pysac Documentation

Release 0.2.dev209

Stuart Mumford

January 09, 2015

Contents

PySAC is a python packaged designed to make it easier to analyse and plot output from VAC and SAC in python.

pySAC currently supports routines for VAC and SAC Input / Output as well as 2D plotting.

Contents:

Part I

pySAC Guide

pySAC is a python package that is designed to enable analysis and plotting of VAC (Verstatile Advection Code) and SAC (Sheffield Advanced Code) as well as smaug, the GPGPU port of SAC.

CHAPTER 1

Installation

pySAC is contained inside a DVS version control system called git, with a hosted repository on BitBucket. To obtain the latest version of the source code you can clone the git repository thus:

```
git clone git@bitbucket.org:swatsheffield/pysac.git
```

which you can then install using the included python setup script. There are two options at this point, you can install the code permantly or you can install it in 'develop' mode which enable you to edit the code in the directory in which you have just cloned it. To install in devlop mode do the following:

```
python setup.py develop
```

which, under linux you will need to run as root.

The following python packages are required to use pySAC:

- matplotlib >= 1.1
- h5py
- numpy
- scipy

1.1 Basic Usage

Once installed pySAC can be used by importing the component you wish to use in a python script or interpreter to open a SAC file you can now do:

```
myfile = pysac.io.SACfile("mysacfile.out")
```

myfile now contains various routines to read the data in the file, as we shall see in more detail later.

CHAPTER 2

Input/Output

The first role of pySAC is to enable the reading and writing of SAC or VAC files in either the VAC style fortran unformatted binary file type or the newer HDF5 file structures.

The basic components of any VAC/SAC filetype are:

- x array: An array containing all coordinates at any point in the domain
- y array: An array containing all the computed variables
- **header: A header, stored in some form consisting of at:**
  - filehead - A not so descriptive file string
  - iteration number
  - physical time
  - number of dimensions
  - number of constant parameters in the equations
  - number of physcical varibles being solved for (in w array)
  - nx
  - the equation parameters
  - names for the varibles in the w array

pySAC implements a 'VACdata' class that will read either the VAC fortran type files or HDF5 files containing VAC/SAC data. There is a specification of VACdata to SACdata for SAC files with magnetic field and pertubation and background varibles in the w array.

SACdata implemetes methods to extract primative variables such as temperature and pressure.

2.1 File Description

2.1.1 Header Contents

The header of VAC / SAC files contains the same information irrespective of the file type. When saving a file using pySAC the routines will automatically reformat the header to fit the file. The keys in the header and examples are listed below:

```
header = {
            'filehead': '3Dmhd_33',
            'ndim': 3,
            'it': 0,
            't': 0,
            'nw': 13,
            'nx': [128,128,128],
            'varnames': ['z', 'x', 'y', 'h', 'm1', 'm2', 'm3', 'e', 'b1', 'b2', 'b3',
                         'eb', 'rhob', 'bg1', 'bg2', 'bg3', 'gamma', 'eta',
                         'grav1', 'grav2', 'grav3'],
            'neqpar': 7,
            'eqpar': [1.66666667, 0.0, 1.0, -274.0, 0.0, 0.0, 0.0]
            }
```

Some extra parameters can be added for HDF5 files, namely: 'filedesc' which is a file desciption.

'varnames' should contain a label for each corrdinate dimension, a label for each varible in the w array, followed by each equation parameter. So

```
len(header['varnames']) == nx + nw + neqpar
```

The above example is not consitent with this.

2.1.2 VAC File Description

The unformatted FORTRAN binary files have the following format for each time step stored a header of the following form preceeds the data:

- 'filehead' a string describing the file i.e. '2D_mhd22'
- **'params' a list containing:**
  iteration number, physical time, ndim, number of equation params, number of vars in w array i.e. [10, 11.2541, 2, 7, 9], [int, float, int, int, int]
- nx coord dimensions e.g. [128,128,128]
- 'eqpar' - equation parameters, neqpars floats.
- varnames a list of strings nw long. Holds the names for all the w vars.
- x array
- w array

These are the defualt binary file type that VAC/SAC uses.

WARNING: These files are compiler and machine dependant, they are not portable and should not be used over the far superior HDF5 files.

Part II

pySAC Code Reference

This contains a very draft API reference for all the pySAC functions and classes.

CHAPTER 3

Contents

3.1 Input / Output

3.1.1 pysac.io Module

3.1.2 pysac.io.yt_fields Module

A set of dervied fields for yt 2.x which combine pertubation and background components and define magnitudes, as well as calculate things like charaistic speeds.

Note: These use yt 3.x like field naming conventions

3.1.3 pysac.io.gdf_writer Module

Routines for the writing of GDF files

Functions

write_field(gdf_file, field[, field_shape, ...])	Write a field to an existing gdf file.
write_field_u(gdf_file, data, field_title, ...)	Write a field to an existing gdf file.
create_file(f, simulation_parameters, ...[, ...])	Do all the structral creation of a gdf file.

write_field

pysac.io.gdf_writer.**write_field**(*gdf_file*, *field*, *field_shape=None*, *arr_slice=slice(None, None, None)*, *collective=False*, *api='high'*)

Write a field to an existing gdf file.

Parameters

gdf_file: h5py.File

Open, writeable gdf file

field: dict

A dict containing the following keys:
'field': ndarray 'field_title': string 'field_name': string 'field_units': string 'field_to_cgs': float 'staggering': 0 or 1

arr_slice: (optional) np.s_

The slice of the whole dataset to write

staggering: (optional) int

The 'staggering' of the gdf field

collective: bool

Use MPI collective write

api: str

'high' or 'low' signifiyng the h5py API to use for write. Used for benchmarking.

write_field_u

pysac.io.gdf_writer.**write_field_u**(*gdf_file*, *data*, *field_title*, *field_name*, *field_shape=None*, *arr_slice=slice(None, None, None)*, *staggering=0*, *collective=False*, *api='high'*)

Write a field to an existing gdf file.

Parameters

gdf_file: h5py.File

Open, writeable gdf file

data: astropy.units.Quantity

The data to be written

field_tile: str

The name of the field dataset

field_name: str

The long name for the field

field_shape: (optional) tuple

The shape of the whole dataset, if not specified use data.shape.

arr_slice: (optional) np.s_

The slice of the whole dataset to write

staggering: (optional) int

The 'staggering' of the gdf field

create_file

pysac.io.gdf_writer.**create_file**(*f*, *simulation_parameters*, *grid_dimensions*, *data_author=None*, *data_comment=None*)

Do all the structral creation of a gdf file.

gdf files should be written in a x,y,z order, please swap them before calling this function!!

Parameters

gdf_path: string or h5py instance

Filename to save out

simulation_parameters: dict

Key value pairs for attributes to be written to the simulation_parameters group.

grid_dimensions

data_author: (optional) string

Author to write to file

data_comment: (optional) string

A comment to write to file

Returns
h5py.File instance

Notes

GDF is defined here: https://bitbucket.org/yt_analysis/grid_data_format/

Classes

SimulationParameters(*args, **kwargs)

SimulationParameters

class pysac.io.gdf_writer.**SimulationParameters**(**args*, ***kwargs*)
Bases: dict

Class Inheritance Diagram

3.1.4 pysac.io.legacy Module

Classes

SACdata(filename[, filetype])	This adds specifications to VACdata designed for SAC simulations in 2D or 3D with magnetic field.
VACdata(filename[, filetype])	This is a file type independant class that should expose a VACfile or VACHDF5 file so it is transparen
VACfile(fname[, mode, buf])	Base input class for VAC Unformatted binary files.
VAChdf5(filename[, mode])	Based on FortranFile has been modified to read VAC / SAC HDF5 files.

SACdata

class pysac.io.legacy.**SACdata**(*filename*, *filetype='auto'*)

Bases: pysac.io.legacy.VACdata

This adds specifications to VACdata designed for SAC simulations in 2D or 3D with magnetic field.

This adds the background and pertubation varibles into a new w_sac dict.

Methods Summary

convert_B()	This function corrects for the scaling of the magentic field units.
get_bgp()	
get_bgtemp()	
get_cs([p])	
get_temp([p])	
get_thermalp([beta])	Calculate Thermal pressure from varibles
get_total_p()	
get_va()	
get_w_yt()	
read_timestep(i)	
update_w_sac()	This method creates the w_sac dictionary for the current timestep.

Methods Documentation

convert_B()

This function corrects for the scaling of the magentic field units.

It will convert the magnetic field into Tesla for the current time step.

WARNING: The conservative variable calculations are in SAC scaled magnetic field units, this conversion should be run after accessing any calculatuions involving the magnetic field

get_bgp()

get_bgtemp()

get_cs(*p=None*)

get_temp(*p=None*)

get_thermalp(*beta=False*)

Calculate Thermal pressure from varibles

get_total_p()

get_va()

get_w_yt()

read_timestep(*i*)

update_w_sac()
This method creates the w_sac dictionary for the current timestep.

VACdata

class pysac.io.legacy.**VACdata**(*filename*, *filetype='auto'*)
Bases: object

This is a file type independant class that should expose a VACfile or VACHDF5 file so it is transparent to the end user.

Create a VACdata class.

Parameters
filename: str

filetype: str {'auto' | 'fort' | 'hdf5' }

Attributes Summary

header	
num_records	
t_end	
t_start	
w	
w_	
w_dict	
x	

Methods Summary

read_timestep(i)	Read in the specified time step

Attributes Documentation

header

num_records

t_end

t_start

w

w_

w_dict

x

Methods Documentation

read_timestep(*i*)
Read in the specified time step

Parameters
i: int

Time Step number

VACfile

class pysac.io.legacy.**VACfile**(*fname*, *mode='r'*, *buf=0*)
Base input class for VAC Unformatted binary files. Based on FortranFile has been modified to read VAC / SAC output files.

Parameters
fname: string

Input Filename

mode: {'r' | 'w'}

I/O mode (only 'r' is fully supported)

buf: int

underlying I/O buffer size

Returns

——-

Reads a iteration into the following structure:

file.header: -Dictionary containging

-filehead: string at begging of file -params: Iteration Parameters, it, t, ndim, neqpar, nw -nx: Size of cordinate array [list] -eqpar: eqpar_ parameters [list] -varnames: list containg varible names for dimensions, nw and eqpar?

file.w : w array from file which is [params,[nx]] in size

file.w_: dict containing the {varname:index} pairs for the w array

file.x : x array from file which is [ndim,[nx]] in size

Methods Summary

close()

Continued on next page

Table 3.7 – continued from previous page

process_step()	Does the raw file processing for each timestep
read_timestep(i)	
write_step()	

Methods Documentation

close()

process_step()
Does the raw file processing for each timestep

Sets up the header and reads and reshapes the arrays

read_timestep(*i*)

write_step()

VAChdf5

class pysac.io.legacy.**VAChdf5**(*filename*, *mode='r'*)
Based on FortranFile has been modified to read VAC / SAC HDF5 files.

Reads a iteration into the following structure:

file.header: -Dictionary containging -filehead: string at begging of file -params: Iteration Parameters, it, t, ndim, neqpar, nw -nx: Size of cordinate array [list] -eqpar: eqpar_ parameters [list] -varnames: list containg varible names for dimensions, nw and eqpar? file.w : w array from file which is [params,[nx]] in size file.w_: dict containing the {varname:index} pairs for the w array file.x : x array from file which is [ndim,[nx]] in size

Also creates HDF5 specific attributes:

file.sac_group - Holds the x and time_group attributes. file.time_group - Holds the series of w arrays.

Largely the HDF5 file is designed so the functionality mimics the VAC binary file, i.e. all the vars are still in the W array etc.

Methods Summary

close()	
read_timestep(i)	
write_step()	Save step data into hdf5 file

Methods Documentation

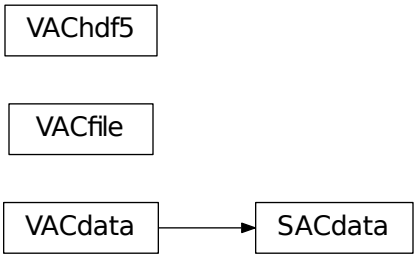
close()

read_timestep(*i*)

write_step()

Save step data into hdf5 file

Class Inheritance Diagram



3.2 pysac.plot Module

3.2.1 Functions

fieldlines(v1, v2, seeds[, dS])	Simple 2D Euler fieldline integrator.

fieldlines

pysac.plot.**fieldlines**(*v1*, *v2*, *seeds*, *dS=1*)

Simple 2D Euler fieldline integrator.

Parameters

v1,v2 : numpy.ndarray (2D)

y and x vector arrays

seeds : numpy.ndarray

array of coordinate (array indexes) pairs

dS : float

step size (default value is 1.0)

3.2.2 Classes

FixedCentre([vmin, vmax, clip])	Normalise with a Fixed Centre If *vmin* or *vmax* is not given, they are taken from the input's minir

FixedCentre

class pysac.plot.**FixedCentre**(*vmin=None*, *vmax=None*, *clip=False*)
Bases: matplotlib.colors.Normalize

Normalise with a Fixed Centre

If *vmin* or *vmax* is not given, they are taken from the input's minimum and maximum value respectively. If *clip* is *True* and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if:

```
vmin==vmax
```

Works with scalars or arrays, including masked arrays. If *clip* is *True*, masked values are set to 1; otherwise they remain masked. Clipping silently defeats the purpose of setting the over, under, and masked colors in the colormap, so it is likely to lead to surprises; therefore the default is *clip* = *False*.

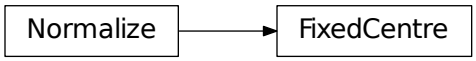
Methods Summary

__call__(value[, clip])	

Methods Documentation

__call__(*value*, *clip=None*)

3.2.3 Class Inheritance Diagram



3.3 pysac.plot.mayavi_plotting_functions Module

This module provides a set of helper functions to make mayavi scenes into nice plots. It also provides routines for the visualisation of flux surfaces.

3.3.1 Functions

set_text(text_property)	Set the text to sane defaults
set_cbar_text(cbar[, lut_manager])	
add_axes(ranges[, obj])	

Table 3.12 – continued from previous page

add_cbar_label(cbar, title)	
add_colourbar(module, position, position2, title)	
draw_surface(surf_poly, cmap[, scalar, ...])	Draw a mayavi surface from a PolyData object.
change_surface_scalars(new_tube, ...[, ...])	Change the surface scalars of an existing surface object, and change the associate

set_text

pysac.plot.mayavi_plotting_functions.**set_text**(*text_property*)

Set the text to sane defaults

Parameters

text_property: mayavi TextProperty object

set_cbar_text

pysac.plot.mayavi_plotting_functions.**set_cbar_text**(*cbar*, *lut_manager='scalar'*)

add_axes

pysac.plot.mayavi_plotting_functions.**add_axes**(*ranges*, *obj=None*)

add_cbar_label

pysac.plot.mayavi_plotting_functions.**add_cbar_label**(*cbar*, *title*)

add_colourbar

pysac.plot.mayavi_plotting_functions.**add_colourbar**(*module*, *position*, *position2*, *title*, *label_fstring='%#4.2f'*, *number_labels=5*, *orientation=1*, *lut_manager='scalar'*)

draw_surface

pysac.plot.mayavi_plotting_functions.**draw_surface**(*surf_poly*, *cmap*, *scalar='vperp'*, *lines=False*, *lim=None*, *log10=False*, *colorbar_label='Velocity Perpendicular\n to Surface [km/s]'*, ***colourbar_args*)

Draw a mayavi surface from a PolyData object.

Parameters

surf_poly: tvtk.PolyData Object

The polydata object containing the surface information.

cmap: ndarray or string

The colour map to scale the scalar data with.

scalar: string

The name of the tvtk scalar to plot.

lines: (optional) bool

If True then lines are not removed from the PolyData object and will be plotted.

lim: [None, float, int or [min,max]]

If lim is None symmetric limits will be created covering the whole dynamic range, if the input is a number symmetric limts will e made with that as a maximum, if a [min,max] pair is passed that will be the limits.

log10: bool

Scale the scalar with log scaling.

colorbar_label: string

Label the colorbar.

****colorar_args: dict**

Other kwargs will be passed to mayavi_plotting_functions.add_colourbar()

Returns

new_tube: mayavi.modules.surface.Surface

The surface object

surf_bar: ScalarBarWidget

The colorbar object

surf_bar_label: mayavi.modules.text.Text

The object created to label the colorbar

change_surface_scalars

pysac.plot.mayavi_plotting_functions.**change_surface_scalars**(*new_tube*, *surf_bar_label*, *scalar_name*, *colorbar_label=None*, *lim=None*, *log10=False*)

Change the surface scalars of an existing surface object, and change the associated colorbar and text objects. Note: This is hard coded to be easier for velocity surfaces.

Parameters

new_tube: mayavi.modules.surface.Surface

The Surface

surf_bar_label: mayavi.modules.text.Text

The text object

scalar_name: string

The name to set the scalar to.

colorbar_label: string

The string to set the colorbar tex object to, if None it will guess velocity ones.

lim:[None, float, int or [min,max]]

If lim is None symmetric limits will be created covering the whole dynamic range, if the input is a number symmetric limts will e made with that as a maximum, if a [min,max] pair is passed that will be the limits.

log10: bool

Scale the scalar with log scaling.

Returns
None

3.4 pysac.plot.mayavi_seed_streamlines Module

This module contains a custom streamlining class derived from the MayaVi2 streamlining class, modified to accept an array of seed points for visulaisation using mayavi.

Warning: The documentation for this class cannot be built on Read The Docs, it is possible to build it locally.

You can use this class thus:

Create a new Streamline instance and add it to a pipeline

```
>>> from pysac.plot.mayavi_seed_streamline import SeedStreamline
>>> field_lines = SeedStreamline(seed_points = np.array(seeds))
>>> myvectorfield.add_child(field_lines)
```

3.5 Analysis

3.5.1 pysac.analysis Module

General routines for the analysis of HD and MHD simulations run with SAC

Functions

get_wave_flux(f, pk)	Calculate the wave energy flux from a SACData instance and the kinetic pressure
get_wave_flux_yt(ds[, B_to_SI, V_to_SI, ...])	Calculate the wave energy flux from a yt dataset.

get_wave_flux

pysac.analysis.**get_wave_flux**(*f*, *pk*)
Calculate the wave energy flux from a SACData instance and the kinetic pressure

Parameters
f: VACdata instance

RAW data file

pk: np.ndarray

Thermal pressure

Returns
Fwave: np.ndarray

The wave energy flux

get_wave_flux_yt

pysac.analysis.**get_wave_flux_yt**(*ds*, *B_to_SI=1*, *V_to_SI=1*, *Pk_to_SI=1*)
Calculate the wave energy flux from a yt dataset.

Parameters
ds: yt dataset

with derived fields

Returns
Fwave: np.ndarray

The wave energy flux

3.5.2 pysac.analysis.tube3D.tvtk_tube_functions Module

This submodule provides routines to generate and analyse "Flux Surfaces" as described in (Mumford et. al. 2014). Flux Surfaces are created by the tracing of a closed loop of fieldlines, from which a surface is reconstructed by creating polygons between the pesudo parallel streamlines.

Functions

move_seeds(seeds, vfield, dt)	Move a list of seeds based on a velocity field.
make_circle_seeds(n, r, **domain)	Generate an array of n seeds evenly spaced in a circle at radius r.
create_flux_surface(bfield, surf_seeds)	Create a flux surface from an array of seeds and a tvtk vector field.
update_flux_surface(surf_seeds, ...)	Update the flux surface streamlines and surface.
make_poly_norms(poly_data)	Extract the normal vectors from a PolyData instance (A surface).
norms_sanity_check(poly_norms)	Check that the normals are pointing radially outwards.
get_surface_vectors(poly_norms, surf_bfield)	Calculate the vector normal, vertically parallel and Azimuthally around
interpolate_scalars(image_data, poly_data)	Interpolate a imagedata scalars to a set points in polydata
interpolate_vectors(image_data, poly_data)	Interpolate a imagedata vectors to a set points in polydata
update_interpolated_vectors(poly_data, ...)	
update_interpolated_scalars(poly_data, ...)	
get_surface_velocity_comp(...)	
get_the_line(bfield, surf_seeds, n)	Generate the vertical line on the surface
update_the_line(the_line, surf_seeds, seed, ...)	Updates the TD line at each time step, while making sure the length is fixed
get_surface_indexes(surf_poly, the_line)	
write_step(file_name, surface, normals, ...)	Write out the surface vectors and velocity compnents.
write_flux(file_name, surface, ...)	
write_wave_flux(file_name, surface_poly, ...)	
read_step(filename)	Read back in a saved surface file
get_data(poly_out, name)	

move_seeds

pysac.analysis.tube3D.tvtk_tube_functions.**move_seeds**(*seeds*, *vfield*, *dt*)

Move a list of seeds based on a velocity field.

> **Warning:** WARNING: THIS IS HARD CODED FOR GRID SIZE!

Parameters

seeds: tvtk.PolyData

Old seed points

vfield: mayavi.sources.array_source.ArraySource object

The velocity field

dt: float

The time step betweent the current and the previous step.

Returns

seeds_arr: ndarray

New Seed points

make_circle_seeds

pysac.analysis.tube3D.tvtk_tube_functions.**make_circle_seeds**(*n*, *r*, ***domain*)

Generate an array of n seeds evenly spaced in a circle at radius r.

Parameters

n: integer

Number of Seeds to Create

r: float

Radius of the Circle in grid points

****domain: Dict**

kwargs specifiying the properties of the domain.

Returns

surf_seeds: tvtk.PolyData

vtkPolyData containg point data with the seed locations. Needs: xmax, ymax, zmax

create_flux_surface

pysac.analysis.tube3D.tvtk_tube_functions.**create_flux_surface**(*bfield*, *surf_seeds*)

Create a flux surface from an array of seeds and a tvtk vector field.

Parameters

bfield: tvtk.ImageData

The vector field to use for streamline traceing

surf_seeds: numpy.ndarray

The array of seed points to start the fieldline tracing from

Returns

surf_field_lines: tvtk.StreamTracer instance

The fieldline tracer with the fieldlines stored inside it.

surface: tvtk.RuledSurfaceFilter instance

The surface built from the StreamTracer instance

update_flux_surface

pysac.analysis.tube3D.tvtk_tube_functions.**update_flux_surface**(*surf_seeds*, *surf_field_lines*, *surface*)

Update the flux surface streamlines and surface.

make_poly_norms

pysac.analysis.tube3D.tvtk_tube_functions.**make_poly_norms**(*poly_data*)

Extract the normal vectors from a PolyData instance (A surface).

Parameters

poly_data: tvtk.PolyData instance

The poly data to extract normal vectors from

Returns

poly_norms: tvtk.PolyDataNormals instance

The normal vectors

norms_sanity_check

pysac.analysis.tube3D.tvtk_tube_functions.**norms_sanity_check**(*poly_norms*)

Check that the normals are pointing radially outwards.

..warning:: THIS IS HARD CODED to grid size and surface size

Parameters

poly_norms: tvtk.PolyDataNormals instance

The normals to check

Returns

poly_normals: tvtk.PolyDataNormals instance

The same normals but flipped if needed

get_surface_vectors

pysac.analysis.tube3D.tvtk_tube_functions.**get_surface_vectors**(*poly_norms*, *surf_bfield*)

Calculate the vector normal, vertically parallel and Azimuthally around the surface cont

interpolate_scalars

pysac.analysis.tube3D.tvtk_tube_functions.**interpolate_scalars**(*image_data*, *poly_data*)

Interpolate a imagedata scalars to a set points in polydata

interpolate_vectors

pysac.analysis.tube3D.tvtk_tube_functions.**interpolate_vectors**(*image_data*, *poly_data*)
Interpolate a imagedata vectors to a set points in polydata

update_interpolated_vectors

pysac.analysis.tube3D.tvtk_tube_functions.**update_interpolated_vectors**(*poly_data*, *surface_probe_filter*)

update_interpolated_scalars

pysac.analysis.tube3D.tvtk_tube_functions.**update_interpolated_scalars**(*poly_data*, *surface_probe_filter*)

get_surface_velocity_comp

pysac.analysis.tube3D.tvtk_tube_functions.**get_surface_velocity_comp**(*surface_velocities*, *normals*, *torsionals*, *parallels*)

get_the_line

pysac.analysis.tube3D.tvtk_tube_functions.**get_the_line**(*bfield*, *surf_seeds*, *n*)
Generate the vertical line on the surface

update_the_line

pysac.analysis.tube3D.tvtk_tube_functions.**update_the_line**(*the_line*, *surf_seeds*, *seed*, *length*)
Updates the TD line at each time step, while making sure the length is fixed

get_surface_indexes

pysac.analysis.tube3D.tvtk_tube_functions.**get_surface_indexes**(*surf_poly*, *the_line*)

write_step

pysac.analysis.tube3D.tvtk_tube_functions.**write_step**(*file_name*, *surface*, *normals*, *parallels*, *torsionals*, *vperp*, *vpar*, *vphi*)
Write out the surface vectors and velocity compnents.

write_flux

pysac.analysis.tube3D.tvtk_tube_functions.**write_flux**(*file_name*, *surface*, *surface_density*, *surface_va*, *surface_beta*, *surface_cs*, *Fpar*, *Fperp*, *Fphi*)

write_wave_flux

pysac.analysis.tube3D.tvtk_tube_functions.**write_wave_flux**(*file_name*, *surface_poly*, *parallels*, *normals*, *torsionals*, *Fwpar*, *Fwperp*, *Fwphi*)

read_step

pysac.analysis.tube3D.tvtk_tube_functions.**read_step**(*filename*)
Read back in a saved surface file

get_data

pysac.analysis.tube3D.tvtk_tube_functions.**get_data**(*poly_out*, *name*)

Classes

PolyDataWriter(filename, polydata)	This class allows you to write tvtk polydata objects to a file, with as many or as few associate

PolyDataWriter

class pysac.analysis.tube3D.tvtk_tube_functions.**PolyDataWriter**(*filename*, *polydata*)
Bases: object

This class allows you to write tvtk polydata objects to a file, with as many or as few associated PointData arrays as you wish.

Methods Summary

add_array(**kwargs)	Add any number of arrays via keyword arguments.
add_point_data([vectors, scalars, ...])	Add a vector comonent and a associated scalar
write()	

Methods Documentation

add_array(***kwargs*)
Add any number of arrays via keyword arguments.

Examples

Add one scalar

```
>>> writer = PolyDataWriter(filename, polydata)
>>> writer.add_array(myscalar=myarray)
>>> writer.write
```

add_point_data(*vectors=None*, *scalars=None*, *vector_name=None*, *scalar_name=None*)
Add a vector comonent and a associated scalar

write()

Class Inheritance Diagram

3.5.3 pysac.analysis.tube3D.process_utils Module

This module contains routines to convert from SACData and yt DataSets into tvtk fields via mayavi.

Functions

get_sacdata_mlab(f, cube_slice[, flux])	Reads in useful variables from a hdf5 file to vtk data structures
get_yt_mlab(ds, cube_slice[, flux])	Reads in useful variables from yt to vtk data structures, converts into SI units be
process_next_step_yt(ds, cube_slice, bfield, ...)	Update all mayavi sources using a yt dataset, in SI units.
process_next_step_sacdata(f, cube_slice, ...)	Update all mayavi sources using a SACData instance
yt_to_mlab_vector(ds, xkey, ykey, zkey[, ...])	Convert three yt keys to a mlab vector field :Parameters: **ds** : yt dataset The data

get_sacdata_mlab

pysac.analysis.tube3D.process_utils.**get_sacdata_mlab**(*f*, *cube_slice*, *flux=True*)
Reads in useful variables from a hdf5 file to vtk data structures

Parameters

f : hdf5 file handle

SAC HDF5 file

flux : boolean

Read variables for flux calculation?

cube_slice : np.slice

Slice to apply to the arrays

Returns
bfield, vfield[, density, valf, cs, beta]

get_yt_mlab

pysac.analysis.tube3D.process_utils.**get_yt_mlab**(*ds*, *cube_slice*, *flux=True*)
Reads in useful variables from yt to vtk data structures, converts into SI units before return.

Parameters
ds : yt dataset

with derived fields

flux : boolean

Read variables for flux calculation?

cube_slice : np.slice

Slice to apply to the arrays

Returns
if flux:

bfield, vfield, density, valf, cs, beta

else:

bfield, vfield

process_next_step_yt

pysac.analysis.tube3D.process_utils.**process_next_step_yt**(*ds*, *cube_slice*, *bfield*, *vfield*, *density*, *valf*, *cs*, *beta*)
Update all mayavi sources using a yt dataset, in SI units.

Parameters
ds : yt dataset

The dataset to use to update the mayavi fields

cube_slice : np.s_

A array slice to cut the yt fields with

bfield, vfield, density, valf, cs, beta : mayavi sources

The sources to update

Returns
bfield, vfield, density, valf, cs, beta : mayavi sources

The updated sources

process_next_step_sacdata

pysac.analysis.tube3D.process_utils.**process_next_step_sacdata**(*f*, *cube_slice*, *bfield*, *vfield*, *density*, *valf*, *cs*, *beta*)
Update all mayavi sources using a SACData instance

Parameters

ds : SACData instance

The dataset to use to update the mayavi fields

cube_slice : np.s_

A array slice to cut the yt fields with

bfield, vfield, density, valf, cs, beta : mayavi sources

The sources to update

Returns

bfield, vfield, density, valf, cs, beta : mayavi sources

The updated sources

yt_to_mlab_vector

pysac.analysis.tube3D.process_utils.**yt_to_mlab_vector**(*ds*, *xkey*, *ykey*, *zkey*, *cube_slice=(slice(None, None, None), slice(None, None, None), slice(None, None, None))*, *SI_scale=1*, *field_name=''*)

Convert three yt keys to a mlab vector field

Parameters

ds : yt dataset

The dataset to get the data from.

xkey, ykey, zkey : string

yt field names for the three vector components.

cube_slice : numpy slice

The array slice to crop the yt fields with.

SI_scale : float

Conversion factor to SI units (multipled by the field).

field_name : string

The mlab name for the field.

Returns

field : mayavi vector field

3.6 pysac.mhs_atmosphere Module

3.6.1 Functions

construct_magnetic_field(x, y, z, x0, y0, S, ...)	Construct self similar magnetic field configuration
construct_pairwise_field(x, y, z, xi, yi, ...)	Construct self similar magnetic field configuration
get_flux_tubes(model, model_pars, coords, ...)	Obtain an array of x,y coordinates and corresponding vertical
get_internal_energy(pressure, magp, ...)	Convert pressures to internal energy – this may need revision if an alternative e
get_logical(model, l_mpi[, l_SI, l_gdf])	This module assigns the logical options for the model.

Table 3.18 – continued from previous page

get_model(model, Nxyz, xyz_SI, scales, ...)	
get_parameters(model, l_mpi, logical_pars, size)	
get_spruit_hs(filenames, Z, scales, ...)	
interpolate_atmosphere(filenames, Z, scales, ...)	This module generates a 1d array for the model plasma preesure, plasma densit
mhs_3D_profile(z, pressure_z, rho_z, ...)	Return the vertical profiles for thermal pressure and density in 3D
save_SACsources(model, sourcesfile, Fx, Fy, ...)	Save the balancing forces for a SAC model with multiple flux tubes in hdf5 (gd
save_SACvariables(model, filename, rho, Bx, ...)	Save the background variables for a SAC model in hdf5 (gdf default) format aft
save_auxilliary1D(model, auxfile, ...)	Save auxilliary variables for use in plotting background setup in hdf5 (gdf defau
vertical_profile(Zint, Z, pdata_i, rdata_i, ...)	Return the vertical profiles for thermal pressure and density in 1D.

construct_magnetic_field

pysac.mhs_atmosphere.**construct_magnetic_field**(*x*, *y*, *z*, *x0*, *y0*, *S*, *model_pars*, *logical_pars*, *physical_constants*, *scales*)

Construct self similar magnetic field configuration Note if model_pars['B_corona'] = 0 then paper3 results otherwise paper 2

construct_pairwise_field

pysac.mhs_atmosphere.**construct_pairwise_field**(*x*, *y*, *z*, *xi*, *yi*, *xj*, *yj*, *Si*, *Sj*, *model_pars*, *logical_pars*, *physical_constants*, *scales*)

Construct self similar magnetic field configuration

get_flux_tubes

pysac.mhs_atmosphere.**get_flux_tubes**(*model*, *model_pars*, *coords*, *scales*, *logical_pars*)

Obtain an array of x,y coordinates and corresponding vertical component value for the photospheric magnetic field

get_internal_energy

pysac.mhs_atmosphere.**get_internal_energy**(*pressure*, *magp*, *physical_constants*)

Convert pressures to internal energy – this may need revision if an alternative equation of state is adopted.

get_logical

pysac.mhs_atmosphere.**get_logical**(*model*, *l_mpi*, *l_SI=True*, *l_gdf=True*)

This module assigns the logical options for the model. If adding new models with additional logical arguments add it to the default list as false, include an if statement for True update the dictionary logical_pars

get_model

pysac.mhs_atmosphere.**get_model**(*model*, *Nxyz*, *xyz_SI*, *scales*, *logical_pars*)

get_parameters

pysac.mhs_atmosphere.**get_parameters**(*model*, *l_mpi*, *logical_pars*, *size*)

get_spruit_hs

pysac.mhs_atmosphere.**get_spruit_hs**(*filenames*, *Z*, *scales*, *model_pars*, *physical_constants*, *logical_pars*, *plot*)

interpolate_atmosphere

pysac.mhs_atmosphere.**interpolate_atmosphere**(*filenames*, *Z*, *scales*, *model_pars*, *physicalconstants*, *logical_pars*, *plot=False*)

This module generates a 1d array for the model plasma preesure, plasma density, temperature and mean molecular weight.

mhs_3D_profile

pysac.mhs_atmosphere.**mhs_3D_profile**(*z*, *pressure_z*, *rho_z*, *pressure_m*, *rho_m*)

Return the vertical profiles for thermal pressure and density in 3D

save_SACsources

pysac.mhs_atmosphere.**save_SACsources**(*model*, *sourcesfile*, *Fx*, *Fy*, *logical_pars*, *physical_constants*, *scales*, *coords*, *Nxyz*)

Save the balancing forces for a SAC model with multiple flux tubes in hdf5 (gdf default) format after collating the data from mpi sub processes if necessary.

save_SACvariables

pysac.mhs_atmosphere.**save_SACvariables**(*model*, *filename*, *rho*, *Bx*, *By*, *Bz*, *energy*, *logical_pars*, *physical_constants*, *scales*, *coords*, *Nxyz*)

Save the background variables for a SAC model in hdf5 (gdf default) format after collating the data from mpi sub processes if necessary.

save_auxilliary1D

pysac.mhs_atmosphere.**save_auxilliary1D**(*model*, *auxfile*, *pressure_m*, *rho_m*, *temperature*, *pbeta*, *alfven*, *cspeed*, *dxB2*, *dyB2*, *val*, *mtw*, *pressure_Z*, *rho_Z*, *Rgas_Z*, *logical_pars*, *physical_constants*, *scales*, *coords*, *Nxyz*)

Save auxilliary variables for use in plotting background setup in hdf5 (gdf default) format after collating the data from mpi sub processes if necessary.

vertical_profile

pysac.mhs_atmosphere.**vertical_profile**(*Zint*, *Z*, *pdata_i*, *rdata_i*, *Tdata_i*, *muofT_i*, *magp*, *physical_constants*, *dz*, *scales*)

Return the vertical profiles for thermal pressure and density in 1D. Integrate in reverse from the corona to the photosphere to remove sensitivity to larger chromospheric gradients.

Part III

Indices and tables

- *genindex*
- *modindex*
- *search*

p

Symbols

A

C

D

F

G

H

I

M

N

P

R

S

T

U

V

W

X

Y