Analyzing rasters, vectors and time series using new Python interfaces in GRASS GIS 7

Václav Petráš[1], Anna Petrášová[1], Yann Chemin[2], Pietro Zambelli[3], Martin Landa[4], Sören Gebbert[5], Markus Neteler[6], and Peter Löwe[7]

[1]North Carolina State University, Raleigh, USA (wenzeslaus@gmail.com, vpetras@ncsu.edu), [2]International Water Management Institute, Pelawatta, Sri Lanka, [3]EURAC Research, Institute for Renewable Energy, Bolzano/Bozen, Italy, [4]Faculty of Civil Engineering, Czech Technical University in Prague, Czech Republic, [5]Thünen Institute of Climate-Smart Agriculture, Braunschweig, Germany, [6]Research and Innovation Centre, Fondazione Edmund Mach, San Michele all'Adige, Italy, [7]TIB Hannover - German National Library of Science and Technology, Hanover, Germany

GRASS GIS

Highlights

GRASS GIS [1] is a platform for geospatial computations.

The functionality is divided into a set of modules (individual tools, functions, algorithms or models).

Core libraries and algorithms implemented in C for high performance.

Both the modules and the libraries are accessible through the Python API.

Specialized Python APIs support different use cases ranging from high level scripting to fine data editing.

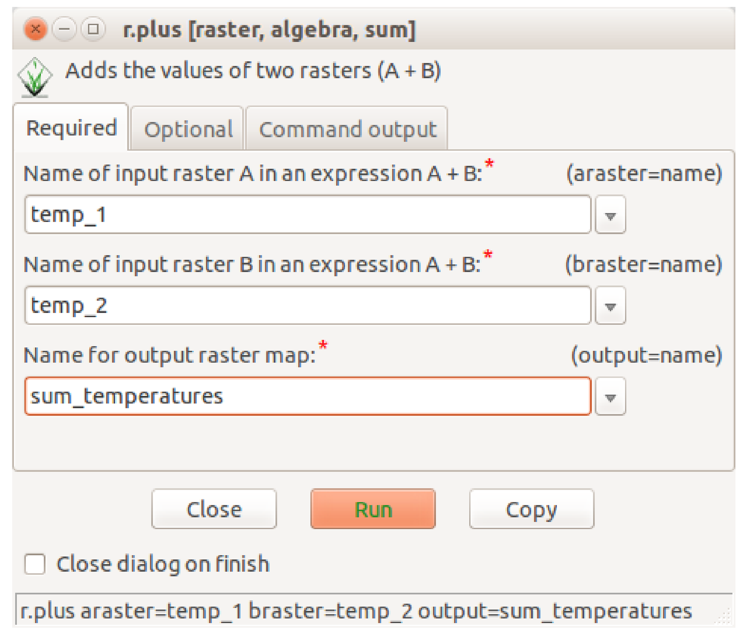
It is simple to build graphical user interface.

NumPy or IPython can be used together with GRASS GIS Python APIs.

Automatic creation of GUI and CLI

The *g.parser* module provides full interface definition support for Python scripts including creation of standardized part of a help page and command line checking. Each script which uses the GRASS GIS parser can publish definition of its parameters (options and flags) in XML. The GRASS GIS graphical user interface is able to use the XML to dynamically generate an interactive graphical dialog with unified styling.

```
#%module
#% description: Adds the values of two rasters (A + B)
#% keywords: raster
#% keywords: algebra
#% keywords: sum
#%end
#%option G_OPT_R_INPUT
#% key: araster
#% description: Raster A in the expression A + B
#%end
#%option G_OPT_R_INPUT
#% key: braster
#% description: Raster B in the expression A + B
#%end
#%option G_OPT_R_OUTPUT
#%end
```




```
Description:
 Adds the values of two rasters (A + B)

Keywords:
 raster, algebra, sum

Usage:
 r.plus araster=name braster=name output=name
    [--overwrite] [--help] [--verbose]

Flags:
 --o    Allow output files to overwrite existing files
 --h    Print usage summary
 --v    Verbose module output
 --q    Quiet module output
 --ui   Force launching GUI dialog

Parameters:
  araster   Name of input raster A in an expression A + B
  braster   Name of input raster B in an expression A + B
   output   Name for output raster map
```

GRASS Scripting Library interface to GRASS GIS modules

The *grass.script* package offers simple and straightforward syntax to call GRASS GIS modules:

```
run_command('r.neighbors', input='elevation',
            output='elevation_smooth',
            method='median', flags='c')
```

PyGRASS interface to GRASS GIS modules

The *grass.pygrass* package provides a object-oriented way to work with GRASS GIS modules and their parameters:

```
r_neighbors = Module('r.neighbors', input='elevation',
                     output='elevation_smooth',
                     method='median', flags='c')
# get result with alternative method
r_neighbors.inputs.method = 'mode'
r_neighbors.outputs.output = 'elevation_smooth_2'
r_neighbors.run()
```

Additionally, the *grass.pygrass* package offers simpler, Python oriented, way of accessing modules:

```
r_neighbors = r.neighbors(
    input='elevation', output='elevation_smooth',
    method='median', flags='c')
```

Using documentation for GRASS GIS modules

Documentation of GRASS GIS modules usually use Bash syntax to provide an example of usage, e.g.:

```
r.neighbors input=elevation output=elevation_smooth method=median -c
```

This syntax can be easily rewritten to the *grass.script* syntax or the *grass.pygrass* syntax shown above.

IPython Notebook

IPython Notebook is a web-based tool to develop, document, and execute code, as well as communicate the results. In combination with GRASS GIS, it is an excellent tool to process and visualize your geospatial data, integrate formulas, explanatory text and maps, and interact with remote servers and clusters.

PyGRASS interface to C libraries

Examples of using the API to convert a raster map to a NumPy array and back to raster:

```
import numpy as np
from grass.pygrass.raster import RasterRow
from grass.pygrass.raster.buffer import Buffer
from grass.pygrass.gis.region import Region

def raster2numpy(name, mapset):
    """Return a numpy array from a raster map"""
    with RasterRow(name, mapset, mode='r') as array:
        return np.array(array)

def numpy2raster(array, mtype, name):
    """Save a numpy array to a raster map"""
    reg = Region()
    if (reg.rows, reg.cols) != array.shape:
        msg = "Region and array shape are different: %r != %r"
        raise TypeError(msg % ((reg.rows, reg.cols), array.shape))
    with RasterRow(name, mode='w', mtype=mtype) as new:
        newrow = Buffer((array.shape[1],), mtype=mtype)
        for row in array:
            newrow[:] = row[:]  # cast array to raster type
            new.put_row(newrow)  # write row to raster map
# note that there is a specialized package which implements this functionality
```

In addition to PyGRASS interface, advanced programmers can use *ctypes* interface to access C functions from GRASS GIS libraries directly.

Testing the algorithms

To show that all promised functionality is available and algorithm works as expected every module should be supplied with a test. This also ensures that functionality can be simply tested any time in the future [3].

```
# run the series interpolation module in a highly controlled way
self.assertModule('r.series.interp',
                  input=['prec_1', 'prec_5'], datapos=(0.0, 1.0),
                  output=['prec_2', 'prec_3', 'prec_4'],
                  samplingpos=(0.25, 0.5, 0.75), method='linear')
# check the interpolated raster prec_2 for minimum and maximum (same here)
self.assertRasterMinMax(map='prec_2', refmin=200, refmax=200)
```

Tests can use standardize datasets or use custom reference data. It is very easy to write a sophisticated test just by checking a statistical summary of computation results.

GRASS GIS modules, addons and Python scripts

A Python script can by turned into a GRASS GIS module by adding a definition of the interface using GRASS GIS parser mechanism. Another difference is that GRASS GIS modules expect to be executed in GRASS GIS session. Python scripts which are using GRASS GIS can be written in a way that GRASS GIS session is not required; the setup of a necessary environment is done in the script itself in this case.

The GRASS GIS Addon repository contains modules from wide range of contributors and ensures vendor-independent long-term preservation of the code and ensures an easy distribution of module to individual users. Well maintained modules in Addons can be moved to GRASS GIS core.

Alternatives to Python

GRASS GIS API is not limited only to Python. GRASS GIS modules are command line tools, so they can be used in shell scripting (e.g. Bash) and as subprocesses in virtually any language as long as the proper environment is set. The GRASS GIS library provides a C API which is commonly used to create GRASS GIS modules in C and C++ programming languages. Finally, GRASS GIS modules can be used within R statistical environment using the *rgrass7* package.

GRASS GIS Temporal Framework

The GRASS GIS Temporal Framework implements the temporal GIS functionality and provides a Python API to implement spatio-temporal processing modules. The framework introduces space-time datasets that represent time series of raster, 3D raster or vector maps. An API example:

```
# Import and initialize the GRASS GIS temporal framework
import grass.temporal as tgis
tgis.init()
# Open an existing space time raster dataset (STRDS)
temp_strds = tgis.open_old_stds(name="temp_daily",
                                type="strds")
# Shift all registered raster map layer 2 days in the future
temp_strds.shift("2 days")
# Open an exitsing time stamped raster map as map object
temp_24Mar77 = tgis.RasterDataset('temp_1977_Mar_24@soeren')
# Unregister the raster map from the STRDS
temp_strds.unregister_map(temp_24Mar77)
# Get a list of all registered raster map layers with
# a start time later 1990-05-22 from the STRDS
maps = temp_strds.get_registered_maps_as_objects(\
                 where="start_time > '1990-05-22'")
# Create a temporal buffer of 6h for each map in the list
for map in maps:
    map.temporal_buffer("6 hours", update=True)
# Compute new spatio-temporal extent and granularity
temp_strds.update_from_registered_maps()
# Get the granuarity of the STRDS
gran = temp_strds.get_granularity()
```

References and Acknowledgements

[1] Neteler, M., Bowman, M. H., Landa, M., Metz, M., 2012. *GRASS GIS: A multi-purpose open source GIS.* Environmental Modelling & Software, 31, 124–130.

[2] Gebbert, S., Pebesma, E., 2014. A temporal GIS for field based environmental modeling. Environmental Modelling & Software 53, 1–12.

[3] Petras, V., Gebbert, S., 2014. Testing framework for GRASS GIS: ensuring reproducibility of scientific geospatial computing. Poster presented at: AGU Fall Meeting, December 15-19, 2014, San Francisco, USA.

[4] Zambelli, P., Gebbert, S., Ciolli, M., 2013. Pygrass: An Object Oriented Python Application Programming Interface (API) for Geographic Resources Analysis Support System (GRASS) Geographic Information System (GIS). ISPRS International Journal of Geo-Information 2, 201–219.

Acknowledgements

GRASS GIS is a OSGeo project. OSGeo provides infrastructure for project websites, mailing lists and source code management.

Initial development of *pygrass* and *gunittest* packages was done during Google Summer of Code 2012 and 2014.

Luca Delucchi, Italy, contributed significantly to development of Python interfaces for GRASS GIS 7 through extensive testing in early stages of development, documenting the APIs and general contributions to the code itself.

More information

GRASS GIS website
`grass.osgeo.org`

GRASS GIS Python libraries documentation
`grass.osgeo.org/grass71/manuals`

LaTeX TikZposter