mock-implementation

December 31, 2019

1 Rough Mock Implementation

This is a Julia notebook which, using some tricks, shows a rough mock implementation of cfgrib in Julia. It goes through:

1. Loading in the data from a grib file via `PyCall` instead of by calling `ecCodes` as that is the easiest solution for this demonstration
2. Using `AxisArrays` to create an object which behaves similarily to `DataArray` in xarrays
3. Using `ImageMetadata` to add in mestadata to the `AxisArrays`, as it does not support metadata by default - this is equivalent to the `Attributes` DataArrays have in xarrays
4. Adding in a custom class `AxisSet` class (collection of AxisArrays), along with metadata for the sets - this is equivalent to a `Dataset` in xarrays

[1]:
```
using Pkg; Pkg.activate("./binder/")
```

```
Activating environment at `~/work/cfgrib-julia-
notebooks/binder/Project.toml`
```

[2]:
```
using AxisArrays
using ImageMetadata
```

1.1 1 - Loading in Data via PyCall

[3]:
```
#  Code file includes the load functions as I have reuse these a few times in
#  other notebooks
include("./src/conversions.jl")
```

[3]:
```
python_cfgrib_load (generic function with 2 methods)
```

[4]:
```
#  Contains the `Coordinates` of the xarray `Dataset`
pythons_dataset_coords = python_cfgrib_coords("./data/era5-levels-members.grib")
```

[4]:
```
OrderedDict{Any,Any} with 7 entries:
  "number"        => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  "time"          => DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00, 2017-01…
  "step"          => 0 nanoseconds
  "isobaricInhPa" => [850, 500]
```

```
  "latitude"      => [90.0, 87.0, 84.0, 81.0, 78.0, 75.0, 72.0, 69.0, 66.0, 63.…
  "longitude"     => [0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0, 27.0  ……
  "valid_time"    => DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00, 2017-01…
```

[5]:
```
#  Order of coordinates is lost in the Python-Julia conversion, so I have to
#  order them manually here
coords = (
    Axis{:number}(pythons_dataset_coords["number"]),
    Axis{:time}(pythons_dataset_coords["time"]),
    Axis{:isobaricInhPa}(pythons_dataset_coords["isobaricInhPa"]),
    Axis{:latitude}(pythons_dataset_coords["latitude"]),
    Axis{:longitude}(pythons_dataset_coords["longitude"])
)
```

[5]:
```
(Axis{:number,Array{Int64,1}}([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
Axis{:time,Array{DateTime,1}}(DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00,
2017-01-02T00:00:00, 2017-01-02T12:00:00]),
Axis{:isobaricInhPa,Array{Int64,1}}([850, 500]),
Axis{:latitude,Array{Float64,1}}([90.0, 87.0, 84.0, 81.0, 78.0, 75.0, 72.0,
69.0, 66.0, 63.0  …  -63.0, -66.0, -69.0, -72.0, -75.0, -78.0, -81.0, -84.0,
-87.0, -90.0]), Axis{:longitude,Array{Float64,1}}([0.0, 3.0, 6.0, 9.0, 12.0,
15.0, 18.0, 21.0, 24.0, 27.0  …  330.0, 333.0, 336.0, 339.0, 342.0, 345.0,
348.0, 351.0, 354.0, 357.0]))
```

[6]:
```
#  Contains the `Attributes` of the xarray `Dataset`
python_dataset_attributes = python_cfgrib_metadata("./data/era5-levels-members.
 ↪grib")
```

[6]:
```
OrderedDict{Any,Any} with 7 entries:
  "Conventions"            => "CF-1.7"
  "history"                => "2019-12-31T02:18:03 GRIB to CDM+CF via cfgrib-0.…
  "GRIB_edition"           => 1
  "GRIB_centreDescription" => "European Centre for Medium-Range Weather Forecas…
  "institution"            => "European Centre for Medium-Range Weather Forecas…
  "GRIB_subCentre"         => 0
  "GRIB_centre"            => "ecmf"
```

[7]:
```
#  Contains the `values` in both `DataArray`s held by the xarray `Dataset`
python_data_t = python_cfgrib_data("./data/era5-levels-members.grib", "t")
python_data_z = python_cfgrib_data("./data/era5-levels-members.grib", "z")
python_datas = Dict(
    :t => python_data_t,
    :z => python_data_z
)

#  Contains the `Attributes` in both `DataArray`s held by the xarray `Dataset`
python_meta_t = python_cfgrib_metadata("./data/era5-levels-members.grib", "t")
```

```
python_meta_z = python_cfgrib_metadata("./data/era5-levels-members.grib", "z")
python_metas = Dict(
    :t => python_meta_t,
    :z => python_meta_z
);
```

1.2 2 - Creating the AxisArray

[8]:
```
println("Data variable `t` is an $(typeof(python_data_t)) of size␣
↪$(size(python_data_t))")
println("With coordinates of $([(axisnames(coord), length(coord)) for coord in␣
↪coords])")
```

```
Data variable `t` is an Array{Float32,5} of size (10, 4, 2, 61, 120)
With coordinates of Tuple{Tuple{Symbol},Int64}[((:number,), 10), ((:time,), 4),
((:isobaricInhPa,), 2), ((:latitude,), 61), ((:longitude,), 120)]
```

[9]:
```
#  Simple to create, only requires an arrray and coordinates with data that
#  matches the lengths of the array
axisarray_t = AxisArray(python_data_t, coords);
```

```
5-dimensional AxisArray{Float32,5,...} with axes:
    :number, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    :time, DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00, 2017-01-02T00:00:00, 2017-01-02T
    :isobaricInhPa, [850, 500]
    :latitude, [90.0, 87.0, 84.0, 81.0, 78.0, 75.0, 72.0, 69.0, 66.0, 63.0  …  -63.0, -66.0, -
    :longitude, [0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0, 27.0  …  330.0, 333.0, 336.
And data, a 10×4×2×61×120 Array{Float32,5}:
[:, :, 1, 1, 1] =
 252.663  251.854  251.142  252.044
 252.277  251.73   250.983  252.548
 252.449  251.733  250.829  252.358
 252.283  252.258  250.811  252.494
 252.049  251.622  250.824  251.921
 252.376  252.039  251.123  252.284
 252.131  251.842  251.281  252.17
 252.173  251.64   251.116  252.252
 251.714  251.768  251.422  252.368
 251.881  251.83   250.935  252.599

 ...
```

1.3 Using ImageMetadata to add in Attributes

AxisArrays do not have metadata, here we use an `ImageMeta` array instead as this adds a convenient metadata wrapper on top and does all of the multiple dispatch to base and AxisArrays methods

[10]:
```
#  Wraps AxisArray with a dictionary of `properties` - equivalent to `Attributes`
imagemeta_t = ImageMeta(axisarray_t, convert(Dict{String,Any}, python_meta_t))
```

[10]:
```
Float32 ImageMeta with:
  data: 5-dimensional AxisArray{Float32,5,…} with axes:
    :number, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    :time, DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00,
2017-01-02T00:00:00, 2017-01-02T12:00:00]
    :isobaricInhPa, [850, 500]
    :latitude, [90.0, 87.0, 84.0, 81.0, 78.0, 75.0, 72.0, 69.0, 66.0, 63.0  …
-63.0, -66.0, -69.0, -72.0, -75.0, -78.0, -81.0, -84.0, -87.0, -90.0]
    :longitude, [0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0, 27.0  …
330.0, 333.0, 336.0, 339.0, 342.0, 345.0, 348.0, 351.0, 354.0, 357.0]
And data, a 10×4×2×61×120 Array{Float32,5}
  properties:
    GRIB_typeOfLevel: isobaricInhPa
    long_name: Temperature
    GRIB_dataType: an
    GRIB_totalNumber: 10
    GRIB_jScansPositively: 0
    GRIB_name: Temperature
    GRIB_gridType: regular_ll
    GRIB_Ny: 61
    GRIB_longitudeOfLastGridPointInDegrees: 357.0
    GRIB_stepUnits: 1
    GRIB_jPointsAreConsecutive: 0
    standard_name: air_temperature
    GRIB_jDirectionIncrementInDegrees: 3.0
    GRIB_gridDefinitionDescription: Latitude/Longitude Grid
    GRIB_latitudeOfLastGridPointInDegrees: -90.0
    GRIB_shortName: t
    GRIB_missingValue: 9999
    GRIB_stepType: instant
    GRIB_numberOfPoints: 7320
    GRIB_NV: 0
    GRIB_latitudeOfFirstGridPointInDegrees: 90.0
    GRIB_cfName: air_temperature
    units: K
    GRIB_cfVarName: t
    GRIB_Nx: 120
    GRIB_iDirectionIncrementInDegrees: 3.0
```

```
    GRIB_iScansNegatively: 0
    GRIB_paramId: 130
    GRIB_longitudeOfFirstGridPointInDegrees: 0.0
    GRIB_units: K
```

1.4 4- AxisArrays with Dataset Structure

In python xarrays the data structure is:

- a `DataArray`, which holds a single multi-dimensional variable, its coordinates, and a metadata dictionary
- a `Dataset`, which holds multiple variables that potentially share the same coordinates

AxisArrays does not follow this model, it only has support for `AxisArray` (equivalent to `DataArray`), but no `AxisSet` exists.

We need to create a similar class to the Dataset class described in the xarray docs

[11]:
```
mutable struct AxisSet
    coords::NTuple{N, Axis} where N
    attrs::OrderedDict
    data::Dict{Symbol,ImageMeta }

    AxisSet(coords::NTuple{N, Axis} where N, attrs::OrderedDict) = new(coords,␣
 ↪attrs)
end
```

The above class is made such that all of the `AxisArrays` will share the same coordinates, it also allows for 'top-level' metadata (equivalent to `Attributes` on an xarray Dataset)

[12]:
```
function AxisSet(coords::NTuple{N, Axis} where N, attrs::OrderedDict,
                 data::Dict{Symbol, <:Array}, meta::Dict{Symbol, <:OrderedDict})

    if any([haskey(data, k) for k in (:coords, :attrs, :data)])
        throw(ErrorException("Data keys cannot be any of (:coords, :attrs, :
 ↪data)"))
    end

    axisset = AxisSet(coords, attrs)
    imagemetas = Dict{Symbol,ImageMeta}()
    for (v, d) in data
        #  Create an ImageMeta array for each of the data variables
        #  with the respective metadata attached
        axisarray = AxisArray(d, axisset.coords)
        imagemetas[v] = ImageMeta(axisarray, convert(Dict{String,Any},␣
 ↪python_metas[v]))
    end
```

```
    axisset.data = imagemetas

    return axisset
end
```

[12]:
```
AxisSet
```

[13]:
```
axisset = AxisSet(coords, python_dataset_attributes, python_datas,␣
↪python_metas);
```

[14]:
```
#  Hacky way to allow for convenient data variable access like `obj.key`
#  instead of the longer `obj.data.key`
function Base.getproperty(obj::AxisSet, sym::Symbol)
    if sym in keys(getfield(obj, :data))
        return getindex(getfield(obj, :data), sym)
    else
        return getfield(obj, sym)
    end
end
```

[15]:
```
axisset.t
```

[15]:
```
Float32 ImageMeta with:
  data: 5-dimensional AxisArray{Float32,5,…} with axes:
    :number, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    :time, DateTime[2017-01-01T00:00:00, 2017-01-01T12:00:00,
2017-01-02T00:00:00, 2017-01-02T12:00:00]
    :isobaricInhPa, [850, 500]
    :latitude, [90.0, 87.0, 84.0, 81.0, 78.0, 75.0, 72.0, 69.0, 66.0, 63.0  …
-63.0, -66.0, -69.0, -72.0, -75.0, -78.0, -81.0, -84.0, -87.0, -90.0]
    :longitude, [0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0, 27.0  …
330.0, 333.0, 336.0, 339.0, 342.0, 345.0, 348.0, 351.0, 354.0, 357.0]
And data, a 10×4×2×61×120 Array{Float32,5}
  properties:
    GRIB_typeOfLevel: isobaricInhPa
    long_name: Temperature
    GRIB_dataType: an
    GRIB_totalNumber: 10
    GRIB_jScansPositively: 0
    GRIB_name: Temperature
    GRIB_gridType: regular_ll
    GRIB_Ny: 61
    GRIB_longitudeOfLastGridPointInDegrees: 357.0
    GRIB_stepUnits: 1
    GRIB_jPointsAreConsecutive: 0
    standard_name: air_temperature
    GRIB_jDirectionIncrementInDegrees: 3.0
```

```
GRIB_gridDefinitionDescription: Latitude/Longitude Grid
GRIB_latitudeOfLastGridPointInDegrees: -90.0
GRIB_shortName: t
GRIB_missingValue: 9999
GRIB_stepType: instant
GRIB_numberOfPoints: 7320
GRIB_NV: 0
GRIB_latitudeOfFirstGridPointInDegrees: 90.0
GRIB_cfName: air_temperature
units: K
GRIB_cfVarName: t
GRIB_Nx: 120
GRIB_iDirectionIncrementInDegrees: 3.0
GRIB_iScansNegatively: 0
GRIB_paramId: 130
GRIB_longitudeOfFirstGridPointInDegrees: 0.0
GRIB_units: K
```

[16]:
```
#  Metadata access like in ImageMetadata
ImageMetadata.properties(obj::AxisSet) = obj.attrs

#  Accesses equivalent of dataset-level attributes in xarrays
properties(axisset)
```

[16]:
```
OrderedDict{Any,Any} with 7 entries:
  "Conventions"            => "CF-1.7"
  "history"                => "2019-12-31T02:18:03 GRIB to CDM+CF via cfgrib-0.…
  "GRIB_edition"           => 1
  "GRIB_centreDescription" => "European Centre for Medium-Range Weather Forecas…
  "institution"            => "European Centre for Medium-Range Weather Forecas…
  "GRIB_subCentre"         => 0
  "GRIB_centre"            => "ecmf"
```

[17]:
```
properties(axisset.t)
```

[17]:
```
Dict{String,Any} with 30 entries:
  "GRIB_typeOfLevel"                      => "isobaricInhPa"
  "long_name"                             => "Temperature"
  "GRIB_dataType"                         => "an"
  "GRIB_totalNumber"                      => 10
  "GRIB_jScansPositively"                 => 0
  "GRIB_name"                             => "Temperature"
  "GRIB_gridType"                         => "regular_ll"
  "GRIB_Ny"                               => 61
  "GRIB_longitudeOfLastGridPointInDegrees" => 357.0
  "GRIB_stepUnits"                        => 1
  "GRIB_jPointsAreConsecutive"            => 0
```

```
  "standard_name"                            => "air_temperature"
  "GRIB_jDirectionIncrementInDegrees"        => 3.0
  "GRIB_gridDefinitionDescription"           => "Latitude/Longitude Grid"
  "GRIB_latitudeOfLastGridPointInDegrees"    => -90.0
  "GRIB_shortName"                           => "t"
  "GRIB_missingValue"                        => 9999
  "GRIB_stepType"                            => "instant"
  "GRIB_numberOfPoints"                      => 7320
  "GRIB_NV"                                  => 0
  "GRIB_latitudeOfFirstGridPointInDegrees" => 90.0
  "GRIB_cfName"                              => "air_temperature"
  "units"                                    => "K"
  "GRIB_cfVarName"                           => "t"
  "GRIB_Nx"                                  => 120
                                            =>
```

[18]:
```
#  Basic show which looks similar to what xarrays displays
function Base.show(io::IO, m::AxisSet)
    println(io, "AxisSet with: $(length(m.data)) data variables:")

    dimensions = ["$(axisnames(x)[1]): $(length(x))" for x in m.coords]
    println(io, "  Dimensions: \t $(join(dimensions, ", "))")

    println(io, "  Data variables:")
    [println(io, "  \t$k: $(typeof(m.data[:t]).types[1].types[1])$(size(d))")␣
 ↪for (k, d) in m.data]

    attributes = properties(m)
    println(io, "  Attributes:")
    println(io, "  \t$(typeof(attributes)) with $(length(attributes)) entries:")
    attributes_str = [join("$(repr(k)) => $(repr(v))") for (k, v) in␣
 ↪properties(axisset)]
    if length(attributes_str) > 5; attributes_str = attributes_str[1:5]; end
    [println(io, "  \t  $str") for str in attributes_str]

    return nothing
end
```

[19]:
```
axisset
```

[19]:
```
AxisSet with: 2 data variables:
  Dimensions:    number: 10, time: 4, isobaricInhPa: 2, latitude: 61, longitude:
120
  Data variables:
        z: Array{Float32,5}(10, 4, 2, 61, 120)
        t: Array{Float32,5}(10, 4, 2, 61, 120)
  Attributes:
```

```
        OrderedDict{Any,Any} with 7 entries:
          "Conventions" => "CF-1.7"
          "history" => "2019-12-31T02:18:03 GRIB to CDM+CF via
cfgrib-0.9.7.3/ecCodes-2.13.1 with {\"source\": \"/home/roscar/work/cfgrib-
julia-notebooks/data/era5-levels-members.grib\", \"filter_by_keys\": {},
\"encode_cf\": [\"parameter\", \"time\", \"geography\", \"vertical\"]}"
          "GRIB_edition" => 1
          "GRIB_centreDescription" => "European Centre for Medium-Range Weather
Forecasts"
          "institution" => "European Centre for Medium-Range Weather Forecasts"
```

Overall this is all extremely rough and would not reliable to use, however it does demonstrate that it is in principle relatively simple to add the required functionality on top of existing packages like `AxisArrays` and `ImageMetadata`.

However I believe that none of these additional features, like metadata and collections of AxisArrays, should exist/be maintained in this cfgrib package.

My plan is to develop them here, as they are essential features which should be a part of any package aiming to provide similar functionality to xarray, and then merge any developments back into their respective packages.

This will help show the development community that these features are desired, should be maintained, and should be present in any future packages like the proposed replacement of AxisArrays, or `DimensionalArrayTraits`, or any other similar multidimensional-arrays-with-coordinates package.