Pre-Conference Workshops

Michael Bussieck
Steve Dirkse
Fred Fiand
Lutz Westermann
Outline

Part I: An Introduction to GAMS

Part II: Stochastic programming in GAMS

Part III: The GAMS (Object-Oriented) API's

Part IV: Code embedding in GAMS
# Excel and GAMS

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Distance</td>
<td>New-York</td>
<td>Chicago</td>
<td>Topeka</td>
<td>Supply</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
<td>350</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>San-Diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
<td>600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Demand</td>
<td>325</td>
<td>300</td>
<td>275</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Freight cost</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SHIPMENT</td>
<td>New-York</td>
<td>Chicago</td>
<td>Topeka</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Seattle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>San-Diego</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>GAMS Directory:</td>
<td>C:\GAMS\win64\24.8.5\</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Working Directory:</td>
<td>c:\tmp\</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Solver:</td>
<td>CPLEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph with line chart showing distances and costs]
Excel and GAMS

- VBA GAMS API to call GAMS from Excel
- Exchange of input data and results using either GDXXRW or GDX API
Embedding GAMS in 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 → 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
Features of the object oriented API

• No modeling capability! Model is still written in GAMS

• Control GAMS execution → \textit{GAMSJob}

• Prepare input data and retrieve results in a convenient way → \textit{GAMSDatabase}

• Scenario Solving: Feature to solve multiple very similar models in a dynamic and efficient way → \textit{GAMSModelInstance}

• Seamless integration of GAMS into other programming environments

• .NET, C++, Java and Python APIs are part of the current GAMS release available at \url{www.gams.com}. Many examples available:
  • Sequence of Transport examples (Tutorial)
  • Cutstock, Warehouse, Benders Decomposition, ...
using System;
using GAMS;

namespace TransportSeq
{
    class Transport1
    {
        static void Main(string[] args)
        {
            GAMSWorkspace ws = new GAMSWorkspace();
            GAMSJob t1 = ws.AddJobFromGamsLib("transport");

            t1.Run();
            foreach (GAMSVariableRecord rec in t1.OutDB.GetVariable("x"))
            {
                Console.WriteLine("x(\" + rec.Key(0) + ",\" + rec.Key(1) + "):\");
                Console.WriteLine("    level=\" + rec.Level);
                Console.WriteLine("    marginal=\" + rec.Marginal);
            }
        }
    }
}
Seamless Integration

- GAMS concept: Separation of tasks

- Use GAMS for modeling and optimization tasks
- Programming languages like C# (.NET), C++, Java and Python are well-suited for developing applications
  - Frameworks available for a wide range of specific task:
    - GUI
    - Web development
    - ...
- The object oriented GAMS API provides a convenient link to run GAMS in such environments
Seamless Integration

- Example: Small Transport Desktop application written in C#
- Convenient data preparation
- Representation of the results in a predefined way
- Modeling details are hidden from the user
Scenario Solving

Solving Transport in a loop with different scenarios for the demand:

\[
\text{Loop}(s, \\
  \text{d}(i,j) = \text{dd}(s,i,j); \\
  \text{solve} \text{ transport using lp minimizing z; } \\
  \text{objrep}(s) = \text{transport.objval}; \\
)\
\]
set dict / s.scenario.

d.param .dd

z.level .objcrep /

solve transport using lp minimizing z;

• Save model generation and solver setup time
• Hot start (keep the model hot inside the solver and use solver’s best update and restart mechanism)

• Apriori knowledge of all scenario data
• Model rim unchanged from scenario to scenario
**Scenario Solving – GAMSModelInstance**

```plaintext
define (string s in scen) {
    f.FirstRecord().Value = v[s];
    modelInstance.Solve();
    objrep[s] = z.FirstRecord().Level;
}
```

- Save model generation and solver setup time
- Hot start (keep the model hot inside the solver and use solver’s best update and restart mechanism)
- Data exchange between solves possible
- Model rim unchanged from scenario to scenario
GAMSJob
• Manages the execution of a GAMS program given by GAMS model source

GAMSCheckpoint
• Captures the state of a GAMSJob

GAMSModelInstance
• A single mathematical model generated by a GAMS solve statement

GAMSModifier
• Marks elements of a GAMSModelInstance to be modifiable
• `bmult` is one parameter of the model which gets modified before we solve the instance:

```csharp
GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
bmult.AddRecord().Value = 1.0;
mi.Instantiate("transport us lp min z", opt, new GAMSModifier(bmult));
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };

foreach (double b in bmultlist)
{
    bmult.FirstRecord().Value = b;
    mi.Solve();
    ...
    Console.WriteLine("Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
}
```
GAMSModelInstance – Example

- Updating bounds of a variable:

```gams
GAMSVariable x = mi.SyncDB.AddVariable("x", 2, VarType.Positive, "");
GAMSPParameter xup = mi.SyncDB.AddParameter("xup", 2, "upper bound on x");
mi.Instantiate("transport us lp min z", modifiers: new GAMSModifier(x, UpdateAction.Upper, xup));

foreach (GAMSSetRecord i in t7.OutDB.GetSet("i"))
    foreach (GAMSSetRecord j in t7.OutDB.GetSet("j"))
    {
        xup.Clear();
        xup.AddRecord(i.Keys[0], j.Keys[0]).Value = 0;
        mi.Solve();
        <...>
        Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
    }
```
GAMSModelInstances in **Parallel**

- Multiple GAMSModelInstances running in parallel with one common data source (work):
GAMS Model Instances in Parallel

- Threads consume data from source dynamically instead of getting a fixed amount of data at thread initialization time.

- Implicit load balancing by architecture:
  - Number of solves in a thread depend on its speed
  - Keeps all threads busy as long as possible

- Typical applications:
  - Scenario analysis
  - Decomposition algorithms (Benders, CG, ...)

- Communication between threads for “dynamic” algorithms.
Summary

• Object Oriented API provides an additional abstraction layer of the low level GAMS APIs

• Powerful and convenient link to other programming languages

• Versions for .NET, C++, Java, and Python are available and part of the current distribution

• Many examples are available:
  • Sequence of Transport examples (Tutorial)
  • Cutstock, Warehouse, Benders Decomposition, …
Thank You

Meet us at the GAMS booth!