Xarray-regex

Release 0.2.2

Clément Haëck

Mar 25, 2021

CONTENTS:

Welcome to the Xarray-regex package documentation !

Xarray-regex allows to find files based on regular expressions, in order to feed them to Xarray. It allows to easily create regular expressions using 'Matchers', to fix some elements of the expressions to select only certain files, and to easily retrieve information from filenames.

CHAPTER

ONE

FINDING FILES

The main entry point of this package is the `FileFinder` class. This is the object that will find files according to a regular expression. An instance is created using the root directory containing the files, and a pre-regular expression (abbreviated pre-regex) that will be transformed into a proper regex later.

When asking to find files, the finder will first create a regular-expression out of the pre-regex. It will then recursively find *all* files in the root directory and its subfolders, though not descending deeper than `FileFinder.max_depth_scan` folders (default is 3). The finder only keeps files that match the regex. The files can be retrieved using `FileFinder.get_files()`.

1.1 Pre-regex

The pre-regex specifies the structure of the filenames relative to the root directory. It is a regular expression with the added feature of *matchers*.

A matcher is a part of the filename that vary from file to file. In the pre-regex, it is enclosed by parenthesis and preceded by '*%*'. It is represented by the `xarray_regex.matcher.Matcher` class.

Warning: Anything outside matchers in the pre-regex will be considered constant across files. For example, if we have daily files '*sst_2003-01-01.nc*' with the date changing for each file, we could use the regex '*sst_.*.nc*' which would match correctly all files, but the finder would in fact consider that *all* files are '*sst_2003-01-01.nc*' (the first file found).

Inside the matchers parenthesis can be indicated multiple elements, separated by colons:

- a group name (optional)
- a name that will dictate the matcher regex using a correspondance table
- a custom regex if correspondances are not enough (optional)
- a keyword that will discard that matcher when retrieving information from a filename (optional)

The full syntax is as follows: '*%([group:]name[:custom=custom regex:][:discard])*'.

Note: The matchers are uniquely identified by their index in the pre-regex (starting at 0).

1.1.1 Name

The name of the matcher will dictate the regex used for that matcher (unless overriden by a custom regex), and how it will be used by functions that retrieve information from the filename. The *Matcher.NAME_RGX* class attribute will make the correspondance between name and regex:

Name	Regex	
idx	\d*	Index
text	[a-zA-Z]*	Letters
char	\S*	Character
F	%Y-%m-%d	Date (YYYY-MM-DD)
x	%Y%m%d	Date (YYYYMMDD)
X	%H%M%S	Time (HHMMSS)
Y	\d\d\d\d	Year (YYYY)
m	\d\d	Month (MM)
d	\d\d	Day of month (DD)
j	\d\d\d	Day of year (DDD)
B	[a-zA-Z]*	Month name
H	\d\d	Hour 24 (HH)
M	\d\d	Minute (MM)
S	\d\d	Seconds (SS)

This table *mostly* follows the strftime format specifications.

So for example, '*%(Y)*' will be replaced by a regex searching for 4 digits, and *library.get_date* will use it to find the date year.

A letter preceded by a percent sign '*%*' in the regex will be recursively replaced by the corresponding name in the table. This can be used in the custom regex. This still counts as a single matcher and its name will not be changed, only the regex. So '*%x*' will be replaced by '*%Y%m%d*', in turn replaced by '*\d\d\d\d\d\d\d\d*'. A percentage character in the regex is escaped by another percentage (*'%%'*).

1.1.2 Custom regex

All the possible use cases are not covered in the *NAME_RGX* table and one might want to use a specific regex:

```
sst_%(Y:custom=\d\d:)-%(doy:custom=\d\d\d:discard)
```

Warning: The custom regex must be terminated with a colon.

1.1.3 Discard keyword

doc:*Information can be retrieved<retrieving_values>* from the matches in the filename, but one might discard a matcher so it would not be used. For example for a file of weekly averages with a filename indicated the start and end dates of the average, we might want to only recover the starting date:

```
sst_%(x)-%(x:discard)
```

1.2 Nesting files

Found files can be retrieved using `FileFinder.get_files()`. This outputs a list of all files (relative to the finder root, or as absolute paths), sorted alphabetically. They can also be returned as a nested lists of filenames. This is aimed to work with xarray.open_mfdataset(), which will merge files in a specific order when supplied a nested list of files.

To this end, one must specify group names to the *nested* argument of the same function. The rightmost group will correspond to the innermost level.

An example is available in the *examples*.

CHAPTER

TWO

RETRIEVE INFORMATION

As some metadata might only be found in the filenames, FileFinder offer the possibility to retrieve it easily using the *FileFinder.get_matches()* method. Thus, a filename can be matched against the regex of the finder and returns a list of the matches found.

The package supply the function *library.get_date* to retrieve a datetime object from those matches:

```
from xarray_regex.library import get_date
matches = finder.get_matches(filename)
date = get_date(matches)
```

2.1 Combine with Xarray

Retrieving information can be used when opening multiple files with xarray.open_mfdataset().

FileFinder.get_func_process_filename() will turn a function into a suitable callable for the *preprocess* argument of *xarray.open_mfdataset*. The function should take an *xarray.Dataset*, a filename, and a *FileFinder*, and eventual additional arguments as input, and return an *xarray.Dataset*. This allows to use the finder and the dataset filename in the pre-processing. This following example show how to add a time dimension using the filename to find the timestamp:

```
def preprocess(ds, filename, finder):
  matches = finder.get_matches(filename)
  date = library.get_date(matches)

  ds = ds.assign_coords(time=pd.to_datetime([value]))
  return ds

ds = xr.open_mfdataset(finder.get_files(),
                       preprocess=f.get_func_process_filename(preprocess))
```

Note: The filename path sent to the function is automatically made relative to the finder root directory, so that it can be used directly with *FileFinder.get_matches()*.

CHAPTER

THREE

FIX MATCHERS

The package allows to dynamically change the regular expression easily. This is done by replacing matchers in the regular expression by a given string, using the *FileFinder.fix_matcher()* method.

Matchers to replace can be selected either by their index in the pre-regex (starting from 0), or by their name, or their group and name following the syntax *'group:name'*. If using a matcher name or group+name, multiple matchers can be fixed to the same value at once.

For instance, when using the following pre-regex:

```
'%(time:m)/SST_%(time:Y)%(time:m)%(time:d)\.nc'
```

we can keep only the files corresponding to january using any of:

```
finder.fix_matcher(0, '01')
finder.fix_matcher('m', '01')
finder.fix_matcher('time:m', '01')
```

We could also select specific days using a regular expression:

```
finder.fix_matcher('d', '01|03|05|07')
```

This would create the following regular expression:

```
'(\d\d)/SST_(\d\d\d\d)(\d\d)(01|03|05|07)\.nc'
```

CHAPTER

FOUR

EXAMPLES

4.1 Plain time serie

Here the files are all in the same folder. Only the timestamp differ from one file to the other:

```
Data
├── SSH
│   ├── SSH_20070101.nc
│   ├── SSH_20070109.nc
│   └── ...
└── SST
    ├── A_2007001_2007008.L3m_8D_sst.nc
    ├── A_2007008_2007016.L3m_8D_sst.nc
    └── ...
```

We will scan for SST files:

```
from xarray_regex import FileFinder, library

root = 'Data/SST'
pregex = 'A_%(Y)%(j)_%(Y)%(j:discard)%(suffix)'
finder = FileFinder(root, pregex, suffix=r'\.L3m_8D_sst\.nc')

files = finder.get_files()
```

We would like to open all these files using Xarray, however the files lacks a defined 'time' dimensions to concatenate all files. To make it work, we can use the 'preprocess' argument of *xarray.open_mfdataset*:

```
def preprocess(ds, filename, finder):
  matches = finder.get_matches(filename)
  date = library.get_date(matches)

  ds = ds.assign_coords(time=pd.to_datetime([value]))
  return ds

ds = xr.open_mfdataset(files,
                       preprocess=f.get_func_process_filename(preprocess))
```

4.2 Nested files

We can scan both variables at the same time but retrieve the files as a *nested list*. We assume the filenames for both variable are structured in the same way. Groups in the pre-regex will define what matchers will be grouped together:

```
pregex = '%(variable:char)/%(variable:char)_%(time:Y)%(time:j)\.nc'
```

We can now group the files by variable or time:

```
>>> finder.get_files(relative=True, nested=['variable'])
[['SSH_20070101.nc',
  'SSH_20070109.nc',
  ...],
 ['SST_20070101.nc',
  'SST_20070109.nc',
  ...]]

>>> finder.get_files(relative=True, nested=['time'])
[['SSH_20070101.nc', 'SST_20070101.nc'],
 ['SSH_20070109.nc', 'SST_20070109.nc'],
 ...]
```

This works for any number of groups in any order.

CHAPTER

FIVE

API

Content

file_finder.FileFinder(root, pregex, ...)	Find files using a regular expression.

Submodules

file_finder	Find files using a pre-regex.
library	Functions to retrieve values from filename.
matcher	Matcher object.

5.1 xarray_regex.matcher

Matcher object.

Classes

Matcher(m, idx)	Manage a matcher inside the pre-regex.

class xarray_regex.matcher.**Matcher**(*m: re.match*, *idx: int = 0*)

Bases: object

Manage a matcher inside the pre-regex.

Parameters

- **m** (*re.match*) – Match object obtained to find matchers in the pre-regex.
- **idx** (*int*) – Index inside the pre-regex.

idx

Index inside the pre-regex.

Type int

group

Group name.

Type str

name
Matcher name.

Type str

custom
If there is a custom regex to use preferentially.

Type bool

rgx
Regex.

Type str

discard
If the matcher should not be used when retrieving values from matches.

Type bool

match
The string that created the matcher *%(match)*.

Type str

NAME_RGX = {'B': '[a-zA-Z]*', 'F': '%Y-%m-%d', 'H': '\\d\\d', 'M': '\\d\\d', 'S': '\\d
Regex str for each type of element.

get_regex() → str
Get matcher regex.

Replace the matchers name by regex from *Matcher.NAME_RGX*. If there is a custom regex, recursively replace '*%*' followed by a single letter by the corresponding regex from *NAME_RGX*. '*%%*' is replaced by a single percentage character.

Raises **KeyError** – Unknown replacement.

set_matcher(*m: re.match*)
Find attributes from match.

Raises

- **NameError** – No name.
- **ValueError** – Empty custom regex.

5.2 xarray_regex.library

Functions to retrieve values from filename.

Content

get_date	Retrieve date from matched elements.
_find_month_number	Find a month number from its name.

xarray_regex.library.**get_date**(*matches: List*, *default_date: Optional[Dict] = None*, *group: Optional[str] = None*) → datetime.datetime
Retrieve date from matched elements.

If any element is not found in the filename, it will be replaced by the element in the default date. If no match is

found, None is returned.

Supports matches with names from *Matcher.NAME_RGX*.

Parameters

- **matches** (*list*) – Matches from a filename, returned by *FileFinder.get_matches*
- **group** (*str*) – If not None, restrict matcher to this group.
- **default_date** (*dict, optional*) – Default date. Dictionnary with keys: year, month, day, hour, minute, and second. Defaults to 1970-01-01 00:00:00

Raises **KeyError** – If no matchers are found to create a date from.:

xarray_regex.library.**_find_month_number**(*name: str*) → int

Find a month number from its name.

Name can be the full name (January) or its three letter abbreviation (jan). The casing does not matter.

5.3 xarray_regex.file_finder

Find files using a pre-regex.

Classes

FileFinder(root, pregex, **replacements)	Find files using a regular expression.

class xarray_regex.file_finder.**FileFinder**(*root: str*, *pregex: str*, ***replacements: str*)

Bases: object

Find files using a regular expression.

Provides abilities to 'fix' some part of the regular expression, to retrieve values from matches in the expression, and to create an advanced pre-processing function for *xarray.open_mfdataset*.

Parameters

- **root** (*str*) – The root directory of a filetree where all files can be found.
- **pregex** (*str*) – The pre-regex. A regular expression with added 'Matchers'. Only the matchers vary from file to file. See documentation for details.
- **replacements** (*str, optional*) – Matchers to replace by a string: *'matcher name' = 'replacement string'*.

max_depth_scan

Maximum authorized depth when descending into filetree to scan files.

Type int

root

The root directory of the finder.

Type str

pregex

Pre-regex.

Type str

regex
Regex obtained from the pre-regex.

Type str

pattern
Compiled pattern obtained from the regex.

Type re.pattern

matchers
List of matchers for this finder, in order.

Type list of Matchers

segments
Segments of the pre-regex. Used to replace specific matchers. *['text before matcher 1', 'matcher 1', 'text before matcher 2, 'matcher 2', ...]*

Type list of str

fixed_matchers
Dictionnary of matchers with a set value. 'matcher index': 'replacement string'

Type dict

files
List of scanned files.

Type list of str

scanned
If the finder has scanned files.

Type bool

create_regex()
Create regex from pre-regex.

find_files()
Find files to scan.

Uses os.walk. Limit search to *max_depth_scan* levels of directories deep. Sort files alphabetically.

Raises

- **AttributeError** – If no regex is set.
- **IndexError** – If no files are found in the filetree.

fix_matcher(*key: Union[int, str]*, *value: str*)
Fix a matcher to a string.

Parameters

- **key** (*int, or str, or tuple of str of lenght 2.*) – If int, is matcher index, starts at 0. If str, can be matcher name, or a group and name combination with the syntax 'group:name'. When using strings, if multiple matchers are found with the same name or group/name combination, all are fixed to the same value.
- **value** (*str*) – Will replace the match for all files.

Raises

- **TypeError** – Value must be a string.:
- **TypeError** – key is neither int nor str.:

fix_matchers (*fixes: Optional[Dict[Union[int, str], str]] = None*)
Fix multiple values at once.

Parameters **fixes** (*dict*) – Dictionnary of matcher key: value. See *fix_matcher()* for details. If None, no matcher will be fixed.

get_files (*relative: bool = False*, *nested: Optional[List[str]] = None*) → List[str]
Return files that matches the regex.

Lazily scan files: if files were already scanned, just return the stored list of files.

Parameters

- **relative** (*bool*) – If True, filenames are returned relative to the finder root directory. If not, filenames are absolute. Defaults to False.
- **nested** (*list of str*) – If not None, return nested list of filenames with each level corresponding to a group in this argument. Last group in the list is at the innermost level. A level specified as None refer to matchers without a group.

Raises **KeyError** – A level in *nested* is not in the pre-regex groups.:

get_func_process_filename (*func: Callable*, *relative: bool = True*, **args*, ***kwargs*) → Callable
Get a function that can preprocess a dataset.

Written to be used as the 'process' argument of *xarray.open_mfdataset*. Allows to use a function with additional arguments, that can retrieve information from the filename.

Parameters

- **func** (*Callable*) – Input arguments (*xarray.Dataset*, filename: *str*, *FileFinder*, *args, **kwargs) Should return a Dataset. Filename is retrieved from the dataset encoding attribute.
- **relative** (If True, *filename* is made relative to finder root.) – This is necessary to match the filename against the finder regex. Defaults to True.
- **args** (*optional*) – Passed to *func* when called.
- **kwargs** (*optional*) – Passed to *func* when called.

Returns Function with the signature of the 'process' argument of *xarray.open_mfdataset*.

Return type Callable

Examples

This retrieve the date from the filename, and add a time dimensions to the dataset with the corresponding value. >>> from xarray_regex import library ... def process(ds, filename, finder, default_date=None): ... matches = finder.get_matches(filename) ... date = library.get_date(matches, default_date=default_date) ... ds = ds.assign_coords(time=[date]) ... return ds ... ... ds = xr.open_mfdataset(finder.get_files(), ... preprocess=finder.get_func_process_filename(... process, default_date={'hour': 12}))

get_matchers (*key: str*) → List[*xarray_regex.matcher.Matcher*]
Return list of matchers corresponding to key.

Parameters **key** (*str*) – Can be matcher name, or group+name combination with the syntax: 'group:name'.

Raises **KeyError** – No matcher found.:

get_matches (*filename: str*, *relative: bool = True*) → Dict[str, Dict]
Get matches for a given filename.

Apply regex to *filename* and return a dictionary of the results.

Parameters

- **filename** – Filename to retrieve matches from.
- **relative** – Is true if the filename is relative to the finder root directory. If false, the filename is made relative before being matched. Default to true.

Returns

[{'match': string matched, 'start': start index in filename, 'end': end index in filename, 'matcher': Matcher object}, . . .]

Return type list of dict

Raises

- **AttributeError** – The regex is empty.:
- **ValueError** – The filename did not match the pattern.:
- **IndexError** – Not as many matches as matchers.:

property n_matchers
Number of matchers in pre-regex.

scan_pregex ()
Scan pregex for matchers.

Add matchers objects to self. Set segments attribute.

set_pregex (*pregex: str*, ***replacements: str*)
Set pre-regex.

Apply replacements.

update_regex ()
Update regex.

Set fixed matchers. Re-compile pattern. Scrap previous scanning.

Source code: https://github.com/Descanonge/xarray-regex

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

Symbols

C

D

F

G

I

M

N

P

R

S

U

X