Python Interface to GrADS

From OpenGrads Wiki

The Python interface to GrADS is an alternative method of scripting GrADS that can take advantage of the unique capabilities of Python

Contents

(http://en.wikipedia.org/wiki/Python), and gives you access to a wealth of numerical and scientific software available for this platform. Here are few reasons for scripting GrADS in Python:

- You are an experienced Python (http://en.wikipedia.org/wiki/Python) programmer new to GrADS and do not want to spend the time learning a new scripting language.
- You need some GrADS functionality inside your regular Python script, say, parse the contents of a GrADS readable dataset or want to store your metadata in a mySQL database.

- You want to query your OPeNDAP (http://www.opendap.org/index.html) server and figure out which is the latest forecast available before actually opening the dataset.
- You would like to use `TKinter` or any other toolkit to write a Graphical User Interface for GrADS from within Python.
- Your script is getting too complex and you could use an object oriented approach to better organize and reuse your code.
- You would like to explore GrADS ability to slice and dice a meteorological dataset or OpenDAP URL, but prefer to use Matplotlib/PyLab (http://matplotlib.sf.net/) to perform further analysis and visualization of your dataset.

The Python interface to GrADS, which is similar to the Perl interface, enables full scripting capability for GrADS in Python, and can be used together with the classic GrADS scripting language (http://grads.iges.org/grads/gadoc /script.html#intro).

Overview

The Python interface to GrADS is implemented in package `grads` which contains the following modules:

H500s bmarble.jpg

gacore
: The module `gacore` provides the basic GrADS client class which allows you to start GrADS, send commands to it and to retrieve the text output produced by GrADS in response to such command. This is a Pure Python module, although it requires the GrADS binaries to have been installed on your system.

ganum
: If you have NumPy installed, the module `ganum` will be loaded. This module extends the GrADS client class in `gacore` by providing methods for exchanging n-dimensional NumPy array data between Python and GrADS. It also provides methods for computing EOFs and least square estimation.

galab
: If PyLab/Matplotlib/Basemap is available, the module `galab` is loaded. This module adds Matplotlib/Basemap specific methods for contours, images and other graphical functionality. This class provides high level methods operating directly on GrADS expressions (or *fields*) while retaining all the configurability that Matplotlib has to offer.

gahandle
: This module provides a simple container class to collect output for `query()`

operations.

gacm

This modules provides additional colormaps, as well as an extension of the `Colormaps` class which allows for the definition of color look-up takes with an alpha channel. It also extends the `LinearSegmentedColormap` with the ability of create derived color tables which are either reversed or provide an arbitrary scaling by means of a lambda function.

numtypes

This module defines `GaField`, a class for representing GrADS variables in Python. It consists of a NumPy masked array with a `grid` containing coordinate/dimension information attached to it.

The grads Package at a Glance

The GrADS client class provided in `gacore` allows you to start any of the GrADS executables (`grads`, `gradsnc`, `gradshdf`, `gradsdods`, `gradsdap`), send commands to it, and retrieve its standard output. Here is a simple example

```
from grads.gacore import GaCore
ga = GaCore(Bin='gradsnc')
fh = ga.open("model.nc")
ga("display ps")
```

If you have NumPy installed, you can exchange array data between GrADS and python with module `ganum`,

```
from grads.ganum import GaNum
ga = GaNum(Bin='gradsnc')
ts = ga.exp("ts")  # export variable ts from GrADS
ts = ts - 273      # convert ts to Celsius
ga.imp("tc",ts)    # send the NumPy array to GrADS
ga("display tc")   # display the just imported variable
```

If you have `Matplotlib` installed you can do the plotting in python. For example you can produce a simple contour plot of the `tc` above with the commands.

```
from pylab import contourf
contourf(ts.grid.lon,ts.grid.lat,tc)
```

In addition, when the Matplotlib/Basemap toolkit is available the module `galab` provides methods to display your data with a map background, and provides a wealth of map transformations useful for display satellite imagery.

```
 from grads.galab import GaLab
 ga = GaLab(Bin='gradsnc')
 ga.blue_marble('on')
 ga("set lon -180 180)
 ga.contour('ua')
 title('Zonal Wind')
```

Indeed, the combination of GrADS/Matplotlib/PyLab is a very powerfull one. In addition, the numerical capabilities of NumPy allows one to extend the GrADS capabilties in areas such statistical data analysis. As a proof of concept we have implement a method of computing empirical ortoghonal functions. Here is a simple example:

```
ga.open("slp.nc")
ga("set t 1 41")
v, d, pc = ga.eof('slp')
```

where `v` contains the eigenvectors, `d` the eigenvalues and `pc` the principal components (EOF coefficients). These eigenvectors can be displayed with GrADS or Matplotlib. A method for computation of multiple regression is also available. For additional information, consult the `grads package` Reference Manual (http://opengrads.org/doc/python/grads).

IPython Based Interactive Shell

Ipy console.png

IPython (http://ipython.scipy.org/moin/) is an enhanced Python shell designed for efficient interactive work. It includes many enhancements over the default Python shell, including the ability for controlling interactively all major GUI toolkits in a non-blocking manner. The script `pygrads` is a wrapper script which starts IPython with a number of aliases and customizations convenient for interactive GrADS work. In particular, it starts PyLab bringing together all the GrADS and Matplotlib capabilities. Here is a sample `pygrads` session

```
 [] ga-> xx ts                 # export ts
 [] ga-> ts = ts - 273         # convert to Celsius
 [] ga-> c                     # clear the screen
 [] ga-> sh                    # shortcut for "set gxout shaded'
 [] ga-> dd ts                 # import modiefied ts into Grads and display it
 [] ga-> cb                    # add a color bar
 [] ga-> . draw title Surface Temperature (Celsius)
```

For additonal information on `pygrads` consult the PyGrADS Interactive Shell documentation.

Using PyGrADS with Jython and IronPython

Jython (http://www.jython.org/) is an implementation of the Python language written in 100% Pure Java. Jython scripts can access Java classes, the same way Java programs can access Jython classes. However, any Python extension written in C cannot be run under Jython. Although PyGrADS is written in 100% Python, some of its dependencies are not. For this reason only the `gacore` module works under Jython. It is conceivable that the JNumeric (http://jnumerical.sourceforge.net/) package could be used for implementing a Jython compatible version of `ganum`. The VisAD (http://www.ssec.wisc.edu/~billh/visad.html) Java component library could provide some of the display capabilities that Matplotlib offers under CPython.

As of this writing, `gacore` has been shown to run under Jython v2.2.1, with the very basic `jython` shell. Currently, the Jython development team is working on Jython 2.5, an upgrade that will likely permit the interactive shell IPython to run under it. A new package called JyGrADS, which is based on the PyGrADS sources, includes additional Java and Matlab classes allowing a GrADS interface from these languages. JyGrADS also includes a self contained **jar** (Java Archive) containing Jython, PyGrADS and support Java classes.

IronPython (http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython) is an implementation of the Python language running under Microsoft's .NET framework. As of this writing I am not aware of any attempt to run PyGrADS under IronPython, but I would be interested in hearing about it. Please drop a note at the open-discussion (http://sourceforge.net/forum/?group_id=161773) forum if you have tried PyGrADS under IronPython, or whether you would like to develop something similar to JyGrADS but for .NET.

Getting Started: Downloading and Installing

Requirements

For the basic functionality provided by module `gacore` you need the following

- Python (http://python.org) Version 2.3 or later, or Jython (http://www.jython.org/)
- GrADS (http://grads.iges.org/grads). Either
  - Version 1.9.0-rc1 or later, or any OpenGrADS release (http://sourceforge.net/project/showfiles.php?group_id=161773). It **does not** work with v1.9b4
  - Version 2.0.a3 or later. For previous versions of GrADS 2.0 only the basic class **GaCore** works.

If you would like GrADS to exchange array data with Python, module `ganum` requires

- For exporting n-dimensional arrays from GrADS you will need:
  - NumPy (http://numpy.scipy.org)
- For importing/exporting data from GrADS you will need in addition:
  - GrADS extension libipc (http://opengrads.org/doc/udxt/libipc/). See the User Defined Extensions documentation for information. This extension is included by default with every OpenGrADS release of GrADS.

For high quality graphics in Python, including map backgrounds and transformations,

- Matplotlib (http://matplotlib.sourceforge.net/)
- Basemap Tookit (http://www.scipy.org/Cookbook/Matplotlib/Maps) for Matplotlib

If you will be working with satellite images it is recommended that you install

- PIL (http://www.pythonware.com/products/pil/), the Python Imaging Library

The additional color tables provided by module `gacm` require both Matplotlib and PIL. And finally, it is *highly recommended* that you also install

- IPython (http://ipython.scipy.org/moin/)

for a more enjoyable interactive experience. Although not a requirement, the following package is highly recommended:

- SciPy - Scientific Tools or Python (http://www.scipy.org/)

These packages are available for most Linux distributions, MacOS X and Microsoft Windows, as well as in many flavors of Unix. Consult the Platform Specific Notes (http://opengrads.org/wiki/index.php?title=Python_Interface_to_GrADS#Platform_Specific_Notes) for additional information.

When you import package `grads` the following attributes are defined, depending on which of the modules were successfully imported: HAS_GALAB, HAS_GANUM, HAS_GACM. An `ImportError` exception will be raised if `gacore` cannot be imported. Therefore, provided you have Python or Jython installed you will always have the functionality to start GrADS and interact with it. Whether you will have the extra functionality to exchange array data with GrADS and do your plotting in Python will depend on the other packages you have installed on your python environment. When you import the package `grads`, the class `GrADS` is aliaded to `GaLab` if `galab` can be imported, otherwise it is aliased to `GaNum`. Failing that, `GrADS` is aliased to `GaCore`. Use the HAS_* attributes to determine which functionality is available to you.

Downloading the software and sample datasets

The PyGrADS modules can be downloaded from the OpenGrADS download (http://sourceforge.net/project/showfiles.php?group_id=161773) area at SourceForge. Examples datasets and satellite images are included in the tarball.

VERY IMPORTANT

You will need at least PyGrADS 1.1.0 (released in the Summer of 2008) for Matplotlib v0.98.1 and later. Use Matplotlib v0.91.x with PyGrADS v1.0.8 and earlier.

Installation

The official Python installation guide can be found at Installing Python Modules (http://docs.python.org/inst/inst.html). A brief summary is presented here.

Microsoft Windows

There are 3 Win32 related packages in the download area:

a) pygrads-x.y.z.win32_superpack.exe
: This is your best choice if you have no Python whatsoever installed on your Windows box and would like to get started with Python and PyGrADS. This self installing package has Python 2.5 proper, as well as PyGrADS itself and all the dependencies you need for a fully functional PyGrADS. Installation of each package is optional, so you can use this installation option if you need to install just some of the required packages. **Important**: If you install "basemaps" be sure to install "httplib2" as well, also included in the superpack; at this point the installer does not enforce this.

b) pygrads-x.y.z.win32.exe
: Now, if you have Python 2.5 and the other required packages already installed on your Windows box, you can install *PyGrADS only* with this self installing file. No dependencies are provided.

c) pygrads-x.y.z.tar.gz
: Or else, get this tarball and do a standard python install (see next subsection). Note that PyGrADS is 100% pure Python, so no compilation is necessary. Again, you will need the dependencies for full functionality, though.

Linux, Mac OS X and Unix

a) If you have administrative privileges

Install it the usual way

```
% tar xvfz pygrads-x.y.z.tar.gz
% cd pygrads-x.y.z
% python setup.py install
```

b) As a regular user

Install it anywhere you have write permission, using the options `--prefix` or `--home` to specify the directory base name:

```
% python setup.py install --prefix=/path/to/some/dir
```

setting the environment variable `PYTHONPATH` to point to the location of your Python scripts. For example, to install it off your home directory:

```
% python setup.py install --home=$HOME
```

In this case, two directories will be created (if they do not exist already), and the PyGraDS files installed there:

```
$HOME/bin
    pygrads
$HOME/lib/python2.5/site-packages
    grads/
    ipygrads.py
    etc.
```

(The `python2.5` in the directory name above may vary from system to system depending on your installed Python version.) For this example, you will need set the following environment variables; if using `bash` and variants:

```
% export PATH=$HOME/bin:$PATH
% export PYTHONPATH=$HOME/lib/python2.5/site-packages:$PYTHONPATH
```

or with the `csh` and variants:

```
% setenv PATH       $HOME/bin:$PATH
% setenv PYTHONPATH $HOME/lib/python2.5/site-packages:$PYTHONPATH
```

(Do not include `$PYTHONPATH` after the `:` if it complains that `$PYTHONPATH` is not defined.)

For additional information,

```
 % python setup.py --help
```

Checking your Installation

a) First things first: is GrADS working?

Before everything make sure you have a functioning GrADS installation:

```
% cd data
% grads -u
ga> sdfopen model.nc
ga> display ts
```

You should see a contour plot of surface temperature. If you intend to import data back into GrADS using GaNum, make sure the IPC extension is functioning:

```
ga> q udct
```

This should list:

```
  ipc_verb ---> cmd_Verb() from <libipc.gex.so>
  ipc_open ---> cmd_Open() from <libipc.gex.so>
 ipc_close ---> cmd_Close() from <libipc.gex.so>
  ipc_save ---> cmd_Save() from <libipc.gex.so>
ipc_define ---> cmd_Define() from <libipc.gex.so>
 ipc_error ---> cmd_Error() from <libipc.gex.so>
```

If not, make sure you have installed the IPC extension and that your `GAUDXT` environment variable is set properly. To actually test the IPC extension type

```
ga> ipc_save ts ts.bin
```

This should create a binary file called `ts.bin` of size 14 kilobytes. If ipc_save cannot find `libipc.gex.so`, make sure you have set your `LD_LIBRARY_PATH` environment variable appropriately. For more information consult the User Defined Extensions documentation.

b) Running the examples

First check the core functionality:

```
 % cd examples
 % ./gacore_examples.py | grep OK
```

If you have NumPy installed you can check your installation by running the `ganum_examples`

```
 % ./ganum_examples.py          # will produce a bunch of PNG images
 % animate -delay 300 *.png     # you can use ImageMagick to look at them
```

Likewise, if you have Matplotlib with the basemaps installed you can run the `galab_examples`

```
% rm *.png                      # remove images from previous examples
% ./galab_examples.py           # will produce a bunch of PNG images
% animate -delay 300 *.png      # you can use ImageMagick animate to look at them
```

The output of these examples can be found at GaNum Examples and GaLab Examples.

Win32 Tip: You need to download the tarball `pygrads-x.x.x.tar.gz` for getting the example files. Once you untar the this file, open a `cmd.exe` window and type something like this:

```
cd \some\path\pygrads-x.x.x\examples
c:\python25\python ganum_examples.py
c:\python25\python galab_examples.py
```

taking a look at the output PNG files. You can also open the `examples` folder with *Windows Explorer* and click on `galab_examples` to run it.

Examples and Tutorials

- Examples:
  - GaCore Examples: Basic functionality
  - GaNum Examples: Exchanging data with Python and some basic numerical computations
  - GaLab Examples: Matplotlib/Basemap related examples
- PyGrADS Interactive Shell
- GrADS Tutorial Using PyGrads
- The PyGrADS Cookbook

Additional References

- Python Tutorial. (http://docs.python.org/tut/) Start here is you are new to Python.
- The Programming Python (http://en.wikibooks.org/wiki/Programming:Python)

Wikibook is also a good starting point, with lots of additional references.

- The `grads` Python Package Reference Manual (http://opengrads.org /doc/python/grads). Here you will find the doc strings for all the available classes.
- PyGrADS Interactive Shell documentation
- Matplotlib Tutorial (http://matplotlib.sourceforge.net/tutorial.html) is a good reference for getting started with the Matlab-like features in Python. In particular look at the Documentation links on this page.
- Matplotlib User's Guide (http://matplotlib.sourceforge.net /users_guide_0.91.2svn.pdf) in pdf, a more in-depth introduction to Matplotlib.

Platform Specific Notes

These section summarizes the experience of users installing PyGrADS on different platforms. Please help us keep this page complete and up to date. Drop us a note at the Open Discussion (http://sourceforge.net/forum /forum.php?thread_id=2028765&forum_id=547756) forum if you would like to make a contribution.

Microsoft Windows

Your best option is to install the PyGrADS Superpack. During installation you have the chance to install Python itself and all the dependencies or only the dependencies you don't have. Remember that the PyGrADS Superpack does not include the Win32 GrADS binaries, you will need to download and install it separately from Sourceforge (http://sourceforge.net/project /showfiles.php?group_id=161773&package_id=270750).

Mac OS X

Arlindo da Silva has had good luck with the Mac OS X binaries from svn from Chris Fonnesbeck's page (http://macinscience.org).

- Make sure you are using OSX 10.5 Leopard's preinstalled Python 2.5.1, ActivePython 2.5 or MacPython 2.5. Note: The Superpack's version detection may fail with other Python distributions (e.g., fink, Darwin Ports), and it will refuse to install.
- Download the SciPy Superpack for Python 2.5 (http://macinscience.org /?page_id=6)
- NumPy is included in the Superpack. For best compatibility, make sure you use the version in the Superpack.
- Note that the Chris Fonnesbeck's Superpacks are based on recent SVN code, and not the latest official release.

Important Update: Quoting from Chris Fonnesbeck site:

A recent post on the Enthought Blog (http://blog.enthought.com/?p=43) announces the availability of the Enthought Python Distribution (EPD) for OSX. Previously only available on Windows, the EPD is a "batteries-included" distribution of Python, geared toward scientific applications. This distro includes the following essentials:

- Python - Core Python
- NumPy - Multidimensional arrays and fast numerics for Python
- SciPy - Scientific Library for Python
- Enthought Tool Suite (ETS) - A suite of tools including: **Traits** - Manifest typing, validation, visualization, delegation, etc. **Mayavi** - 3D interactive data exploration environment. **Chaco** - Advanced 2D plotting toolkit for interactive 2D visualization. **Kiva** - 2D drawing library in the spirit of DisplayPDF. **Enable** - Object-based canvas for interacting with 2D components and widgets.
- Matplotlib - 2D plotting library
- wxPython - Cross-platform windowing and widget library.
- Visualization Toolkit (VTK) - 3D visualization framework

These sorts of bundles are very attractive for scientists that would rather not invest the time in compiling each of these packages from scratch, particularly in the case of the visualisation packages, which can be rather fussy to build.

Red Hat Enterprise Linux/Cent OS

Extra Packages for Enterprise Linux (EPEL) is a volunteer-based community effort from the Fedora project to create a repository of high-quality add-on packages that complement the Fedora-based Red Hat Enterprise Linux (RHEL) and its compatible spinoffs such as CentOS or Scientific Linux. RPMS can be found at this URL

```
http://fedoraproject.org/wiki/EPEL
```

The following packages are needed by PyGrADS:

- python-matplotlib
- python-matplotlib-tk
- python-basemap
- python-basemap-data

I have not found *ipython* and the *Python Imaging Library* (PIL) on the EPEL repository. However, Dag Wieers maintains another repository (http://dag.wieers.com/rpm) where you can find these packages:

- ipython
- python-imaging

(I am not sure if you can combine packages from these repositories; let me know either way so that I can update this page; dasilva@opengrads.org)

Ubuntu

Numpy, Matplotlib and PIL (Python Imaging Library) are available through the Synaptic Package Manager. However, the required Basemap package is not, and you will need to build it from sources (http://sourceforge.net/project/showfiles.php?group_id=80706&package_id=142792). Make sure your Basemap version is compatible with your Numpy/Matplotlib: you'll need Basemap 0.99 to use with Matplotlib 0.98; with Ubuntu 8.04 and earlier you will need Basemap v0.98 since it uses an earlier Matplotlib. Before installing Basemap from sources, make sure to install the `libgeos` package from Synaptic (or using `apt-get` at the command line) or else build it from sources - it is bundled with the Basemap sources.

FreeBSD

The **Ports** subsystem has all the required dependencies. Make sure to install the following packages:

- py-matplotlib
- py-basemap
- py-basemap-data
- py-imaging

Under *Desktop BSD* you can use the Package Manager under the KDE [Main Menu]/[System]/[Software Management] to search for and install these packages.

Retrieved from "http://opengrads.org/wiki/index.php?title=Python_Interface_to_GrADS&oldid=930"

- This page was last modified on 4 February 2016, at 08:55.