

# epygram-2.0.1 *cheatsheet*

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

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

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

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

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

Geometries	
<code>g.name, g.structure, g.grid, g.dimensions, [g.projection]</code>	what defines a geometry
<code>g.what()</code>	comprehensive description of the geometry
<code>lons, lats = g.get_lonlat_grid()</code>	get the longitudes and latitudes arrays of all gridpoints of the geometry
<code>g.make_point_geometry(lon, lat)</code>	extract a point geometry from the geometry, keeping as much properties as possible
<code>g.make_profile_geometry(lon, lat)</code>	extract a profile geometry from the geometry, keeping as much properties as possible
<code>g.make_section_geometry((lon1, lat1), (lon2, lat2))</code>	extract a section geometry from the geometry, keeping as much properties as possible
<code>g.gimme_corners_ll()</code>	get the lon/lat position of the grid's corners
<code>g.point_is_inside_domain(lon, lat)</code>	check that a point is inside the domain
<code>g.plotgeometry()</code>	plot the borders of the domain
<code>g.distance((lon1, lat1), (lon2, lat2))</code>	compute the distance between two points
<code>g.linspace((lon1, lat1), (lon2, lat2), N)</code>	compute a series of N linearly spaced points between two ends
<code>g.azimuth((lon1, lat1), (lon2, lat2))</code>	compute an azimuth from one point to another
<code>g.default_cartopy_CRS()</code>	compute a <b>cartopy</b> CRS object, used to map geographical coordinates to a <b>Matplotlib</b> axis plotting coordinates
<code>g.nearest_points(lon, lat, request=dict(n="2*2"))</code>	get the points of the grid nearest to (lon, lat)
<code>g.ij2ll(), g.ll2ij()</code>	conversion methods between (i, j) indexes of the grid and (lon, lat) coordinates
<code>g.map_factor(*position)</code>	get the map factor at given position (depend on the geometry)
<code>g.map_factor_field()</code>	get the map factor over the whole grid as a Field

Gauss grids	
<code>g.*resolution*(...)</code>	various local resolution computation methods
Vertical Geometry	
<code>g.vcoordinate</code>	the vertical coordinate within a Geometry
<code>from epygram.geometries.VGeometry import &lt;function&gt;</code>	
<code>hybridP2pressure(...), hybridP2altitude(...)</code>	vertical coordinate type conversion functions
<code>hybridH2pressure(...), hybridH2altitude(...)</code>	vertical coordinate type conversion functions
<code>hybridP_coord_and_surfpressure_to_3D_pressure_field(hybridP_geometry, Psurf, vertical_mean)</code>	compute a 3D pressure field from a hybrid-pressure coordinate and a surface pressure field
<code>hybridP_coord_to_3D_altitude_field(hybridP_geometry, Psurf, vertical_mean, t3D, q3D)</code>	compute a 3D altitude field from a hybrid-pressure coordinate, a surface pressure field and 3D temperature and specific humidity fields

Validities	
<code>f.validity.get()</code>	instantaneous validity ( <b>datetime.datetime</b> )
<code>f.validity.getbasis()</code>	initial (basis) validity ( <b>datetime.datetime</b> )
<code>f.validity.term()</code>	term ( <b>datetime.timedelta</b> )
<code>f.validity.cumulativeveduration()</code>	cumulative duration ( <b>datetime.timedelta</b> )
<code>f.validity.set(...)</code>	modify any of the validity attributes (with <b>datetime</b> objects)

<sup>1</sup>uses spectral derivative operators,  $\Rightarrow$  components must be spectral