?
Reihenfolge der Elemente in y2 entspricht nicht der Definitions-Reihenfolge

Grund: Element ‚y_2009' ist das zuletzt zum *universal set* hinzugefügte Element!
Elemente ‚y_2010' und ‚'y_2011' wurden bereits bei der Definition von Menge y1 hinzugefügt.

```
----      5 SET pool  Aliased with *
y_2010,    y_2011,    y_2012,    y_2009
```

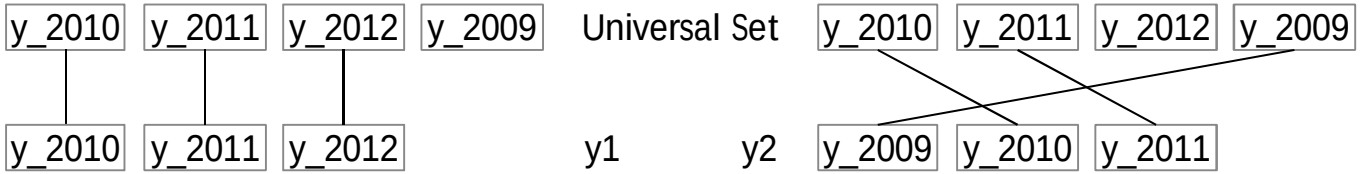
Reihenfolge der Elemente in y2 entspricht nicht der Reihenfolge seiner Elemente im *universal set*

- Ø y2 ist sog. *unordered set*
- Ø Ord()-Funktion nicht verfügbar (!)

Universal Set - Reihenfolgen und ord()

```
set y1 Jahre /y_2010*y_2012/;
set y2 Jahre /y_2009*y_2011/;
parameter c(y2);
c(y2) = ord(y2);
```

```
*** Error 198 in C:\Users\matth\CloudStation\UHH\Adv.GAMS\Untitled_3.gms
    Set used in 'ord' or lag is not ordered.
       Hint: Some of the elements of the set were used before this
       was initialized and the order was different from the order used
       in this set. Try to initialize the set earlier.
```



geordnete Menge (kreuzungsfrei)
ord()-Funktion verfügbar

ungeordnete Menge (nicht kreuzungsfrei)
ord()-Funktion nicht verfügbar

Hinweis: ord() Funktion ist auch bei dynamischen Mengen nicht verfügbar:

```
*dynamisches set (Zuweisung)
set y3(*);
y3('y_2010')=yes;
y3('y_2012')=yes;
```

Ø Workaround: vorab Dummy-Menge definieren

```
set y0 Dummy /y_2009*y_2012/;
set y1 Jahre /y_2010*y_2012/;
set y2 Jahre /y_2009*y_2011/;
alias(*,pool); display y2;
```

```
----      4 SET y2  Jahre

y_2009,    y_2010,    y_2011
```

Universal Set - Nutzen

- Einfaches Zufügen zum Universal Set, z.b. bei der Aufbereitung von Modelstatistiken (*Model attributes*)
- Parameter/Subset etc. abhängig von (*) deklarieren:

```
model transport /all/;
solve transport using lp minimizing z;
parameter Stats(*) Model Statistiken;
Stats('Rechenzeit')      = transport.resusd;
Stats('Iterationen')     = transport.iterUsd;
Stats('#Variablen')      = transport.numVar;
Stats('#Variablen_int') = transport.numDVar;
Stats('#Gleichungen')    = transport.numEqu;
Stats('ModelStatus')     = transport.modelStat;
Stats('LoesungsStatus') = transport.solveStat;
execute_unload "stats.gdx" stats;
```

Entry	Symbol	Type	Dim	Nr Elem
1	Stats	Par	1	6

Stats(*): Model Statistiken

Rechenzeit	0.031
Iterationen	7
#Variablen	16
#Gleichungen	9
ModelStatus	1
LoesungsStatus	1

Tipp: Auslagern von Solver-Statistiken

- Mit $batInclude Datei einbinden und Argumente übergeben
- GAMS-Code der eingebundenen Datei ist wiederverwendbar (z.B. innerhalb einer Schleife oder projektübergreifend)
- Beispiel: ein Skript wird eingebunden, dabei Übergabe von Modellname und Hilfsparameter, im eingebundenen Skript: Nutze %1, %2,... als Platzhalter für Argumente

Haupt GAMS-Skript

```
(...)
model netConstruct /all/;
(...)
parameter statPar(*,*);

$batInclude "stats.inc" netConstruct statPar

execute_unload "stats.gdx" statPar;
```

Datei: stats.inc

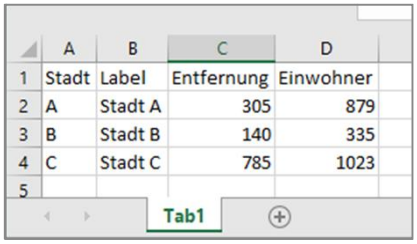
```
%2('%1','Rechenzeit')      = %1.resusd;
%2('%1','Iterationen')     = %1.iterusd + eps;
%2('%1','Variablen')       = %1.numVar + eps;
%2('%1','Variablen_int')   = %1.numDVar + eps;
%2('%1','Gleichungen')     = %1.numequ + eps;
%2('%1','Modelstatus')     = %1.modelstat;
%2('%1','LoesungsStatus')  = %1.solvestat;
```

Datei: stats.gdx

	netConstruct
Rechenzeit	0.218
Iterationen	902
Variablen	1266
Variablen_int	55
Gleichungen	716
Modelstatus	8
LoesungsStatus	1

Datentabellen

Ø Oftmals liegen Daten in Tabellenform vor, etwa aus MS Excel:



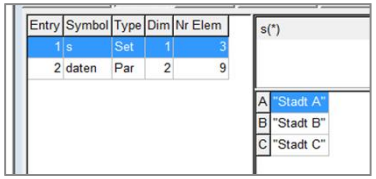
	A	B	C	D
1	Stadt	Label	Entfernung	Einwohner
2	A	Stadt A	305	879
3	B	Stadt B	140	335
4	C	Stadt C	785	1023
5				

Tab1

Datentabelle

zweidimensionaler Parameter

```
$onUNDF
set S Staedte;
parameter
 d(S) Entfernung,
 e(S) Einwohner;

Parameter daten(s,*) Datentabelle (Excel);
*Tabelle in GDX Datei laden:
$call "gdxxrw daten.xlsx se=0 set=s rng=Tab1!A2 rdim=1 cdim=0 par=daten rng=Tab1!A1:D4"
*GDX Daten in GAMS laden:
$GDXIN "daten.gdx"
$load s daten
$gdxin
*Datentabelle in Gams Parameter überführen:
d(s) = daten(s,'Entfernung');
e(s) = daten(s,'Einwohner');
*Geladene Symbole in GAMS anzeigen:
display S,d,e;
```

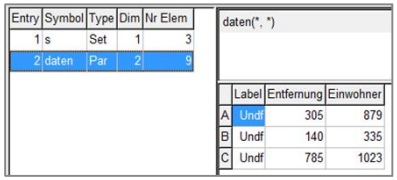


Entry	Symbol	Type	Dim	Nr Elem
1	s	Set	1	3
2	daten	Par	2	9

s(*)

A	"Stadt A"
B	"Stadt B"
C	"Stadt C"

Ø GDX Datei:

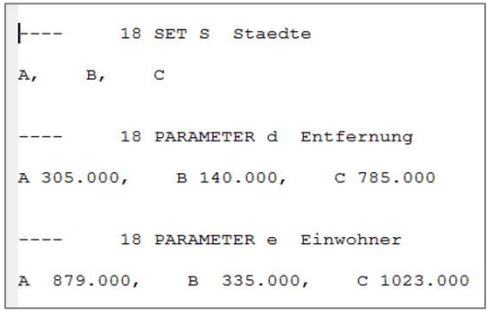


Entry	Symbol	Type	Dim	Nr Elem
1	s	Set	1	3
2	daten	Par	2	9

daten(*, *)

	Label	Entfernung	Einwohner
A	Undf	305	879
B	Undf	140	335
C	Undf	785	1023

Ø Nutzung des flexiblen universal sets: Spaltenüberschriften (Attribute der Zeilen) können Elemente des Universal Sets sein und eingelesen werden



```
----      18 SET S  Staedte

A,    B,    C


----      18 PARAMETER d  Entfernung

A 305.000,    B 140.000,    C 785.000


----      18 PARAMETER e  Einwohner

A  879.000,    B  335.000,    C 1023.000
```

Ø GAMS Listing

Mengenelemente mit Kommentaren

- Mengenelemente können individuelle Kommentare erhalten
- Zugriff im GAMS-Skript über .TE
- Zugriff auf den Namen des Elements über .TL

```
set S Staedte /S1 Hamburg, S2 Berlin, S3 Dortmund /
parameter d(s) Bevoelkerungsdichte /S1 2367, S2 3948, S3 2088/;

file out /out.txt/;
put out;
put @1 "Stadt", @10 "Code", @15 "Dichte" //;
loop(s,
 put @1 s.TE(s), @10 "[", @11 s.TL, @13 "]", @15 d(s):<5:0 /;
);
putclose;
```

- Ausgabedatei out.txt:

```
Stadt     Code Dichte

Hamburg   [S1] 2367
Berlin    [S2] 3948
Dortmund  [S3] 2088
```

Spezielle Werte: EPS

- EPS:
  - bildet einen numerischen Wert = 0 ab
  - in logischen statements (boolsche Vergleiche) ist eps stets wahr
  - der Wert = 0 ist dagegen in logischen statements stets falsch
  - eps: explizites Ausdrücken von Werten = 0 (die normalerweise in GAMS nicht mitgespeichert/ausgegeben werden und logisch „falsch“ sind)

$-Bedingung ist wahr, Berechnung aufgrund der Division durch 0 fehlerhaft:

```
scalar x,y;
x = eps; y = 5;
y$x = 1/x;
display x,y;
```

```
**** Exec Error at line 3: division by eps

----      4 PARAMETER x                  =          EPS
            PARAMETER y                  =        UNDF
```

$-Bedingung ist falsch, Berechnung wird nicht ausgeführt:

```
scalar x,y;
x = 0; y = 5;
y$x = 1/x;
display x,y;
```

```
----      4 PARAMETER x                  =        0.000
            PARAMETER y                  =        5.000
```

Spezielle Werte: EPS

- Degeneration: Nichtbasisvariablen mit reduzierten Kosten = 0 ?
- in GAMS: marginals solcher Nichtbasisvariablen = EPS
- für BV gilt marginals = 0 (reduzierte Kosten)

Optimierungsproblem

$$\max F = x_1 + x_2$$

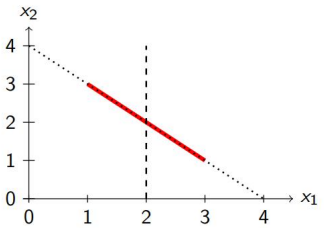
unter den Nebenbedingungen

$$\begin{aligned} x_1 + x_2 + y_1 &= 4 \\ x_1 + y_2 &= 2 \end{aligned}$$

$$x_1, x_2, y_1, y_2 \geq 0$$

Simplextableau

BV	x_1	x_2	y_1	y_2	b
x_2	1	1	1	0	4
y_2	1	0	0	1	2
F	0	0	1	0	4



ZF liegt auf erster Nebenbedingung (gepunktete Gerade): (2,2), (0,4) sowie alle Linearkombinationen dieser beiden Punkte sind optimal

```
---- VAR x

     LOWER      LEVEL      UPPER     MARGINAL

1      .          .         +INF        EPS
2      .        4.000       +INF         .
```

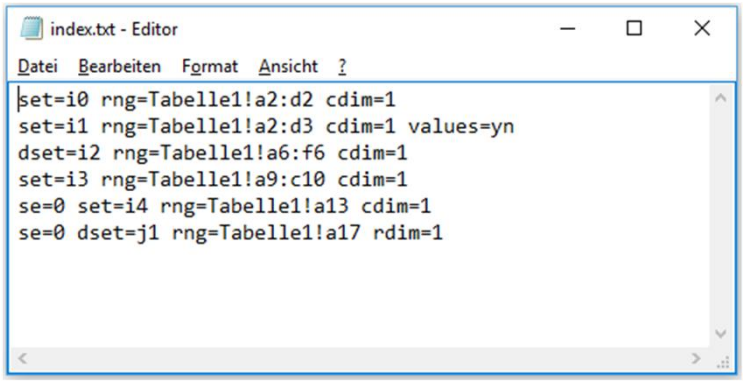
- reduzierte Kosten = 0, Basisvariable
- reduzierte Kosten = EPS, Nichtbasisvariable, aber rc = 0
- EPS also „0 mit Wert“

Nebenbedingungen im Listing:

```
---- EQU NB

     LOWER      LEVEL      UPPER     MARGINAL

1      4.000      4.000      4.000      1.000
2      2.000      2.000      2.000      EPS
```

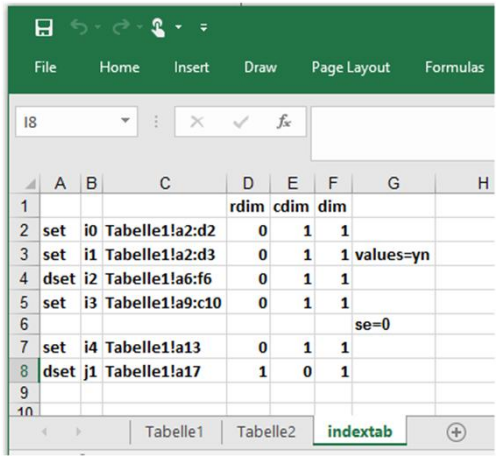
Viele Symbole lesen: Index-Datei / Index-Sheet

```
sets
 i0 Elemente der Zeile
 i1 ausgewählte Elemente einer Zeile
 i2 Menge deren Input Duplikate enthielt
 i3 Elemente mit Kommentar ausgelesen
 i4 Menge mit Hilfe von SE=0 begrenzt
 j1 Elemente aus einer Spalte ausgelesen;

*GDX-XRW Aufruf mit Index Text-Datei:
$call "gdxxrw beispiel.xls @index.txt"
*GDX Daten laden:
$gdxin beispiel.gdx
$load i0 i1 i2 i3 i4 j1
$gdxin
display
i0,i1,i2,i3,i4,j1;
```



Index-Datei als .txt

```
sets
 i0 Elemente der Zeile
 i1 ausgewählte Elemente einer Zeile
 i2 Menge deren Input Duplikate enthielt
 i3 Elemente mit Kommentar ausgelesen
 i4 Menge mit Hilfe von SE=0 begrenzt
 j1 Elemente aus einer Spalte ausgelesen;

*GDX-XRW Aufruf mit Index in Exceldatei:
$call gdxxrw beispiel.xls index=indextab!a1

$gdxin beispiel.gdx
$load i0 i1 i2 i3 i4 j1
$gdxin
display
i0,i1,i2,i3,i4,j1;
```



	A	B	C	D	E	F	G	H
1				rdim	cdim	dim		
2	set	i0	Tabelle1!a2:d2	0	1	1		
3	set	i1	Tabelle1!a2:d3	0	1	1	values=yn	
4	dset	i2	Tabelle1!a6:f6	0	1	1		
5	set	i3	Tabelle1!a9:c10	0	1	1		
6							se=0	
7	set	i4	Tabelle1!a13	0	1	1		
8	dset	j1	Tabelle1!a17	1	0	1		
9								
10								

Index als Tabellenblatt in Excel

Optionsdateien für Solver

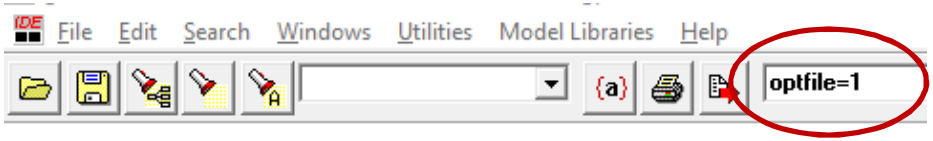
- Fine-Tuning der Lösungsverfahren mit Optionsdateien
- Einfache Textdateien: solvername.opt (z.B. cplex.opt)
- .opt Datei muss im aktuellen Projektverzeichnis liegen
- möglich: .opt Datei mit Befehlen innerhalb des .gms Skript schreiben (vor solve) – vgl. Folie 17

Verwendung im .gms Skript

- Für jedes Modell ein Attribut setzen

```
model Transportmodell /all/;
Transportmodell.optfile=1;
```

- Für alle Modelle des Skriptes per Command Line



Mehrere Optionsdateien im Skript

Modelattribut	Optionsdatei
.optfile=1	Solvername.opt
.optfile=2	Solvername.op2
.optfile=15	Solvername.o15
.optfile=200	Solvername.200
.optfile=1234	Solvername.1234
Optfile=0	Keine Datei (Standard)

Optionsdatei: Häufig genutzte Optionen

- Optionen sind in den Solverdokumentationen beschrieben (GAMSIDE -> Help -> GAMS Documentation)

Häufig genutzte Optionen in CPLEX.opt (Auszug):

`threads <Zahl>` : Anzahl paralleler CPU-Threads die dem Solver zur Verfügung stehen sollen. Bei negativen Werten: Anzahl nicht durch den solver nutzbarer CPU Threads (für andere Aufgaben freigehalten) -> Wert auf 0 stellt alle verfügbaren CPU Threads für den Solver zur Verfügung

`parallelmode <Zahl>` : Art der parallelen Verfahren:

- -1: Opportunisitsch: Maximales Ausnutzen von Parallelität, Rechenverlauf und Rechenergebnisse können sich aber von Lauf zu Lauf unterscheiden (z.B. falls verschiedene Optima möglich)
- 0: Automatisch: falls Threads>1 gesetzt, opportunistisch; falls Threads=0 gesetzt, deterministisch, falls Threads=1, kein parallleles Rechnen
- 1: deterministisch: Ausnutzen von Parallelität, aber Rechenverlauf und Rechenergebnisse sind rekonstruierbar
- Standard = 1

Cplex.opt: Häufig genutzte Optionen

`Lpmethod <Zahl>` : Bestimmt Lösungsverfahren für lineare Programme, bei Wahl des Parallelmodus, d.h. „lpmethod 6“ werden verschiedene Verfahren auf verschiedenen Threads durchgeführt. Das schnellste Verfahren gibt dann die Lösung zurück,
Standard = 0

Wert	0	1	2	3	4	5	6
Bedeutung	Automatisch	Simplex Algorithmus			Barrier	Sifting	Parallel
		Primal	Dual	Netzwerk			

`memoryemphasis <Zahl>` : Wenn auf 1 gesetzt, versucht cplex speichersparend zu arbeiten – Optimierung kann langsamer werden, bei sehr großen Problemen und limitierter Hardware aber manchmal sinnvoll,
Standard = 0

`preind <Zahl>` : Presolve an (1) oder aus (0) stellen. Wenn auf 1 gesetzt, versucht der Solver, das Problem vor dem Lösen zu vereinfachen und Redundanzen zu entfernen. In einigen Fällen, kann das Problem ohne Presolve schneller gelöst werden.
Standard = 1

Cplex.opt: Häufig genutzte Optionen

`mipstart <Zahl>` : Solver probiert, aktuelle Variablenwerte als Startlösung des Optimierungsverfahrens zu verwenden, Standard = 0

Wert	Erklärung
0	Keine Startlösung verwenden
1	Nutze Ganzzahlige Variablenwerte mit Test auf Zulässigkeit
2	Nutze alle Variablenwerte mit Test auf Zulässigkeit
6	Nutze ganzzahlige Variablenwerte ohne Zulässigkeitstest vor der Optimierung

`writelp <Dateiname>` : Solver erstellt eine Ausgabedatei im Projektverzeichnis, in der das gesamte Lineare Programm explizit ausgeschrieben wird.

`probe <Zahl>` : Voruntersuchung (*Probing*) von MIP Modellen kann Lösungszeit verkürzen, bei einigen Problemen aber auch sehr zeitaufwändig werden:

- Ø -1: Probing ausschalten
- Ø 0: Automatisch (Standard)
- Ø 1 – 3: je höher der Wert, desto umfangreicher das Probing-Verfahren

Solution Pool

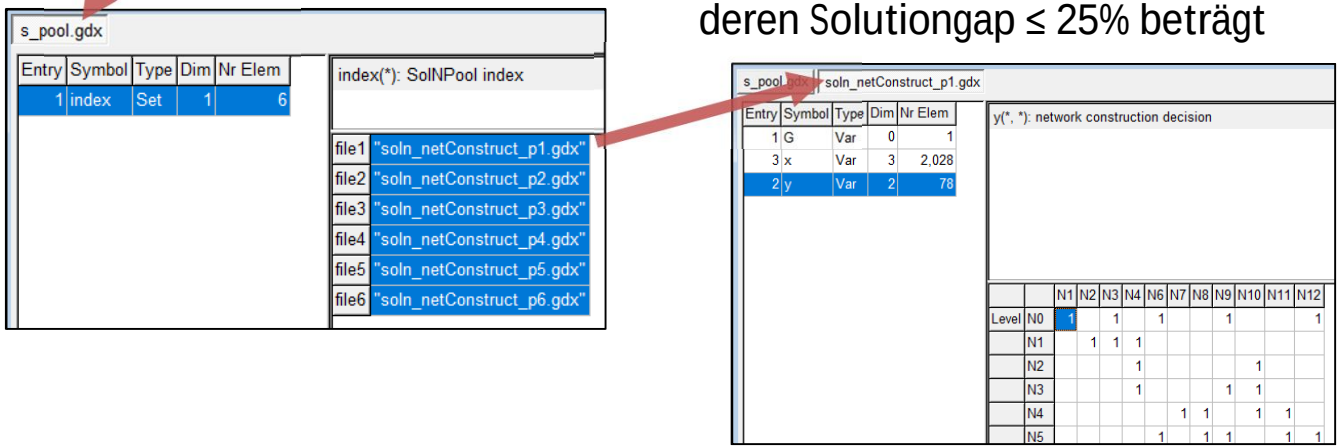
- Anweisung zum Speichern zusätzlicher Lösungen für MIP/MIQCP Modelle
- Lösungen werden im Solutionpool abgelegt
- Anwendungsbeispiele:
  - Modelle mit Mehrfachzielsetzung, einzelne Ziele sind schwer in Zielfkt. zu formalisieren (insb. in linearer Form)
  - Mehrere optimale/gute Lösungen sind in der Entscheidungsunterstützung explizit erwünscht
  - Numerische Studien

Solution Pool

Aktivierung des Solution Pool Features

```
$Onecho > cplex.opt
parallelmode 0
threads 0
solnpool s_pool.gdx
solnpoolgap 0.25
$Offecho
```

- Aktivierung durch Option solnpool
- Jede gefundene Lösung wird in einer .gdx Datei gespeichert
- s_pool.gdx enthält nach dem GAMSlauf alle Namen der .gdx Dateien (wie ein Verzeichnis)
- viele Konfigurationsmöglichkeiten, hier z.B.
  - suche alle Lösungen während des Branch-&-Bound Prozesses, deren Solutiongap ≤ 25% beträgt

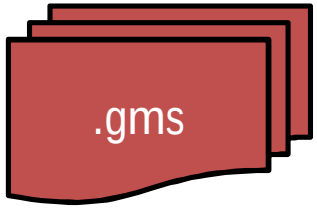


Solution Pool

- Lösungen des Pools können im GAMS Skript weiter verwertet werden
- Konfigurationsmöglichkeiten:
  - Auswahl des Lösungen
    - SolutionGap
    - Unterschiedlichkeit der Lösungen
    - Auswahl aufgrund von Eigenschaften der Lösungen
    - FIFO
    - Solutionpoolgröße
  - Generieren der Lösungen
    - Während B&B Prozess (Incumbents)
    - Spezielle Prozedur

> Solver Dokumentation

Kompilieren vs. Ausführen



Kompilieren

- $CALL / $CALL.ASYNCNC
- $INCLUDE
- $GDXIN
- $LOAD | $LOADDC
- $GDXOUT
- $UNLOAD
- $Echo
- $Onecho / $Offecho

Ausführen

- Execute / Execute.ASyncNC
- Execute_Unload
- Execute_Load
- Put ($Onput / $Offput)
- Solve

- Ergebnisse aus Wertberechnungen und solve statements können erst während Ausführung des Skriptes genutzt werden
- $-Anweisungen werden vor Ausführung des Skriptes durchgeführt (*compile-time*)
- keine Ergebnisse aus GAMS Skript in $-Commands nutzen!

Kompilieren vs. Ausführen

Beispiele

```
Model mo1 /all/;
mo1.optfile = 1;
solve mo1 max F using mip;

****** // Solver Option file ******
$Onecho > cplex.opt
parallelmode 0
threads 0
mipstart 1
writelp lpmodel.txt
$Offecho
```

Solve-Statement: Ausführung
$Onecho: Kompilierung

Ø Während der Ausführung des solve statements steht die während der Kompilierung erzeugte *cplex.opt* Optionsdatei schon zur Verfügung

```
file f1 /f.inc/;
put f1; put "display 'Hallo';" /;
putclose;
$include f.inc;
*** Unable to open include file
```

put: Ausführung
$Include: Kompilierung

Ø Include-Datei liegt zum Zeitpunkt der Kompilierung noch nicht vor
Ø es resultiert ein Fehler

Tipp: Logik in Mengen verpacken

- Komplexe Auswahllogik für Summenindices und Modellgleichungen können oft in selbsterstellte Hilfs-Mengen verpackt werden
  - Vereinfacht Fehlersuche durch Ansicht/Filtern der Hilfsmengen in GDX-Files
  - Verschlankt Notation der Modelle im GAMS Skript

Beispiel: Summenindex mit mehreren Bedingungen

```
Equation TpLst Transportleistung;

TpLst..
 F =E=   SUM( (i,j,h,k)$( ord(i)< ord(j) AND ord(h) <> ord(k) ),
         d(h,k) * ( t(i,j)+t(j,i) ) * x(i,j,h,k)  )  ;
```

Zwei Hilfsmengen E und HH werden getrennt definiert und können einfach überprüft werden

```
Set E(i,j) relevante Verknüpfungen, HH(h,k) ungleiche Hubs;
E(i,j)  = ( ord(i)  < ord(j) );
HH(h,k) = ( ord(h) <> ord(k) );

TpLst..
 F =E=  SUM( (E(i,j),HH(h,k) ),  d(h,k) * ( t(i,j)+t(j,i) ) * x(i,j,h,k)  )  ;
```

Szenarien: Iterativ

```
set i Zeilen /1*4/, j Spalten /A,B,C,D,E/ ;
Parameter a(i,j), b(i), c(j);
Variable x(j) Entscheidungsvariable, z Zielfkt-Var;
Equation obj Zielfunktion, nb(i) Nebenbedingung;
obj.. z =E= sum(j, c(j) * x(j) );
nb(i).. sum(j, a(i,j) * x(j) ) =G= b(i);
model mod1 /all/;
```

Szenarien vorbereiten: Parametervariation

```
***** Szenarien *****
Set S Szenarien / s1*s3 /
Parameter
 c_scen(s,j) Kostenparameter des Szenarios s
 xLevel(s,j) Szenario-Lösung für x (x.L)
;
* Daten bereitstellen (GDX, externe Quelle, Zufallsziehungen...)
* [...]
```

```
* Szenarien mit Loop lösen:
Loop(s,
  c(j) = c_scen(s,j);
  solve mod1 min z using LP;
  xLevel(s,j) = x.l(j);
);
```

```
* Szenarien mit GUSS lösen:
Set gussMapping
         / S.   scenario. ''
           c.   param.    c_scen
           x.   level.    xLevel/;

solve mod1 min z using LP scenario gussMapping;
```

Loop vs GUSS

- GUSS: Gather-Update-Solve-Scatter
  - Wiederholtes Lösen des Modells durch Solver ohne anschließendes sofortiges Konvertieren von Modelstatistik und Lösung in die GAMS Datenstrukturen (Models,Variables, Equations mit ihren Eigenschaften)
  - dadurch deutlich schneller, aber nicht alle Modellstatistiken stehen sofort nach Lösung zur Verfügung
  - Parametervariation (keine Variation von Sets, Struktur des Modells unverändert)
  - voneinander unabhängige Szenarien
  - spezielle Syntax muss eingehalten werden
- Loop():
  - Wiederholtes Lösen, allerdings wird Modell mit jedem solve-Statement von GAMS für den Solver vorbereitet, dann gelöst und zurück in die GAMS Datenstrukturen geschrieben – dadurch deutlich langsamere Ausführung
  - flexiblere Modellmanipulation zwischen den Iterationen z.B. durch Veränderung von Domain-Mengen einzelner Equations oder Spalten
  - Mehr Modellstatistiken verfügbar

GUSS Syntax

```
* Szenarien mit GUSS lösen:
Set gussMapping
         / S.   scenario. ''
           c.   param.    c_scen
           x.   level.    xLevel/;

solve mod1 min z using LP scenario gussMapping;
```

- 3-dimensionales mapping set
- Erster Eintrag definiert Menge der Szenarien, hier S, dritte Dimension ist leer

Restliche Einträge:

- Dimension 1: Modell-Symbole (hier z.b. Parameter c, Variable x)
- Dimension 2: Schlüsselwörter, möglich:
  - *param*: Input-Mapping eines Parameters
  - *lower / upper / fixed*: Input-Mapping von Schranken (oder Fixierungen) einer Variable
  - *level*: Output-Mapping von Szenario-Ergebnissen
  - *marginal*: Output-Mapping von Szenario-Dualvariablen
- Dimension 3: Szenario-Symbole, hier z.B. c_scen (Input des Szenarios), xLevel (Output des Szenarios)

GUSS Optionen und Modell-Output

```
Set h solution headers / System.GUSSModelAttributes /;
Parameter
 go / SkipBaseCase 1, UpdateType 0, Optfile 1 /
 stats_s(s,h) Loesungsstatistiken;
;
Set gussMapping
          / S.   scenario. ''
            go.  opt.      stats_s
            c.   param.    c_scen
            x.   level.    xLevel/;
```

- Hilfsparameter zum Festlegen von GUSS-Optionen, hier
  - es werden nur Szenariowerte verwendet, nicht die Ursprungswerte des zu variierenden Paramters c im Modell (*SkipBaseCase 1*)
  - die Parameterwerte werden komplett überschrieben (*UpdateType* 0, default)
  - es wird eine Optionsdatei (z.B. cplex.opt) genutzt
- Optionsparameter durch Schlüsselwort *.opt* in Mapping-Set aufnehmen
- Modell/Lösungs-Statistiken die von GUSS unterstützt werden wie etwa Lösungszeit, Anzahl B&B-Knoten etc, können in den Output-Parameter des *.opt*-Mappings geschrieben werden

Put_Utility

Ø Für Szenarien/Iterative Algorithmen: Erstellen separater Outputdateien mit put_utility

```
Sets sc scenarios /sc1*sc5/
     vn variations /VNx,VNy,VNz/;
file myput;
put  myput;
*Iteratives Lösen unterschiedlicher Szenarien:
LOOP(sc,
 LOOP(vn,
* << Iteration, solve oder ähnliches  >> *
*Ausgabe der Ergebnisse dieser Iteration in ein separates GDX:
  put_utility 'gdxout' / 'study2017_' sc.tl:0, '_', vn.tl:0;
  execute_unload x y;
 );
);
putclose;
```

Ø Put_utility erlaubt Zusammensetzen von strings z.b. aus Set-Elementen
Ø Allgemeine Syntax: **Put_utility** 'feature' / 'arguments' ;
Ø Mit GDXOUT nutzen nachfolgende **execute_unload** Befehle den in arguments eingesetzten string als Dateiname +++ weitere features vorhanden, bspw. für Input: GDXIN

Prioritäten für ganzzahlige Variablen

- Reihenfolge für das Branching von MIP Variablen bestimmen: Attribut .prior *(Standard: automatische Bestimmung durch Solver)*
- Gezieltes Relaxieren einzelner Variablen, z.B. für Fix-And-Optimize Strategien
- Dazu: .prior Wert auf +inf setzen

```
Option optcr=0;
set i Zeilen /1*4/, j Spalten /A,B,C,D,E/ ;
Parameter a(i,j), b(i), c(j);
Variable z Zielfkt-Var;
integer variable  x(j) Entscheidungsvariable;
Equation obj Zielfunktion, nb(i) Nebenbedingung;
obj.. z =E= sum(j, c(j) * x(j) );
nb(i).. sum(j, a(i,j) * x(j) ) =G= b(i);
model mod1 /all/;
a(i,j) = uniformint(0,10);
b(i) = uniform(0,10) * 100;
c(j) = uniformint(1,10);
solve mod1 min z using mip;
display z.l,x.l;
x.prior('A') = +inf;
solve mod1 min z using mip;
display z.l,x.l;
```

```
--- Restarting execution
--- relaxample.gms(14) 2 Mb
--- Reading solution for model mod1
--- Executing after solve: elapsed 0:00:00.247
--- relaxample.gms(16) 3 Mb
--- Generating MIP model mod1
--- relaxample.gms(17) 3 Mb
---   5 rows  6 columns  25 non-zeroes
---   4 discrete-columns
***   1 relaxed-columns WARNING
***
***   5 Integer +INF Bounds have been reset to 100 (see Option IntVarUp)
***
--- Executing CPLEX: elapsed 0:00:00.261

IBM ILOG CPLEX   24.9.1 r63795 Released Aug 30, 2017 WEI x86 64bit/MS Windows
Cplex 12.7.1.0
```

```
----     18 VARIABLE z.L                  =      205.516  Zielfkt-Var

----     18 VARIABLE x.L  Entscheidungsvariable

A  3.586,    C 59.000,    D  1.000
```

Erweiterbarkeit: externe Funktionen

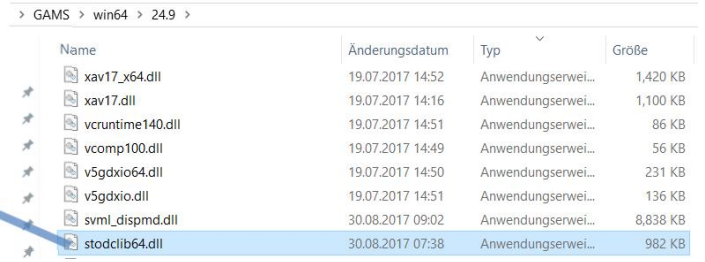
- Externe Funktionsbibliotheken können in GAMS eingebunden werden, Syntax:

```
$FuncLibIn <InternalLibName> <ExternalLibName>
Function <InternalFuncName> /<InternalLibName>.<FuncName>/;
```

- E*xternalLibName* ist die Programmbibliothek (.dll), ggf. mit Pfadangabe, InternalLibName ist für die Verwendung im GAMS Skript
- Standardmäßig wird im GAMS Programmordner nach der .dll gesucht, andernfalls ist der gesamte absolute Pfad zur Bibliothek einzusetzen
- So deklarierte *Functions* werden im Skript analog zu den bereits eingebauten GAMS-Funktionen (ord,card etc) genutzt

Externe Funktionen: Beispiel

```
$funclibin stolib stodclib
function icdfnorm /stolib.icdfnormal/;

set r /1*100/;
parameter q, e(r);
loop(r,
 q = 1/(card(r)+1) * ord(r);
 e(r) = icdfnorm(q,0,4);
);
```



- Es wird die Bibliothek *stodclib* (Stochastic Library) eingebunden, die bereits im GAMS Programmordner vorliegt und GAMS um diverse stochastische Funktionen erweitert
- Es wird die Funktion *icdfnormal* der Bibliothek für das GAMS-Skript verfügbar gemacht
- Verwendung im Skript mit dem internen Namen *icdfnorm*
- Weitere bereitgestellte Bibliotheken
  - pwpcclib (Piecewise Polynomial Library)
  - fitfclib (Fitpack Library)
  - lsadclib (LINDO Sampling Library)