PythonFmask Documentation

Release 0.5.6

Neil Flood, Sam Gillingham

Oct 18, 2021

Contents

CHAPTER 1

Introduction

A set of command line utilities and Python modules that implement the 'fmask' algorithm as published in:

Zhu, Z. and Woodcock, C.E. (2012). Object-based cloud and cloud shadow detection in Landsat imagery Remote Sensing of Environment 118 (2012) 83-94.

and

Zhu, Z., Wang, S. and Woodcock, C.E. (2015). Improvement and expansion of the Fmask algorithm: cloud, cloud shadow, and snow detection for Landsats 4-7, 8, and Sentinel 2 images Remote Sensing of Environment 159 (2015) 269-277.

Also includes optional extension for Sentinel-2 from Frantz, D., Hass, E., Uhl, A., Stoffels, J., & Hill, J. (2018). Improvement of the Fmask algorithm for Sentinel-2 images: Separating clouds from bright surfaces based on parallax effects. Remote Sensing of Environment 215, 471-481.

Installation requires Python, numpy, scipy, GDAL and RIOS and the ability to compile C extensions for Python. It is licensed under GPL 3.

Originally developed by Neil Flood at DSITI and additional work funded by Landcare Research.

CHAPTER 2

Later Fmask4 Methods

A later publication by Qiu et al, 2019, suggests a number of further changes to Fmask, and gives this newer version the name of Fmask4. We have done some testing of these suggested methods, using Landsat and Sentinel-2 data over large areas of Australia. We found that overall, the losses outweighed the benefits, and so are unsure whether to implement these changes. Currently we have chosen not to.

Qiu, S., Zhu, Z., & He, B. (2019). Fmask 4.0: Improved cloud and cloud shadow detection in Landsats 4-8 and Sentinel-2 imagery. Remote Sensing of Environment, 231, 111205.

CHAPTER 3

Disclaimer and Acknowledgement

This Python implementation has followed the work of the authors cited above, and we offer our thanks for their work. None of them were involved in the creation of this Python implementation, and all errors made are our own responsibility.

CHAPTER 4

Philosophy

This package implements the Fmask algorithm as a Python module. It is intended that this can be wrapped in a variety of main programs which can handle the local details of how the image files are named and organised, and is intended to provide maximum flexibility. It should not be tied to expecting the imagery to be layed out in a particular manner.

This modular design also simplifies the use of the same core algorithm on either Landsat and Sentinel imagery. The wrapper programs take care of differences in file organisation and metadata formats, while the core algorithm is the same for both.

However, we have also supplied some example wrapper scripts, based around the image organisation as supplied by the usual distributors of the imagery. In the case of Landsat, we have supplied main programs which can cope with the data as it comes from USGS, while in the case of Sentinel-2 we have supplied wrappers to deal with the data as supplied by ESA.

It is expected that some users will use these directly, while larger organisations will wish to create their own wrappers specific to their own file naming and layout conventions.

The output from the core algorithm module is a single thematic raster, with integer codes representing null, clear, cloud, shadow, snow, water respectively.

The examples shown below use the given example wrappers.

CHAPTER 5

Command Line Examples

All the commandline programs given use argparse to handle commandline arguments, and hence will respond sensibly to the -h option by printing their own help. Some have options to modify their behaviour.

Please note that the output format used is defined by RIOS. This defaults to HFA (.img). See RIOS documentation for more information and how to change this using the environment variable $RIOS_DFLT_DRIVER.

5.1 USGS Landsat

Update: Since the USGS released their Collection-1 data set (completed globally in 2017), they now distribute cloud, shadow and snow masks included in their QA layer. These are calculated using CFMask, which should, in principle, be equivalent to the results of this code. Therefore, when processing USGS Collection-1 data, users may prefer the USGS-supplied masks. See USGS QA Layer.

The command line scripts supplied can process an untarred USGS Landsat scene. Here is an example of how to do this. This command will take a given scene directory, find the right images, and create an output file called cloud.img:

```
fmask_usgsLandsatStacked.py -o cloud.img --scenedir LC08_L1TP_150033_20150413_
↪20170410_01_T1
```

If the thermal band is empty (for Landsat-8 with the SSM anomaly, after 2015-11-01) then it is ignored gracefully.

There are command line options to modify many aspects of the algorithm's behaviour.

There are four options which are now obsolete, for manually specifying a pre-stacked file of reflectance bands, thermal bands, a saturation mask and the image of angles. These should be considered obsolete, and are replaced with the –scenedir option, which takes care of all internally.

5.2 Sentinel2

The command line scripts supplied can process a Sentinel2 Level C granule from the image directory. Here is an example of how to do this. This example works at 20m resolution, but the recipe can be varied as required. Be warned,

processing at 10m resolution would be considerably slower, and is unlikely to be any more accurate.

This command will take a given .SAFE directory, find the right images, and create an output file called cloud.img:

```
fmask_sentinel2Stacked.py -o cloud.img --safedir S2B_MSIL1C_20180918T235239_N0206_
↪R130_T56JNQ_20180919T011001.SAFE
```

When working with the old ESA zipfile format, which packed multiple tiles into a single SAFE-format zipfile, this approach will not work, as it won't know which tile to process. So, instead, use the option to specify the granule directory, as follows:

```
fmask_sentinel2Stacked.py -o cloud.img --granuledir S2A_OPER_PRD_MSIL1C_PDMC_
↪20160111T072442_R030_V20160111T000425_20160111T000425.SAFE/GRANULE/S2A_OPER_MSI_L1C_
↪TL_SGS__20160111T051031_A002887_T56JNQ_N02.01
```

This would also work on a new-format directory, but specifying the top .SAFE directory is easier.

There are command line options to modify many aspects of the algorithm's behaviour.

There are two options which are now obsolete, for manually specifying a pre-stacked file of reflectance bands, and the image of angles. These should be considered obsolete, and are replaced with the –safedir or –granuledir option, which take care of all internally.

CHAPTER 6

Re-wrapping and Re-configuring

To build a different set of wrappers, and configure things differently, the default wrappers are a good place to start. The configuration is mainly handled by the *fmask.config.FmaskConfig* class. For example, one would call *fmask.config.FmaskConfig.setReflectiveBand()* to change which layer of the stack corresponds to which wavelength band.

CHAPTER 7

Downloads

Get the source as a bundle from GitHub. Release notes for each version can be read in releasenotes. To install from source, read the INSTALL.txt file included inside the source bundle.

Pre-built binary Conda packages are available under the 'conda-forge' channel. Once you have installed Conda, run the following commands on the command line to install python-fmask:

```
conda config --add channels conda-forge
conda config --set channel_priority strict
conda create -n myenv python-fmask
conda activate myenv
```

CHAPTER 8

Applications that use python-fmask

- Cloud Masking: It is a Qgis plugin for cloud masking the Landsat (4, 5, 7 and 8) products using different process and filters such as Fmask, Blue Band, Cloud QA, Aerosol and Pixel QA.

CHAPTER 9

Issues

Please log bugs encountered with the Issue Tracker.

CHAPTER 10

Python developer documentation

10.1 fmask

Implement the cloud and shadow algorithms known collectively as Fmask, as published in

Zhu, Z. and Woodcock, C.E. (2012). Object-based cloud and cloud shadow detection in Landsat imagery Remote Sensing of Environment 118 (2012) 83-94.

and

Zhu, Z., Wang, S. and Woodcock, C.E. (2015). Improvement and expansion of the Fmask algorithm: cloud, cloud shadow, and snow detection for Landsats 4-7, 8, and Sentinel 2 images Remote Sensing of Environment 159 (2015) 269-277.

Taken from Neil Flood's implementation by permission.

The notation and variable names are largely taken from the paper. Equation numbers are also from the paper.

Input is a top of atmosphere (TOA) reflectance file.

The output file is a single thematic raster layer with codes representing null, clear, cloud, shadow, snow and water. These are the values 0-5 respectively, but there are constants defined for the different codes, as fmask.fmask.OUTCODE_*

`fmask.fmask.`**`accumHist`**(*counts*, *vals*)
Accumulate the given values into the given (partial) counts

`fmask.fmask.`**`calcBTthresholds`**(*otherargs*)
Calculate some global thresholds based on the results of the first pass

`fmask.fmask.`**`calcCDI`**(*ref*, *fmaskConfig*, *refBands*)
Calculate the Cloud Displacement Index, as per Frantz et al (2018).

`fmask.fmask.`**`cloudFinalPass`**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS

Final pass of cloud mask layer

fmask.fmask.**cloudShapeFunc**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS.

Calculate the 3d cloud shape image. Requires that RIOS be run on the whole image at once, as substantial spatial structure is required. Needed to use RIOS because of the possibility of the thermal input being on a different resolution to the cloud input

Returns the result as whole arrays in otherargs, rather than writing them to output files, as they would just be read in again as whole arrays immediately.

fmask.fmask.**clumpClouds**(*cloudmaskfile*)
Clump cloud pixels to make a layer of cloud objects. Currently assumes that the cloud mask contains only zeros and ones.

fmask.fmask.**doCloudLayerFinalPass**(*fmaskFilenames*, *fmaskConfig*, *pass1file*, *pass2file*, *landThreshold*, *Tlow*, *missingThermal*)
Final pass

fmask.fmask.**doFmask**(*fmaskFilenames*, *fmaskConfig*)
Main routine for whole Fmask algorithm. Calls all other routines in sequence. Parameters:

- **fmaskFilenames** an instance of *fmask.config.FmaskFilenames* that contains the files to use
- **fmaskConfig** an instance of *fmask.config.FmaskConfig* that contains the parameters to use

If *fmask.config.FmaskConfig.setKeepIntermediates()* has been called with True, then a dictionary of intermediate files will be returned. Otherwise None is returned.

fmask.fmask.**doPotentialCloudFirstPass**(*fmaskFilenames*, *fmaskConfig*, *missingThermal*)
Run the first pass of the potential cloud layer. Also finds the temperature thresholds which will be needed in the second pass, because it has the relevant data handy.

fmask.fmask.**doPotentialCloudSecondPass**(*fmaskFilenames*, *fmaskConfig*, *pass1file*, *Twater*, *Tlow*, *Thigh*, *missingThermal*, *nonNullCount*)
Second pass for potential cloud layer

fmask.fmask.**doPotentialShadows**(*fmaskFilenames*, *fmaskConfig*, *NIR_17*)
Make potential shadow layer, as per section 3.1.3 of Zhu&Woodcock.

fmask.fmask.**finalizeAll**(*fmaskFilenames*, *fmaskConfig*, *interimCloudmask*, *interimShadowmask*, *pass1file*)
Use the cloud and shadow masks to mask the snow layer (as per Zhu & Woodcock). Apply the optional extra buffer to the cloud mask, and write to final file.

fmask.fmask.**focalVariance**(*img*, *winSize*)
Calculate the focal variance of the given 2-d image, over a moving window of size winSize pixels.

fmask.fmask.**getIntersectionCoords**(*filelist*)
Use the RIOS utilities to get the correct area of intersection for a set of files, although we are not going to read the files using RIOS itself, but with GDAL directly.

fmask.fmask.**make3Dclouds**(*fmaskFilenames*, *fmaskConfig*, *clumps*, *numClumps*, *missingThermal*)
Create 3-dimensional cloud objects from the cloud mask, and the thermal information. Assumes a constant lapse rate to convert temperature into height. Resulting cloud heights are relative to cloud base.

Returns an image of relative cloud height (relative to cloud base for each cloud object), and a dictionary of cloud base temperature, for each cloud object, and valueindexes.ValueIndexes object for use in extracting the location of every pixel for a given cloud object.

fmask.fmask.**makeBufferKernel**(*buffsize*)
Make a 2-d array for buffering. It represents a circle of radius buffsize pixels, with 1 inside the circle, and zero outside.

fmask.fmask.**makeCloudShadowShapes**(*fmaskFilenames*, *fmaskConfig*, *cloudShape*, *cloudClumpNdx*)
Project the 3d cloud shapes onto horizontal surface, along the sun vector, to make the 2d shape of the shadow.

fmask.fmask.**maskAndBuffer**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS

Apply cloud and shadow masks to snow layer, and buffer cloud layer

The main aims of all this re-masking are:

1) A pixel should be either cloud, shadow, snow or not, but never more than one

2) Areas which are null in the input imagery should be null in the mask, even after buffering, etc.

fmask.fmask.**matchOneShadow**(*cloudmask*, *shadowEntry*, *potentialShadow*, *Tcloudbase*, *Tlow*, *Thigh*, *xRes*, *yRes*, *cloudID*, *nullmask*)
Given the temperatures and sun angles for a single cloud object, and a shadow shape, search along the sun vector for a matching shadow object.

fmask.fmask.**matchShadows**(*fmaskConfig*, *interimCloudmask*, *potentialShadowsFile*, *shadowShapesDict*, *cloudBaseTemp*, *Tlow*, *Thigh*, *pass1file*)
Match the cloud shadow shapes to the potential cloud shadows. Write an output file of the resulting shadow layer. Includes a 3-pixel buffer on the final shadows.

fmask.fmask.**potentialCloudFirstPass**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS.

Calculate the first pass potential cloud layer (equation 6)

fmask.fmask.**potentialCloudSecondPass**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS

Second pass of potential cloud layer

fmask.fmask.**refDNtoUnits**(*refDN*, *fmaskConfig*)
Convert the given reflectance pixel value array to physical units, using parameters given in fmaskConfig.

Scaling is ref = (dn+offset)/scaleVal

fmask.fmask.**scoreatpcnt**(*counts*, *pcnt*)
Given histogram counts (binned on the range 0-255), find the value which corresponds to the given percentile value (0-100).

fmask.fmask.**singleRefDNtoUnits**(*refDN*, *scaleVal*, *offset*)
Apply the given scale and offset to transform a single band of reflectance from digital number (DN) to reflectance units.

Calculation is ref = (refDN + offset) / scaleVal

fmask.fmask.**B4_SCALE = 500.0**
Gain to scale b4 reflectances to 0-255 for histograms

fmask.fmask.**BT_OFFSET = 176**
An offset so we can scale brightness temperature (BT, in deg C) to the range 0-255, for use in histograms.

fmask.fmask.**OUTCODE_CLEAR = 1**
Output pixel value for clear land

fmask.fmask.**OUTCODE_CLOUD = 2**
Output pixel value for cloud

fmask.fmask.**OUTCODE_NULL = 0**
Output pixel value for null

fmask.fmask.**OUTCODE_SHADOW = 3**
Output pixel value for cloud shadow

fmask.fmask.**OUTCODE_SNOW = 4**
Output pixel value for snow

fmask.fmask.**OUTCODE_WATER = 5**
Output pixel value for water

fmask.fmask.**PROB_SCALE = 100.0**
For scaling probability values so I can store them in 8 bits

fmask.fmask.**RIOS_WINDOW_SIZE = 512**
Global RIOS window size

- genindex
- modindex
- search

10.2 config

Configuration classes that define the inputs and parameters for the fmask function.

class fmask.config.**AngleConstantInfo**(*solarZenithAngle*, *solarAzimuthAngle*, *viewZenithAngle*, *viewAzimuthAngle*)
An implementation of AnglesInfo that uses constant angles accross the scene.

getSolarAzimuthAngle(*indices*)
Return the solar azimuth angle

getSolarZenithAngle(*indices*)
Return the solar zenith angle

getViewAzimuthAngle(*indices*)
Return the view azimuth angle

getViewZenithAngle(*indices*)
Return the view zenith angle

class fmask.config.**AnglesFileInfo**(*solarZenithFilename*, *solarZenithBand*, *solarAzimuthFilename*, *solarAzimuthBand*, *viewZenithFilename*, *viewZenithBand*, *viewAzimuthFilename*, *viewAzimuthBand*)
An implementation of AnglesInfo that reads the information from GDAL supported files.

getSolarAzimuthAngle(*indices*)
Return the average solar azimuth angle for the given indices

getSolarZenithAngle(*indices*)
Return the average solar zenith angle for the given indices

getViewAzimuthAngle(*indices*)
Return the average view azimuth angle for the given indices

getViewZenithAngle(*indices*)
Return the average view zenith angle for the given indices

prepareForQuerying()
Called when fmask is about to query this object for angles.

static readData(*filename*, *bandNum*)

releaseMemory()
Called when fmask has finished querying this object.

setScaleToRadians(*scale*)
Set scaling factor to get radians from angles image values.

class fmask.config.**AnglesInfo**
Abstract base class that Contains view and solar angle information for file (in radians).

getSolarAzimuthAngle(*indices*)
Return the average solar azimuth angle for the given indices

getSolarZenithAngle(*indices*)
Return the average solar zenith angle for the given indices

getViewAzimuthAngle(*indices*)
Return the average view azimuth angle for the given indices

getViewZenithAngle(*indices*)
Return the average view zenith angle for the given indices

prepareForQuerying()
Called when fmask is about to query this object for angles. Derived class should do any reading of files into memory required here.

releaseMemory()
Called when fmask has finished querying this object. Can release any allocated memory.

setScaleToRadians(*scale*)
Set scaling factor to get radians from angles image values.

class fmask.config.**FmaskConfig**(*sensor*)
Class that contains the configuration parameters of the fmask run.

setAnglesInfo(*info*)
Set an instance of AnglesInfo. By default this is None and will need to be set before fmask will run.

The *fmask.config.readAnglesFromLandsatMTL()* function can be used to obtain this from a Landsat .mtl file.

setCirrusBandTestThresh(*thresh*)
Change the threshold used by Zhu et al 2015, section 2.2.1 for the cirrus band test. Defaults to 0.01.

setCirrusProbRatio(*ratio*)
Change the ratio used by Zhu et al 2015 Equation 1 to determine the cirrus cloud probability. Defaults to 0.04.

setCloudBufferSize(*bufferSize*)
Extra buffer of this many pixels on cloud layer. Defaults to 5.

setDefaultExtension(*extension*)
Sets the default extension used by temporary files created by fmask. Defaults to the extension of the driver that RIOS is configured to use.

Note that this should include the '.' - ie '.img'.

setEqn17CloudProbThresh(*thresh*)
Change the threshold used by Equation 17. The threshold given here is the constant term added to the end of the equation for the land probability threshold. Original paper had this as 0.2, although Zhu et al's MATLAB code now defaults it to 0.225 (i.e. 22.5%)

setEqn19NIRFillThresh (*thresh*)
Change the threshold used by Equation 19 to determine potential cloud shadow from the difference between NIR and flood filled NIR. Defaults to 0.02.

setEqn1Swir2Thresh (*thresh*)
Change the threshold used by Equation 1 for the SWIR2 band. This defaults to 0.03

setEqn1ThermThresh (*thresh*)
Change the threshold used by Equation one for BT. This defaults to 27.

setEqn20GreenSnowThresh (*thresh*)
Change the threshold used by Equation 20 (snow) for green reflectance. This defaults to 0.1

setEqn20NirSnowThresh (*thresh*)
Change the threshold used by Equation 20 (snow) for NIR reflectance. This defaults to 0.11

setEqn20ThermThresh (*thresh*)
Change the threshold used by Equation 20 (snow) for BT. This defaults to 3.8.

setEqn2WhitenessThresh (*thresh*)
Change the threshold used by Equation 2 to determine whiteness from visible bands. This defaults to 0.7.

setEqn7Swir2Thresh (*thresh*)
Change the threshold used by Equation 7 (water test) for the Swir2 band. This defaults to 0.03.

setGdalDriverName (*driverName*)
Change the GDAL driver used for writing the final output file. Default value is taken from the default for the RIOS package, as per $RIOS_DFLT_DRIVER.

setKeepIntermediates (*keepIntermediates*)
Set to True to keep the intermediate files created in processed. This is False by default.

setMinCloudSize (*minCloudSize*)
Set the minimum cloud size retained. This minimum is applied before any buffering of clouds. Size is specified as an area, in pixels.

setReflectiveBand (*band*, *index*)
Tell fmask which band is in which index in the reflectance data stack file. band should be one of the BAND_* constants. index is zero based (ie 0 is first band in the file).

These are set to default values for each sensor which are normally correct, but this function can be used to update.

setSen2displacementTest (*useDisplacementTest*)
Set whether or not to use the Frantz (2018) parallax displacement test to remove false clouds. Pass True if the test is desired, False otherwise.

setShadowBufferSize (*bufferSize*)
Extra buffer of this many pixels on cloud layer. Defaults to 10.

setStrictFmask (*strictFmask*)
Set whatever options are necessary to run strictly as per Fmask paper (Zhu & Woodcock). Setting this will override the settings of other parameters on this object.

setTOARefOffsetDict (*offsetDict*)
Set the reflectance offsets to the given list. This should contain an offset value for each band used with the Fmask code. The keys are the named constants in the config module, BAND_*.

The offset is added to the corresponding band pixel values before dividing by the scaling value.

This facility is made available largely for use with Sentinel-2, after ESA unilaterally starting using non-zero offsets in their Level-1C imagery (Nov 2021). However, it can be used with Landsat if required.

setTOARefScaling(*scaling*)
Set the scaling used in the Top of Atmosphere reflectance image. The calculation is done as

ref = (dn + dnOffset) / scaling

and so is used in conjunction with the offset values (see setTOARefOffsets).

The dnOffset was added in 2021 to cope with ESA's absurd decision to suddenly introduce an offset in their Sentinel-2 TOA reflectance imagery. For Landsat, there is no need for it ever to be non-zero.

setTempDir(*tempDir*)
Temporary directory to use. Defaults to '.' (the current directory).

setThermalInfo(*info*)
Set an instance of ThermalFileInfo. By default this is None and fmask assumes there is no thermal data available.

The *fmask.config.readThermalInfoFromLandsatMTL()* function can be used to obtain this from a Landsat .mtl file.

setVerbose(*verbose*)
Print informative messages. Defaults to False.

Eqn17CloudProbThresh = 0.2

Eqn19NIRFillThresh = 0.02

Eqn1Swir2Thresh = 0.03

Eqn1ThermThresh = 27

Eqn20GreenSnowThresh = 0.1

Eqn20NirSnowThresh = 0.11

Eqn20ThermThresh = 3.8

Eqn2WhitenessThresh = 0.7

Eqn7Swir2Thresh = 0.03

TOARefDNoffsetDict = None

TOARefScaling = 10000.0

cirrusBandTestThresh = 0.01

cirrusProbRatio = 0.04

cloudBufferSize = 5

gdalDriverName = <Mock name='mock.applier.DEFAULTDRIVERNAME' id='139918829389648'>

keepIntermediates = False

minCloudSize_pixels = 0

sen2cdiWindow = 7

sen2displacementTest = False

shadowBufferSize = 10

strictFmask = False

tempDir = '.'

verbose = False

class fmask.config.**FmaskFilenames**(*toaRefFile=None*, *thermalFile=None*, *outputMask=None*, *saturationMask=None*)

Class that contains the filenames used in the fmask run.

setOutputCloudMaskFile(*cloudMask*)

Set the output cloud mask path.

Note that this file will be written in the format that RIOS is currently configured to use. See the RIOS documentation for more details. Note that the default is HFA (.img) and can be overridden using environment variables.

setSaturationMask(*mask*)

Set the mask to use for ignoring saturated pixels. By default no mask is used and all pixels are assumed to be unsaturated. This will cause problems for the whiteness test if some pixels are in fact saturated, but not masked out.

Use the `fmask.saturation.makeSaturationMask()` function to create this from input radiance data.

This mask should be 1 for pixels that are saturated, 0 otherwise.

Note that this is not in the original paper so cannot be considered 'strict', but if provided is used no matter the strict setting in *fmask.config.FmaskConfig*.

This file should be in any GDAL readable format.

setTOAReflectanceFile(*toaRefFile*)

Set the path of the input top of atmosphere (TOA) file. It pays to check that the default set of bands match what fmask expects in the *fmask.config.FmaskConfig* class and update if necessary.

This should have numbers which are reflectance * 1000

Use the *fmask.landsatTOA.makeTOAReflectance()* function to create this file from raw Landsat radiance (or the fmask_usgsLandsatTOA.py command line program supplied with fmask).

It is assumed that any values that are nulls in the original radiance image are set to the ignore values in the toaRefFile.

This file should be in any GDAL readable format.

setThermalFile(*thermalFile*)

Set the path of the input thermal file. To make use of this, the *fmask.config.FmaskConfig.setThermalInfo()* function must also be called so that fmask knows how to use the file.

This file should be in any GDAL readable format.

outputMask = None

saturationMask = None

thermal = None

toaRef = None

class fmask.config.**ThermalFileInfo**(*thermalBand1040um*, *thermalGain1040um*, *thermalOffset1040um*, *thermalK1_1040um*, *thermalK2_1040um*)

Contains parameters for interpreting thermal file. See *fmask.config.readThermalInfoFromLandsatMTL()*.

scaleThermalDNtoC(*scaledBT*)

Use the given params to unscale the thermal, and then convert it from K to C. Return a single 2-d array of the temperature in deg C.

thermalBand1040um = None

thermalGain1040um = None

thermalK1_1040um = None

thermalK2_1040um = None

thermalOffset1040um = None

fmask.config.**readAnglesFromLandsatMTL**(*mtlfile*)
Given the path to a Landsat USGS .MTL file, read the angles out and return an instance of AngleConstantInfo.

This is no longer supported, and this routine now raises an exception.

fmask.config.**readMTLFile**(*mtl*)
Very simple .mtl file reader that just creates a dictionary of key and values and returns it

fmask.config.**readThermalInfoFromLandsatMTL**(*mtlfile*, *thermalBand1040um=0*)
Returns an instance of ThermalFileInfo given a path to the mtl file and the index of the thermal band.

fmask.config.**BAND_BLUE = 0**
~475nm

fmask.config.**BAND_CIRRUS = 4**
~1360nm

fmask.config.**BAND_GREEN = 1**
~560nm

fmask.config.**BAND_NIR = 3**
~780nm

fmask.config.**BAND_RED = 2**
~660nm

fmask.config.**BAND_S2CDI_NIR7 = 8**
~783nm

fmask.config.**BAND_S2CDI_NIR8A = 7**
~865nm

fmask.config.**BAND_SWIR1 = 5**
~1610nm

fmask.config.**BAND_SWIR2 = 6**
~2200nm

fmask.config.**FMASK_LANDSAT47 = 0**
Landsat 4 to 7

fmask.config.**FMASK_LANDSAT8 = 1**
Landsat 8

fmask.config.**FMASK_SENTINEL2 = 2**
Sentinel 2

- genindex
- modindex
- search

10.3 landsatTOA

Module that handles convertion of scaled radiance (DN) values from USGS to Top of Atmosphere (TOA) reflectance (*1000).

fmask.landsatTOA.**earthSunDistance**(*date*)
Given a date in YYYYMMDD will compute the earth sun distance in astronomical units

fmask.landsatTOA.**makeTOAReflectance**(*infile*, *mtlFile*, *anglesfile*, *outfile*)
Main routine - does the calculation

The eqn for TOA reflectance, p, is p = pi * L * d^2 / E * cos(theta)

d = earthSunDistance(date) L = image pixel (radiance) E = exoatmospheric irradiance for the band, and theta = solar zenith angle.

Assumes infile is radiance values in DN from USGS. mtlFile is the .mtl file. outfile will be created in the default format that RIOS is configured to use and will be top of atmosphere reflectance values *10000. Also assumes that the angles image file is scaled as radians*100, and has layers for satAzimuth, satZenith, sunAzimuth, sunZenith, in that order.

fmask.landsatTOA.**readGainsOffsets**(*mtlInfo*)
Read the gains and offsets out of the .MTL file

fmask.landsatTOA.**riosTOA**(*info*, *inputs*, *outputs*, *otherinputs*)
Called from RIOS

- genindex
- modindex
- search

10.4 saturationcheck

Module for doing checks of visible band and reporting and saturation. Note that this works off the original radiance file, not the TOA reflectance.

fmask.saturationcheck.**makeSaturationMask**(*fmaskConfig*, *radiancefile*, *outMask*)
Checks the radianceFile and creates a mask with 1's where there is saturation in one of more visible bands. 0 otherwise.

The fmaskConfig parameter should be an instance of *fmask.config.FmaskConfig*. This is used to determine which bands are visible.

This mask is advisible since the whiteness test Eqn 2. and Equation 6 are affected by saturated pixels and may determine a pixel is not cloud when it is.

The format of outMask will be the current RIOS default format.

It is assumed that the input radianceFile has values in the range 0-255 and saturated pixels are set to 255.

fmask.saturationcheck.**riosSaturationMask**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS. Does the actual saturation test. Currently assumes that only 8-bit radiance inputs can be saturated, but if this turns out not to be true, we can come back to this.

- genindex
- modindex
- search

10.5 landsatangles

Functions relating to estimating the per-pixel sun and satellite angles for a given Landsat image. These are rough estimates, using the generic characteristics of the Landsat 5 platform, and are not particularly accurate, but good enough for the current purposes.

Historically, the USGS have not supplied satellite zenith/azimuth angles, and have only supplied scene-centre values for sun zenith/azimuth angles. Since the satellite view geometry is important in correctly tracking a shadow when matching shadows to their respective clouds, the Fmask algorithm requires good estimates of all these angles. The routines contained here are used to derive per-pixel estimates of these angles.

As of mid-2016, the USGS are planning to supply sufficient information to calculate these angles directly from orbit ephemeris data. When that comes about, it seems likely that the need for the routines here will diminish, but any data downloaded from USGS prior to then will still require this approach, as the associated angle metadata will not be present.

The core Fmask code in this package is adaptable enough to be configured for either approach.

The general approach for satellite angles is to estimate the nadir line by running it down the middle of the image data area. The satellite azimuth is assumed to be at right angles to this nadir line, which is only roughly correct. For the whisk-broom sensors on Landsat-5 and Landsat-7, this angle is not 90 degrees, but is affected by earth rotation and is latitude dependent. For Landsat-8, the scan line is at right angles, due to the compensation for earth rotation, but the push-broom is made up of sub-modules which point in slightly different directions, giving slightly different satellite azimuths along the scan line. None of these effects are included in the current estimates. The satellite zenith is estimated based on the nadir point, the scan-line, and the assumed satellite altitude, and includes the appropriate allowance for earth curvature.

Because this works by searching the imagery for the non-null area, and assumes that this represents a full-swath image, it would not work for a subset of a full image.

The sun angles are approximated using the algorithm found in the Fortran code with 6S (Second Simulation of the Satellite Signal in the Solar Spectrum). The subroutine in question is the POSSOL() routine. I translated the Fortran code into Python for inclusion here.

`fmask.landsatangles.`**`bilinearInterp`**(*xMin*, *xMax*, *yMin*, *yMax*, *cornerVals*, *x*, *y*)
Evaluate the given value on a grid of (x, y) points. The exact value is given on a set of corner points (top-left, top-right, bottom-left, bottom-right). The corner locations are implied from xMin, xMax, yMin, yMax.

`fmask.landsatangles.`**`findCorners`**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS

Checks non-null area of image block. Finds extremes, records coords of extremes against those already in otherargs.

Note that the logic is very specific to the orientation of the usual Landsat descending pass imagery. The same logic should not be applied to swathes oriented in other, e.g. for other satellites.

`fmask.landsatangles.`**`findImgCorners`**(*img*, *imgInfo*)
Find the corners of the data within the given template image Return a numpy array of (x, y) coordinates. The array has 2 columns, for X and Y. Each row is a corner, in the order:

top-left, top-right, bottom-left, bottom-right.

Uses RIOS to pass through the image searching for non-null data, and find the extremes. Assumes we are working with a full-swathe Landsat image.

Each list element is a numpy array of (x, y)

`fmask.landsatangles.`**`findNadirLine`**(*corners*)
Return the equation of the nadir line, from the given corners of the swathe. Returns a numpy array of [b, m], for the equation

y = mx + b

giving the y coordinate of the nadir as a function of the x coordinate.

fmask.landsatangles.**getCtrLatLong**(*imgInfo*)

Return the lat/long of the centre of the image as (ctrLat, ctrLong)

fmask.landsatangles.**localRadius**(*latitude*)
Calculate a local radius of curvature, for the given geodetic latitude. This approximates the earth curvature at this latitude. The given latitude is in degrees. This is probably overkill, given some of the other approximations I am making....

fmask.landsatangles.**makeAngles**(*info*, *inputs*, *outputs*, *otherargs*)
Called from RIOS

Make 4-layer sun and satellite angles for the image block

fmask.landsatangles.**makeAnglesImage**(*templateimg*, *outfile*, *nadirLine*, *extentSunAngles*, *satAzimuth*, *imgInfo*)
Make a single output image file of the sun and satellite angles for every pixel in the template image.

fmask.landsatangles.**satAzLeftRight**(*nadirLine*)
Calculate the satellite azimuth for the left and right sides of the nadir line. Assume that the satellite azimuth vector is at right angles to the nadir line (which is not really true, but reasonably close), and that there are only two possibilities, as a pixel is either to the left or to the right of the nadir line.

Return a numpy array of [satAzLeft, satAzRight], with angles in radians, in the range [-pi, pi]

fmask.landsatangles.**sunAnglesForExtent**(*imgInfo*, *mtlInfo*)
Return array of sun azimuth and zenith for each of the corners of the image extent. Note that this is the raster extent, not the corners of the swathe.

The algorithm used here has been copied from the 6S possol() subroutine. The Fortran code I copied it from was .... up to the usual standard in 6S. So, the notation is not always clear.

fmask.landsatangles.**sunAnglesForPoints**(*latDeg*, *longDeg*, *hourGMT*, *jdp*)
Calculate sun azimuth and zenith for the given location(s), for the given time of year. jdp is the julian day as a proportion, ranging from 0 to 1, where Jan 1 is 1.0/365 and Dec 31 is 1.0. Location is given in latitude/longitude, in degrees, and can be arrays to calculate for multiple locations. hourGMT is a decimal hour number giving the time of day in GMT (i.e. UTC).

Return a tuple of (sunAz, sunZen). If latDeg and longDeg are arrays, then returned values will be arrays of the same shape.

- genindex
- modindex
- search

10.6 zerocheck

Utility function to determine if a particular band in a file is all zeros. This is useful for Landsat8 since although the file and metadata exist for thermal, it may be all zeroes and hence unusable form cloud masking.

fmask.zerocheck.**isBandAllZeroes**(*filename*, *band=0*)
Checks the specified band within a file to see if it is all zeroes. band should be 0 based.

Returns True if all zeroes, False otherwise.

This function firstly checks the stats if present and uses this and assumes that they are correct.

If they are not present, then the band is read and the values determined.

fmask.zerocheck.**riosAllZeroes**(*info*, *inputs*, *outputs*, *otherArgs*)
Called from RIOS. Does the actual work.

- genindex
- modindex
- search

10.7 fillminima

Module to implement filling of local minima in a raster surface.

The algorithm is from

Soille, P., and Gratin, C. (1994). An efficient algorithm for drainage network extraction on DEMs. J. Visual Communication and Image Representation. 5(2). 181-189.

The algorithm is intended for hydrological processing of a DEM, but is used by the Fmask cloud shadow algorithm as part of its process for finding local minima which represent potential shadow objects.

fmask.fillminima.**fillMinima**(*img*, *nullval*, *boundaryval*)
Fill all local minima in the input img. The input array should be a numpy 2-d array. This function returns an array of the same shape and datatype, with the same contents, but with local minima filled using the reconstruction-by-erosion algorithm.

- genindex
- modindex
- search

10.8 valueindexes

exception fmask.valueindexes.**NonIntTypeError**

exception fmask.valueindexes.**RangeError**

exception fmask.valueindexes.**ValueIndexesError**

class fmask.valueindexes.**ValueIndexes**(*a*, *nullVals=[]*)
An object which contains the indexes for every value in a given array. This class is intended to mimic the reverse_indices clause in IDL, only nicer.

Takes an array, works out what unique values exist in this array. Provides a method which will return all the indexes into the original array for a given value.

The array must be of an integer-like type. Floating point arrays will not work. If one needs to look at ranges of values within a float array, it is possible to use numpy.digitize() to create an integer array corresponding to a set of bins, and then use ValueIndexes with that.

Example usage, for a given array a:

```
valIndexes = ValueIndexes(a)
for val in valIndexes.values:
    ndx = valIndexes.getIndexes(val)
    # Do something with all the indexes
```

This is a much faster and more efficient alternative to something like:

```
values = numpy.unique(a)
for val in values:
    mask = (a == val)
    # Do something with the mask
```

The point is that when a is large, and/or the number of possible values is large, this becomes very slow, and can take up lots of memory. Each loop iteration involves searching through a again, looking for a different value. This class provides a much more efficient way of doing the same thing, requiring only one pass through a. When a is small, or when the number of possible values is very small, it probably makes little difference.

If one or more null values are given to the constructor, these will not be counted, and will not be available to the getIndexes() method. This makes it more memory-efficient, so it doesn't store indexes of a whole lot of nulls.

A ValueIndexes object has the following attributes:

- **values** Array of all values indexed
- **counts** Array of counts for each value
- **nDims** Number of dimensions of original array
- **indexes** Packed array of indexes
- **start** Starting points in indexes array for each value
- **end** End points in indexes for each value
- **valLU** Lookup table for each value, to find it in the values array without explicitly searching.
- **nullVals** Array of the null values requested.

Limitations: The array index values are handled using unsigned 32bit int values, so it won't work if the data array is larger than 4Gb. I don't think it would fail very gracefully, either.

getIndexes(*val*)
Return a set of indexes into the original array, for which the value in the array is equal to val.

- genindex
- modindex
- search

10.9 fmaskerrors

Exceptions used within fmask

exception fmask.fmaskerrors.**FmaskException**
An exception rasied by Fmask

exception fmask.fmaskerrors.**FmaskFileError**
Data in file is incorrect

exception fmask.fmaskerrors.**FmaskInstallationError**
Problem with installation of Fmask or a required package

exception fmask.fmaskerrors.**FmaskNotSupportedError**
Requested operation is not supported

exception fmask.fmaskerrors.**FmaskParameterError**
Something is wrong with a parameter

exception fmask.fmaskerrors.**Sen2MetaError**
Error with Sentinel-2 metadata

- genindex
- modindex
- search
- modindex
- search

Python Module Index

f

Index

G

I

K

L

M

N

O

P

T

V