MoogPy Manual

C. Deen[a,b]

[a]Max Planck Institut für Astronomie, Königstuhl 17, 69117 Heidelberg Germany;
[b]Max Planck Institut für Extraterrestriche Physik, Giessenbachstraße, 80797 Garching bei München Germany;

ABSTRACT

MoogPy is a python interface to the MoogStokes suite of spectral synthesis tools.

1. INTRODUCTION

MoogStokes uses a full Stokes vector treatment of the polarized radiative transfer to synthesize the emergent spectra of stars with magnetic fields in their photospheres. As the emergent spectra of a magnetic star depends upon the viewing angle, direction, and strength of the magnetic field, the data produced by MoogStokes to synthesize the integrated spectrum of a single star are numerous and complicated. Therefore, to keep one from descending into the depths of a spaghetti-code hell, I have devised an accompanying suite of Python tools to shepherd the output of MoogStokes through a pipeline, producing an output suitable for comparison with observed spectra.

2. MOOGPY ARCHITECTURE

The MoogPy suite of python tools are hierarchically arranged into four separate python objects. The MoogStokesPy object contains the python-wrapped FORTRAN77 code. MoogTools contains low-level functions which interact directly with MoogStokes and the raw data products. Moog960 uses the MoogTools interfaces to organize and process the raw data into a form useful for comparison to observed spectra. MoogSymphony provides an interface to the Moog960 objects with an aim of allowing the user to generate large grids of spectra.

2.1 MoogStokesPy.so

MoogStokesPy.so is a **s**hared **o**bject file created when f2py compiles the FORTRAN77 code and attaches the Python. The creation of this unholy hybrid FORTRAN77/python monster is not within the scope of this document. Suffice it to say, that if you would like to control MoogStokes from Python, you will need to include MoogStokesPy in your import section (either directly or indirectly).

2.2 MoogTools.py

The MoogTools.py object defines the low-level functions necessary to interact directly with the MoogStokes Fortran Code. The end-user does not directly interact with these routines, but knowledge of them is useful for development and processing

Please direct all correspondence to deen@mpia.de

2.2.1 MoogStokes

The Python wrapper for MoogStokes. This object allows the user to spawn an instance of MoogStokes to generate a synthetic spectrum for given magnetic field strength and set of atmospheric parameters (alternatively a user-defined model atmosphere file). The user is allowed to chose between three different versions of MoogStokes: The original (scalar) version of Moog (*TENNISBALL*), a full Stokes **IQUV** vector treatment (*DISKOBALL*), and a scaled down Stokes **IV** vector treatment (*BEACHBALL*). As most of my work has been focused on the effect on Stokes **I** of uniform magnetic fields, most of the tools described below have only been tested with any degree of confidence with BEACHBALL-produced spectra.

The MoogStokes object contains instances of the following objects: A ParameterFile object (section 2.2.2) which creates and modifies the input parameter file used by Moog to determine the parameters for the synthesis. A LineList object (section 2.2.3) which handles the details of the Zeeman splitting of absorption lines (both atomic and molecular) also writes these lines into the Moog input line list. The LineList object is in turn comprised of many SpectralLine objects (section 2.2.4) each of which contains all quantitative descriptions (λ, $\log gf$, element or molecular species, etc. . .) of an individual spectral line. Going further, each SpectralLine object contains two EnergyLevel objects (section 2.2.6) to describe the upper and lower quantum levels, and zeemanTransition objects (section 2.2.5) to describe the π and σ components which compose each spectral line.

- **Constructor**:
  - *MS* = MoogTools.MoogStokes(configurationFile, fileBase="moogstokes", **kwargs):
- **Input**:
  - *configurationFile* : The name of a file containing configuration parameters
  - *fileBase* : the base name for the MoogStokes parameter file/linelists
  - *kwargs*: Keyword arguments. Possible keyword arguments are:
    - `MODELFILE`: Allows user to specify directly the atmospheric model to be used in the calculation, overriding the model specified in the configuration file
    - `diskInt`: Allows the user to specify which disk-integration algorithm to use. Possible values are `TENNISBALL`, `BEACHBALL`, or `DISKOBALL`.
    - `wlStart`: Specifies the starting wavelength for the synthesis.
    - `wlStop`: Specifies the ending wavelength for the synthesis.
    - `moogInstance`: Specifies which instance of MoogStokesPy to import. Possible values are `ALPHA`, `BRAVO`, `CHARLIE`, or `DELTA`. If this keyword argument is not given, the generic version of MoogStokesPy is imported.
    - `progressBar`: Boolean specifying whether or not to display synthesis progress through a progress bar.
- **Member Variables**:
  - *config* - a python dictionary created from a top-level configuration file
  - *lineList* - a LineList object (section 2.2.3) containing lines within the defined wavelength range
  - *parameterFile* - a ParameterFile object (section 2.2.2) useful for configuring and writing Moog-Readable parameter files
  - *modelFile* - If a specific model atmosphere file is desired to be used, its name will be stored here. If a model file is specified, the values of the next two parameters are unused
  - *T* - Temperature in Kelvin. If the modelFile is unused, *T* is used to specify the temperature of the MARCS model atmosphere selected by default.
  - *logg* - Surface gravity (logarithm of cgs acceleration due to gravity at surface) If the modelFile is unused, *logg* is used to specify the surface gravity of the MARCS model atmosphere selected by default.

- *B* - Magnetic field strength in kG. This value is used by the Zeeman splitting routine to shift the energy levels (and hence the wavelengths) of the different Zeeman components
- *fileBase* - File name base.
- *fileName* - name of the MoogStokes parameter file. Constructed with the following structure: [MoogSandbox]+fileBase+'.par'
- *diskInt* = Flag indicating which disk integration algorithm is to be used. Possible values include:
  - `TENNISBALL`: Use original MOOG to calculate a disk-integrated spectrum. No further processing is require to compare to observed spectra (other than convolution with a $v \sin i$ kernel, if needed.
  - `BEACHBALL`: Use MoogStokes to calculate an emergent spectrum for a meridianal stripe along the stellar disk (from pole to pole). In this case, the Stokes **Q** and **U** vectors are valid only at the equator. In order to compare to observed spectra, spectra from different meridianal stipes must be convovled together using a disk-integration (a python translation of Jeff Valenti's IDL *rtint*). Most of the subsequent processing tools have been developed with BEACHBALL spectra in mind.
  - `DISKOBALL`: Use MoogStokes to calculate an emergent spectrum for a single location on the stellar disk. In this case, all Stokes vectors are valid, but it is significantly more computationally intensive. In order to compare to observed spectra, it is necessary to do a more thorough disk integration. At current, this functionality doesn't really exist.
- *lineListFormat*: The format of the linelists to be used. Possible values include:
  - `MOOGSCALAR`: If the original, non-polarized version of Moog is run, the linelist should follow the format of the normal Moog.
  - `MOOGSTOKES`: If polarized radiative transfer is requested (i.e. MoogStokes), the linelist needs to include a few more variables (i.e. polarization, radiative broadening coefficient, Stark broadening coefficient).
- *MoogSandbox*: Directory which MoogStokes can use as a "sandbox," to write line lists and temporary files during the computation of the emergent spectrum.
- *Spectra*: List of emergent spectra. Each spectrum in this list an object of type SpectralTools.Spectrum (section 2.3.1).
- *logtau*: A list of opacities through the stellar photosphere. Only used in flux-tracing mode.
- *moogInstance*: Which instance of MoogStokesPy to import. A single instance can only be imported by one program at a time. Otherwise, the two programs will step on each other's toes. Bad things will happen.
- *MoogPy*: The instance of MoogStokesPy imported when specified by the moogInstance variable. This variable provides access into the memory banks of the F77 code, allowing variables to be passed in to and out of the Fortran program. Specifically, the following access points are available:
  - `MoogPy.charstuff.moogpath`: Allows MoogStokesPy to react to changes in the location of the MoogStokes source code (through the environment variable `MOOGSTOKESSOURCE`) without modifying source code and recompiling.
  - `MoogPy.recorder`: Fortran hook for recorder function, called during scalar MOOG. Connected to python function `recorder(x, y)`
  - `MoogPy.stokesrecorder`: Fortran hook for stokesrecorder function, called during MoogStokes. Connected to python function `stokesrecorder(i, wave, Stokes, continuum)`
  - `MoogPy.tennisball`: Fortran hook for beachball function to prepare the python wrapper to accept tennisball-format sythetic spectra. Connected to function `tennisball()`
  - `MoogPy.beachball`: Fortran hook for beachball function to prepare the python wrapper to accept beachball-format sythetic spectra. Connected to member function `beachball()`
  - `MoogPy.diskoball`: Fortran hook for beachball function to prepare the python wrapper to accept diskoball-format sythetic spectra. Connected to member function `diskoball()`

- **Member Functions**
  - *fluxtracer(logtau, dtau, Stokes, continuum)*: Fortran hook which allows the Fluxtracer option to trace the flux through the stellar atmosphere at a single wavelength. Each time it is called, the current values of the opacity Stokes **IQUV** fluxes, and continuum are appended into their respective arrays.
  - *recorder(x, y)*: Fortran hook which allows access to the scalar version of Moog, to extract the emergent flux at each wavelength point for which it is calculated. Each time it is called, the current value of the wavelength and emergent flux are appended into the proper locations in the last Spectrum object in the MoogStokes.Spectrum list.
  - *stokesrecorder(i, wave, Stokes, continuum)*: Fortran hook which allows access to MoogStoke, to extract the emergent Stokes vectors at each wavelength point for which it is calculated. Each time it is called, the current value of the wavelength, emergent Stokes fluxes, and continuum, are appended into the proper locations in the $(i-1)^{\text{th}}$ Spectrum object in the MoogStokes.Spectrum list.
  - *prepareFluxes()*: Creates a .FITS header and populates a list of blank Spectrum objects to be associated with a to-be generated spectrum. The exact contents of the FITS header and number of associated Spectrum objects depend upon the type of disk integration. Each individual spectrum is tagged with the date and time of its creation, the wavelength range, as well as the version of MoogStokes used to create it. The `SPECTRUM_TYPE` FITS keyword is used to store what type of disk integration was used.
  - *tennisball()*: Fortran hook called to prepare the python wrapper to accept `TENNISBALL`-format spectra (i.e. scalar moog).
  - *beachball()* Fortran hook called to prepare the python wrapper to accept `BEACHBALL`-format spectra (i.e. Polarized radiative transfer, Only stellar equator is synthesized.).
  - *diskoball()* Fortran hook called to prepare the python wrapper to accept `DISKOBALL`-format spectra (i.e. Polarized radiative transfer, Different stellar latitudes are treated differently).
  - *finishSpectrum()*: Once the synthesis is finished, for each Spectrum object in the Spectra list, call the `preserve()` function to prepare the .FITS table.
  - *run(saveRaw=False, **kwargs)*: After the Fortran and Python programs have been initialized, perform the Zeeman splitting, write the line lists and parameter files, launch the MoogStokes (or scalar Moog) FORTAN subroutine (`MoogPy.moogstokessilent()`). After the FORTRAN subroutine is finished, call `finishSpectrum()` and load the resulting raw spectra into a Moog960.Phrase object (see section2.4.1). Optionally, allows the user to save the raw data into a filename specified by the atmospheric parameters.
  - *trace(save=False)*: After the Fortran and Python programs have been initialized, perform the Zeeman splitting, write the line lists and parameter files, launch the MoogStokes FORTAN subroutine (`MoogPy.moogstokessilent()`) using the driver fluxtracer. After the FORTRAN subroutine is finished, returns python lists of the opacity, Stokes vectors, and continuum. Optionally, allows the user to save the raw data.

2.2.2 ParameterFile

A ParameterFile object handles the creation and customization of Moog .par files, which MoogStokes uses to determine the parameters for the synthesis.

- **Constructor**
  - *ParameterFile* = MoogTools.ParameterFile(parent, config, **kwargs):
- **Input**
  - *parent*: The parent object of the ParameterFile. Usually a MoogStokes object.

  - *config*: A python dictionary containing variables and values from the top-level configuration file.
  - *kwargs*: Possible keyword arguments are:
    * *wlProbe*: Boolean telling MoogStokes to probe the creation of a spectral line through the stellar atmosphere at a specific wavelength. Default is False/None
    * *PARFILENAME*: Allows user-defined override of value in moogPars['parFileName']
    * *atmos_dir*: over-writes value in moogPars['atmos_dir']

- **Member Variables**
  - *parent*: The parent object of the ParameterFile. Usually a MoogStokes object.
  - *config*: A python dictionary containing variables and values from the top-level configuration file.
  - *moogParCfgFile*: The name of the configuration file that stores Moog-specific (as opposed to MoogStokes specific) parameters.
  - *moogPars*: A python dictionary containing Moog-specific variables and values from the configuration file specified in *moogParCfgFile*.
  - *synlimits*: Numpy array containing the values passed to the synlimits line in a Moog .par file. [λ_{start}, λ_{stop}, $\delta\lambda$, $\delta\lambda_{\mathrm{lc}}$] ($\delta\lambda_{\mathrm{lc}}$ is the distance from line center that a line's opacity is considered.
  - *wlProbe*: The wavelength at which to do the wavelength probe through the atmosphere.
  - *parFileName*: The name of the parameter file to be written.
  - *mode*: Moog driver to use. Possible values are:
    * `synth`: Scalar Moog. The original
    * `synsto`: Stokes Synthesis.
    * `stoktra`: Stokes tracing of the creation of the emergent spectrum through the atmosphere at a single wavelength.
  - *labels*: A python dictionary containing input variables in the MOOG .par file. More detailed descriptions of the effect of these variables can be found in writemoog.ps, the Moog manual. The default value for any entry can be overwritten by placing the desired parameter and value (separated by a `':'`) in the MoogParCfgFile. Entries include: (default values in parentheses)
    * `terminal`: Terminal for Moog to use (`'x11'`)
    * `strong`: Whether or not to consider strong lines (`1`)
    * `atmosphere`: (`1`)
    * `molecules`: How Moog handles molecular lines (`2`)
    * `lines`: (`0`)
    * `damping`: (`1`)
    * `freeform`: (`2` - This value is not available in the standard Moog, and is related to additional data incorporated in the line list, i.e. angular momentum, additional damping parameters, etc. . .)
    * `flux/int`: Whether or not compute intensities or flux. Only used with driver *synth* (`0`)
    * `diskflag`: Controls whether MoogStokes calculates a disco-ball (0, `DISKOBALL`) or annuli (1, `BEACHBALL`) (`1`)
    * `testflag`: Unused, except in debugging mode. (`0`)
  - *file_labels*: A python dictionary containing file names to be written in the MOOG .par file. More detailed descriptions of the effect of these variables can be found in writemoog.ps, the Moog manual. The default value for any entry can be overwritten by placing the desired parameter and value (separated by a `':'`) in the MoogParCfgFile. I have not spent time to ensure all these files include the proper values, but simply made sure they are created for the sake of Moog running. Entries include: (default values in parentheses)
    * `summary_out`: Summary of synthesis produced by Moog. (`'./Output/summary.out'`)

    * `standard_out`: Standard output (`'./Output/out1'`)
    * `smoothed_out`: Smoothed output (`'./Output/smoothed.out'`)
    * `model_in`: Input model (`''`)
    * `lines_in`: Weak line-list (`config['Weak_FileName']`)
    * `stronglines_in`: Strong line-list (`config['Strong_FileName']`)

- **Member Functions**
  - *setName(name)*: Applies the base name `name` to the following input files in the following format:
    * `file_labels['lines_in']` = parent.MoogSandbox + config['Weak_FileName']+'_'+name
    * `file_labels['stronglines_in']` = parent.MoogSandbox + config['Strong_FileName']+'_'+name
    * `parFileName` = parent.MoogSandbox + name + '.par'

2.2.3 LineList

blah

2.2.4 SpectralLine

blah

2.2.5 zeemanTransition

blah

2.2.6 EnergyLevel

blah

2.3 SpectralTools.py

SpectralTools.py contains the nuclear-level building blocks which MoogStokes uses to construct a spectrum.

2.3.1 Spectrum

This is the fundamental building block of all MoogStokes spectra.

- **Constructor**: sp = Spectrum(wl, I, Q, U, V, continuum, header, spectrum_type, filename, ext, preserve)
  - *wl*: numpy array of wavelength values (in microns)
  - *I*: numpy array of Stokes I flux values
  - *Q*: numpy array of Stokes Q flux values
  - *U*: numpy array of Stokes U flux values
  - *V*: numpy array of Stokes V flux values
  - *continuum*: numpy array of continuum values. May be omitted if the flux values are normalized
  - *header*: pyfits Header object containing the .FITS header to be saved with the Spectrum object if/when it is saved to disk.
  - *spectrum_type*: Label describing the contents of the Spectrum object. Possible values are:
    * `RAW`
    * `INTERPOLATED`
    * `DISK INTEGRATED`
    * `CONVOLVED`
    * `BLENDED`
    * `MERGED`

        * `ROTATED`
        * `DIFFERENCE`
        * `SCALED`
        * `MOOG DISK INTEGRATED`
        * `MOOG EMERGENT`
    - *filename*: Filename for saving (or reading)
    - *ext*: For .FITS files with multiple parts, `ext` is the extension number
    - *preserve*: When True, prepares the fluxes and wavelengths for saving into a .FITS table.

- **Constructor (from File)**: sp = Spectrum.from_file(header, data, filename, ext)
    -
- **Member Variables**
- **Member Functions**

2.4 Moog960.py

The original version of Moog was named for the Moog synthesizer, one of the first electronic music makers. In this spirit, the assorted software architecture has been constructed with this musical heritage in mind. Moog960 is the name of an electronic mixer, used to combine inputs from multiple musical instruments into a unified sound. The Moog960.py hierarchy assembles the outputs from different MoogStokes objects, treating them like individual instruments in an orchestra. The most fundamental unit comprising the symphonic metaphor is the phrase. In music, a phrase is the smallest musical building block, sometimes consisting only of a few notes. For Moog960, a Phrase object (section 2.4.1) represents a region of a single spectrum from λ_{Start} to λ_{Stop}. A melody represents the next largest musical building block. Melodies are composed of multiple phrases strung together. Similarly, a Melody object in Moog960 (section 2.4.2) is comprised of multiple regions of the same spectrum. Finally, the largest unit of music (as far as this extremely simplistic metaphor is concerned) is a score. Scores contain melodies for multiple instruments, and is what the conductor uses to direct the orchestra and produce the music she desires. In Moog960, a Score object (section 2.4.3) contains multiple Melody objects. Similarly, the user may use the Score object to perform many different operations (i.e. convolutions, wavelength shifts, etc. . .) on the raw or observed spectra contained within.

2.4.1 Moog960:Phrase

There are two types of Phrases, Synthetic Phrases, and Observed Phrases. Synthetic Phrases are synthesized for a single set of stellar parameters (T_{eff}, $\log g$, and magnetic field strength B), while Observed Phrases are continuous sections of stellar spectra observed with a spectrograph (i.e. H-band IGRINS observations of TW Hydra)

2.4.2 Moog960:Melody

As with Phrases, Melodies also come in two flavors, Synthetic and Observed. Synthetic Melodies are composed of Synthetic Phrases with the same stellar parameters, while Observed Melodies are comprised of different regions of the same observed spectrum (i.e. H and K band IGRINS observations of TW Hydra)

2.4.3 Moog960:Score

Scores contain multiple Melody objects, and can be used to select individual melodies for further processing, generate new melodies corresponding to different smoothings or resolving powers, and output final processed synthetic spectra in a form suitable for comparison with observed spectra.

Currently, Scores only come in one type: Synthetic. However, as MoogStokes evolves with its (hopefully expanding) userbase, it may make sense to extend the Score object to hold multiple observed Melodies.

2.5 MoogSymphony.py

The MoogSymphony object is used to synthesize a grid of synthetic spectra.

3. INTERFACE DESCRIPTION