An Introduction

Frederik Fiand & Tim Johannessen
GAMS Software GmbH
Agenda

GAMS at a Glance

GAMS - Hands On Examples

APIs - Application Programming Interfaces to GAMS

Outlook - Some Advanced GAMS Features
Company

- Roots: World Bank, 1976
- Went commercial in 1987
- Locations
  - GAMS Development Corporation (Washington)
  - GAMS Software GmbH (Germany)
- Product: The General Algebraic Modeling System
What did this give us?

Simplified model development & maintenance

Increased productivity tremendously

Made mathematical optimization available to a broader audience (domain experts)

2012 INFORMS Impact Prize
GENERAL ALGEBRAIC MODELING SYSTEM

Broad User Community and Network

- 11,500+ licenses
- Users: 50% academic, 50% commercial
- GAMS used in more than 120 countries
- Uniform interface to more than 30 solvers

25+ Years GAMS Development
**Broad Range of Application Areas**

<table>
<thead>
<tr>
<th>Agricultural Economics</th>
<th>Applied General Equilibrium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemical Engineering</td>
<td>Economic Development</td>
</tr>
<tr>
<td>Econometrics</td>
<td>Energy</td>
</tr>
<tr>
<td>Environmental Economics</td>
<td>Engineering</td>
</tr>
<tr>
<td>Finance</td>
<td>Forestry</td>
</tr>
<tr>
<td>International Trade</td>
<td>Logistics</td>
</tr>
<tr>
<td>Macro Economics</td>
<td>Military</td>
</tr>
<tr>
<td>Management Science/OR</td>
<td>Mathematics</td>
</tr>
<tr>
<td>Micro Economics</td>
<td>Physics</td>
</tr>
</tbody>
</table>

**25+ Years**

GAMS Development
Foundation of GAMS

- Powerful algebraic modeling language
- Open architecture and interfaces to other systems, independent layers
Powerful Declarative Language

Similar to mathematical notation

Easy to learn - few basic language elements: sets, parameters, variables, equations, models

Model is executable (algebraic) description of the problem
Mix of Declarative and Imperative Elements

Control Flow Statements (e.g. loops, for, if,...), macros and functions

Advantages:
- Build complex problem algorithms within GAMS
- Simplified interaction with other systems:
  - Data exchange
  - GAMS process control
Strong Development Environment

GAMS IDE

- Project management
- Editor / Syntax coloring / Spell checks
- Tree view / Syntax-error navigation
- Model Debugging & Profiling
- Solver selection & setup
- Data viewer
  - Export
  - Charting
- GAMS Processes Control
Independence of Model and Operating System

Platforms supported by GAMS:

Models can be moved between platforms with ease!
Independence of **Model and Solver**

One environment for a wide range of model types and solvers

- All major commercial LP/MIP solver
- Open Source Solver (COIN)
- Also solver for NLP, MINLP, global, and stochastic optimization

Switching between solvers with one line of code!

Model

Platform  Solver  Data  Interface
Independence of Model and Data

- Declarative Modeling
- ASCII: Initial model development

- GDX: Data layer ("contract") between GAMS and applications
  - Platform independent
  - No license required
  - Direct GDX interfaces and general API
  - ...

Model

Platform  Solver  Data  Interface
Independence of **Model and User Interface**

**API’s**

- *Low Level*
- *Object Oriented*: .Net, Java, Python
- No modeling capability: Model is written in GAMS
- Wrapper class that encapsulates a GAMS model
Smart Links to other Applications

- User keeps working in his productive tool environment
- Application accesses all optimization capabilities of GAMS through API
- Visualization and analysis of model data and results in the application
Smart Links to other Applications

- User keeps working in his productive tool environment
- Application accesses all optimization capabilities of GAMS through API
- Visualization and analysis of model data and results in the application

MatLab

Figure 1: US dollar short rate scenarios

Figure 2: Short vs. long rates
Smart Links to other Applications

- User keeps working in his productive tool environment
- Application accesses all optimization capabilities of GAMS through API
- Visualization and analysis of model data and results in the application
Striving for **Innovation and Compatibility**

**Models must benefit from:**
- Advancing hardware / New Platforms
- Enhanced / new solver and solution technology
- Improved / upcoming interfaces to other systems
- New Modeling Concepts

**Protect investments of Users**
- Life time of a model: 15+ years
- New maintainer, platform, solver, user interface
- Backward Compatibility
- Software Quality Assurance
Free Model Libraries

More than 1400 models!
Why GAMS?

- Experience of 25+ years
- Broad user community from different areas
- Lots of model templates
- Strong development interface

- Consistent implementation of design principles
  - Simple, but powerful modeling language
  - Independent layers
  - Open architecture: Designed to interact with other applications

- Open for new developments
- Protecting investments of users
Agenda

GAMS at a Glance

GAMS - Hands On Examples

APIs - Application Programming Interfaces to GAMS

Outlook - Some Advanced GAMS Features
A Simple **Transportation Problem**

What does this example show?

- It gives a first glimpse of how a problem can be formulated in GAMS
- It shows some basics of data exchange with GAMS
- It shows how easy it is to change model type and, consequently, solver technology
Model types in this example

- **LP**
  - Determine minimum transportation cost.
  - Result: city to city shipment volumes.

- **MIP**
  - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**
  - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**
  - Allows uncertainty, e.g. uncertain demand
Model types in this example

- **LP**: Determine minimum transportation cost. Result: city to city shipment volumes.

- **MIP**: Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**: Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**: Allows uncertainty, e.g. uncertain demand
A Simple Transportation Problem

Canning Plants (supply)  shipments  Markets (demand)
(Number of cases)

Seattle (350)
San Diego (600)
Topeka (275)
Chicago (300)
New York (325)

Freight: $90 case / thousand miles
Minimize Transportation cost subject to Demand satisfaction at markets Supply constraints
Indices:  
i (Canning plants)  
j (Markets)  

Decision variables:  
\( x_{ij} \) (Number of cases to ship)  

Data:  
\( c_{ij} \) (Transport cost per case)  
\( a_i \) (Capacity in cases)  
\( b_i \) (Demand in cases)  

\[
\begin{align*}
\text{min} & \quad \sum_i \sum_j c_{ij} \cdot x_{ij} \\
\text{subject to} & \quad \sum_j x_{ij} \leq a_i \quad \forall i \quad \text{(Shipments from each plant \( \leq \) supply capacity)} \\
& \quad \sum_i x_{ij} \geq b_j \quad \forall j \quad \text{(Shipments to each market \( \geq \) demand)} \\
& \quad x_{ij} \geq 0 \quad \forall i, j \quad \text{(Do not ship from market to plant)} \\
& \quad i, j \in \mathbb{N}
\end{align*}
\]
**GAMS Syntax (LP Model)**

Variables
- \( x(i,j) \)  shipment quantities in cases
- \( z \)  total transportation costs in thousands of dollars

Positive Variable \( x \):

Equations
- \( \text{cost} \)  define objective function
- \( \text{supply}(i) \)  observe supply limit at plant \( i \)
- \( \text{demand}(j) \)  satisfy demand at market \( j \):

\[
\text{cost} .. \quad z \; =e= \; \text{sum}((i,j), \; c(i,j) \cdot x(i,j)) ;
\]

\[
\text{supply}(i) .. \quad \text{sum}(j, \; x(i,j)) \; =l= \; a(i) ;
\]

\[
\text{demand}(j) .. \quad \text{sum}(i, \; x(i,j)) \; =g= \; b(j) ;
\]

Model modellLP  /cost, supply, demand/ ;

Solve modellLP using lp minimizing \( z \) ;
GAMS Syntax (LP Model)

Variables
   x(i,j)   shipment quantities in cases
   z       total transportation costs in thousands of dollars;

Positive Variable x:

Equations
cost     define objective function
supply(i) observe supply limit at plant i
demand(j) satisfy demand at market j;

cost ..   z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;

Model modellp /cost, supply, demand/ ;

Solve modellp using lp minimizing z ;
GAMS Syntax (Data Input)

Sets
canning plants / seattle, san-diego /
j markets / new-york, chicago, topeka /

Parameters
a(i) capacity of plant i in cases
   / seattle 350
   san-diego 600 /

b(j) demand at market j in cases
   / new-york 325
   chicago 300
   topeka 275 /

Table d(i,j) distance in thousands of miles

   new-york    chicago    topeka
seattle      2.5        1.7     1.8
san-diego     2.5        1.2     1.4

Scalar f freight in dollars per case

Parameter c(i,j) transport cost in thousand dollars per case

\[ c(i,j) = f \times d(i,j) / 1000 \]

$include data.gms

Variables
x(i,j) shipment quantities in cases
z total transportation costs in thousands

Positive Variable x;

Equations
cost define objective function
supply(i) observe supply limit at plant i
demand(j) satisfy demand at market j;
Sets
d i canning plants
 j markets ;

Parameters
 a(i) capacity of plant i in cases
 b(j) demand at market j in cases
 d(i,j) distance in thousands of miles
 c(i,j) transport cost in thousands of dollars per case :

$call gams data.gms gdx-data.gdx
$if errorlevel 1 $abort Error preparing data
$gdxin data.gdx
$load i j a b d
$gdxin

Scalar f freight in dollars per case per thousand miles /80/ ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
 x(i,j) shipment quantities in cases
 z total transportation costs in thousands of dollars ;

Positive Variable x ;

Equations
cost define objective function
 supply(i) observe supply limit at plant i
 demand(j) satisfy demand at market j ;
GAMS Syntax (Data Input)

Sets
  i  canning plants
  j  markets ;

Parameters
  a(i)  capacity of plant i in cases
  b(j)  demand at market j in cases
  d(i,j) distance in thousands of miles
  c(i,j) transport cost in thousands of dollars per case ;

$onecho > instructions.txt
par=d rng=Sheet1!A1 rdim=1 cdim=1
par-b rng=Sheet1!B6 rdim=0 cdim=1
par=a rng=Sheet1!G2 rdim=1 cdim=0
$offecho
$call gdxrw data.xlsx @instructions.txt
$if errorlevel 1 $abort Error preparing data
$gdxin data.gdx
$load c<i,d.dim1 j<d.dim2 d a b
$gdxin

Scalar f  freight in dollars per case per thousand miles /90/ ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
  x(i,j)  shipment quantities in cases
  z  total transportation costs in thousands of dollars ;
Solution to LP model

Canning Plants (supply) \[\quad\] shipments \[\quad\] Markets (demand)

Seattle (350) \[\quad\] San Diego (600)

Topeka (275) \[\quad\] Chicago (300) \[\quad\] New York (325)

Seattle to Topeka: 300 cases
Seattle to Chicago: 275 cases
San Diego to New York: 50 cases

Freight: $90 case / thousand miles
Total cost: $153,675
Model types in this example

- **LP**
  - Determine minimum transportation cost. Result: city to city shipment volumes.

- **MIP**
  - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**
  - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**
  - Allows uncertainty, e.g. uncertain demand
Model types in this example

- **LP**
  - Determine minimum transportation cost.
  - Result: city to city shipment volumes.

- **MIP**
  - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**
  - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**
  - Allows uncertainty, e.g. uncertain demand
MIP Model: Minimum Shipment of 100 cases

- Shipment volume: $x$ (continuous variable)
- Discrete decision: $\text{ship}$ (binary variable)

add constraints:

- $x_{i,j} \geq 100 \cdot \text{ship}_{i,j}$ $\forall i, j$ (if ship=1, then ship at least 100)
- $x_{i,j} \leq \text{bigM} \cdot \text{ship}_{i,j}$ $\forall i, j$ (if ship=0, then do not ship at all)

$\text{ship}_{i,j} \in \{0,1\}$
MIP Model: GAMS Syntax

demand(j) .. sum(i, x(i,j)) =g= b(j) ;

model transportLP / all /;

Solve transportLP using LP minimizing z ;

parameter rep(i,j,*) report parameter;
rep(i,j,'LP') = x.l(i,j):

MIP
scalar minS minimum shipment / 100 /
bigM big M;
bigM = min(smax(1,a(1)), smax(j,b(j)));

binary variable ship(i,j) '1 if we ship from i to j, otherwise 0';

equation minship(i,j) minimum shipment
maxship(i,j) maximum shipment;

minship(i,j).. x(i,j) =g= minS * ship(i,j);
maxship(i,j).. x(i,j) =l= bigM * ship(i,j);

Model transportMIP / transportLP, minship, maxship / ;
option optcr = 0;

Solve transportMIP using MIP minimizing z ;

rep(i,j,'MIP') = x.l(i,j);
display rep;
MIP Model: GAMS Syntax

demand(j) ..  sum(i, x(i,j)) =g=  b(j) ;

model transportLP / all /;

Solve transportLP using LP minimizing z :

parameter rep(i,j,*) report parameter;
rep(i,j,'LP') = x.l(i,j):

* MIP
scalar minS minimum shipment / 100 /
bigM big M;
bigM = min(smax(1,a(i)), smax(j,b(j)));

binary variable ship(i,j) '1 if we ship from i to j, otherwise 0';

equation minship(i,j) minimum shipment
    maxship(i,j) maximum shipment;

minship(i,j) .. x(i,j) =g=  minS * ship(i,j);
maxship(i,j) .. x(i,j) =l=  bigM * ship(i,j):

Model transportMIP / transportLP, minship, maxship / ;

option optcr = 0;

Solve transportMIP using MIP minimizing z :

rep(i,j,'MIP') = x.l(i,j);
display rep;
MIP Model: Results
MIP Model: Solution

Canning Plants (supply)  shipments  Markets (demand)

Seattle (350)  Topeka (275)  Chicago (300)  New York (325)

San Diego (600)

Freight: $90 case / thousand miles  Total cost: $153,675
Model types in this example

1. **LP**
   - Determine minimum transportation cost.
   - Result: city to city shipment volumes.

2. **MIP**
   - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

3. **MINLP**
   - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

4. **SP**
   - Allows uncertainty, e.g. uncertain demand
Model types **in this example**

- **LP**
  - Determine minimum transportation cost. Result: city to city shipment volumes.

- **MIP**
  - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**
  - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**
  - Allows uncertainty, e.g. uncertain demand
MINLP: Cost Savings

The cost per case decreases with a increasing shipment volume

x (Number of cases)

Replace:

\[
\min \sum_{i} \sum_{j} c_{ij} \cdot x_{ij}
\]

(Minimize total transportation cost)

With

\[
\min \sum_{i} \sum_{j} c_{ij} \cdot x_{ij}^{\beta}
\]

(Minimize total transportation cost)
MINLP Model: GAMS Syntax

* MINLP
Scalar  beta / 0.95 /
Equation  costnlp define non-linear objective function;
costnlp..  z  =e=  sum((i,j),  c(i,j)*x(i,j)**beta) ;
Model  transportMINLP / transportMIP - cost + costnlp ;
Solve  transportMINLP using MINLP minimizing  z ;
rep(i,j,'MINLP') = x.l(i,j);
display  rep;
MINLP Model: GAMS Syntax

```gams
* MINLP
Scalar beta / 0.95 /
Equation costnlp define non-linear objective function;
costnlp.. z =e= sum((i,j), c(i,j)*x(i,j)**beta) ;
Model transportMINLP / transportMIP - cost + costnlp /;
Solve transportMINLP using MINLP minimizing z ;
rep(i,j,'MINLP') = x.l(i,j);
display rep;|
```
MINLP Model: Results
MINLP Model: Solution

Canning Plants (supply) → shipments (Number of cases) → Markets (demand)

Seattle (350)  →  Topeka (275)
San Diego (600) → Chicago (300)
New York (325)

Freight: $90 case / thousand miles
Total cost: $153,675
Model types in this example

- **LP**
  - Determine minimum transportation cost. Result: city to city shipment volumes.

- **MIP**
  - Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.

- **MINLP**
  - Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows

- **SP**
  - Allows uncertainty, e.g. uncertain demand
Model types **in this example**

- **LP**: Determine minimum transportation cost. Result: city to city shipment volumes.
- **MIP**: Allows discrete decisions, e.g. if we ship, then we ship at least 100 cases.
- **MINLP**: Allows non-linearity, e.g. a smooth decrease in unit cost when shipping volumes grows.
- **SP**: Allows uncertainty, e.g. uncertain demand.
Stochastic Programming in GAMS

**EMP/SP**
- Simple interface to add uncertainty to existing deterministic models
- (EMP) Keywords to describe uncertainty include: discrete and parametric random variables, stages, chance constraints, Value at Risk, ...
- Available solution methods:
  - Automatic generation of Deterministic Equivalent (can be solved with any solver)
  - Specialized commercial algorithms (DECIS, LINDO)
Transport Example - **Uncertain Demand**

- **Uncertain demand factor** \( bf \)

### Decisions to make

- **First-stage decision**: How many units should be shipped “here and now” (without knowing the outcome)
- **Second-stage (recourse) decision**:
  - How can the model react if we do not ship enough?
  - Penalties for “bad” first-stage decisions, e.g. buy additional cases \( u(j) \) at the demand location:

```plaintext
costsp ..  z  =e=  sum((i,j), c(i,j)*x(i,j)) + 
              sum(j, 0.3*u(j));
demandsp(j) ..  sum(i, x(i,j))  =g=  bf*b(j) - u(j) ;
```

<table>
<thead>
<tr>
<th></th>
<th>Prob: 0.3</th>
<th>Val: 0.90</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-york</td>
<td></td>
<td></td>
</tr>
<tr>
<td>chicago</td>
<td>Prob: 0.5</td>
<td>Val: 1.00</td>
</tr>
<tr>
<td>topeka</td>
<td>Prob: 0.2</td>
<td>Val: 1.10</td>
</tr>
</tbody>
</table>
Uncertain Demand - GAMS Algebra

* Stochastic Program with uncertain demand
Positive variable u(j) unsatisfied demand;
Scalar br demand factor / 1/;
Equation costsp define objective function for SP
demandsp(j) demand satisfaction in SP;

costsp.. z =e= sum((i,j), c(i,j)*x(i,j)) + sum(j, 0.3*u(j));
demandsp(j).. sum(i, x(i,j)) - u(j) - br*b(j) =e= u(j);

Model transportSP / costsp, demandsp, supply /;
File emp / '%emp.info%' / put emp;
Smpout
randvar bf discrete 0.3 0.9
0.5 1.0
0.2 1.1

stage 2 bf u demandsp
Smpout
Putclose emp;

Set scen scenarios / s1*s4 /;
Parameter
  s_bf(scen) demand factor for realization by scenario
  s_x(scen,i,j) shipment per scenario
  s_u(scen,j) unsatisfied demand per scenario (bought cases);

Set dict / scen . scenario . ''
  bf . randvar . s_bf
  x . level . s_x
  u . level . s_u /;

option emp=linprog;
Solve transportSP min z use emp scenario dict;
Uncertain Demand - GAMS Algebra

* Stochastic Program with uncertain demand
Positive variable u(j) unsatisfied demand;
Scalar br demand factor / 1 /;
Equation costsp define objective function for SP
   demandsp(j) demand satisfaction in SP;
costsp.. z =e= sum((i,j), c(i,j)*x(i,j)) + sum(j, 0.3*u(j));
demandsp(j).. sum(i, x(i,j)) =g= br*b(j) - u(j);

Model transportSP / costsp, demandsp, supply /;
File emp / '$emp.info' /; put emp;
Solve transportSP using lindo;
Putclose emp;

Set scen scenarios / a1*a4 /;
Parameter
   s_br(scen) demand factor for realization by scenario
   s_x(scen,i,j) shipment per scenario
   s_u(scen,j) unsatisfied demand per scenario (bought cases);

Set dict / scen . scenario . ''
   bf . randvar . s_br
   x . level . s_x
   u . level . s_u /;
option emp=lindo;
Solve transportSP min z use emp scenario dict;
### Uncertain Demand - Results

---
**PARAMETER s_bf** demand factor for realization by scenario

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0.900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>1.100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
**PARAMETER s_b** demand per scenario

<table>
<thead>
<tr>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>292.500</td>
<td>270.000</td>
</tr>
<tr>
<td>s2</td>
<td>325.000</td>
<td>300.000</td>
</tr>
<tr>
<td>s3</td>
<td>357.500</td>
<td>330.000</td>
</tr>
</tbody>
</table>

---
**PARAMETER s_x** shipment per scenario

<table>
<thead>
<tr>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1.seattle</td>
<td>50.000</td>
<td>300.000</td>
</tr>
<tr>
<td>s1.san-diego</td>
<td>242.500</td>
<td>300.000</td>
</tr>
<tr>
<td>s2.seattle</td>
<td>50.000</td>
<td>300.000</td>
</tr>
<tr>
<td>s2.san-diego</td>
<td>242.500</td>
<td>300.000</td>
</tr>
<tr>
<td>s3.seattle</td>
<td>50.000</td>
<td>300.000</td>
</tr>
<tr>
<td>s3.san-diego</td>
<td>242.500</td>
<td>300.000</td>
</tr>
</tbody>
</table>

---
**PARAMETER s_u** unsatisfied demand per scenario (bought cases)

<table>
<thead>
<tr>
<th>new-york</th>
<th>chicago</th>
<th>topeka</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>32.500</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>65.000</td>
<td>30.000</td>
</tr>
</tbody>
</table>
Stochastic Program: Solution

Canning Plants (supply)  shipments  Markets (demand)

Seattle (350)  \[\text{Number of cases}\]
San Diego (600)

Topeka (~275)
Chicago (~300)
New York (~325)

Freight: $90 case / thousand miles
Total cost: $158,588
Stochastic Programming in GAMS

• The Extended Mathematical Programming (EMP) framework is used to replace parameters in the model by random variables

• Support for Multi-stage recourse problems and chance constraint models

• Easy to add uncertainty to existing deterministic models, to either use specialized algorithms or create Deterministic Equivalent (new free solver DE)

Agenda

GAMS at a Glance

GAMS - Hands On Examples

APIs - Application Programming Interfaces to GAMS

Outlook - Some Advanced GAMS Features
Calling GAMS from your Application

Creating Input for GAMS Model
→ Data handling using GDX API

Callout to GAMS
→ GAMS option settings using Option API
→ Starting GAMS using GAMS API

Reading Solution from GAMS Model
→ Data handling using GDX API
Low level APIs \rightarrow Object Oriented API

- Low level APIs
  - GDX, OPT, GAMSX, GMO, ...
  - High performance and flexibility
  - Automatically generated imperative APIs for several languages (C, Delphi, Java, Python, C#, ...)  

- Object Oriented GAMS API
  - Additional layer on top of the low level APIs
  - Object Oriented
  - Written by hand to meet the specific requirements of different Object Oriented languages
Transport Application GUI Example

- Scenario solves of the transportation problem

- Features:
  - Preparation of input data
  - Loading data from Access file
  - Solving multiple scenarios of a model
  - Displaying results

- Four implementation steps:
  1. Graphical User Interface
  2. Preparation of GAMS model
  3. Implementation of scenario solving using GAMSJob
  4. GAMSMModelInstance for performance improvements
Transport Application **GUI Example**

- Scenario solves of the transportation problem

- Features:
  - Preparation of input data
  - Loading data from Access file
  - Solving multiple scenarios of a model
  - Displaying results

- Four implementation steps:
  1. Graphical User Interface
  2. Preparation of GAMS model
  3. Implementation of scenario solving using GAMSJob
  4. GAMSModelInstance for performance improvements
Agenda

GAMS at a Glance

GAMS - Hands On Examples

APIs - Application Programming Interfaces to GAMS

Outlook - Some Advanced GAMS Features
Solvelink Option
controls GAMS function when linking to solve

Model transport /all/;
Option solvelink = {
  %Solvelink.ChainScript%,
  %Solvelink.CallScript%,
  %Solvelink.CallModule%,
  %Solvelink.AsyncGrid%,
  %Solvelink.AsyncSimulate%,
  %Solvelink.LoadLibrary%};

solve transport using lp minimizing z;
Solvelink Option
controls GAMS function when linking to solve

Model transport /all/ ;
Option solvelink = {
  %Solvelink.ChainScript%,
  %Solvelink.CallScript%,
  %Solvelink.CallModule%,
  %Solvelink.AsyncGrid%,
  %Solvelink.AsyncSimulate%,
  %Solvelink.LoadLibrary%};

solve transport using lp minimizing z;

• ChainScript [0]: Solver process, GAMS vacates memory
  + Maximum memory available to solver
  + protection against solver failure (*hostile* link)
  - swap to disk
Solvelink Option – cont.

- Call{Script [1]/Module [2]}: Solver process, GAMS stays live
  + protection against solver failure (*hostile* link)
  + no swap of GAMS database
  - file based model communication
Call{Script [1]/Module [2]}: Solver process, GAMS stays live
  + protection against solver failure (*hostile* link)
  + no swap of GAMS database
  - file based model communication

LoadLibrary [5]: Solver DLL in GAMS process
  + fast memory based model communication
  + update of model object inside the solver (hot start)
  - not supported by all solvers
transport.gms (LP) solved 500 times with CPLEX:
**Simple Serial Solve - Performance**

trnsport.gms (LP) solved 500 times with CPLEX:

The `solvelink` option controls GAMS function when linking to solve.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Solve time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solvelink=%Solvelink.ChainScript%</td>
<td>54.368</td>
</tr>
<tr>
<td>Solvelink=%Solvelink.CallModule%</td>
<td>42.909</td>
</tr>
<tr>
<td>Solvelink=%Solvelink.LoadLibrary%</td>
<td>05.039</td>
</tr>
</tbody>
</table>

**Hands-On**
Generates model once and updates the algebraic model *keeping the model “hot”* inside the solver.
Scenario Solver/GUSS - Performance

trnsport.gms (LP) solved 500 times with CPLEX:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Solve time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solvelink=%Solvelink.ChainScript%</td>
<td>54.368</td>
</tr>
<tr>
<td>Solvelink=%Solvelink.CallModule%</td>
<td>12.909</td>
</tr>
<tr>
<td>Solvelink=%Solvelink.LoadLibrary%</td>
<td>05.039</td>
</tr>
<tr>
<td>GUSS</td>
<td>01.947</td>
</tr>
</tbody>
</table>
# Scenario Solver/GUSS - Performance

Example: Stochastic model with 66,320 linear problems

<table>
<thead>
<tr>
<th>Setting</th>
<th>Solve time (secs)</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop: Solvelink=%Solvelink.Chainscript (default)</td>
<td>7,204</td>
<td></td>
</tr>
<tr>
<td>Loop: Solvelink=%Solvelink.LoadLibrary%</td>
<td>2,481</td>
<td>18.3</td>
</tr>
<tr>
<td>GAMS Scenario Solver</td>
<td>392</td>
<td>1.86</td>
</tr>
<tr>
<td>CPLEX Concert Technology</td>
<td>210</td>
<td></td>
</tr>
</tbody>
</table>
Grid Computing Facility

GAMS jobs in a **distributed** environment

- Scalable: supports large grids, but also works on local machine
- Platform independent, works with all solvers/model types
- Only minor changes to model required

---

**GAMS**

- **Submit Job 1**
  - **Solve Job 1**
    - Process solution information
  - ...  
    - **Solve Job n**
    - Process solution information

**GAMS**

- **Job Submission**
  - **Solve Job 1**
  - **Solve Job 2**
  - **Solve Job ..**
  - **Solve Job n**

- **Solution Collection**
  - **Solve Job 1**
  - **Solve Job 2**
  - **Solve Job ..**
  - **Solve Job n**
Grid Computing Facility – Example 1

GAMS Model Library: trnsgrid

```
transport.solvlink = %solvlink.AsyncGrid%; // turn on
transport.limcol = 0;
transport.limrow = 0;
transport.solprint = %solprint.Quiet%;

set s scenarios / 1*5 /;

parameter dem(s,j) random demand
     h(s) store the instance handle;

dem(s,j) = b(j)*uniform(0.85,1.15); // create some random demands

loop(s,
    b(j) = dem(s,j);
    Solve transport using lp minimizing z;
    h(s) = transport.handle; // save instance handle

parameter repx(s,i,j) solution report
    repy summary report:

  repy(s,'solvestat') = na;
  repy(s,'modelstat') = na;

  if we use the handle parameter to indicate that the solution has been collected
  repeat
    loop(s#$handlecollect(h(s)),
        repx(s,i,j) = x.l(i,j);
        repy(s,'solvestat') = transport.solvestat;
        repy(s,'modelstat') = transport.modelstat;
        repy(s,'sumstat') = transport.sumstat;
        repy(s,'obval') = transport.objval;
        display$h,s(handlecollect(h(s)) 'trouble deleting handles'
        h(s) = 0; // indicate that we have loaded the solution
        display$sleep(card(h)*0.2) 'was sleeping for some time';
    until card(h) = 0 or timesleaped > 10; // wait until all models are loaded

display repx, repy;

abort$sum((s(repy(s,'solvestat')=na),1) 'Some jobs did not return';
```
Grid Computing Facility – Example 1

GAMS Model Library: trnsgrid

```
transport.solvlink = %solvlink.AsyncGrid%;  // turn on the parallel solver
transport.limcol   = 0;
transport.limrow   = 0;
transport.solprint = %solprint.Quiet%;

set s scenarios / 1*5 /;
parameter dem(s,j) random demand
      h(s) store the instance handle;
dem(s,j) = b(j)*uniform(0.35,1.15);  // create some random demands
loop(s,
    b(j) = dem(s,j);
    Solve transport using lp minimizing z;
    h(s) = transport.handle;  // save instance handle
)

parameter repx(s,i,j) solution report
    repy summary report;
repy(s,'solvestatus') = na;
repy(s,'modelstatus') = na;
* we use the handle parameter to indicate that the solution has been collected
repeat
    loop(s$handlecollect(h(s)),$)
        repx(s,i,j) = x.l(i,j);
        repy(s,'solvestatus') = transport.solvestatus;
        repy(s,'modelstatus') = transport.modelstatus;
        repy(s,'resnum') = transport.resnum;
        repy(s,'objval') = transport.objval;
    display$h(handledelete(h(s)) 'trouble deleting handles';
    h(s) = 0;  // indicate that we have loaded the solution
    display$sleep(card(h)*0.2) 'was sleeping for some time';
until card(h) = 0 or timeslept > 10;  // wait until all models are loaded
display repx, repy;
abort$sum(s(repy(s,'solvestatus')=na),1) 'Some jobs did not return';
```
Grid Computing Facility – Example 2

GAMS Model Library: tgridmix

```plaintext

dem(s,s) = b(j)*uniform(.95,1.15); // create some random demands

repeat
  while (card(actS)<maxs and card(nexS)<maxs),
    loop(nexS(s),
      b(j) = dem(s,j)
      Solve transport using lp minimizing z:
      h(s) = transport.handle;
      actS(s) = yes;
    );
    nexS(s) = nexS(s-1); // advance nexS
  
  co1S(s) = no;
  display$ReadyCollect(h) 'Waiting for next instance to collect';
  loop(actS(s)$handlecollect(h(s)),
    rcpx(s1,s,j) = x.l(i,j);
    reply(s1,s,'solvestatus') = transport.solvestat;
    reply(s1,s,'modelstat') = transport.modelstat;
    reply(s1,s,'resud') = transport.resud;
    reply(s1,s,'objval') = transport.objval;
    display$handledelete(h(s)) 'trouble deleting handles';
    co1S(s) = yes; h(s) = 0;
  );

until (card(nexS)=0 and card(actS)=0) or timeelapsed > 10; // wait until a
reply(s1,'time','elapsed') = (tstart - tstart)*3600*24;
abort&sum((s1,s,'solvestatus')=no,1) 'Some jobs did not return';

display repx, reply;
```

---

FOR/WHILE = 2
FOR/WHILE = 1

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.742

tgridmix.gms(89) 4 Mb

Generating LP model transport

tgridmix.gms(100) 4 Mb

LOOPS s1 = Grid

FOR/WHILE = 2
FOR/WHILE = 2

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.759

tgridmix.gms(89) 4 Mb

Generating LP model transport

tgridmix.gms(100) 4 Mb

LOOPS s1 = Grid

FOR/WHILE = 2
FOR/WHILE = 2

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.776

tgridmix.gms(89) 4 Mb

Generating LP model transport

tgridmix.gms(100) 4 Mb

LOOPS s1 = Grid

FOR/WHILE = 2
FOR/WHILE = 3

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.765

tgridmix.gms(89) 4 Mb

Generating LP model transport

tgridmix.gms(100) 4 Mb

LOOPS s1 = Grid

FOR/WHILE = 2
FOR/WHILE = 2

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.776

tgridmix.gms(89) 4 Mb

Generating LP model transport

tgridmix.gms(100) 4 Mb

LOOPS s1 = Grid

FOR/WHILE = 2
FOR/WHILE = 3

6 rows 7 columns 19 non-zeros

Executing after solve: elapsed 0:00:00.789

tgridmix.gms(121) 4 Mb

Status: Normal completion

Job tgridmix.gms Stop 08/25/16 06:22:43 elapsed 0:00:00.899
```

Close Open Log □ Summary only □ Update

75
```
Grid Computing Facility – Example 2

GAMS Model Library: tgridmix

dem(s,j) = b(j)*uniform(.95,1.15); // create some random demands
        tStart = jnow;
        reply(s,j,'solvestat') = no;
        reply(s,j,'modelstat') = no;
        actS(s) = no; h(s) = 0; nexS(s) = sammem('1',s);
        transport.solveLink = snum(s);
repeat
    while (card(actS) < maxS and card(nexS),
        loop(nexS(s),
            b(j) = dem(s,j)
            Solve transport using lp minimizing z;
            h(s) = transport.handle;
            actS(s) = yes;
        );
        nexS(s) = nexS(s-1); // advance nexS
    );
colS(s) = no;
display$ReadyCollect(h) 'Waiting for next instance to collect';
        loop (actS(s)@handleCollect(h(s)),
            rpx(s,l,i,j) = x.l(i,j);
            reply(s,l,'solvestat') = transport.solvestat;
            reply(s,l,'modelstat') = transport.modelstat;
            reply(s,l,'resud') = transport.resud;
            reply(s,l,'objval') = transport.objval;
            displayHandleDelete(h(s)) 'trouble deleting handles';
            colS(s) = yes; h(s) = 0;
        );
        actS(colS) = no;
until (card(nexS) = 0 and card(actS) = 0) or timeelapsed > 10; // wait until all
        reply(sl,'time', 'elapsed') = (jnow - tStart)*3600*24;
        abort$sum (s(reply(sl,s,'solvestat') = no),1) 'Some jobs did not return';
    );
display repx, reply;

Hands-On
Solving “many” Scenarios
How to find the right approach?

1. Small Ratio of solver time / GAMS time → Scenario Solver
2. Large ratio i.e. only solver time is relevant (pre/post processing not critical) → Grid Computing Facility
3. Entire model run including pre processing / optimization / post processing is costly → Parallel execution of entire model in the cloud
### Application - Scenario Solver

#### Scenario Solver and Parallel Combined

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Number of MIP models</th>
<th>Solve time</th>
<th>Rest of algorithm</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional GAMS loop</td>
<td>100,000</td>
<td>1068 sec</td>
<td>169 sec</td>
<td>1237 sec</td>
</tr>
<tr>
<td>Scenario Solver</td>
<td>100,000</td>
<td>293 sec</td>
<td>166 sec</td>
<td>459 sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Number of MIP models</th>
<th>Worker Threads</th>
<th>Parallel sub-problem time</th>
<th>Rest of algorithm (serial)</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel + Scenario Solver</td>
<td>100,000</td>
<td>4</td>
<td>116 sec</td>
<td>67 sec</td>
<td>183 sec</td>
</tr>
</tbody>
</table>

http://yetanothermathprogrammingconsultant.blogspot.de/2012/04/parallel-gams-jobs-2.html
Where to Find Help?

• Documentation Center: http://gams.com/help/index.jsp

• Support Wiki: http://support.gams.com/

• Mailing List(s): http://gams.com/maillist/index.htm

• YouTube Channel: https://www.youtube.com/user/GAMSLessons

• GAMS support: support@gams.com

Meet us at the GAMS booth!
Other GAMS Talks

**BEAM-ME: Acceleration Strategies for Energy System Models**

*Given by:* Frederik Fiand  
*When:* Thursday (Sep. 01), 09:00-09:30  
*Where:* Hörsaal 3  
*Abstract:* BEAM-ME is a project funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) and addresses the need for new and improved solution approaches for energy system models. The project unites various partners with complementary expertise from the fields of algorithms, computing and application development. The considered problems result in large-scale LPs that are computationally intractable for state-of-the-art solvers. Hence, new solution approaches combining decomposition methods, algorithm development and high performance computing are developed. We provide an overview on the large variety of challenges we are facing within this project, present current solutions approaches and provide first results.

**Recent Enhancements in GAMS**

*Given by:* Franz Nelissen  
*When:* Thursday (Sep. 01), 11:30-12:00  
*Where:* Seminarraum 105  
*Abstract:* Algebraic Modeling Languages (AML) are one of the success stories in Operations Research. GAMS is one of the prominent AMLs and has evolved continuously in response to user requirements, changes in computing environments and advances in the theory and practice of mathematical programming. In this talk we will begin with some fundamental principles and outline several recent enhancements of GAMS supporting efficient and productive development of optimization based decision support applications.