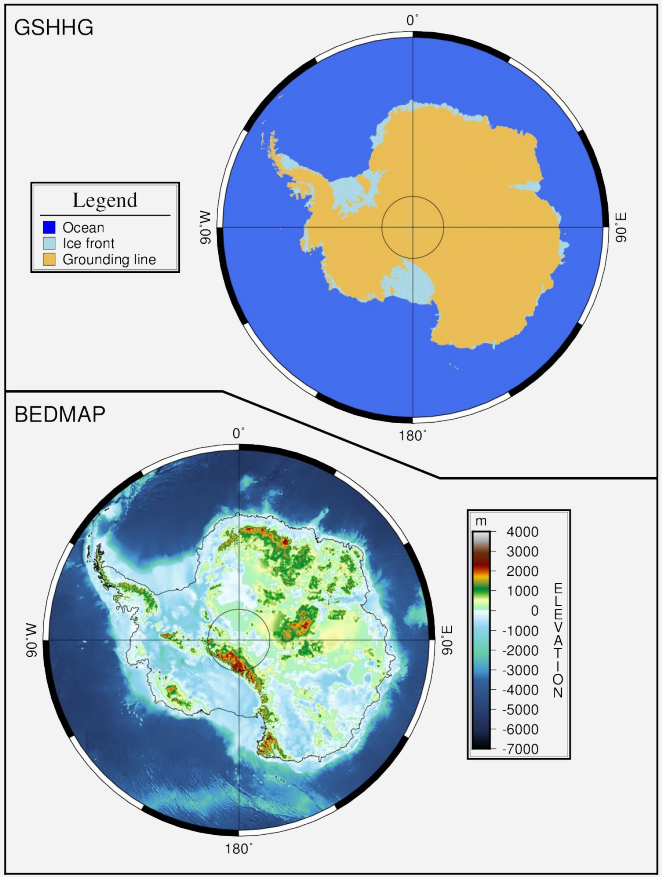
Building an Object-Oriented Python interface for the Generic Mapping Tools

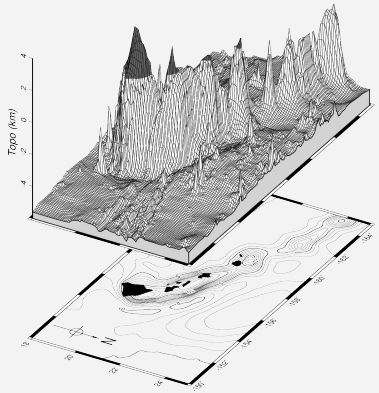
Leonardo Uieda and Paul Wessel
@leouieda
Department of Geology and Geophysics, SOEST, University of Hawaii at Manoa

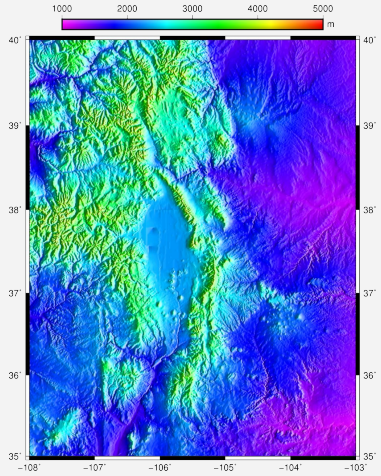
Scipy2018 - 2018/07/13

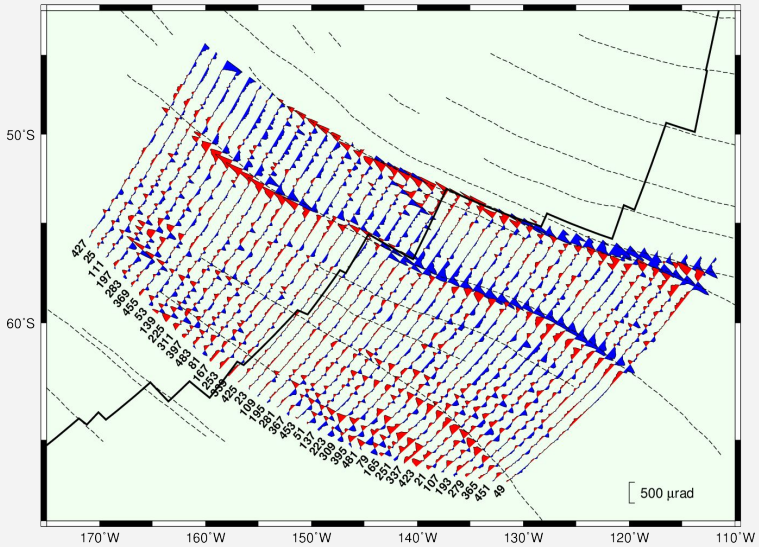
GMT

HAWAIIAN RIDGE









- BC
  - blockmean
  - blockmedian
  - blockmode
- EF
  - filter1d
  - fitcircle
- I
  - isogmt
- K
  - kml2gmt
- MN
  - makecpt
  - mapproject
  - movie
  - nearneighbor
- P
  - postscriptlight
  - project
  - psconvert
- S
  - sample1d
  - spectrum1d
  - sph2grd
  - sphdistance
  - sphinterpolate
  - sphtriangulate
  - splitxyz
  - surface
- T
  - trend1d
  - trend2d
  - triangulate
- X
  - xyz2grd

- GM
  - gmt.conf
  - gmt
  - gmt2kml
  - gmt5syntax
  - gmtcolors
  - connect
  - convert
  - defaults
  - get
  - info
  - logo
  - math
  - regress
  - select
  - set
  - gmt_shell_functions.sh
  - simplify
  - spatial
  - gmtswitch
  - vector
  - which

- GR
  - greenspline
  - grd2cpt
  - grd2kml
  - grd2xyz
  - grdblend
  - grdclip
  - grdconvert
  - grdcut
  - grdedit
  - grdfft
  - grdfill
  - grdfilter
  - grdgradient
  - grdhisteq
  - grdinfo
  - grdlandmask
  - grdmask
  - grdmath
  - grdpaste
  - grdproject
  - grdsample
  - grdtrack
  - grdtrend
  - grdvolume

- Mapping
  - basemap
  - clip
  - coast
  - colorbar
  - contour
  - grdimage
  - grdcontour
  - grdvector
  - grdview
  - histogram
  - image
  - legend
  - mask
  - plot
  - plot3d
  - rose
  - solar
  - ternary
  - text
  - wiggle

- GSHHG
  - gshhg
- IMGSRC
  - img2grd
  - img2google
- MISC
  - dimfilter

- MECA
  - coupe
  - meca
  - polar
  - velo
  - sac

- POTENTIAL
  - earthtide
  - gmtgravmag3d
  - gmtflexure
  - gpsgridder
  - gravfft
  - grdflexure
  - grdgravmag3d
  - grdredpol
  - grdseamount
  - talwani2d
  - talwani3d

- SEGY
  - segy
  - segyz
  - segy2grd

- MGD77
  - mgd77convert
  - mgd77header
  - mgd77info
  - mgd77list
  - mgd77magref
  - mgd77manage
  - mgd77path
  - mgd77sniffer
  - mgd77track

- SPOTTER
  - backtracker
  - gmtpmodeler
  - grdpmodeler
  - grdrotater
  - grdspotter
  - hotspotter
  - originater
  - polespotter
  - rotconverter
  - rotsmoother

- X2SYS
  - x2sys_binlist
  - x2sys_cross
  - x2sys_datalist
  - x2sys_get
  - x2sys_init
  - x2sys_list
  - x2sys_merge
  - x2sys_put
  - x2sys_report
  - x2sys_solve

Project Goals

Make GMT more accessible

Python wrapper that looks like Python

Integration with the Scipy stack

Comprehensive documentation

Demo

try.gmtpython.xyz

After Scipy 2017:

- Mac conda packages for GMT (Mike Hearne, Filipe Fernandes)
- gmt.Figure class
- Figure.savefig -> pytest-mpl
- Input numpy arrays, pandas Series, and xarray DataArray
- gmt-plot:: sphinx extension
- Many changes and fixes upstream

Challenges

```
#!/bin/bash

gmt begin
    gmt which @hotspots.txt -Gl
    gmt figure hotspots png
    gmt grdimage @earth_relief_10m -JG200/30/6i -Cgeo -I+
    gmt coast -W -Dc -Bafg
    gmt plot hotspots.txt -Sc0.2c -Gred
gmt end
```

NORMAL +0 ~0 -0 master command-line-example.sh[+]

File Edit View Run Kernel Tabs Settings Help

Launcher | command-line-

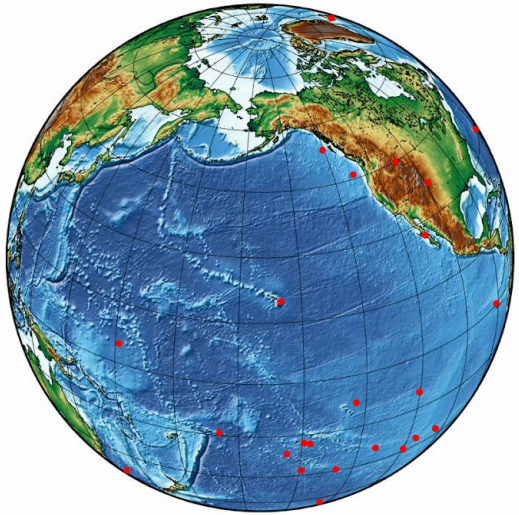
Code

In [9]:
```
import gmt
import numpy as np
```

In [10]:
```
lon, lat = np.loadtxt("hotspots.txt", usecols=[0, 1], unpack=True)
```

In [11]:
```
fig = gmt.Figure()
fig.grdimage("@earth_relief_10m", projection="G200/30/6i", cmap="geo", shading=True)
fig.coast(shorelines=True, frame="afg")
fig.plot(x=lon, y=lat, style="c0.2c", color="red")
fig.show()
```

Out[11]:



In [ ]:

Keeping
“conda install gmt”
alive.

Success

context managers

In [13]:
```
grid = gmt.datasets.load_earth_relief()
grid
```

Out[13]:
```
<xarray.DataArray 'z' (lat: 181, lon: 361)>
array([[ 2762.,  2762.,  2762., ...,  2762.,  2762.,  2762.],
       [ 2983.,  2980.,  2977., ...,  2989.,  2986.,  2983.],
       [ 3074.,  3074.,  3074., ...,  3072.,  3073.,  3074.],
       ...,
       [-3727., -3715., -3706., ..., -3759., -3742., -3727.],
       [-2294., -2282., -2271., ..., -2322., -2308., -2294.],
       [-4181., -4181., -4181., ..., -4181., -4181., -4181.]], dtype=float32)
Coordinates:
  * lon      (lon) float64 -180.0 -179.0 -178.0 -177.0 -176.0 -175.0 -174.0 ...
  * lat      (lat) float64 -90.0 -89.0 -88.0 -87.0 -86.0 -85.0 -84.0 -83.0 ...
Attributes:
    long_name:     z
    actual_range:  [-8425.  5551.]
```

In [19]:
```
with gmt.clib.Session() as ses:
    with ses.virtualfile_from_grid(grid) as f_in:
        print(f_in)
        with gmt.helpers.GMTTempFile() as f_out:
            ses.call_module("grdinfo", "{} ->{}".format(f_in, f_out.name))
            print("\n", f_out.read(), sep="")
```

```
@GMTAPI@-000000

: Title:
: Command:
: Remark:
: Gridline node registration used [Cartesian grid]
: Unrecognized grid file format! Probably not a GMT grid
: x_min: -180 x_max: 180 x_inc: 1 name: x n_columns: 361
: y_min: -90 y_max: 90 y_inc: 1 name: y n_rows: 181
: z_min: -8425 z_max: 5551 name: z
: scale_factor: 1 add_offset: 0
```

In [19]:
```
with gmt.clib.Session() as ses:
    with ses.virtualfile_from_grid(grid) as f_in:
        print(f_in)
        with gmt.helpers.GMTTempFile() as f_out:
            ses.call_module("grdinfo", "{} ->{}".format(f_in, f_out.name))
            print("\n", f_out.read(), sep="")
```

```
@GMTAPI@-000000

: Title:
: Command:
: Remark:
: Gridline node registration used [Cartesian grid]
: Unrecognized grid file format! Probably not a GMT grid
: x_min: -180 x_max: 180 x_inc: 1 name: x n_columns: 361
: y_min: -90 y_max: 90 y_inc: 1 name: y n_rows: 181
: z_min: -8425 z_max: 5551 name: z
: scale_factor: 1 add_offset: 0
```

In [19]:
```
with gmt.clib.Session() as ses:
    with ses.virtualfile_from_grid(grid) as f_in:
        print(f_in)
        with gmt.helpers.GMTTempFile() as f_out:
            ses.call_module("grdinfo", "{} ->{}".format(f_in, f_out.name))
            print("\n", f_out.read(), sep="")
```

```
@GMTAPI@-000000

: Title:
: Command:
: Remark:
: Gridline node registration used [Cartesian grid]
: Unrecognized grid file format! Probably not a GMT grid
: x_min: -180 x_max: 180 x_inc: 1 name: x n_columns: 361
: y_min: -90 y_max: 90 y_inc: 1 name: y n_rows: 181
: z_min: -8425 z_max: 5551 name: z
: scale_factor: 1 add_offset: 0
```

In [19]:
```
with gmt.clib.Session() as ses:
    with ses.virtualfile_from_grid(grid) as f_in:
        print(f_in)
        with gmt.helpers.GMTTempFile() as f_out:
            ses.call_module("grdinfo", "{} ->{}".format(f_in, f_out.name))
            print("\n", f_out.read(), sep="")
```

```
@GMTAPI@-000000

: Title:
: Command:
: Remark:
: Gridline node registration used [Cartesian grid]
: Unrecognized grid file format! Probably not a GMT grid
: x_min: -180 x_max: 180 x_inc: 1 name: x n_columns: 361
: y_min: -90 y_max: 90 y_inc: 1 name: y n_rows: 181
: z_min: -8425 z_max: 5551 name: z
: scale_factor: 1 add_offset: 0
```

In [19]:
```
with gmt.clib.Session() as ses:
    with ses.virtualfile_from_grid(grid) as f_in:
        print(f_in)
        with gmt.helpers.GMTTempFile() as f_out:
            ses.call_module("grdinfo", "{} ->{}".format(f_in, f_out.name))
            print("\n", f_out.read(), sep="")
```

```
@GMTAPI@-000000

: Title:
: Command:
: Remark:
: Gridline node registration used [Cartesian grid]
: Unrecognized grid file format! Probably not a GMT grid
: x_min: -180 x_max: 180 x_inc: 1 name: x n_columns: 361
: y_min: -90 y_max: 90 y_inc: 1 name: y n_rows: 181
: z_min: -8425 z_max: 5551 name: z
: scale_factor: 1 add_offset: 0
```

numpy + ctypes

```
@ctp.CFUNCTYPE(ctp.c_int, ctp.c_void_p, ctp.c_char_p)
def print_func(file_pointer, message):  # pylint: disable=unused-argument
    """
    Callback function that the GMT C API will use to print log and error
    messages. We'll capture the messages and print them to stderr so that they
    will show up on the Jupyter notebook.
    """
    message = message.decode().strip()
    self._error_log.append(message)
    # flush to make sure the messages are printed even if we have a crash.
    print(message, file=sys.stderr, flush=True)
    return 0

# Need to store a copy of the function because ctypes doesn't and it will be
# garbage collected otherwise
self._print_callback = print_func

padding = self["GMT_PAD_DEFAULT"]
session_type = self["GMT_SESSION_EXTERNAL"]
session = c_create_session(name.encode(), padding, session_type, print_func)
```

```
@ctp.CFUNCTYPE(ctp.c_int, ctp.c_void_p, ctp.c_char_p)
def print_func(file_pointer, message):  # pylint: disable=unused-argument
    """
    Callback function that the GMT C API will use to print log and error
    messages. We'll capture the messages and print them to stderr so that they
    will show up on the Jupyter notebook.
    """
    message = message.decode().strip()
    self._error_log.append(message)
    # flush to make sure the messages are printed even if we have a crash.
    print(message, file=sys.stderr, flush=True)
    return 0
```

```
# Need to store a copy of the function because ctypes doesn't and it will be
# garbage collected otherwise
self._print_callback = print_func

padding = self["GMT_PAD_DEFAULT"]
session_type = self["GMT_SESSION_EXTERNAL"]
session = c_create_session(name.encode(), padding, session_type, print_func)
```

```
@ctp.CFUNCTYPE(ctp.c_int, ctp.c_void_p, ctp.c_char_p)
def print_func(file_pointer, message):  # pylint: disable=unused-argument
    """
    Callback function that the GMT C API will use to print log and error
    messages. We'll capture the messages and print them to stderr so that they
    will show up on the Jupyter notebook.
    """
    message = message.decode().strip()
    self._error_log.append(message)
    # flush to make sure the messages are printed even if we have a crash.
    print(message, file=sys.stderr, flush=True)
    return 0

# Need to store a copy of the function because ctypes doesn't and it will be
# garbage collected otherwise
self._print_callback = print_func

padding = self["GMT_PAD_DEFAULT"]
session_type = self["GMT_SESSION_EXTERNAL"]
session = c_create_session(name.encode(), padding, session_type, print_func)
```

```python
@ctp.CFUNCTYPE(ctp.c_int, ctp.c_void_p, ctp.c_char_p)
def print_func(file_pointer, message):  # pylint: disable=unused-argument
    """
    Callback function that the GMT C API will use to print log and error
    messages. We'll capture the messages and print them to stderr so that they
    will show up on the Jupyter notebook.
    """
    message = message.decode().strip()
    self._error_log.append(message)
    # flush to make sure the messages are printed even if we have a crash.
    print(message, file=sys.stderr, flush=True)
    return 0

# Need to store a copy of the function because ctypes doesn't and it will be
# garbage collected otherwise
self._print_callback = print_func

padding = self["GMT_PAD_DEFAULT"]
session_type = self["GMT_SESSION_EXTERNAL"]
session = c_create_session(name.encode(), padding, session_type, print_func)
```

Conclusion

Working out kinks in the GMT API

Experimenting with the Python API

Improvements to docs (gmt-plot, sphinx-gallery)

Help with conda-forge Windows build

GMT API is more robust (but still needs work).

GMT 6 release at the end of the year (hopefully).

Stabilize C API so we can move forward.

Welcome community involvement.

Acknowledgements

Thanks to Dongdong Tian (@seisman) for contributions.

This project is supported by **NSF** grant OCE-1558403.

Download slides + demo code: **leouieda.com**

Project website: **gmtpython.xyz**