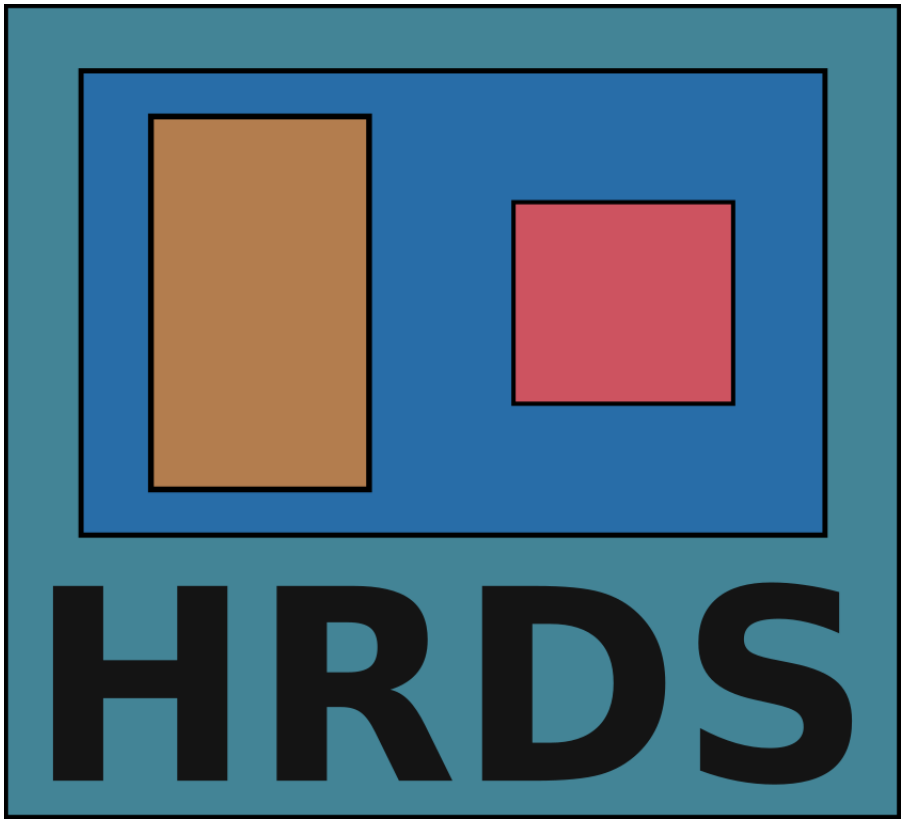


hrds Documentation

Release 0.1.1

Jon Hill

May 09, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 What is hrds?

hrds is a python package for obtaining points from a set of rasters at different resolutions. You can request a point and hrds will return a value based on the highest resolution dataset (as defined by the user) available at that point, blending datasets in a buffer region to ensure consistency. The software assumes all rasters are in the same projection space and using the same datum.

1.2 About this document

This document is the main manual for the software. It includes installation instructions, some examples and comprehensive API documentation.

1.3 Prerequisites

- python 3.6+
- numpy
- scipy
- osgeo.gdal (pygdal) to read and write raster data

hrds is available on conda-forge, so you can install easily using:

```
conda config --add channels conda-forge
conda install hrds
```

It is possible to list all of the versions of hrds available on your platform with:

```
conda search hrds --channel conda-forge
```

On Debian-based Linux you can also install manually. To install pygdal, first install the libgdal-dev packages and binaries:

```
sudo apt-get install libgdal-dev gdal-bin
```

To install pygdal, we neeed to check which version of gdal is installed:

```
gdal-config --version
```

pygdal can be installed using pip, specifying the version obtained from the command above. Note that you may need to increase the minor version number, e.g. from 2.1.3 to 2.1.3.3.

```
pip install pygdal==2.1.3.3
```

Replace 2.1.3.3 with the output from the `gdal-config` command.

You can install hrds using the standard:

```
python setup.py install
```

1.4 Examples

This example loads in an XYZ file and obtains data at each point, replacing the Z value with that from hrds.

```
from hrds import hrds

points = []
with open("test_mesh.csv",'r') as f:
    for line in f:
        row = line.split(",")
        # grab X and Y
        points.append([float(row[0]), float(row[1])])

bathy = hrds("gebco_uk.tif",
             rasters=("emod_utm.tif",
                      "inspire_data.tif"),
             distances=(700, 200))
bathy.set_bands()

print len(points)

with open("output.xyz","w") as f:
    for p in points:
        f.write(str(p[0])+"\t"+str(p[1])+"\t"+str(bathy.get_val(p))+"\n")
```

This will turn this:

```
$ head test_mesh.csv
805390.592314,5864132.9269,0
805658.162910036,5862180.30440542,0
805925.733505999,5860227.68191137,0
806193.304101986,5858275.05941714,0
806460.874698054,5856322.43692232,0
806728.445294035,5854369.81442814,0
806996.015889997,5852417.19193409,0
807263.586486046,5850464.56943942,0
807531.157082069,5848511.94694493,0
807798.727678031,5846559.32445088,0
```

into this:

```
$ head output.xyz
805390.592314       5864132.9269    -10.821567728305235
805658.16291        5862180.30441   2.721575532084955
805925.733506       5860227.68191   2.528217188012767
806193.304102       5858275.05942   3.1063558741547865
806460.874698       5856322.43692   5.470234157891056
806728.445294       5854369.81443   1.382685066254607
806996.01589        5852417.19193   1.8997482922322515
807263.586486       5850464.56944   4.0836843606647335
807531.157082       5848511.94694   -2.395080795759155
807798.727678       5846559.32445   -2.401006071401176
```

Example of use via [thetis](http://thetisproject.org/):

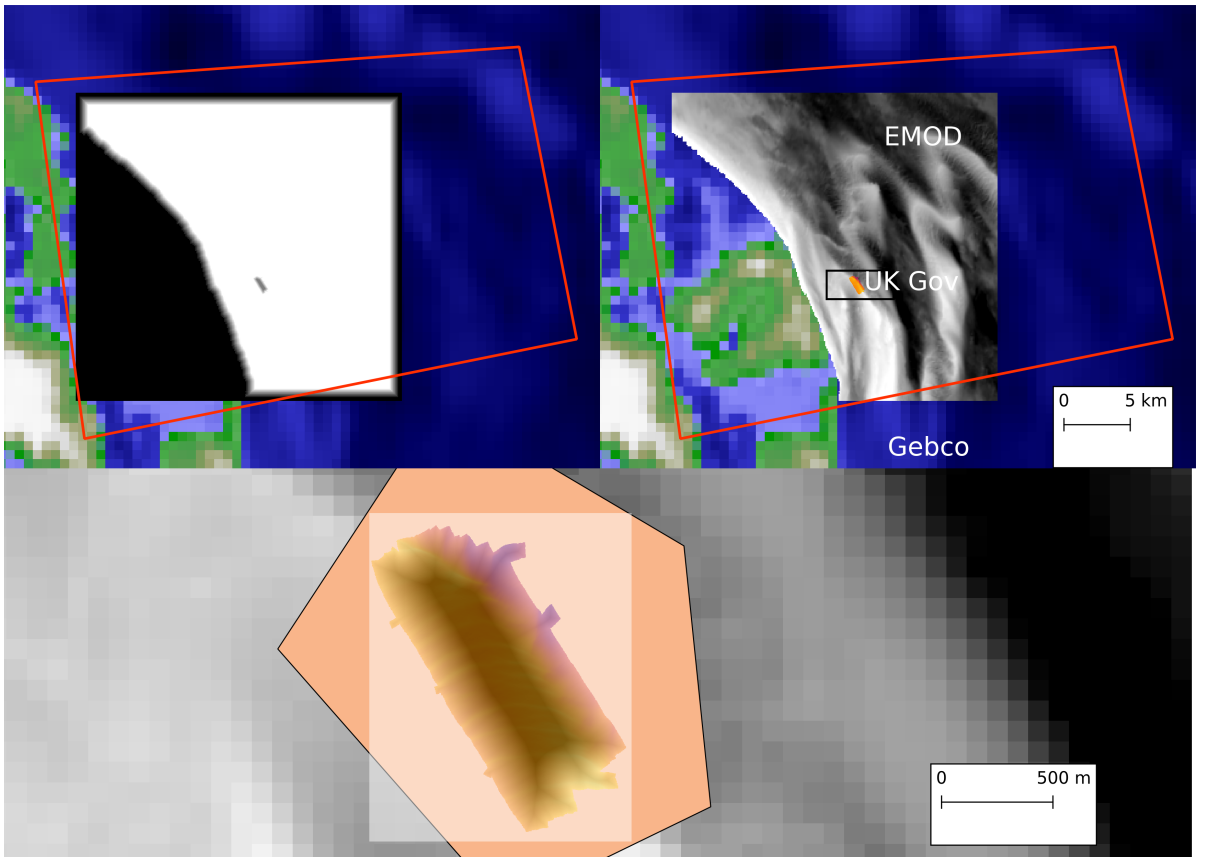
```
import firedrake
import thetis
from hrds import HRDS

mesh2d = firedrake.Mesh('test_mesh.msh') # mesh file

P1_2d = firedrake.FunctionSpace(mesh2d, 'CG', 1)
bathymetry2d = firedrake.Function(P1_2d, name="bathymetry")
bvector = bathymetry2d.dat.data
bathy = HRDS("gebco_uk.tif",
         rasters=("emod_utm.tif",
                  "inspire_data.tif"),
         distances=(700, 200))
bathy.set_bands()
for i, (xy) in enumerate(mesh2d.coordinates.dat.data):
    bvector[i] = bathy.get_val(xy)
thetis.File('bathy.pvd').write(bathymetry2d)
```

rest of thetis code

These images show the original data in QGIS in the top right, with each data set using a different colour scheme (GEBCO - green-blue; EMOD - grey; UK Gov - plasma - highlighted by the black rectangle).The red line is the boundary of the mesh used (see figure below). Both the EMOD and UK Gov data has NODATA areas, which are shown as transparent here, hence the curved left edge of the EMOD data. The figure also shows the buffer regions created around the two higher resolution datasets (top left), with black showing that data isn't used to white where it is 100% used. The effect of NODATA is clear here. The bottom panel shows a close-up of the UK Gov data with the buffer overlayed as a transparancy from white (not used) to black (100% UK Gov). The coloured polygon is the area of the high resolution mesh (see below).

After running the code above, we produce this blended dataset. Note the coarse mesh used here - it's not realistic for a model simulation!

If we then zoom-in to the high resolution area we can see the high resolution UK Gov data being used and with no obvious lines between datasets.



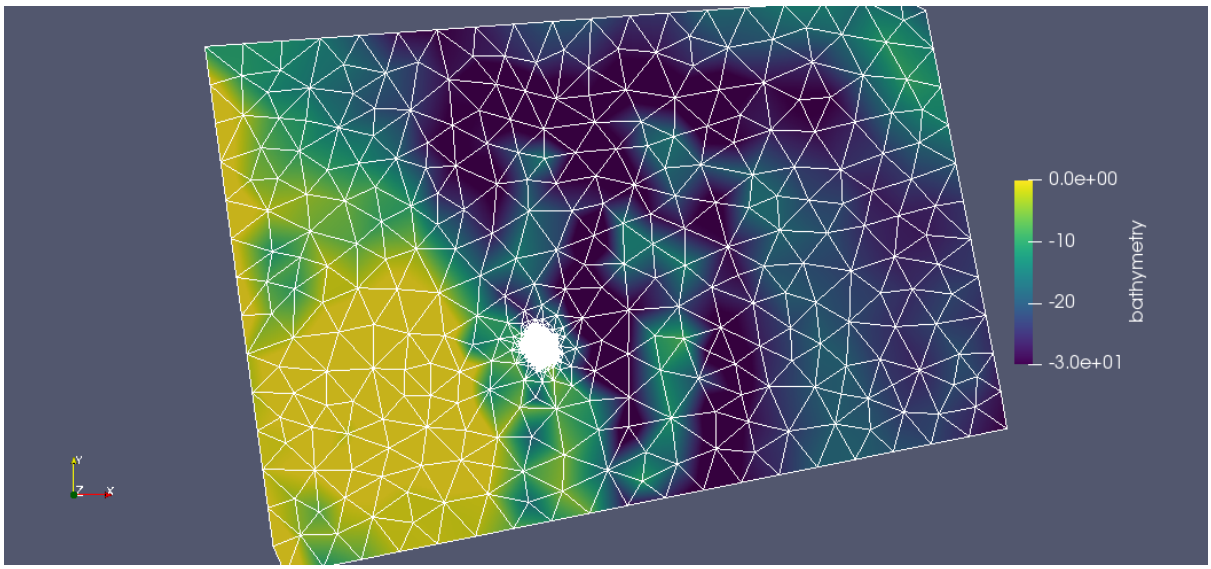


Fig. 1: A contrived mesh used in the example. We have a very high resolution area in the centre.

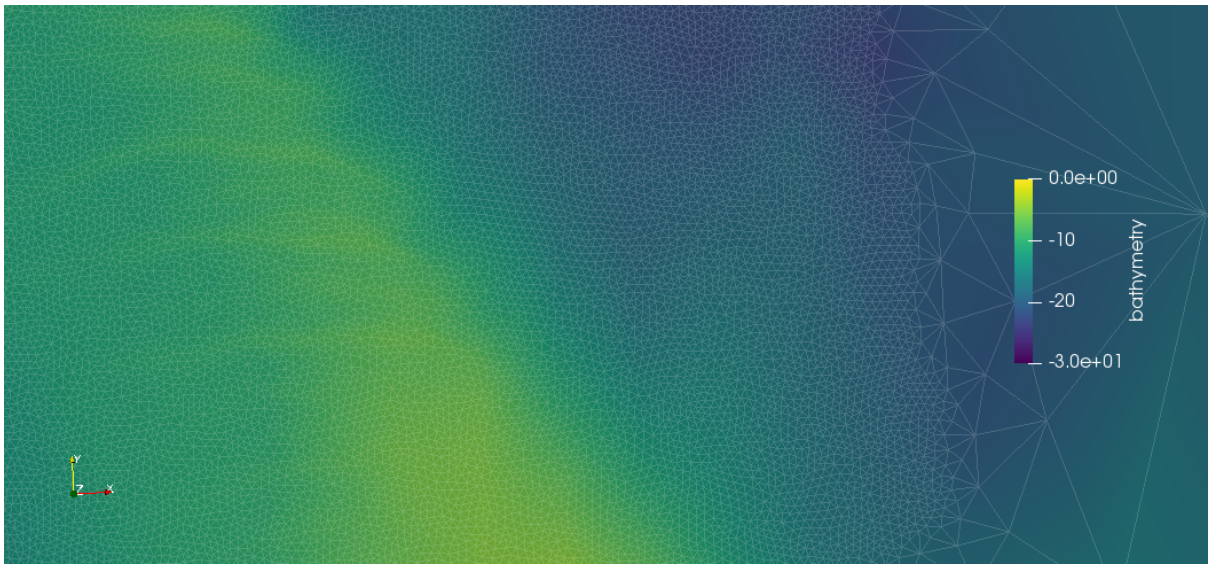


Fig. 2: The bathymetry data, which al lhave different resolution, are smoothly blended onto our mesh.

CHAPTER

TWO

HRDS API

hrds must be called within a python script. It has no user interface of its own.

```
import hrds
```

and all the functions below will be available hrds.xxxx.

Below is a description of the functions that are available in the API.

hrds contains the following components:

- buffer - functions to create buffer files
- hdrs - the main hdrs object.
- RasterInterpolator - objects to interpolate individual rasters

class hrds.hrds.**HRDS**(*baseRaster*, *rasters=None*, *distances=None*, *buffers=None*, *minmax=None*, *saveBuffers=False*)

The main HRDS class. Create a raster stack and initialise:

```
bathy = HRDS("gebco_uk.tif",
     rasters=("emod_utm.tif",
              "marine_digimap.tif"),
     distances=(10000, 5000),
     minmax=None)
bathy.set_bands()
```

The first argument is the base raster filename. *rasters* is a list of raster filenames, with corresponding *distances* over which to create the buffer. Distances are in the same units as the rasters. The min/max argument allows you to specify a minimum or maximum (or both!) values when returning data. This is useful for ocean simulations where you want a minimum depth to prevent "drying". To set this, do:

```
bathy = HRDS("gebco_uk.tif",
     rasters=("emod_utm.tif",
              "marine_digimap.tif"),
     distances=(10000, 5000),
     minmax=[[None,-5],[None,-3],[None,None]])
```

which would set a maximum depth of -5m on the gebco data (-ve = below sea level, +ve above), maximum of -3m on the emod data and no limits on the marine_digimap data. You must supply the same number of min-max pairs are there are total rasters. Temporary buffer files will be created and deleted at clean up

If is possible to supply the buffer rasters directly (e.g. if you want to use different distances on each edge of your raster, or some other such thing). Buffer extent must match or exceed the corresponding raster extent:

```
bathy = HRDS("gebco_uk.tif",
     rasters=("emod_utm.tif",
              "marine_digimap.tif"),
     buffers=(buffer1.tif,
              buffer2.tif,
              buffer3.tif))
bathy.set_bands()
```

Once set up, you can ask for data at any point:

```
bathy.get_val(100,100)
```

get_val(*point*)
Performs bilinear interpolation of your raster stack to give a value at the requested point.

Parameters **point** – a length 2 list containing x,y coordinates

Returns The value of the raster stack at that point

Raises

- CoordinateError – The point is outside the rasters
- RasterInterpolatorError – Generic error interpolating data at that point

set_bands(*bands=None*)
Performs bilinear interpolation of your raster stack to give a value at the requested point.

Parameters **bands** – a list of band numbers for each raster in the stack or None (uses the first band in each raster). Default is None.

exception hrds.hrds.**HRDSError**

class hrds.raster_buffer.**CreateBuffer**(*filename*, *distance*, *over=None*)
Implements the creation of a distance buffer from the edge of a raster to the centre:

```
rbuff = CreateBuffer('myRaster.tif',10000.0)
```

Will create a buffer raster with the same extents as myRaster.tif with a buffer that goes from 0 at the edge to 1.0 at a distance of 10,000 units from the edge. The distance should be in the same units as the raster file.

You can also specify the "resolution" of your buffer using the 'over' argument. Using '10' would use ten units resolve the buffer from edge to distance (e.g. is distance was 1.5, and over was 10, your output buffer would have a dx of 0.15). This will probably alter the extents of your buffer raster such that it no longer matches the actual raster, so proceed with caution. It may however, be useful if your input raster has very high resolution and you want to prevent multiple large raster files.

Once the object is made, write out the buffer using:

```
rbuff.make_buffer('output_buffer.tif')
```

Any GDAL-understood file format is supported for input or output.

extend_mask(*array*, *iterations*)
Extend the mask a number of "cells" in all directions"

Parameters

- **array** – the numpy aray to extend
- **iterations** – integer for how many cells to extend

Returns a numpy array

make_buffer(*output_file*)
Create a buffer raster from 0 to 1 over a set distance.

Parameters **output_file** – where to save this raster

exception hrds.raster.**CoordinateError**(*message*, *x*, *i*, *j*)
Raised when a point is outside the raster or the land mask

class hrds.raster.**Interpolator**(*origin*, *delta*, *val*, *mask=None*, *minmax=None*)
Implements an object to interpolate values from a Raster-type data set.

Used by the RasterInterpolator. A separate object as in the future we may switch bands and hence have to reload the val data.

get_val(*point*)
Get the value of this raster at the desired point via bi-linear interpolation.

Parameters **point** – a length 2 list containing x,y coordinates

Returns The value of the raster stack at that point

Raises

- *CoordinateError* – The point is outside the rasters
- *RasterInterpolatorError* – Generic error interpolating data at that point

set_mask(*mask*)
Set a mask to use. Not yet implemented.

class hrds.raster.**RasterInterpolator**(*filename*, *minmax=None*)
Implements an object to interpolate values from a Raster-stored data set:

```
rci = RasterInterpolator('foo.tif')
```

Any GDAL supported raster format should be fine. The origin is assumed to be the lower-left corner (i.e. south west) and the projection space is stored with the raster.

To indicate the band to be interpolated:

```
rci.set_band(2)
```

The default is Band 1 (ie. with no number given)

To interpolate this field in any arbitrary point:

```
rci.get_val((-3.0, 58.5))
```

It is allowed to switch between different fields using multiple calls of set_band().

get_array()
Get the raw data in the raster

Returns a numpy array containing the raster data

get_extent()
Return list of corner coordinates from a geotransform

Returns List continaing the corner coordinates of the raster

get_val(*x*)
Interpolate the field chosen with set_field(). The order of the coordinates should correspond with the storage order in the file.

Parameters **point** – a length 2 list containing x,y coordinates

Returns The value of the raster stack at that point

Raises

- *CoordinateError* – The point is outside the rasters
- *RasterInterpolatorError* – Generic error interpolating data at that point

point_in(*point*)
Does a point lay inside a raster's extent?

Parameters **point** – a length 2 list containing x,y coordinates

Returns Boolean. True is point is in the raster. False otherwise.

set_band(*band_no=1*)
Set the number of the band to be used. Usually 1, which is default

Parameters **band_no** – an integer which is the band number to use. Default is 1.

exception hrds.raster.**RasterInterpolatorError**
A generic error created by the RasterInterpolator class

PYTHON MODULE INDEX

h

INDEX