Embedded Code in GAMS Using Python as an Example

Lutz Westermann
24.9.1 Major release (August 30, 2017)

Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Etienne Ayotte-Sauvé, Wolfgang Britz, Florian Habermacher, Florian Häberlein, Maximilian Held, Ignacio Herrero, Hanspeter Höschle, Erwin Kalvelagen, Toni Lastusilta, John Ross, and Tom

GAMS System

New feature, the Embedded Code Facility: This extends the connectivity of GAMS to other programming languages. It allows the use of Python code during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary.

The embedded code feature is available on Linux, MacOS X, and Windows. For these platforms, a Python 3.6 installation is included with the GAMS distribution. If the user wants to work with a different Python 3.6, installed separately, for models with embedded code the new command line option pySetup needs to be set to 0.

Note

This feature is currently in beta status. Any feedback to support@gams.com is appreciated.

• New command line option procDirPath: Specifies the directory where the process directory should be created.
Motivation – Avoid Unreadable/Slow Code

- GAMS code for *parallel assignment* and *equation definition* is compact, elegant, and efficient
- GAMS uses relational data tables as a base data structure
- Traditional data structure are not available in GAMS
  - No arrays, lists, dictionaries, trees, graphs, ...
- GAMS can represent such traditional data structures but ...
  - GAMS code becomes quickly unreadable
    \[
    t(tt) = t(tt-1); \quad /\!/ \text{advances set element } t \text{ to } t+1
    \]
  - Performance with naïve representation is very inefficient
    \[
    t(tt) = \text{ord}(tt)=tCnt; \quad /\!/ \text{advances set element } t \text{ to } t+1
    \]
  - Writing code that executes efficiently requires deep understanding of underlying GAMS internal data structures and often results in even more unreadable code
Motivation –
Data Input/Transformation at Compile Time

• GAMS data input (ASCII) follows strict syntax
• Practical GAMS models get data (via ASCII input files) that is often not in a proper shape
  • Hence GAMS code is often augmented with execution of scripts and programs to get data files into a GAMS readable shape
  • GAMS even ships a selection of POSIX text utilities (sed, grep, awk, ...) on Windows to support a somewhat standardized way of transforming text files into GAMS readable format
  • Scripts spawned by GAMS cannot (easily) access data that is already available in GAMS
• GAMS has no string editing facilities to e.g.
  • modify labels
  • change content of compile time variables
    • “Solution”: $xxx new and weird compile time constructs, e.g. $setNames, $splitOption, ...
Motivation – Other

• Connecting libraries for special algorithms (e.g. graph algorithms like connected components, matrix operations like Cholesky factorization) to GAMS is not easy

• Current “solution” has issues
  • $unload/$call/$load or execute_unload/execute/execute_load
    • Performance: disk IO + process creation
    • Knowledge of data API (GDX or OO-API)
    • Remapping of relational data (plus concept of UELs) into other data structures
  • Add compile time directives to perform a single special task (e.g. $splitOption)
  • Introduce unreadable option or put_utility syntax to perform a single special task (e.g. option a<b;)

• Note: Object Oriented API/Framework versus Embedded Code
  • OO-API: Framework in control
  • Embedded Code: GAMS in control
Embedded Code

• Support the use of external code during GAMS compile and execution time
• Provide support for off-line debugging of embedded code
• Share GAMS symbols (sets, parameters, variables, and equations) structure and content with the external code in memory
• Communication of the data between GAMS and the embedded code inspired by the existing interface to GDX in many ways:
  • Records access by both labels and label indices
  • Data in GAMS can be merged with or replaced by data from embedded code
  • Data from embedded code can be send to GAMS database filtered or domain checked
• Provide automatically generated, additional source code for common tasks

→ Allows the user to concentrate on the task at hand and not the mechanics
Quantiles
Calculate $|i||j|$ quantiles of $|k|$ numbers:

Set $i / ... /, j / ... /, k / ... /$;
Table $p(i,j,k)$

    k1      k2      ...
 i1.j1  18.002966 84.483404 ...
 i1.j2  7.644259 50.520856 ...
    ...
 Set q; Parameter quantiles(i,j,q);

$\text{onEmbeddedCode} \begin{align*}
    \text{Python:} \\
    &\text{import pandas as pd} \\
    &\text{df} = \text{pd.DataFrame(list(gams.get('p')))} \\
    &\text{q} = \text{df.groupby([0,1]).quantile([0,.25,.5,.75,1])} \\
    &\text{gams.set('quantiles',} \\
    &\quad [ (*k[:-1], \text{str(k[-1])}, v[3]) \text{ for } k,v \text{ in q.iterrows() } ] \\
    \text{)}$
$\text{offEmbeddedCode}$

$q<quantiles$

Display $q$, quantiles;

Inspired by E. Kalvelagen
country / system.empty /
city / system.empty /
mccCountry(cc,country)
mccCity (cc,city);
Split Example – **Embedded Code**

```python
$onEmbeddedCode Python:
country = set()
city = set()
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
    r = str.split(cc, " - ", 1)
country.add(r[0])
city.add(r[1])
imcCountry.append((cc,r[0]))
imcCity.append((cc,r[1]))
gams.set("country",list(country))
gams.set("city",list(city))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode
country city mccCountry mccCity
```
**Split Example – Output**

**Display** country, city;

---

27 SET country

France, USA, Spain, Germany

---

27 SET city

Seville, Washington DC, New York, Paris
Munich, Madrid, Toulouse, Berlin
Bonn, Lille, Houston, Bilbao
Cordoba
Plot Example

[added after GAMSlib model meanvar]

Parameter mean(p), var(p);
mean(p) = xres('mean',p);
var(p) = xres('var',p);

embeddedCode Python:
    import matplotlib
    matplotlib.use('WXAgg')
    import matplotlib.pyplot as plt
    plt.plot(gams.get('mean', keyFormat=KeyFormat.SKIP), \
             gams.get('var', keyFormat=KeyFormat.SKIP), \
             marker='.', markersize=10)
    plt.xlabel('return')
    plt.ylabel('variance')
    plt.show()
Plot Example – Output
Exchange via Files

$onEmbeddedCode Python: 10
  f = open('i.txt', 'w')
  for i in range(int(gams.arguments)):
    f.write('i'+str(i+1)+'\n')
  f.close()
$offEmbeddedCode

Set i /
$include i.txt /
Display i;

----  21 SET i

i1 ,   i2 ,   i3 ,   i4 ,   i5 ,
i6 ,   i7 ,   i8 ,   i9 ,   i10
Exchange via Environment Variables

Set  i    / i1*i5    ;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47    ;
Set    k "from 0 to max(b)" / k0*k?    ;

$onEmbeddedCode Python:
    import os
    kmax = int(max([b[1] for b in list(gams.get("b"))]))
    gams.printLog('max value in b is ' + str(kmax))
    os.environ['MAXB'] = str(kmax)
$offEmbeddedCode

$if x%sysEnv.MAXB%==x $abort MAXB is not set
Set    k "from 0 to max(b)" / k0*k%sysEnv.MAXB%    ;
Scalar card_k;
    card_k = card(k);
Display card_k;

----  15 PARAMETER card_k = 60.000
Performance Considerations

Set i / i1*i50/, p(i,i); Alias (i,ii);
Parameter c(i,i); c(i,ii) = uniform(-50,50);

Set iter / 1*100/;
Scalar tcost, minTCost / +inf /

loop(iter,
    embeddedCode Python:
    import random
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
    endEmbeddedCode
    tcost = sum(p, c(p));
    if (tcost < minTCost, minTCost = tcost);
)
Display minTCost;

EXECUTION TIME = 16.375 SECONDS
Performance Considerations

Set i / i1*i50 /, p(i,i); Alias (i,ii);
Parameter c(i,i); c(i,ii) = uniform(-50,50);

embeddedCode Python:
   import random
pauseEmbeddedCode

Set iter / 1*1000 /;
Scalar tcost, minTCost / +inf /;
loop(iter,
   continueEmbeddedCode:
      i = list(gams.get("i"))
      p = list(i)
      random.shuffle(p)
      for idx in range(len(i)):
         p[idx] = (i[idx], p[idx])
      gams.set("p", p)
pauseEmbeddedCode p
      tcost = sum(p, c(p));
      if (tcost < minTCost, minTCost = tcost);
   );
continueEmbeddedCode:
   pass
endEmbeddedCode
Display minTCost;

EXECUTION TIME      =      1.797 SECONDS
Performance Considerations

Set \( i / i1*i50 /, \) \( p(i,i); \) Alias \((i,ii); \)
Parameter \( c(i,i); c(i,ii) = \text{uniform}(-50,50); \)

embeddedCode Python:
import random
i = list(gams.get("i"))
pauseEmbeddedCode

Set \( \text{iter} / 1*1000 /; \)
Scalar \( \text{tcost, minTCost} / +\text{inf} /; \)
loop(iter,
  continueEmbeddedCode Python:
  p = list(i)
  random.shuffle(p)
  for idx in range(len(i)):
    p[idx] = (i[idx], p[idx])
    gams.set("p", p)
pauseEmbeddedCode p
tcost = sum(p, c(p));
  if (tcost < minTCost, minTCost = tcost);
);
continueEmbeddedCode:
pass
endEmbeddedCode
Display minTCost;

EXECUTION TIME = 1.593 SECONDS
Performance Considerations

Set \( i / i1*i50 /, \ p(i,i); \) Alias \( (i,\alpha); \) Parameter \( c(i,i); \ c(i,\alpha) = \text{uniform}(-50,50); \)

eMBEDDED CODE Python:
    
    import random
    i = list(gams.get("i", keyType=KeyType.INT))

PAUSE EMBEDDED CODE

Set \( \text{iter} / 1*1000 /; \)
Scalar \( \text{tcost}, \minTCost / +\text{inf} /; \)
loop(\text{iter},
    continue EMBEDDED CODE Python:
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
    pause EMBEDDED CODE p
    tcost = sum(p, c(p));
    if (tcost < \minTCost, \minTCost = tcost);
)
    continue EMBEDDED CODE:
        pass
END EMBEDDED CODE

Display \( \minTCost; \)

EXECUTION TIME = 1.437 SECONDS
Some **Examples** of Python Embedded Code

- Splitting of labels (compile time)
- Permutation
- Sorting
- Calculation of quantiles
- Power set
- Matching
- Parsing of specially structured ASCII input
- TSP subtour elimination
- Benders Decomposition using **Message Passing Interface (MPI)** on **High-Performance Computing (HPC)** infrastructure
Next steps ...

• More examples
  • High Performance Libraries for specific tasks
    • FORTRAN (Factorization of matrix)
    • C/C++ (Expansion Planning Power Systems)
  • Support of other popular frameworks (compiled and interpreted)
    • C/C++
    • C#/.NET, Java, R, ...
    • Connect of powerful libraries e.g. boost::graph, ...

• Provide a configurable build system that supports building the required libraries (for compiled languages) at GAMS compile time

• Provide a documented API to allow integration of individual user embedded code libraries

• Asynchronous/parallel use of embedded code

This feature is currently in beta status.
Any feedback to support@gams.com is highly appreciated.
Thank You

Meet us at the GAMS booth!
Syntax: **GAMS**

Compile Time:

```plaintext
$onEmbeddedCode[S|V] Python: [arguments]
    Python code
{Python code}
$offEmbeddedCode {output symbol}
```

- $onEmbeddedCode[S] Python: [arguments]
  - Starts a section with Python code
  - Parameter substitution is activated
  - The optional arguments can be accessed in the Python code

- $onEmbeddedCodeV Python: [arguments]
  - Same as $onEmbeddedCode but parameter substitution is disabled (the Python code is passed on verbatim)

- $offEmbeddedCode {output symbol}
  - Ends a section with Python code
  - The optional output symbol(s) get updated in the GAMS database
Syntax: **GAMS**

Execution Time:

```
EmbeddedCode[S|V] Python: [arguments]
    Python code
{Python code}
endEmbeddedCode {output symbol}
```

- **EmbeddedCode[S]** Python: [arguments]
  - Starts a section with Python code
  - Parameter substitution is activated
  - The optional arguments can be accessed in the Python code

- **EmbeddedCode[V]** Python: [arguments]
  - Same as **EmbeddedCode** but parameter substitution is disabled
    (the Python code is passed on verbatim)

```
endEmbeddedCode {output symbol}
```

- Ends a section with Python code
- The optional output symbol(s) get updated in the GAMS database
Syntax: **GAMS**

**Execution Time:**

pauseEmbeddedCode {output symbols}
continueEmbeddedCode[S|V] [handle]: [arguments]

- **pauseEmbeddedCode** {output symbol}
  - Pauses a section with Python code
  - The optional output symbol(s) get updated in the GAMS database

- **continueEmbeddedCode[S]** [handle]: [arguments]
  - Continues a previously paused section with Python code
  - Parameter substitution is activated
  - The optional arguments can be accessed in the Python code
  - The optional handle (pointing to a specific paused embedded code section) could be retrieved by the function `embeddedHandle`. If omitted, the last section paused will be continued.

- **continueEmbeddedCodeV** [handle]: [arguments]
  - Same as `continueEmbeddedCode` but parameter substitution is disabled (the Python code is passed on verbatim)
Syntax: **Python**

The Python Class `ECGamsDatabase` serves as interface between GAMS and Python. An instance of this class is automatically created when an embedded code section is entered and can be accessed using the identifier `gams`. Several methods can be used to interact with GAMS:

- `gams.get(symbolName, [...])`
  - Retrieves iterable object representing the symbol `symbolName`
  - Several optional parameters allow to modify format of the data
- `gams.set(symbolName, data[, merge][, domCheck])`
  - Sets data for the GAMS symbol `symbolName`
  - Data takes a Python list of items representing records of the symbol
  - Optional parameter `merge` specifies if data in a GAMS symbol is merged or replaced
  - Optional parameter `domCheck` specifies if Domain Checking is applied
Syntax: **Python**

- `gams.getUel(idx)`
  - Returns the label corresponding to the label index `idx`
- `gams.mergeUel(label)`
  - **Adds label** to the GAMS universe if it was unknown and returns the corresponding label index
  - **Note:** New labels cannot be added at execution time
- `gams.getUelCount()`
  - Returns the number of labels
- `gams.printLog(msg)`
  - Print `msg` to log
- `gams.arguments`
  - Contains the arguments that were passed to the Python interpreter at start-up of the embedded code section
- `gams.epsAsZero`
  - Flag to read GAMS EPS as 0 [True] or as a small number (4.94066E-300) [False]
- `gams._debug`
  - Debug flag for additional output