epygram-1.4.10 *cheatsheet*

Please refer to the complete HTML documention for description of the arguments and options of each method.

Miscellaneous	
`epygram.init_env()`	initialize environment for inner libraries
`epygram.showconfig()`	show config variables, tunable in `$HOME/.epygram/userconfig.py`
`% epy_doc.py -o [-s cartoplot]`	open the `epygram` documentation in a web browser [option: search for "`cartoplot`"]

Resources	
`r = epygram.formats.resource("path/to/file", "r")`	open an existing file with "r" = read mode (or "a" = append)
`r = epygram.formats.resource("path/to/file", "w", fmt="FA"`	open a new file with "w" = write mode; required format to be specified
`r.format, epygram.formats.guess("path/to/file")`	get the format of a resource, a file
`r.what()`	comprehensive description of the resource contents
`r.listfields()`	list the fields contained in the resource
`r.find_fields_in_resource(dict(level=850))`	filter the fields list within resource (GRIB)
`r.find_fields_in_resource("S001*")`	filter the fields list within resource (FA)
`f = r.readfield(dict(shortName="2t"))`	read the field uniquely identified by fid
`r.extract_profile("S*TEMPERATURE", lon, lat)`	extract a vertical profile from the series of horizontal fields in resource (not all formats)
`r.extract_section("S*HUMI.SPECIFI", (lon1, lat1), (lon2, lat2))`	extract a vertical section ...
`r.writefield(f)`	write an `epygram` field `f` in resource
Meta-Resources	
`r = epygram.resources.meta_resource("a_file", "r", "CL")`	a *meta* "**C**ombine**L**evels" resource (takes resource or filename)
`r = epygram.resources.meta_resource([r1, r2,], "r", "MV")`	a *meta* "**M**ulti**V**alidities" resource (id.)

Fields	
`f.fid, f.validity, f.geometry, f.spectral_geometry`	access to inner metadata objects
`f.copy(), f.deepcopy()`	get a shallow (all inner data/metadata remain common) or deep copy of the field
`f.what()`	comprehensive description of the field metadata
`f.data`	r/w access to the field data array
`f.getvalue_ll(lon, lat, interpolation=...)`	get value of field at *lon/lat* point (nearest point by default)
`f.min(), f.std(), ..., f.stats()`	get basic statistics about the field
`h = f + g`	do an operation, losing fid and validity metadata
`f.operation("+", other_field_or_scalar)`	do an operation, without losing metadata (in place)
`f.sp2gp()`	convert a spectral field to gridpoint (in place)
`f.gp2sp(a_spectral_geometry)`	convert a gridpoint field to the spectral space defined by `a_spectral_geometry`
`f.shave(minval=0.)`	cut values lower than `minval`, resp. $\geq$ `maxval` (in place)
`f.resample(target_geometry)`	resample field onto `target_geometry`
`f.resample_on_regularll(borders, resolution)`	resample field onto regular lon/lat grid
`f.global_shift_center(180.)`	for global lon/lat regular grids, do a zonal *modulo* rotation of the grid (in place)
`f.extract_zoom(dict(lonmin=-2.5, ...))`	extract a zoom given lon/lat borders
`f.extract_point(lon, lat)`	extract a PointField (from a H2DField only)
`f.extract_subdomain(subdomain_geometry)`	provided that `subdomain_geometry` is contained within `f.geometry` (e.g. a profile from a 3D Field, cf. `g.make_profile_geometry`)
`f.dump_to_nc("newoutfile.nc", variablename="t2m")`	dump any field to netCDF file, using specified variable name
Fields data representation	
`fig, ax = f.cartoplot()`	plot the field (w/ cartopy) and get back the underlying `matplotlib` figure and axis
`fig, ax = f.histogram()`	compute and plot a data histogram
`spectrum = f.dctspectrum()`	compute a DCT spectrum of data
`fig, ax = spectrum.plotspectrum()`	and plot it

n-D Fields	
`f.extend(g)`	extend field `f` along time dimension with field `g` (in place)
`f.time_reduce("mean")`	do a reduction along time dimension (in place)
`f.time_smooth(3)`	do a smoothing along time dimension (in place)
`f.decumulate()`	do a decumulation operation along time dimension (in place)
`f.getvalidity(index_or_validity)`	extract a sub-field at requested validity
`f.getlevel(level)`	extract a sub-field at requested level
Vector Fields	
`wind = epygram.fields.make_vector_field(fx, fy)`	make a vector field from two vectorial components Fields (typically u/v for wind)
`wind = epygram.fields.psikhi2uv(psi, khi)`	make a u/v vector field from stream-function/velocity-potential components [1]
`wind.reproject_wind_on_lonlat()`	reproject x/y coordinates of vector onto lon/lat (in place)
`ff = wind.to_module()`	compute the module of the vector field
`d = wind.compute_direction()`	compute the direction of vector field
`wind.map_factorize(reverse=False)`	multiply/divide by the map factor (in place)
`vor, div = wind.compute_vordiv()`	compute vorticity and divergence from u/v components [1]

Geometries	
`g.name, g.structure, g.grid, g.dimensions, [g.projection]`	what defines a geometry
`g.what()`	comprehensive description of the geometry
`lons, lats = g.get_lonlat_grid()`	get the longitudes and latitudes arrays of all gridpoints of the geometry
`g.make_point_geometry(lon, lat)`	extract a point geometry from the geometry, keeping as much properties as possible
`g.make_profile_geometry(lon, lat)`	extract a profile geometry from the geometry, keeping as much properties as possible
`g.make_section_geometry((lon1, lat1), (lon2, lat2))`	extract a section geometry from the geometry, keeping as much properties as possible
`g.gimme_corners_ll()`	get the lon/lat position of the grid's corners
`g.point_is_inside_domain(lon, lat)`	check that a point is inside the domain
`g.plotgeometry()`	plot the borders of the domain
`g.distance((lon1, lat1), (lon2, lat2))`	compute the distance between two points
`g.linspace((lon1, lat1), (lon2, lat2), N)`	compute a series of N linearly spaced points between two ends
`g.azimuth((lon1, lat1), (lon2, lat2))`	compute an azimuth from one point to another
`g.default_cartopy_CRS()`	compute a `cartopy` CRS object, used to map geographical coordinates to a `Matplotlib` axis plotting coordinates
`g.nearest_points(lon, lat, request=dict(n="2*2"))`	get the points of the grid nearest to (lon, lat)
`g.ij2ll(), g.ll2ij()`	conversion methods between (i, j) indexes of the grid and (lon, lat) coordinates
`g.map_factor(*position)`	get the map factor at given position (depend on the geometry)
`g.map_factor_field()`	get the map factor over the whole grid as a Field
Gauss grids	
`g.*resolution*(...)`	various local resolution computation methods
Vertical Geometry	
`g.vcoordinate`	the vertical coordinate within a Geometry
`from epygram.geometries.VGeometry import <function>`	
`hybridP2pressure(...), hybridP2altitude(...)`	vertical coordinate type conversion functions
`hybridH2pressure(...), hybridH2altitude(...)`	vertical coordinate type conversion functions
`hybridP_coord_and_surfpressure_to_3D_pressure_field( hybridP_geometry, Psurf, vertical_mean)`	compute a 3D pressure field from a hybrid-pressure coordinate and a surface pressure field
`hybridP_coord_to_3D_altitude_field( hybridP_geometry, Psurf, vertical_mean, t3D, q3D)`	compute a 3D altitude field from a hybrid-pressure coordinate, a surface pressure field and 3D temperature and specific humidity fields

Validities	
`f.validity.get()`	instantaneous validity (`datetime.datetime`)
`f.validity.getbasis()`	initial (basis) validity (`datetime.datetime`)
`f.validity.term()`	term (`datetime.timedelta`)
`f.validity.cumulativeduration()`	cumulative duration (`datetime.timedelta`)
`f.validity.set(...)`	modify any of the validity attributes (with `datetime` objects)

[1] uses spectral derivative operators, ⇒ components must be spectral