GAMS: General Algebraic Modeling System

- GAMS: Modeling Language and its implementation
- Goal: concise specification of Math Programming models
  - Quick implementation of models
  - Maintainable models
  - Use of state-of-the-art solvers (Cplex, ...)
  - Support for large scale models
  - Support for linear and nonlinear models
History

- Developed at World Bank to achieve
  - Self documenting models
  - Quick turnaround when model changes
  - Maintainability
  - Solver independence
  - Support for nonlinear models
  - Automatic derivatives for NLP’s
  - Initial versions developed in 1978-1979
**GAMS: The Modelling Language**

- **Sets** are used for indexing.
- **Decision variables**:
  - Variables
    - \( x(i,j) \): shipment quantities in cases
    - \( z \): total transportation costs in thousands of dollars
- **Parameters** that don’t change inside a solve:
  - **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
    - seattle new-york 2.5 1.7
    - seattle topeka 2.5 1.8
    - san-diego new-york 2.5 1.8
    - san-diego topeka 2.5 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
    - \( c(i,j) = f \times d(i,j) / 1000 \);
- **Equations** are declared and then defined:
  - **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), \( c(i,j) \times x(i,j) \));
    - supply(i) ..
      - \( \) sum(j, \( x(i,j) \)) = l = \( a(i) \);
    - demand(j) ..
      - \( \) sum(i, \( x(i,j) \)) = g = \( b(j) \);
  - **Model** transport /all/;
  - **Solve** calls external optimizer:
    - transport using lp minimizing \( z \);
    - Display \( x.l \), \( x.m \);
Set Declarations

- **Set elements are strings**
- Even if declared as
  - Set i /1*10/;
  - Set i /1,2,3,4,5,6,7,8,9,10/;
- Sets can have explanatory text:
  - Set y ‘years’ /year2000*year2010/;
- To get sequence number use ord()
  - P(i) = ord(i);
- Parameters, equations are expressed in terms of sets.
Set element names

- If contain blanks then need to be quoted

Set jx 'for use with X/XB variable' / Imports "Food,Seed & Industrial" Production 'Paid Diversion' /;

Explanatory text: these quotes are not needed if we had no / in the text

Double quotes

A valid set element can not contain both ‘ and “

Single quotes. This can be important if the string already contains a single or double quote.
Often the same set is used in different index positions. E.g.

- Parameter p(i,i);
- p(i,i) = 1;  // assigns only diagonal

Use Alias:

- Alias(i,j);
- Parameter p(i,j); // in declaration same as p(i,i)
- p(i,j) = 1;  // assigns all i × j
Sub sets

• Subset:
  • Set j(i)
  • Hierarchy: start with supersets, then define subsets
  • You can have a subset of a subset
  • GAMS will check if elements are in superset (domain checking)

```
1 sets
2     i0     /a,b,c,d/ 
3     i1(i0) /a,b,c/  
4     i2(i1) /b,c,d/  
5 ****                $170
6 **** 170  Domain violation for element
7 ;
```
Multi-dimensional Sets

- Specification of multi-dimensional sets

```plaintext
sets
i  /a,b,c,d/
j  /1,2,3/
k(i,j) /
    a.1
    b.(2,3)
    (c,d).(1,3)
/
;

display k;

----  12 SET k

    1     2     3

a  YES
b                        YES
YES
YES
YES

This is also domain checked

Multidimensional sets can not be used as domain.
```
Dynamic Sets

• Calculate sets dynamically.
• A.k.a. assigned sets
• Dynamic sets can not be used as domains.

set i /i1*i5/;
alias(i,j);
offdiag(i,j) = yes;
offdiag(i,i) = no;
display offdiag;

----

8 SET offdiag exclude diagonal

<table>
<thead>
<tr>
<th></th>
<th>i1</th>
<th>i2</th>
<th>i3</th>
<th>i4</th>
<th>i5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>i2</td>
<td>YES</td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>i3</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>i4</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>i5</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

• Can be entered as
  • Scalar s ‘scalar parameter’ / 3.14/;
  • Parameter p(i) ‘one dimensional parameter’ / 
    i1  2.5
    i2  4.8
  /;
  • Table t(i,j) ‘tabular specification of data’
    j1    j2    j3
    i1   12    14
    i2     8.5
  ;
  • Assignment
    p("i2") = 4.8;
    t(i,j) = p(i) + 3;
The famous $ operator

• ‘Such that’ operator
• Used very often in GAMS models
  – Assignment of parameters
    – \( P(i,j)\$(q(i,j)>0) = q(i,j); \)
    – \( P(i,j) = q(i,j)$$(q(i,j)>0); \)
    – Note: these are different
  – Assignment of sets
  – Sum, prod, smax, smin, loop etc
    – \( S = \text{Sum}((i,j)$$(q(i,j)>0),q(i,j)); \)
  – In equation definitions (discussed later...)

• Assignment of sets
• Sum, prod, smax, smin, loop etc
set i /i1,i2/;
alias(i,j);

parameter p(i,j);

parameter q(i,j);
q(i,j) = -2;
q(i,i) = 2;
p(i,j) = 1;
P(i,j)$(q(i,j)>0) = q(i,j);
display p;

p(i,j) = 1;
P(i,j) = q(i,j)$q(i,j)>0);
display p;
Parallel Assignment

• Parallel assignment:
  – P(i,j) = xxx;
  – No loop needed

• With loop

  \[
  \text{Loop((i,j),}
  \begin{align*}
  &p(i,j)=xxx; \\
  &); \\
  \end{align*}
  \]

• Sometimes beginners use loops too much
Sparse storage

• Only nonzero elements are stored
  – Zero and ‘do not exist’ is identical in GAMS

```gams
set i/ i1,i2/;
alias (i,j);

table t(i,j)
  i1  i2
  i1   1
  i2   3
;

scalar n1,n2;
n1 = card(t);
n2 = sum((i,j)$t(i,j),1);
display n1,n2;
```
Domain Checking

• Makes models more reliable
• Like strict type checking in a programming language

```plaintext
1  set
2     i /a,b,c/
3     j /d,e,f/
4  ;

5

6  parameter p(i);
7   p(i) = 1;
8   p(j) = 2;

****     $171
**** 171  Domain violation for set
9   p('g') = 3;
****     $170
**** 170  Domain violation for element
```
Bypassing domain checking

- Use * as set to prevent domain checking
  - Parameter p(*);

- This is not often needed, sometimes useful to save a few key-strokes.

```
  table unitdata(i,*)
    capacity minoutput mindown minup inistate coefa coefb coefc chot ccold tcool
    *         MW     MW     H     H     H    $/h    $/MWh  $/MW^2h  $/h   $/h   h
  unit1  455  150   8   8   8     1000 16.19 0.00048 4500  9000   5
  unit2  455  150   8   8   8      970 17.26 0.00031 5000 10000   5
  unit3  130  20    5   5   5     700 16.60 0.00200  700  1100   4
  unit4  130  20    5   5   5      680 16.50 0.00211  560  1120   4
  unit5  162  25    6   6   6     450 19.70 0.00398  900  1800   4
  unit6  80   20    3   3   3      370 22.26 0.00712  170   340   2
  unit7  85   25    3   3   3      480 27.74 0.00079  260  520   2
  unit8  55   10    1   1   1     660 25.92 0.00413   30   60   0
  unit9  55   10    1   1   1     665 27.27 0.00222   30   60   0
  unit10 55   10    1   1   1     670 27.79 0.00173   30   60   0
  ;
```
Data Manipulation

• Operate on parameters
• Often large part of the complete model
• Operations:
  – Sum, prod, smax, smin,
  – Functions: sin, cos, max, min, sqr, sqrt etc
  – $ conditions
  – If, loop
  – For, while (not used much)
Checks

- Abort allows to add checks:

```plaintext
scalars total_demand, total_capacity;
total_demand = sum(j, b(j));
total_capacity = sum(i, a(i));
display total_demand, total_capacity;

abort$(total_demand > total_capacity + 0.001) "Capacity too small to meet demand";
```

* *
* check for balanced demand and supply
* *
scalar totalsupply, totaldemand;
totalsupply = sum(i, s(i));
totaldemand = sum(j, d(j));
abort$(abs(totalsupply - totaldemand) > 0.01) "Unbalanced supply and demand";
```
• Declaration:
  – Free variable \(x(i)\); // default!
  – Positive variable \(y(i,j)\); // this means non-negative
  – Binary variable \(z\);
  – Integer variable \(d\);
  – Can be declared in steps, as long as no contradiction:
    • Variable \(x,y,z\); Positive Variable \(x(i)\);

• For MIP/MINLP models extra variable types:
  – Sos1, sos2, semicont, semiint

• Free variable is the default. Most other systems have positive variables as the default.
Variables (2)

- `x.lo=1`; sets lower bound
- `Y.up(i)=100`; sets upper bound
- `Z.L` is level
- `X.M` is marginal (reduced cost, dual)
- `Z.Scale` sets scale for NLP
- `Z.prior` sets priorities for MIP
- `X.fx=1` is shorthand for `x.lo=1;x.up=1;x.L=1`; (cannot by used in rhs)
Equations

• Declaration:
  – Equation e(i) ‘some equation’;

• Definition:
  – e(i).\text{.. sum}(j, x(i,j)) \text{=} 1;

• This generates card(i) equations

• $ conditions:
  – e(i)\text{$subset}(i)\text{.. sum}(j, x(i,j)) \text{=} 1;

• Equation types
  • =E=, =L=, =G=
  • =X= (external functions)
  • =N= (nonbinding, not used much)
  • =C= (conic equation, not used much)
Maps

\[ \text{distance}(i,j):(\text{lt}(i,j)) \quad \Rightarrow \quad d = \sqrt{\text{sqr}(x(i) - x(j)) + \text{sqr}(y(i) - y(j))} \]

identical to

\[ \text{distance}(\text{lt}(i,j)) \quad \Rightarrow \quad d = \sqrt{\text{sqr}(x(i) - x(j)) + \text{sqr}(y(i) - y(j))} \]

A map is a filter

In the rhs both \( i,j \) and \( \text{lt} \) can be used:

\[ \text{distance}(\text{lt}(i,j)) \quad \Rightarrow \quad d(\text{lt}) = \sqrt{\text{sqr}(x(i) - x(j)) + \text{sqr}(y(i) - y(j))} \]
Parameter vs variable

• Nonlinear
  
  Variable y;
  e.. x =e= sqr(y);

• Linear
  
  Parameter p;
  e.. x =e= sqr(p);

  Variable y;
  e.. x =e= sqr(y.L);
Special Values

- **INF**
  - Infinity: often used for bounds
- **-INF**
  - Minus infinity: mostly for bounds
- **NA**
  - Not available: not much used
- **EPS**
  - Numerically zero
  - Marginal is zero but nonbasic → EPS
- **UNDF**
  - Eg result if division by zero

```plaintext
1  parameter x,y;
2  x=0;
3  y=1/x;
4  display y;

**** Exec Error at line 3: division by zero (0)

----
4  PARAMETER y = UNDF
```
Model statement

- Model m /all/;
- Model m /cost, supply, demand/;
- Special syntax for MCP models to indicate complementarity pairs:
  - Model m /demand.Qd, Psupply.Qs, Equilibrium.P/
Solve m minimizing z using lp;

- GAMS uses objective variable instead of objective function

- Model types
  - LP: linear programming
  - NLP: nonlinear programming
  - DNLP: NLP with discontinuities (max, min, abs)
  - MIP: linear mixed integer, RMIP: relaxed MIP
  - MINLP: nlp with integer vars, RMINP: relaxed minlp
  - QCP, MIQCP: quadratically constrained
  - CNS: constrained non-linear system (square)
  - MCP: mixed complementarity
  - MPEQ: NLP with complementarity conditions
GAMS Flow of Control
To select solver

- Option lp=cplex;
- Command line parameter: lp=cplex
- Change defaults (IDE or GAMSINST)

Switching solvers is easy and cheap
<table>
<thead>
<tr>
<th>Solver/Model type availability - 22.7 May 1, 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>ALPHAECP</td>
</tr>
<tr>
<td>BARON 8.1</td>
</tr>
<tr>
<td>BDMLP</td>
</tr>
<tr>
<td>COIN</td>
</tr>
<tr>
<td>CONOPT 3</td>
</tr>
<tr>
<td>CPLEX 11.0</td>
</tr>
<tr>
<td>DECIS</td>
</tr>
<tr>
<td>DICOPT</td>
</tr>
<tr>
<td>KNITRO 5.1</td>
</tr>
<tr>
<td>LINDOGLOBAL 5.0</td>
</tr>
<tr>
<td>LGO</td>
</tr>
<tr>
<td>MILES</td>
</tr>
<tr>
<td>MINOS</td>
</tr>
<tr>
<td>MOSEK 5</td>
</tr>
<tr>
<td>MPSGE</td>
</tr>
<tr>
<td>MSNLPl</td>
</tr>
<tr>
<td>NLPEC</td>
</tr>
<tr>
<td>OQNLP</td>
</tr>
<tr>
<td>OSL V3</td>
</tr>
<tr>
<td>OSLSE</td>
</tr>
<tr>
<td>PATH</td>
</tr>
<tr>
<td>SBB</td>
</tr>
<tr>
<td>SNOPT</td>
</tr>
<tr>
<td>XA</td>
</tr>
<tr>
<td>XPRESS 18.00</td>
</tr>
</tbody>
</table>

**Contributed Plug & Play solvers**

| AMPLwrap | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| DEA | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Kestrel | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
Linear Programming

• Very large models can be solved reliably
• Primal and Dual Simplex and interior point (barrier) methods.
  – Free solvers:
    • BDMLP
    • COINGLPK
    • COINCBC
  – CPLEX (Ilog)
    • commercial, parallel, state-of-the-art, simplex+barrier
  – XPRESS (Fair Isaac)
    • commercial, parallel, state-of-the-art, simplex+barrier
  – MOSEK
    • Very good parallel interior point
  – XA
    • cheaper alternative
Many additional algorithms determine success
- Scaling
- Presolver (reduce size of model)
- Crash (find good initial basis)
- Crossover (interior point solution → basic solution)

Very large models (> 10 million nonzero elements) require much memory

64 bit architecture can help then (available on pc’s, so no need for super computers like this Cray C90)
Performance improvement

- Indus89 model ran for 6-7 hours on a DEC MicroVax in 1990 using MINOS as LP solver
- This model runs now with Cplex on a laptop well within 1 second
LP Modeling

• Almost anything you throw at a good LP solver will solve without a problem
• If presolver reduces the model a lot or if you have many $x.fx(i)=0$ then revisit equations and exclude unwanted variables using $\$$ conditions.
• Don’t reduce \#vars,\#equs if this increases the number of nonzero elements significantly

\[ e(k).. \quad x(k) = \text{L} = \text{sum}(j, y(j)) \]

\[ e(k).. \quad x(k) = \text{L} = ysum; \]
\[ \text{Ydef.} \quad ysum = \text{e} = \text{sum}(j, y(j)); \]

<table>
<thead>
<tr>
<th>K equations</th>
<th>K+1 equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>K+J variables</td>
<td>K+J+1 variables</td>
</tr>
<tr>
<td>K\times(J+1) nonzeroes</td>
<td>2K+J+1 nonzeroes</td>
</tr>
</tbody>
</table>

e.g.
100 equations
200 variables
10100 nonzeroes
e.g.
101 equations
201 variables
301 nonzeroes
Part 1: echo listing of the model. Occasionally useful to look at syntax errors or run time errors.

The compilation time is usually small
Part 2: equation listing
- Shows first 3 equations for each block
- INFES is for initial point, so don’t worry
- Note how explanatory text is carried along
- Especially useful for difficult equations with leads and lags
- More or less can be shown with OPTION LIMROW=nnn;

---- demand =G= satisfy demand at market j

demand(new-york) .. x(seattle,new-york) + x(san-diego,new-york) =G= 325 ; (LHS = 0, INFES = 325 ****)
demand(chicago) .. x(seattle,chicago) + x(san-diego,chicago) =G= 300 ; (LHS = 0, INFES = 300 ****)
demand(topeka) .. x(seattle,topeka) + x(san-diego,topeka) =G= 275 ; (LHS = 0, INFES = 275 ****)

This was generated by: demand(j) .. sum(i, x(i,j)) =g= b(j) ;
Part 3: Column Listing

- Shows variables appearing in the model and where
- First 3 per block are shown
- Can be changed with `OPTION LIMCOL=nnn;`
- By definition feasible (GAMS will project levels back on their bounds)

---

\[
x(\text{seattle,new-york})
\begin{align*}
& (.LO, .L, .UP, .M = 0, 0, +\infty, 0) \\
& -0.225 \text{ cost} \\
& 1 \text{ supply(\text{seattle})} \\
& 1 \text{ demand(\text{new-york})}
\end{align*}
\]

\[
x(\text{seattle,chicago})
\begin{align*}
& (.LO, .L, .UP, .M = 0, 0, +\infty, 0) \\
& -0.153 \text{ cost} \\
& 1 \text{ supply(\text{seattle})} \\
& 1 \text{ demand(\text{chicago})}
\end{align*}
\]

\[
x(\text{seattle,topeka})
\begin{align*}
& (.LO, .L, .UP, .M = 0, 0, +\infty, 0) \\
& -0.162 \text{ cost} \\
& 1 \text{ supply(\text{seattle})} \\
& 1 \text{ demand(\text{topeka})}
\end{align*}
\]

REMAINING 3 ENTRIES SKIPPED
Part 4

- Model statistics
- Model generation time: time spent in SOLVE statement generating the model
- Execution time: time spent in GAMS executing all statements up to the point where we call the solver

MODEL STATISTICS

<table>
<thead>
<tr>
<th>Blocks of Equations</th>
<th>Blocks of Variables</th>
<th>Single Equations</th>
<th>Single Variables</th>
<th>Non Zero Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>19</td>
</tr>
</tbody>
</table>

Generation Time = 0.000 Seconds
Execution Time = 0.000 Seconds
LP Listing File (5)

- Solve info
  - Search for ‘S O L’
  - Solver/model status can also be interrogated programmatically
  - Resource usage, limit means time used, limit

<table>
<thead>
<tr>
<th>SOLVE SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
</tr>
<tr>
<td>TYPE</td>
</tr>
<tr>
<td>SOLVER</td>
</tr>
<tr>
<td>OBJECTIVE</td>
</tr>
<tr>
<td>DIRECTION</td>
</tr>
<tr>
<td>FROM LINE</td>
</tr>
<tr>
<td>**** SOLVER STATUS</td>
</tr>
<tr>
<td>**** MODEL STATUS</td>
</tr>
<tr>
<td>**** OBJECTIVE VALUE</td>
</tr>
<tr>
<td>RESOURCE USAGE, LIMIT</td>
</tr>
<tr>
<td>ITERATION COUNT, LIMIT</td>
</tr>
<tr>
<td>MODEL STATUS CODE</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
</tbody>
</table>
Model/Solver Status (2)

abort$(m.solvestat <> 1) 'bad solvestat';

model m /all/;
option nlp=conopt2;
option mip=cplex;
option rminlp=conopt2;
option minlp=dicopt;
*
* solve relaxed model
*
solve m using rminlp minimizing z;
abort$(m.modelstat > 2.5) "Relaxed model could not be solved";
*
* solve minlp model
*
solve m using minlp minimizing z;
Part 6: messages from solver

ILOG CPLEX       BETA 1Apr 22.7.0 WEX 3927.4246 WEI x86_64/MS Windows
Cplex 11.0.1, GAMS Link 34

Optimal solution found.
Objective : 153.675000

More information can be requested by OPTION SYSOUT=on;

Note: this part is especially important if something goes wrong with the solve. In some cases you also need to inspect the log file (some solvers don’t echo all important messages to the listing file).
• Part 7: Solution listing
  – Can be suppressed with m.solprint=0;

<table>
<thead>
<tr>
<th></th>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-york</td>
<td>325.0000</td>
<td>325.0000</td>
<td>+INF</td>
<td>0.2250</td>
</tr>
<tr>
<td>chicago</td>
<td>300.0000</td>
<td>300.0000</td>
<td>+INF</td>
<td>0.1530</td>
</tr>
<tr>
<td>topeka</td>
<td>275.0000</td>
<td>275.0000</td>
<td>+INF</td>
<td>0.1260</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LOWER</th>
<th>LEVEL</th>
<th>UPPER</th>
<th>MARGINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle .new-york</td>
<td>.</td>
<td>50.0000</td>
<td>+INF</td>
<td>.</td>
</tr>
<tr>
<td>seattle .chicago</td>
<td>.</td>
<td>300.0000</td>
<td>+INF</td>
<td>.</td>
</tr>
<tr>
<td>seattle .topeka</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
<td>0.0360</td>
</tr>
<tr>
<td>san-diego.new-york</td>
<td>.</td>
<td>275.0000</td>
<td>+INF</td>
<td>.</td>
</tr>
<tr>
<td>san-diego.chicago</td>
<td>.</td>
<td>.</td>
<td>+INF</td>
<td>0.0090</td>
</tr>
<tr>
<td>san-diego.topeka</td>
<td>.</td>
<td>275.0000</td>
<td>+INF</td>
<td>.</td>
</tr>
</tbody>
</table>
• Write file solver.opt
• Tell solver to use it: m.optfile=1;
• Option file can be written from GAMS

$onecho > cplex.opt
lpmethod 4
$offecho

Model m/all/;
m.optfile=1;
Solve m minimizing z using lp;

--- Executing CPLEX: elapsed 0:00:00.007
ILOG CPLEX May 1, 2008 22.7.1 WIN 3927.4700 VIS x86/MS Windows Cplex 11.0.1, GAMS Link 34
Reading parameter(s) from "C:\projects\test\cplex.opt"
>> lpmethod 4
Finished reading from "C:\projects\test\cplex.opt"
Integer Programming

- Combinatorial in nature
- Much progress in solving large models
- Modeling requires
  - Skill
  - Running many different formulations: this is where modeling systems shine
  - Luck
- Often need to implement heuristics
MIP Solvers

• Free solvers:
  – Bdmlp, coinglpk, coincbc, coinscip

• Commercial solvers:
  – Cplex, Xpress (market leaders)
  – XA, Mosek
MIP Modeling

• Difficult, not much automated
• Many MINLPs can be linearized into MIPs.
• Eg

\[ z = x \cdot y, \quad x, y \in \{0,1\} \]

can be formulated as:

\[ z \leq x \]
\[ z \leq y \]
\[ z \geq x + y - 1 \]

\( x, y \in \{0,1\}, z \in [0,1] \)
Nonlinear Programming

- Large scale, sparse, local solvers:
  - Conopt (ARKI)
    - Reliable SQP, 2nd derivatives
    - Scaling, presolve, good diagnostics
    - Often works without options
  - Minos (Stanford)
    - Older augmented Lagrangian code
    - Good for models that are mildly nonlinear
  - Snopt (Stanford, UCSD)
    - SQP based code
    - Inherits much from Minos but different algorithm
  - Knitro (Ziena)
    - Interior point NLP
    - Sometimes this works very well on large problems
  - CoinIpOpt (IBM, CoinOR, CMU)
    - Free, interior point
Special Nonlinear Programming

- PathNLP
  - Reformulate to MCP
- BARON
  - Global solver
  - Only for small models
- Other global solvers:
  - LGO, OQNLP, Lindoglobal
- Mosek
  - For convex NLP and QCP only
- Cplex
  - For QCP
MINLP Solvers

- Free Solvers
  - CoinBonmin
- Dicopt
- SBB
- AlphaEcp
- Baron, lgo, oqnlp (global)
NLP Modeling

• Models fail mostly because of:
  – Poor starting point
    • Specify X.L(i)=xx; for all important nonlinear variables
  – Poor scaling
    • You can manually scale model use x.scale, eq.scale
  – Poorly chosen bounds
    • Choose x.lo,x.up so that functions can be evaluated

• Note: changing bounds can change initial point
NLP Modeling

• Minimize nonlinearity
• Measure
  – --- 429 nl-code 30 nl-non-zeros
• Example:

\[ e1.. Z = \log[\text{sum}(i, x(i))] \]

\[ e1.. z = \log(y); \]
\[ e2.. y = \text{sum}(i, x(i)); \]

Additional advantage:
We can protect log by \[ y.lo=0.001; \]
<table>
<thead>
<tr>
<th>Function</th>
<th>Allowed In equations</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>DNLP</td>
<td>Non-differentiable, use alternative: variable splitting</td>
</tr>
<tr>
<td>execSeed</td>
<td>no</td>
<td>Seed for random number generation. Can also be set.</td>
</tr>
<tr>
<td>Exp, log, log2, log10</td>
<td>NLP</td>
<td>Add lowerbound for log</td>
</tr>
<tr>
<td>Ifthen(cond, x, y)</td>
<td>DNLP</td>
<td>Non-differentiable, use binary variables</td>
</tr>
<tr>
<td>Min(x, y), max(x, y, z), smin(i,..), smax(i,..)</td>
<td>DNLP</td>
<td>Non-differentiable, use alternative formulation</td>
</tr>
<tr>
<td>Prod</td>
<td>NLP</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>LP/NLP</td>
<td></td>
</tr>
<tr>
<td>Round, trunc, fract</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Sqr, sqrt, power</td>
<td>Yes</td>
<td>Protect sqrt with lowerbound</td>
</tr>
</tbody>
</table>
| Power(x, y), x**y | NLP                  | Power: integer y  
x**y = exp(y*log(x)), add x.lo=0.001;              |
<p>| Cos, sin, tan, arccos, arcsin, arctan, arctan2, cosh, sinh, tanh, | NLP                  |                                                                     |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Allowed In equations</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact</td>
<td>no</td>
<td>In equations use gamma</td>
</tr>
<tr>
<td>Gamma, Beta, BetaReg, Gamma Reg, LogGamma, LogBeta</td>
<td>DNLP</td>
<td></td>
</tr>
<tr>
<td>Binomial(x,y)</td>
<td>NLP</td>
<td>Generalized binomial function</td>
</tr>
<tr>
<td>Errorf</td>
<td>NLP</td>
<td>Error function. Inverse not available: use equation: $z = e^{- \text{erf}(x)}$ to find $x$.</td>
</tr>
<tr>
<td>Mod</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Normal, uniform, uniformint</td>
<td>No</td>
<td>Random number generation</td>
</tr>
<tr>
<td>Pi</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Edist, entropy, ncpf, ncpcm, poly</td>
<td>Yes</td>
<td>Not often used</td>
</tr>
<tr>
<td>Calendar functions</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>
Command Line Version

1. Edit .gms file
2. Run GAMS
3. View .lst
4. Go back to 1.
IDE
• Syntax coloring can help detect syntax errors very early.

• Block commands are often useful
• F8 to find matching parenthesis
• Search in files
• The project file determines where files (.gms, .lst, .log) are located.

• Start new model by creating new project file in new directory
- After hitting Run Button (or F9), process window shows errors.
- Clicking red line brings you to location in .gms file.
- Clicking back line bring you to location in .lst file.
- This is only needed for obscure errors.
Lst File Window

- Use tree to navigate
- Search for ‘S O L’ to find ‘S O L V E    S U M M A R Y’
Debug Models

- Use DISPLAY statements
- Use GDX=xxx on command line
- Then click on blue line
GDX Viewer

Blank means same as above

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>ku</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>m</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>mp</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>np</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>unit</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>xinit</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>a</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>b</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>lambda</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>stateq</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>V</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>utilde</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>wk</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>x</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>xtilde</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>w</td>
<td>3</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u: control variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane Index (empty)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>u</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>utilde</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>x</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gov-expend</th>
<th>Level</th>
<th>Marginal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1964-i</td>
<td>113.950545458027</td>
<td>-4.8989786116671E-9</td>
</tr>
<tr>
<td>1964-ii</td>
<td>115.263041877291</td>
<td>-5.19740461779605E-9</td>
</tr>
<tr>
<td>1964-iii</td>
<td>116.614838161443</td>
<td>-2.97532565252112E-9</td>
</tr>
<tr>
<td>1964-iv</td>
<td>118.02902087123</td>
<td>3.269659876181666E-9</td>
</tr>
<tr>
<td>1965-i</td>
<td>119.522189413067</td>
<td>1.06708721858695E-8</td>
</tr>
<tr>
<td>1965-ii</td>
<td>121.11805667791</td>
<td>1.22211514241855E-8</td>
</tr>
<tr>
<td>1965-iii</td>
<td>122.924386749315</td>
<td>2.6000309630275E-9</td>
</tr>
<tr>
<td>1965-iv</td>
<td>110.5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>u</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>utilde</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>x</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>xtide</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>w</td>
<td>Par</td>
<td>3</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol search</th>
<th>Next</th>
<th>Prev</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>u</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>utilde</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>x</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>xtide</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>w</td>
<td>Par</td>
<td>3</td>
<td>16</td>
</tr>
</tbody>
</table>

Reset | Sort | Decimals | Search | Ordering: 1 2 3
|---|---|---|---|---|

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>u</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>utilde</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>x</td>
<td>Var</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>xtide</td>
<td>Par</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2001</td>
<td>w</td>
<td>Par</td>
<td>3</td>
<td>16</td>
</tr>
</tbody>
</table>
Index positions can be placed:
1. On the plane
2. On the left (row header)
3. On the top (column header)
Generating GDX files

• From command line (gdx=xxx)
• $gdxout (not used much)
• Execute_unload ‘xxx.gdx’,a,b,x;
• Or via some external tool:
  – Gdxxrw can create a gdx file from an Excel spreadsheet
  – Mdb2gms can create a gdx file from an Access database
  – Sql2gms can create a gdx file from any sql database
Reading GDX file

• \$gdxin
  Set i;
  Parameter p(i);

  \$gdxin a.gdx
  \$load i
  \$load p

  Display i,p;

• Execute_load

  set i /i1*i3/;
  alias (i,j);

  table a(i,j) 'original matrix'
     i1  i2  i3
     i1  1   2   3
     i2  1   3   4
     i3  1   4   3
  ;

  parameter inva(i,j) 'inverse of a';

  execute_unload 'a.gdx',i,a;
  execute '=invert.exe a.gdx i a b.gdx inva';
  execute_load 'b.gdx',inva;

  display a,inva;
GDX is hub for external I/O

- Excel
- Csv
- Access
- Etc.

GAMS MODEL

- gdx

- Excel
- Csv
- Etc.
Gdx xxrw: read xls

```
$onecho > x.txt
trace=2
i=transp ort.xls
par=c
rng=A1
rdim=1
cdim=1
$offecho
$call =gdxxrw.exe @x.txt

parameter c(*,*);
.gdxin transp ort.gdx
$load c
$display c;
```

<table>
<thead>
<tr>
<th></th>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>seattle</td>
<td>2.500</td>
<td>1.700</td>
<td>1.800</td>
</tr>
<tr>
<td>san-diego</td>
<td>2.500</td>
<td>1.800</td>
<td>1.400</td>
</tr>
</tbody>
</table>