pyGDX Documentation

Release 2

Paul Natsuo Kishimoto

Jan 15, 2020

Contents

pyGDX is a Python package for accessing data stored in *GAMS Data eXchange* (GDX) files. GDX is a proprietary, binary file format used by the General Algebraic Modelling System (GAMS); pyGDX uses the Python bindings for the GDX API.

pyGDX uses xarray to provide labeled, multidimensional data structures for accessing data. A *gdx.File* is a thinly-wrapped xarray.Dataset.

Report bugs, suggest feature ideas or view the source code on GitHub.

CHAPTER 1

Documentation

1.1 Installation

pyGDX depends on the low-level application programming interface (API) provided with GAMS, that allows Python code to access the contents of GDX files.

1.1.1 All platforms

1. Install the latest version of GAMS.

The remaining steps depend on the platform:

1.1.2 Linux, Mac OS X

2. Navigate to the GAMS Python API directory. If gams is installed at (for instance) */opt/gams*, this will be */opt/gams/apifiles/Python/api_34* (Python 3.4+, on Linux) or */opt/gams/apifiles/Python/api* (Python 2.7, on Mac OS X).
3. Run either `python setup.py install` (to install all the GAMS bindings) or `python gdxsetup.py install` (to install only the GDX bindings needed by pyGDX).
4. Navigate to the directory containing pyGDX, and again run `python setup.py install`

1.1.3 Windows

Note: There are multiple ways to get a working pyGDX on Windows, but the following is the simplest for new users.

2. Install Anaconda for Python 3.5. Install into your home directory (e.g. *C:\Users\Yourname\Anaconda*) instead of the system-wide install—this avoids later issues with permissions.
3. Create a new Anaconda environment using Python 3.4:[1] open a Command Prompt and run `conda create --name py34 python=3.4 anaconda xarray [PACKAGES]`, where `[PACKAGES]` are the names of any other packages you may need in this environment.[2] Activate the new environment with `activate py34`.
4. In the same command prompt, navigate to the GAMS Python API directory. If GAMS is installed at (for instance) *C:\GAMS\24.6*, this will be *C:\GAMS\24.6\apifiles\Python\api_34*. Run either `python setup.py install` (to install all the GAMS bindings) or `python gdxsetup.py install` (to install only the GDX bindings needed by pyGDX). The bindings will be installed in the *py34* Anaconda environment.
5. Navigate to the directory containing pyGDX, and again run `python setup.py install`.

Steps 4 and 5 may be repeated for any new Anaconda environment in which pyGDX is needed.

1.2 GDX data terminology

Objects in GDX files are termed **Symbols**, and are of several types:

- **Sets** are ordered collections of labels.
- **Parameters** contain numerical data.
- **Variables** are scalar values.
- **Aliases** are alternate names for other Symbols.
- **Equations**, not currently supported by PyGDX.

For clarity (e.g., Python has a built-in class `python.set`), these terms are capitalized throughout this documentation.

[1] This is necessary because GAMS only ships bindings for Python 3.4, and not the newest Python 3.5. Unlike on Linux, the Python 3.4 bindings do not work with Python 3.5.

[2] The Anaconda documentation recommends adding packages when creating the environment, if possible, instead of installing them later.

Both Sets and Parameters may be declared with one-dimensional Sets for each dimension. An example:

```
set  s  'Animals'  /
  a  Aardvark
  b  Blue whale
  c  Chicken
  d  Dingo
  e  Elephant
  f  Frog
  g  Grasshopper
  /;

set  t  'Colours'  /
  r  Red
  o  Orange
  y  Yellow
  g  Green
  b  Blue
  i  Indigo
  v  Violet
  /;

set  u  'Countries'  /
  CA  Canada
  US  United States
  CN  China
  JP  Japan
  /;

set v(s,t) 'Valid animal colours'
  / set.s.set.t yes /;

parameter p(s,t,u) 'Counts of nationalistic, colourful animals'
  / set.s.set.t.set.u 5 /;

parameter total(s) 'Total populations of each type of animal';
total(s) = sum((t, u), p(s, t, u));

execute_unload 'example.gdx';
```

In the resulting file *example.gdx*:

- `s`, `t` and `u` are 1-dimensional Sets.
- `v` is a 2-dimensional Set, defined over the *parent* Sets `s` and `t`. Any Set defined with reference to others, in this way, may include or exclude each element of the parent set. For instance, the following GAMS code defines a subset of `u`:

```
set na(u)  'North American countries' / CA, US /;
```

- `p` and `total` are Parameters containing numerical data.

1.2.1 Other concepts

The **universal Set**, `*`, contains every element appearing in any Set in the GDX file.

- In the above example, `*` would contain: `a b c d e f g r o y b i v CA US CN JP`.
- GAMS allows defining Sets and Parameters over the universal set:

```
parameter new(*)  'More data';
new('L') = 3;
```

  This would add `L` to the universal Set.

The **descriptive text** provided on declaration of Symbols or Set elements is stored in GDX files along with the data contained in those variables.

- For Set `v`, the string `"Valid animal colours"`.
- For Set element `o`, the string `"Orange"`.

1.3 Accessing data from GDX files

class gdx.**File**(*filename=''*, *lazy=True*, *implicit=True*, *skip={}*)

Load the file at *filename* into memory.

If *lazy* is `True` (default), then the data for GDX Parameters is not loaded until each individual parameter is first accessed; otherwise all parameters except those listed in *skip* (default: empty) are loaded immediately.

If *implicit* is `True` (default) then, for each dimension of any GDX Parameter declared over '*' (the universal set), an implicit set is constructed, containing only the labels appearing in the respective dimension of that parameter.

Note: For instance, the GAMS Parameter `foo(*,*,*)` is loaded as `foo(_foo_0, _foo_1,_foo_2)`, where `_foo_0` is an implicit set that contains only labels appearing along the first dimension of `foo`, etc. This workaround is essential for GDX files where `*` is large; otherwise, loading `foo` as declared raises `MemoryError`.

`File` is a subclass of `xarray.Dataset`. The GDX data is represented as follows:

- One-dimensional GDX Sets are stored as xray *coordinates*.
- GDX Parameters and multi-dimensional GDX Sets are stored as `xarray.DataArray` variables within the `xarray.Dataset`.
- Other information and metadata on GDX Symbols is stored as attributes of the *`File`*, or attributes of individual data variables or coordinates.

Individual Symbols are thus available in one of three ways:

1. As dict-like members of the `xarray.Dataset`; see the xarray documentation for further examples.

```
>>> from gdx import File
>>> f = File('example.gdx')
>>> f['myparam']
```

2. As attributes of the *`File`*:

```
>>> f.myparam
```

3. Using *`get_symbol_by_index()`*, using the numerical index of the Symbol within the GDX file.

dealias(*name*)
Identify the GDX Symbol that *name* refers to, and return the corresponding `xarray.DataArray`.

extract(*name*)
Extract the GAMS Symbol *name* from the dataset.

The Sets and Parameters in the *`File`* can be accessed directly, as e.g. *f['name']*; but for more complex xarray operations, such as concatenation and merging, this carries along sub-Sets and other Coordinates which confound xarray.

`extract()` returns a self-contained `xarray.DataArray` with the declared dimensions of the Symbol (and *only* those dimensions), which does not make reference to the *`File`*.

get_symbol_by_index(*index*)
Retrieve the GAMS Symbol from the *index*-th position of the *`File`*.

info(*name*)
Informal string representation of the Symbol with *name*.

parameters()
Return a list of all GDX Parameters.

set(*name*, *as_dict=False*)
Return the elements of GAMS Set *name*.

Because `xarray` stores non-null labels for each element of a coord, a GAMS sub-Set will contain some `''` elements, corresponding to elements of the parent Set which do not appear in *name*. *`set()`* returns the elements without these placeholders.

sets()
Return a list of all GDX Sets.

1.4 Internals

Most methods in the GDX API have similar semantics:

- Names are in CamelCase, e.g. gdxMethodName.
- A list is returned; the first element is a return code.

GDX hides these details, allowing for simpler code. Methods can be accessed using `call()`. For instance, the following code calls the API method gdxFileVersion:

```
>>> g = GDX()
>>> g.call('FileVersion')
```

Alternately, methods can be accessed as members of *GDX* objects, where the CamelCase API names are replaced by lowercase, with underscores separating words:

```
>>> g.file_version()   # same as above
```

See *GDX.__valid* for the list of supported methods.

class gdx.api.**GDX**
Wrapper around the GDX API.

__valid = None
Methods that conform to the semantics of *call()*.

call(*method*, **args*)
Invoke the GDX API method named gdx*Method*.

Optional positional arguments *args* are passed to the API method. Returns the result of the method call, with the return code stripped. Refer to the GDX API documentation for the type and number of arguments and return values for any method.

If the call fails, raise an appropriate exception.

gdx.api.**type_str = {<class 'GMS_DT_SET'>: 'set', <class 'GMS_DT_PAR'>: 'pa**
String representations of API constants for G(a)MS D(ata) T(ypes)

gdx.api.**vartype_str = {<class 'GMS_VARTYPE_UNKNOWN'>: 'unknown', <class 'GM**
String representations of API constants for G(a)MS VAR(iable) TYPE(s)

CHAPTER 2

License

PyGDX is provided under the MIT license.

CHAPTER 3

History

PyGDX was inspired by the similar package, also named py-gdx, by Geoff Leyland.

Python Module Index

g

Index

Symbols

C

D

E

F

G

I

P

S

T

V