CLEDB solar-coronal-inversion

Release update-readthedocs

Alin Paraschiv

Feb 01, 2023

CONTENTS:

CLEDB is a publicly hosted on Github

Purpose:

The **C**oronal **L**ine **E**mission **D**ata**B**ase inversion is a Python based tool and pipeline that is used to infer magnetic field information from SpectroPolariemtric observations of the Solar Corona. This document describes the main concepts, functions, and variables, comprising the CLEDB inversion algorithm for inverted coronal magnetic fields.

Authors and Contact:

Alin Paraschiv and Philip Judge
– High Altitude Observatory NCAR|UCAR
– National Solar Observatory, AURA

arparaschiv *at* ucar edu or paraschiv.alinrazvan+cledb *at* gmail

Note: Last updated at the **update-readthedocs** commit tag.

CHAPTER

ONE

SYNOPSIS AND MOTIVATION

1.1 Synopsis

CLEDB aims to invert coronal magnetic field information from observations of polarized light. The algorithm takes arrays of one or two sets of SpectroPolarimetric *Stokes IQUV* observations `sobs_in` along with their *header* metadata information. The data and metadata are pre-processed, and optimal corresponding sets of databases resulting from forward calculations are selected and read from disk storage.

The data processing is split into two branches, based on the available polarized coronal Stokes observations:

- **1-line branch:** 4 input IQUV observations *(one coronal emission line)*.
- **2-line branch:** 8 input $I_1Q_1U_1V_1I_2Q_2U_2V_2$ observations *(two coronal emission lines)*.

Spectroscopic analysis products are computed for each line for both 1-line or 2-line branches.

The 1-line branch employs analytical approximations to calculate line of sight (*LOS*) integrated magnetic field products, while the 2-line branch offers access to additional magnetic field products. This second setup benefits from more degrees of freedom allowing us to break degeneracies intrinsic in the inversion. Thus, the two-line algorithm branch performs a χ^2 fitting between the observation and a forward-modeled database to recover full 3D vector magnetic field components.

The databases are generated via forward modeling of combinations of input magnetic field and geometric parameters. In this setup, databases are used as a static input with respect to the inversion scheme and **should not** be computed dynamically for each observation.

1.2 Motivation for the CLEDB approach

By utilizing 2-line observations, we can recover the 3D magnetic field information for single point *voxel* using a χ^2 fitting approach. Theoretically, we can employ the *CLE* (Coronal Line Emission) spectral synthesis code to generate forward-model calculations. About 10^7-10^9 atomic plasma and magnetic configurations are needed in order to satisfy a reasonable solution resolution criteria. Directly forward modeling such solutions for one *pixel*/voxel in a dynamic fashion would be time consuming. Such a calculation has execution times in the order of 5-10 hours, on a single CPU thread when using a fast implementation of the Fortran CLE code.

Building a static database (via the `CLEDB_BUILD` module of our algorithm) to store the vast set of synthetic Stokes observations, along with the input plasma and magnetic field configurations responsible for producing polarized emission, proved to be a significantly more feasible approach.

Additionally, the database theoretical calculations gain intrinsic access to otherwise un-observable input parameters (e.g. atomic alignment σ_0^2, intrinsic magnetic field angles ϑ, φ etc.) that can be used to break inherent degeneracies encountered when attempting analytical inversions (as for example occurring in the 1-line branch implementation). The dimensionality of the problem at hand can be further reduced by 1-2 orders of magnitude by using native symmetries

when building and querying databases. Detailed discussions on the physics aspects of dimensionality reduction and *degeneracy* breaking effects can be found in the sources below.

1.3 List of Relevant Publications

Academic journal papers that helped build and justify CLEDB:

1. Paraschiv & Judge, SolPhys, 2022 covered the scientific justification of the algorithm, and the setup of the CLEDB inversion.
2. Judge, Casini, & Paraschiv, ApJ, 2021 discussed the importance of scattering geometry when solving for coronal magnetic fields.
3. Ali, Paraschiv, Reardon, & Judge, ApJ, 2022 performed a spectroscopic exploration of the infrared regions of the emission lines available for inversion with CLEDB.
4. Dima & Schad, ApJ, 2020 discussed potential degeneracies in using certain line combinations. The one-line CLEDB inversion directly utilizes the methods and results described in this work.
5. Schiffmann, Brage, Judge, Paraschiv & Wang, ApJ, 2021 performed large-scale Lande g factor calculations for our ions of interest and discusses degeneracies in context of their results.
6. Casini & Judge, ApJ, 1999 and Judge & Casini, ASP proc., 2001 described the theoretical line formation process implemented by CLE, the coronal forward synthesis Fortran code that is currently utilized by CLEDB.

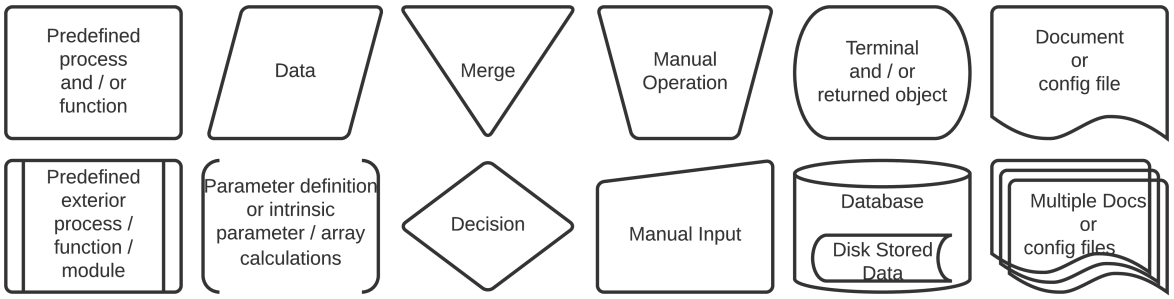
CHAPTER

TWO

MODULE OVERVIEW

2.1 Algorithm Flowchart Definitions

We illustrate our definitions for the flowcharts presented throughout this document that describe the principal algorithm operations. We note that the flowcharts are not exhaustive and are not meant to describe the code to the level of individual operations. The flowcharts list the main variables and functions used along with how they are processed by the different modules and intrinsic functions, and the resulting outputs.

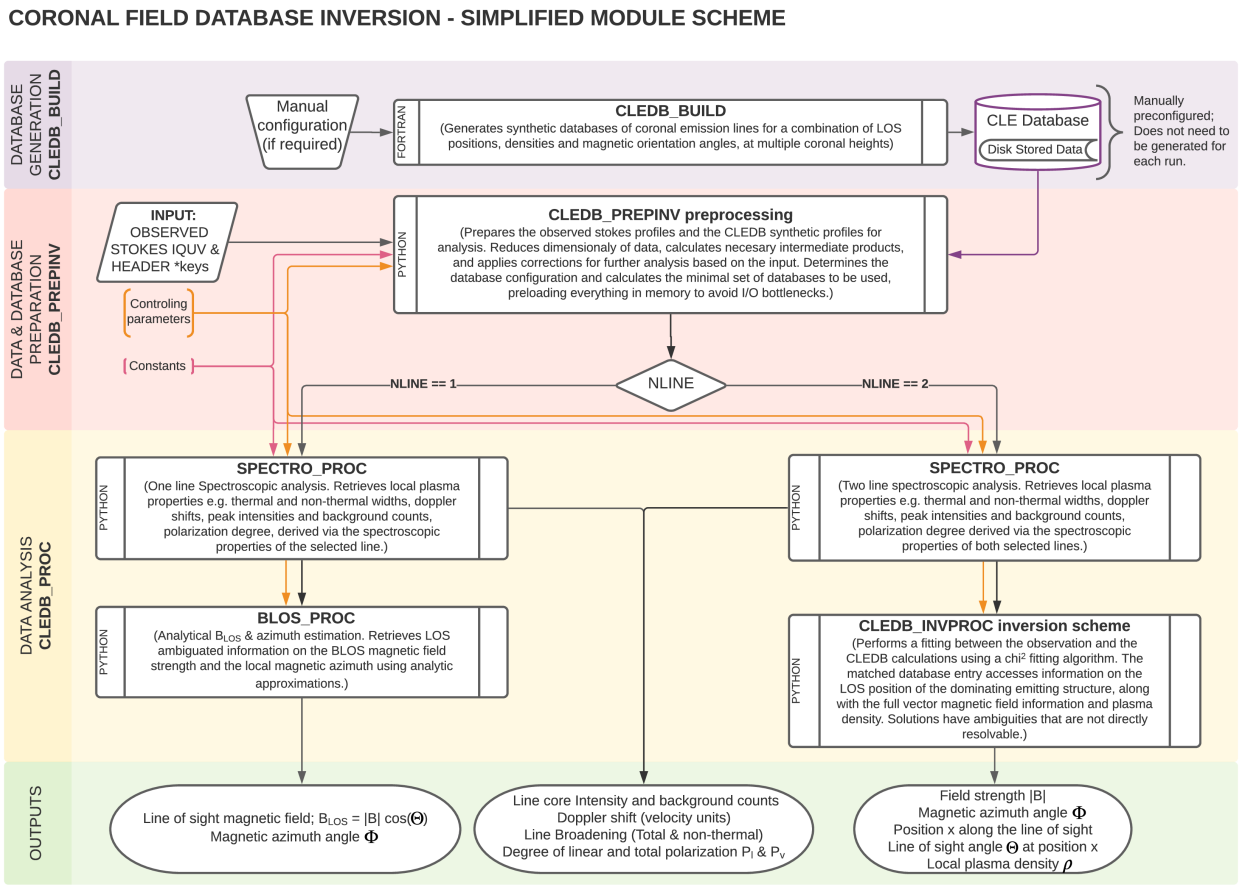


2.2 The CLEDB Modules

The CLEDB inversion algorithm can be split into three parts:

- The `CLEDB_BUILD` module contains pre-configured scripts for generating a database used downstream to fit the observations. Fortran binaries and Bash scripting is used by this module. Running the default configured `rundb_1line.sh` script for each line of your observation is enough in most cases. Please see the dedicated *README-RUNDB* for more detailed instructions.
- The `CLEDB_PREPINV` module prepares the data for analysis and matches required databases to read into memory (for the 2-line branch). Two main functions **SOBS_PREPROCESS** and **SDB_PREPROCESS** match and prepare the data and databases for analysis. The *ctrlparams* and *constants* classes are imported separately and fed to the module.
- The `CLEDB_PROC` module encompasses the main data analysis functions **SPECTRO_PROC**, **BLOS_PROC**, and/or **CLEDB_INVPROC**. These apply *analytical solutions* or database inversion schemes on the input observational data to recover the desired plasma and magnetic field parameters (e.g. the `OUTPUTS`).

Hint: The *MAIN README* contains instructions on how to end-to-end run the provided examples.



The flowchart schematic presents the modules along with the main inputs and outputs. Each module is described separately in the following sections along with detailed operation flowcharts. The most important variables and functions are described for each inversion module component. The definitions and accompanying diagrams are not meant to be 1:1 mirrors of the coding, but merely to trace the most crucial operations and resulting outputs. Common terminology is defined in the last section.

Note: Additionally, more extended comments can be found in each module's Python/Bash scripts.

2.3 The Python Modules

The following Python packages are required. For numerical computation efficiency, the inversion heavily relies on the Numpy and Numba packages.

- **Numpy**
  Numpy provides fast vectorized operations on its self implemented-ndarray datatypes. All Python based modules are written in a Numpy-centric way. Functional equivalent pure Python coding is avoided when possible due significantly slower runtimes. Numpy version specific (1.23) documentation is found here.
- **Numba**
  Numba implements just in time (JIT) compilation decorators and attempts where possible to perform loop-lifting and scale serial tasks on available CPU threads. Numba has two modes of operation, object-mode and

non-python mode. Non-python mode is the desired target. It will maximize optimization and runtime speed, but is significantly limited in terms of Python and/or Numpy function compatibility. Object-mode has full Python compatibility but the applicable optimizations are significantly less effective in most situations.

A Numba fully-enabled implementation can utilize only a small subset of Python and Numpy functions. Significant data sanitation and statically defined function I/O are required in order to enable runtime optimization and parallelization. Due to these sacrifices, coding implementations are not always clear and straightforward.

> **Danger:** Numba non-python mode is not directly equivalent to parallel/loop-lifted runs. A decision on running a specific function in parallel needs manual consideration. Loop-lifting "all" non-python functions leads to significant worsening of execution efficiency. We use a *control parameter* described later-on to control the use of loop-lifting/parallelization, but only on functions that would benefit from the effect. A significant number of non-python compatible functions have implicit parallelization set to disabled **for good reason**.

Extensive documentation and examples can be found in the Numba documentation. The version specific (0.56.4) documentation is available here.

> **Attention:** The `CLEDB_PREPINV` module can only be compiled in Numba object-mode due to disk I/O operations that are not implemented in non-python mode. Object-mode is usually not faster than normal Python, but it does benefit from loop-lifting parallelization, that is beneficial to our usecase.

- **pyyaml**
  YAML format library utilized in the *ctrlparams* class to enable or disable Numba global options.
- **Scipy**
  Used for spectroscopic fitting and statistics.
- **Jupyter, Jupyterlab, Matplotlib and Ipympl**
  Optional libraries for data visualization, plotting, widgets, etc.
- **Glob, and OS**
  Additional modules used primarily by `CLEDB_PREPINV` for disk I/O operations.
- **Time and Sys**
  Used during debug runs with high level of *verbosity*.
- **Sphinx, Sphinx-rtd-theme and Mist-parser**
  Libraries for building documentation and processing markdown files. Disabled as these are not required by the inversion.

CHAPTER

THREE

INSTALLATION AND RUN INSTRUCTIONS

3.1 Code Distribution Download

The CLEDB coronal field inversion code distribution is publicly hosted on Github:

https://github.com/arparaschiv/solar-coronal-inversion

To create a local deployment use the git clone function:

```
git clone https://github.com/arparaschiv/solar-coronal-inversion
```

`CLEDB_BUILD` uses CLE precompiled GNU compatible Fortran binary executable files to generate databases. The module is run by utilizing a Bash script that enables parallel runs of serial computations. Binaries for both Darwin and Linux architectures are provided. More details are found in the *CLEDB_BUILD - Database Generation* module.

Note: The CLE FORTRAN source code is not included in this package. It is hosted in a separate repository https://github.com/arparaschiv/coronal-line-emission.

3.2 A CLEDBenv Python Environment

The `CLEDB_PREPINV` and `CLEDB_PROC` modules of CLEDB are written in Python. The Anaconda environment system is utilized. Anaconda documentation and installation instructions can be found here.

We provide a configuration file `CLEDBenv.yml` to create a custom Anaconda environment that groups the CLEDB utilized Python modules briefly *described above*.

```
name: CLEDBenv
channels:
 - defaults
 - numba
 - conda-forge
dependencies:
 - python=3.10
 - numpy=1.23
 - numba=0.56
 - pyyaml          ## used only in ctrlparams for NUMBA global options
 - scipy=1.9
 - ipython=8.6
 - ipympl
```

(continues on next page)

(continued from previous page)

```
 - jupyter
 - jupyterlab
 - matplotlib
# - sphinx=5.0.2        ## Not required by CLEDB; used only to build the documentation##␣
↪fixed version also stated in docs/requirements.txt
# - myst-parser=0.18.1  ## Not required by CLEDB; used to add markdown parsing to␣
↪documentation building.
# - sphinx-rtd-theme=1.1.1 ## Not required by CLEDB; theme for the documentation
```

The configuration file is used to configure the required and optional CLEDB Python packages. In a terminal session you can create the environment via:

```
conda env create -f CLEDBenv.yml
```

After installing all packages the environment can be activated via

```
conda activate CLEDBenv
```

The user can return to the standard Python package base by running

```
conda deactivate
```

If dependency problems arise for any reason, CLEDBenv can be deleted and recreated with the default fixed-version packages from `CLEDBenv.yml`.

```
conda remove --name CLEDBenv --all
```

Danger: The `CLEDBenv` anaconda environment installs specific version packages. Cross-compatibility is verified by us. This feature ensures additional codebase stability. Updating the individual Python packages inside the CLEDBenv environment is not recommended and might break code functionality.

3.3 Basic Run Example

1. Databases can be built with:

   ```
   ./CLEDB_BUILD/rundb_1line.sh
   ```

   See detailed database build instructions via the dedicated *README-RUNDB* found in the CLEDB_BUILD directory.

2. Two examples of running the full inversion package (assuming databases are already built) are provided as Jupyter notebooks and/or lab sessions.

   ```
   ./jupyter-lab test_1line.ipynb
   ./jupyter-lab test_2line.ipynb
   ```

Attention: Script versions for `test_1line` and `test_2line` are also available. These are tailored to be used in headless runs.

3.4 Headless Slurm Runs Overview

A few optimizations and modifications are provided in order to ensure a straightforward run of CLEDB on headless systems like research computing clusters. The Slurm environment is utilized.

Namely:

- Instructions for resource allocation, installing, and running the inversion in both interactive and batch modes of Slurm research computing setups are provided.
- The database building bash script has a dedicated headless version, *rundb_1line_slurm.sh*, where user options are hard-coded.
- Pure python test scripts (test_*.py) are exported/generated from the Jupyter notebooks (test_*.ipynb) to be compatible with batch allocations.

A dedicated readme covering this topic can be *consulted here* or as standalone in the mian CLEDB directory. The instructions are provided following the templates set by the Colorado University Research Computing User Guide.

CHAPTER

FOUR

INPUT VARIABLES AND PARAMETERS

4.1 Input Data and Metadata

header *keys

Set of input *header* metadata information that should describe the `sobs_in` *variable*. Expected keywords with simplified naming are detailed in this section. Detailed keyword information can be found for DKIST observations in the SPEC_0214 documentation.

`*keys to crpixn [] int` Reference pixel along x y or w (wavelength) direction.

`*keys to crvaln [] float` Coordinate value at crpix along x y or w (wavelength) direction.

`*keys to cdeltn [] float` Spatial (x,y) or spectral (w) platescale sampling along a direction.

`*keys to linpolref [] float` (0, 2π) range; Direction of reference for the linear polarization. This is a physical convention based on the fixed orientation of a spectrograph retarder. `linpolref` = 0 implies the direction is corresponding to a horizontal axis, analogous to the unit circle reference. Direction is trigonometric. The units are in radians.

`*keys to instwidth [] float` Measure of the utilized instrument's intrinsic line broadening coefficient. The units are in nm or km s^{-1}.

`*keys to nline [] int` Number of targeted lines; CLEDB can accept 1-line or 2-line observations.

`*keys to tline [:12, nline] string array` String array containing the name of lines to process. Naming convention follows the database *directory structure* defined as part of the `CLEDB_BUILD` module.

`*keys to xs/naxis1 [] int` Pixel dimension of `sobs_in` array along the horizontal spatial direction.

`*keys to ys/naxis2 [] int` Pixel dimension of `sobs_in` array along the vertical spatial direction.

`*keys to ws [] int` Pixel dimension of `sobs_in` array along the spectral dimension.

`*keys to skybright [] float` Sky brightness measurement used to judge observation quality and rms.

`*keys to grtngba \& grtngang [] float` The grating order and position; used to find central wavelength of input observation and judge suitability for inverting.

keyvals [16] list of variables

Packing of nx, ny, nw, nline, tline, crpix1, crpix2, crpix3, crval1, crval2, crval3, cdelt1, cdelt2, cdelt3, linpolref, instwidth in a list container to more easily feed the necessary keywords to other modules and/or functions.

sobs_in [nline][xs,ys,ws,4] float array; nline = 1 || 2 for (1-line) or (2-line)

`sobs_in` is passed as a numba typed list at input. Data are input Stokes IQUV observations of one or two lines respectively. The list will be internally reshaped as a numpy float array of [xs,ys,ws,4] or [xs,ys,ws,8] size.

4.2 Ctrl. Parameters *ctrlparams.py* Class

```
# """
# @author: Alin Paraschiv paraschiv.alinrazvan+cledb@gmail.com
# """

## To load the class:
#par=ctrlparams()
#print(vars(par))
#print(a.__dict__)

class ctrlparams:
    def __init__(self):
        ## general params
        self.dbdir        = '/home/noxpara/Documents/physics_prog/cle/db204_R500_UPDT/' ␣
↪                    ## directory for database
        self.verbose      = 1                                                           ␣
↪                ## verbosity parameter

        ## Used in CLEDB_PREPINV
        self.integrated = False                                                         ␣
↪                ## Boolean; parameter for switching to line-integrated data such as␣
↪CoMP/uCoMP/COSMO

        ## Used in CLEDB_PROC
        self.nsearch      = 4                                                           ␣
↪                ## number of closest solutions to compute
        self.maxchisq     = 100000000                                                   ␣
↪                ## Stop computing solutions above this chi^2 threshold
        self.gaussfit     = 2                                                           ␣
↪                ## Gaussian parametric fitting to use instead of the standard CDF␣
↪fitting
        self.bcalc        = 0                                                           ␣
↪                ## control parameter for computing the B magnitude for two line␣
↪observations.
        self.reduced      = False                                                       ␣
↪                ## Boolean; parameter for reduced database search
        self.iqud         = False                                                       ␣
↪                ## Boolean; parameter for IQU + Doppled data matching when Stokes V␣
↪is not measurable

        ##numba jit flags
        self.jitparallel  = True                                                        ␣
↪                ## Boolean; Enable or disable numba jit parralel interpreter
        self.jitcache     = False                                                       ␣
↪                ## Boolean; Jit caching for slightly faster repeated execution.␣
↪Enable only after no changes to @jit functions are required. Otrherwise kernel␣
↪restarts are needed to clear caches.
        self.jitdisable   = False                                                       ␣
↪                ## Boolean; enable or disable numba jit entirely; Requires python␣
↪kernel restart!
```

(continues on next page)

(continued from previous page)

```
        import yaml                                                                    ␣
↪                   ## Workaround to save the jitdisable keyword to a separate config␣
↪file.
        names={'DISABLE_JIT' : self.jitdisable}                                        ␣
↪                   ## Working kernel needs to be reset for numba to pick up the change
        with open('.numba_config.yaml', 'w') as file:                                  ␣
↪                   ## more info on numba flags can be found here: https://numba.
↪readthedocs.io/en/stable/reference/envvars.html
            yaml.dump(names, file)
```

Python class that unpacks control parameters used in all modules of the inversion setup. This is an editable imported module that users access and modify. The yaml import seen here is used to configure Numba global options.

Hint: The python importlib module is used in the example notebooks to reload changes.

4.2.1 General Parameters

dbdir [] string
Directory where the database is stored after being built with `CLEDB_BUILD`. This is the main directory containing all ions, and not one of the individual ion subdirectories (e.g. fe-xiii_1074, etc.).

verbose [] uint
Verbosity controlling parameter that takes vales 0-3. Levels are incremental (e.g. lev 3 includes outputs from levels 1 and 2) Due to Numba library incompatibilities, enabling higher level verbosity will block Numba optimization of the code.

- verbose == 0: Production; silent run.
- verbose == 1: Debug; prints the current module and operation being run.
- verbose == 2: Debug; implements warnings for common caveats.
- verbose == 3: Debug; will enable execution timing for selected sections. Numba will fall-back to object-mode.

4.2.2 PREPINV Parameters

integrated [] boolean
To use for calibrated COMP/UCOMP data. In this case, the profiles are integrated across the line sampling points. This parameter defaults to 0 to be applicable to spectroscopic data such as DKIST.

4.2.3 PROC Parameters

nsearch [] uint
Number of solutions to compute and return for each pixel.

maxchisq [] float
Stops searching for solutions in a particular pixel if fitting residuals surpassed this threshold.

gaussfit [] uint
Used to switch between CDF fitting and Gaussian parametric fitting with optimization.

- gaussfit == 0: Process the spectroscopic line parameters using only the CDF method.
- gaussfit == 1: Fit the line using an optimization based Gaussian procedure. This approach requires a set of 4 guess parameters. These are the approximate maximum of the emission (max of curve), the approximate wavelength of the core of the distribution(theoretical center of the line), its standard deviation (theoretical width of 0.16 nm), and an offset (optional, hard-coded as 0).
- gaussfit == 2: (Default) Fit the line using a optimization based Gaussian procedure. In this case, the initial guess parameters are fed in from the results of the CDF solution. In this case, the curve fitting theoretically optimizes for a more accurate solution, with sub-voxel resolution.

bcalc [] uint
Controls how to compute the field strength in the case of 2-line observations.

- bcalc == 0: Use the field strength ratio of the first coronal line in the list. Only applicable when Stokes V measurements exist; e.g. iqud is disabled.
- bcalc == 1: Use the field strength ratio of the second coronal line in the list. Only applicable when Stokes V measurements exist; e.g. iqud is disabled.
- bcalc == 2: Use the average of field strength ratios of the two coronal lines. Only applicable when Stokes V measurements exist; e.g. iqud is disabled.
- bcalc == 3: Assigns the field strength from the Doppler oscillation inputs. Only applicable when iqud is enabled.

reduced [] boolean
Parameter to reduce the database size before searching for solutions by using the linear polarization measurements. Dimensionality of db is reduced by over 1 order of magnitude, enabling significant sped-ups. Solution ordering might be altered in certain circumstances.

iqud [] boolean
Switches the main matching function of `CLEDB_PROC` in order to utilize either Stokes V or Doppler oscillations to compute the magnetic field strength and orientation.

4.2.4 Numba Jit Parameters

jitparallel [] boolean
When Jit is enabled (jitdisable == False), it controls whether parallel loop-lifting allocations are requested, as opposed to just optimize the execution in single-thread-mode.

jitcache [] boolean
Jit caching for slightly faster repeated execution. Enable only after no changes to @jit or @njit functions are required. Otherwise kernel restarts are needed to clear caches.

jitdisable [] boolean
Debug parameter to control the enabling of Numba Just in Time compilation (*JIT*) decorators throughout. Higher level verbosity requires disabling the JIT decorators. This functionality can only be done via Numba GLOBAL flags that need to be written to a configuration file `.numba_config.yaml`. Any change of this parameter requires a kernel restart.

4.3 Constants *constants.py* Class

```
## -*- coding: utf-8 -*-
## """
## @author: Alin Paraschiv paraschiv.alinrazvan+cledb@gmail.com
##
## """
## TODO: update final form of constants and units

## To load the class:
#consts=Constants()
#print(vars(consts))
#print(consts.__dict__)

class Constants:
    def __init__(self,ion):
        ## Solar units in different projections
        #self.solar_diam_arc = 1919
        #self.solar_diam_deg = self.solar_diam_arc/3600.
        #self.solar_diam_rad= np.deg2rad(0.0174533self.solar_diam_deg)
        #self.solar_diam_st = 2.*np.pi*(1.-np.cos(self.solar_diam_rad/2.))

        ##Physical constants
        self.l_speed       = 2.9979e+8                    ## speed of light [m s^-1]
        self.kb            = 1.3806488e-23                ## Boltzmann constant SI [m^2␣
↪kg s^-2 K^-1];
        self.e_mass        = 9.10938356e-31               ## Electron mass SI [Kg]
        self.e_charge      = 1.602176634e-19              ## Electron charge SI [C]
        self.bohrmagneton  = 9.274009994e-24*1.e-4        ## Bohr magneton [kgm^2s^2 G^-
↪1]; Mostly SI; T converted to G;
        self.planckconst   = 6.62607004e-34               ## Planck's constant SI [m^2␣
↪kg s^-1];

        # ion/line specific constants
        if (ion == "fe-xiii_1074"):
            self.ion_temp = 6.25                      ## Ion temperature SI [K]; li+2017
↪<--Chianti
            self.ion_mass = 55.847*1.672621E-27       ## Ion mass SI [Kg]
            self.line_ref = 1074.62686                ## CLE Ion referential wavelength␣
↪[nm]
            #self.line_ref = 1074.68                   ## Ion referential wavelength [nm]
            self.width_th = self.line_ref/self.l_speed*(4.*0.69314718*self.kb*(10.**self.
↪ion_temp)/self.ion_mass)**0.5   ## Line thermal width
            self.F_factor= 0.0                                ## Dima & Schad 2020 Eq. 9
            self.gu = 1.5                                     ## upper level g factor
            self.gl = 1                                       ## lower level g factor
            self.ju = 1                                       ## upper level angular␣
↪momentum
            self.jl = 0                                       ## lower level angular␣
↪momentum
            self.g_eff=0.5*(self.gu+self.gl)+0.25*(self.gu-self.gl)*(self.ju*(self.ju+1)-
↪self.jl*(self.jl+1))    ## LS coupling effective Lande factor; e.g. Landi& Landofi␣
↪2004 eq 3.44; Casini & judge 99 eq 34
```

(continues on next page)

(continued from previous page)

```
        elif (ion == "fe-xiii_1079"):
            self.ion_temp = 6.25                        ## Ion temperature SI [K]; li+2017
↪<--Chianti
            self.ion_mass = 55.847*1.672621E-27         ## Ion mass SI [Kg]
            self.line_ref = 1079.78047                  ## CLE Ion referential wavelength␣
↪[nm]
            #self.line_ref = 1079.79                     ## Ion referential wavelength [nm]
            self.width_th = self.line_ref/self.l_speed*(4.*0.69314718*self.kb*(10.**self.
↪ion_temp)/self.ion_mass)**0.5  ## Line thermal width
            self.F_factor= 0.0                          ## Dima & Schad 2020 Eq. 9
            self.gu = 1.5                               ## upper level g factor
            self.gl = 1.5                               ## lower level g factor
            self.ju = 2                                 ## upper level angular momentum
            self.jl = 1                                 ## lower level angular momentum
            self.g_eff=0.5*(self.gu+self.gl)+0.25*(self.gu-self.gl)*(self.ju*(self.ju+1)-
↪self.jl*(self.jl+1))    ## LS coupling effective Lande factor; e.g. Landi& Landofi␣
↪2004 eg 3.44; Casini & judge 99 eq 34

        elif (ion == "si-x_1430"):
            self.ion_temp = 6.15                        ## Ion temperature SI [K]; li+2017
↪<--Chianti
            self.ion_mass = 28.0855*1.672621E-27        ## Ion mass SI [Kg]
            self.line_ref = 1430.2231                   ## CLE Ion referential wavelength␣
↪[nm] ;;needs to be double-checked with most current ATOM
            #self.line_ref = 1430.10                     ## Ion referential wavelength [nm]
            self.width_th = self.line_ref/self.l_speed*(4.*0.69314718*self.kb*(10.**self.
↪ion_temp)/self.ion_mass)**0.5  ## Line thermal width
            self.F_factor= 0.5                          ## Dima & Schad 2020 Eq. 9
            self.gu = 1.334                             ## upper level g factor
            self.gl = 0.665                             ## lower level g factor
            self.ju = 1.5                               ## upper level angular momentum
            self.jl = 0.5                               ## lower level angular momentum
            self.g_eff=0.5*(self.gu+self.gl)+0.25*(self.gu-self.gl)*(self.ju*(self.ju+1)-
↪self.jl*(self.jl+1))    ## LS coupling effective Lande factor; e.g. Landi& Landofi␣
↪2004 eg 3.44; Casini & judge 99 eq 34

        elif (ion == "si-ix_3934"):
            self.ion_temp = 6.05                        ## Ion temperature SI [K]; li+2017
↪<--Chianti
            self.ion_mass = 28.0855*1.672621E-27        ## Ion mass SI [Kg]
            self.line_ref = 3926.6551                   ## CLE Ion referential wavelength␣
↪[nm] ;;needs to be double-checked with most current ATOM
            #self.line_ref = 3934.34                     ## Ion referential wavelength [nm]
            self.width_th = self.line_ref/self.l_speed*(4.*0.69314718*self.kb*(10.**self.
↪ion_temp)/self.ion_mass)**0.5  ## Line thermal width
            self.F_factor= 0.0                          ## Dima & Schad 2020 Eq. 9
            self.gu = 1.5                               ## upper level g factor
            self.gl = 1                                 ## lower level g factor
            self.ju = 1                                 ## upper level angular momentum
            self.jl = 0                                 ## lower level angular momentum
            self.g_eff=0.5*(self.gu+self.gl)+0.25*(self.gu-self.gl)*(self.ju*(self.ju+1)-
↪self.jl*(self.jl+1))    ## LS coupling effective Lande factor; e.g. Landi& Landofi␣
↪2004 eg 3.44; Casini & judge 99 eq 34
```

(continues on next page)

(continued from previous page)

```
        else:
            print("Not supported ion or wrong string. Ion not Fe fe-xiii_1074, fe-xiii_
↪1079, si-x_1430 or si-ix_3934.\nIon specific constants not returned!")
```

Python class that unpacks physical constants needed during the inversion. The constants are mainly utilized by the SPECTRO_PROC and BLOS_PROC modules. Ion specific and general atomic and plasma constant parameters are packed herein. The class self-initializes for each requested ion providing its **ion specific** parameters in a dynamic fashion.

4.3.1 Physical Constants

solar_diam [float*4]
Solar diameter in arcsecond, degrees, radians, and steradian units.

l_speed [] float
Speed of light; Units in SI [m s^{-1}]

kb [] float
Boltzmann constant; Units in SI [m^{-2} kg s^{-2} K^{-1}]

e_mass [] float
Electron mass; Units in SI [Kg]

e_charge [] float
Electron charge; Units in SI [C]

planckconst [] float
Planck's constant; Units in SI [m^{-2} kg s^{-1}]

bohrmagneton [] float
Bohr Magneton; Units in mostly in SI. T converted to Gauss units [kg m^{-2} s^{-2} G^{-1}]

4.3.2 Ion Specific Constants

Note: Four sets of these constants are provisioned for the four possible lines to invert.

ion_temp [] float
Ion temperature; Units in SI [K]

ion_mass [] float
Ion mass; Units in SI [Kg]

line_ref [] float
Theoretical line core wavelength position; Units in [nm]

Caution: Simulation examples might have different set line centers based on the spectral synthesis code used. Doppler shift products might not compute correctly.

width_th [] float
Thermal width analytical approximation; Units in [nm]

F_factor [] float
Additional factor described by Dima & Schad, ApJ, 2020. Useful when calculating *LOS* products in the BLOS_PROC module

gu and gl [] float
LS coupling atomic upper and lower energy levels factors

ju and jl [] float
Atomic upper and lower level angular momentum terms

g_eff [] float
LS coupling effective Landé g factor

CHAPTER

FIVE

CLEDB_BUILD - DATABASE GENERATION

Purpose:

The CLEDB_BUILD module is used to generate a database of synthetic IQUV profiles for the four provisioned ions, with a range of density estimations, range of possible *LOS* positions, and all possible magnetic angle configurations, for one magnetic field strength B = 1. In normal circumstances this module is only run once per system where the inversion is installed. A module diagram is provided in this section.

5.1 CLEDB_BUILD Configuration

Here we describe the scripts included in the config directory.

DB.INPUT

Main configuration file for the database generation. It contains the *physical parameters* configurations for the databases to be generated.

```
*  NY NED  NX  NBPHI NBTHETA
* standard 51 10 61 180 90
51 10 61 180 90
* ELNMIN ELNMAX   YMIN   YMAX    XMIN   XMAX   BPMIN     BPMAX      BTMIN     BTMAX␣
↪
   -1.5   2.3   1.000   1.500   -1.50   1.50   0.000   6.28318548   0.000   3.
↪14159274
*  above line is important to be kept with 3 spaces and same number of decimals for␣
↪bash script to work
```

Danger: It is critical to keep the same number of parameter decimals and white spaces between the values when modifying the DB.INPUT configuration file. The automated job-scripts that run the jobs are dependent on precisely reading each entry.

ATOM.ion

This set of files contain the atomic configuration data to be used for calculations. Full level atoms would have a too high computational requirement to use. To avoid this, we use reduced calculations. For example take the Fe XIII lines. The atom configurations are set up as reduced 4-level and 6-transition calculations including the M1 and E1/E2 transitions from upper levels to M1 upper levels. See Fig. 3 of Casini & Judge, ApJ, 1999 This level/transition setup mimics the IQUV fluxes from a full level calculation for each of the the selected infrared coronal lines.

Caution: Advanced understanding required. In general, users should not modify the ATOM files.

INPUT.ion(a/b)
These are input and configuration files that are read when generating databases. The *wlmin* and *wlmax* parameters control which lines described in the ATOM.ion files are processed. In the case of Fe XIII, a separate INPUT.ion configuration (a/b) is needed for each line to produce distinct database entries.

Hint: In the case of Fe XIII, a custom INPUT.ion configuration with *wlmin* and *wlmax* constraints that includes both lines can be created. This would lead to the synthesis of a direct 2-line database. The *sdb_preprocess function* in the CLEDB_PREPINV module is provisioned to process such a database configuration. This is an alternate configuration that can be fairly straightforward to implement for a setup aimed at inverting only for the Fe XIII pair. Please note that this is a legacy feature that should not be treated as a default/expected configuration for generating databases.

IONEQ
Ionization equilibrium data from CHIANTI.

GRID.DAT
Defines the range and resolution of a CLE simulation. In the case of database building it has no significant functionality and is only required due to CLE's implicit dependency on it's import.

db"xxxx"_"arch"
Executable CLE binaries for generating databases. *xxxx* is the used version of the CLE Fortran code. arch can be *linux*, *rclinux* or *darwin*. The three different versions are provided in the distribution for cross-platform compatibility.

- linux – Debian compiled
- rclinux – CentOs compiled on research computing system.
- darwin – mac osx x86 compiled.

Attention: Ideally, the *xxxx* version of the CLE code should match its latest stable release.

5.1.1 DB.INPUT Parameters

ny, ymin, ymax
Number of y (horizontal) heights in $R_\odot$ units for which to compute database entries. The ny heights are spanned between ymin and ymax values. Regardless of user input, polarization signal can not computed at this time for $R_\odot$ < 1 due to the assumptions and interpretation focused on off-limb coronal emission.

Attention: Observations show that the amount of polarization in Fe XIII drastically decreases with height. One should not normally expect to reasonably recover full Stokes polarization signal at y > $1.5_\odot$.

ned, elnmin, elnmax
Number and range of ambient electron density values for which to compute calculations. elnmin and elnmax define a logarithmic range in which to spread the ned densities. The center of this range is an analytical approximation of a standard electron density expected for a y height above the limb following the Baumbach formulation. See equation 12 and discussion in Paraschiv & Judge, SolPhys, 2022 . For example, at y = $1.1R_\odot$ we expect a logarithm of density $\log(n_e) \sim 8$ cm^{-3}. Setting ned = 10, elnmin = -2 and elnmax = 2 will generate databases for 10 density values logarithmically scaled between $\log(n_e) \approx 6$ - 10 cm^{-3}.

Attention: Please keep in mind some potential inversion breaking assumptions. A reasonable density range of $\log(n_e)$ 7-10 is compatible with:

i. low enough densities so that collisional depolarization becomes unimportant inside the Hanle saturated regime;

ii. compatible with expected plasma densities in a standard 1.0-1.5$R_\odot$ observation range (also remember above point about polarization vs. height).

`nx, xmin, xmax`
Number of x (depth along the *LOS*) positions to compute databases for in $R_\odot$ units. The `nx` positions are linearly spanned between `xmin` and `xmax` values.

Attention: Due to geometric considerations, setting `xmin` and `xmax` values to more than $\pm$ 1.0 $R_\odot$ will most probably not result in practical benefits. This is because a higher 1.5 $R_\odot$ apparent height, a 1.0 $R_\odot$ depth would correspond to an actual height above the limb of 1.8 $R_\odot$. This is in the more extreme range of the polarization formation vs height issue described above.

`nbphi, bpmin, bpmax`
Number and range of CLE magnetic *LOS* angles to compute. The `nbphi` angles are spread along a `bpmin` - `bpmax` range set to 0 - 2π by default.

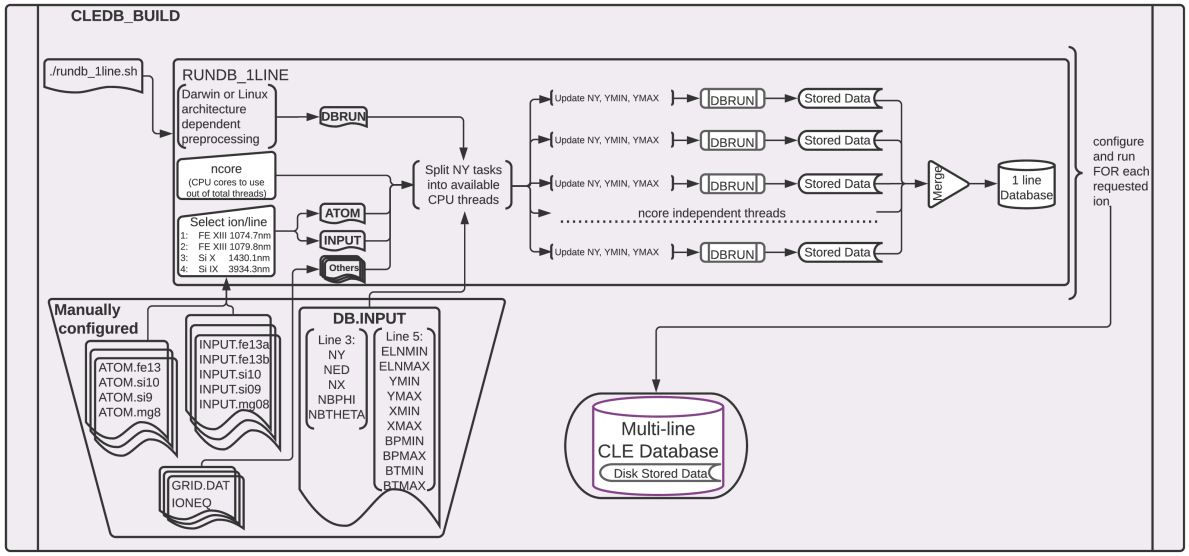
`nbtheta, btmin, btmax`
Number and range of magnetic CLE azimuth angles to compute. The `nbtheta` range is set to `btmin` - `btmax` . By default this is set to a 0 - 1π reduced range due to spherical transformation definitions.

Danger: Due to how the problem is posed, please do not interchange the maximum ranges between the two magnetic angles, as it would lead to execution errors.

5.2 The CLEDB_BUILD Job Script

The *rundb_1line.sh* job script will ingest the ATOM, INPUT, DB.INPUT, etc. files and split the job into available CPU threads. The user is asked for keyboard input on how many threads to use and for which line/ion to generate a database.

DATABASE GENERATION DIAGRAM (CLEDB_BUILD)



The script runs in a Bash shell terminal session. It can handle both Linux and Darwin (OSX) environments. For OSX, an additional dependency is required. Users need to install the GNU implementation of the sed command. The simplest way is to achieve this is by using the homebrew environment:

```
brew install gnu-sed
```

The job script will split the serial `ny` tasks on the requested CPU threads and run in dedicated folders that will be sanitized upon completion, preserving only the output database files and metadata headers.

Logs for each script ("X") are written in real time and can be checked interactively while the job is running.

```
tail BASHJOB_"X".LOG
```

A Slurm enabled version, *rundb_1line_slurm* which has hard-coded choices to be compatible with headless runs is also provided. The parameters need to be checked manually before running. Detailed information about the Slurm enabled routines can be found in the detailed *README-SLURM* section.

Note: A standalone README-SLURM.MD readme is included in the inversion root directory.

Extensive notes about the parallel job script implementations are found in the detailed *README-RUNDB* section.

Note: A standalone README-RUNMD.md readme is included with the `CLEDB_BUILD` module.

5.3 CLEDB_BUILD Output

Databases for one up to four of the currently available ions/lines can be constructed by running the job script successively.

Tip: As long as enough free CPU threads are available, multiple *rundb_1line.sh* jobs can be started simultaneously for **different** ions as there is no storage or computational overlap.

The output database is written to the storage disk. Each individual line will be written in its dedicated folder.

Note: Prior to git commit *update-iqud* `CLEDB_BUILD` wrote compressed data using a simple float64 $\rightarrow$ int16 conversion using a division constant, set to -2.302585092994046e15. Same constant needs to be used when writing but also when reading databases into memory as part of the `CLEDB_PREPINV` module. **This approach proved to create numerical instabilities and is currently disabled.**

A database folder hierarchical system is needed in order to ingest the selected database calculations by the `CLEDB_PREPINV` module. The folder system is defined as: *element-ionstage_line*.

1. **fe-xiii_1074**
2. **fe-xiii_1079**
3. **si-x_1430**
4. **si-ix_3934**

Note: A fifth option for directly writing two line databases for Fe XIII is still preserved as a legacy option as *described above*. The *.hdr* and *.DAT* database files need to be placed in the main *ctrlparams dbdir key* without a specific line subfolder.

This convention is used by all three modules of CLEDB.

> **Warning:** Running successive jobs for the **same** ion/line will **erase** its database calculations if they exist!

Individual data stores for each computed height are created to ease I/O operations when reading databases into memory for inverting. A db"xxxx".dat file is generated at each y height in the `ny` set, where "xxxx" represents the distance *above the limb* in units of $R_\odot$ (DB0000.dat corresponds to the solar limb or a height of $1.00R_\odot$). A metadata *db.hdr* file is produced in the individual line directory that contains the range dimensions and parameters applicable to any one database set of files.

> **Danger:** The user should not change the parameter configurations in `DB.INPUT` between multiple ion/line runs that should be part of the same database.

Generating $\sim 5 \cdot 10^8$ calculations per line for two lines will occupy $\approx$ 32 Gb of disk space with no storage compression.

CHAPTER

SIX

CLEDB_PREPINV - PRE-PROCESSING

Purpose:

The `CLEDB_PREVINV` module processes both the input data and the `CLEDB_BUILD` generated databases to prepare for the main inversion processing.

For 1-line cases, only the observation is pre-processed. Observation keywords are ingested, the geometric height is computed and the spectroscopic IQUV profiles are integrated.

In addition for 2-line observations, the observation linear polarization is rotated to match the database calculation as described in Paraschiv & Judge, SolPhys, 2022. The module then performs a height match between the input data and database configuration. Only the optimal subset of database height entries are pre-loaded into memory to minimize I/O operations but also to avoid I/O bottlenecks when running the analysis routines of the `CLEDB_PROC` module.

6.1 CLEDB_PREPINV Module Functions

Note: The ◇, ▷, and ▷▷ symbols respectively denote main, secondary, and tertiary (helper) level functions. Main functions are called by the example scripts. Secondary functions are called by the main functions, and tertiary from either main or secondary functions.

◇ **SOBS_PREPROCESS**

Main function to process an input observation and ingest the relevant *header* keywords. Generally, this function iterates over the observation maps and sends each pixel to the internal functions. It returns a processed observation array (input dependent) that is ready for analysis. Additional products are calculated. e.g. a height map (used to match databases), signal statistics, etc. via its subfunctions.

▷ **OBS_CALCHEIGHT**

Calculates height map of the same xy dimensions as the input array. Each pixel encodes the solar height in units of $R_\odot$.

▷ **OBS_INTEGRATE**

Estimates background counts using a cumulative distribution function (*CDF*) statistical method, then integrates along the wavelength dimension, in all IQUV components of all input lines. Profile integration is required because the database dimensionality and inversion computational times would not be feasible when processing full-spectra observations. See Paraschiv & Judge, SolPhys, 2022 for additional information.

▷▷ **OBS_CDF**

Computes the CDF distribution from spectra corresponding to one voxel.

▷ **OBS_QUROTATE**

If ingested a 2-line observation, the *Stokes Q and U* components are rotated to match the database's reference direction with the pbservation reference direction. The observation reference direction should be read

as input from the header metadata. This enables using just a 1D database computation (along y heights) to match the any observed Stokes profiles, under any linear polarization reference in a 2D map with these applied rotational transforms. See Paraschiv & Judge, SolPhys, 2022 for extended information.

◇ **SDB_PREPROCESS**

Main function for selecting and reading into memory the optimal database calculations that are compatible with the observations processed via **SOBS_PREPROCESS**.

▷ **SDB_FILEINGEST**

Glob the database directory to ingest available database heights and process the database configuration using the header metadata.

▷ **SDB_FINDELONGATION**

Compares the database entries (along *ny*) with the heights covered by the observation to deduce the closest matching database entries and minimizes the number of database DBXXXX.dat files to be read into memory.

▷ **SDB_PARSEHEADER**

Parses the database header information from the db.hdr file.

▷▷ **SDB_LCGRID**

Computes the grid spacing for the logarithm of density ranges covered in the database. The grid is correspondent to the density configuration in *DB.INPUT* of the database calculations.

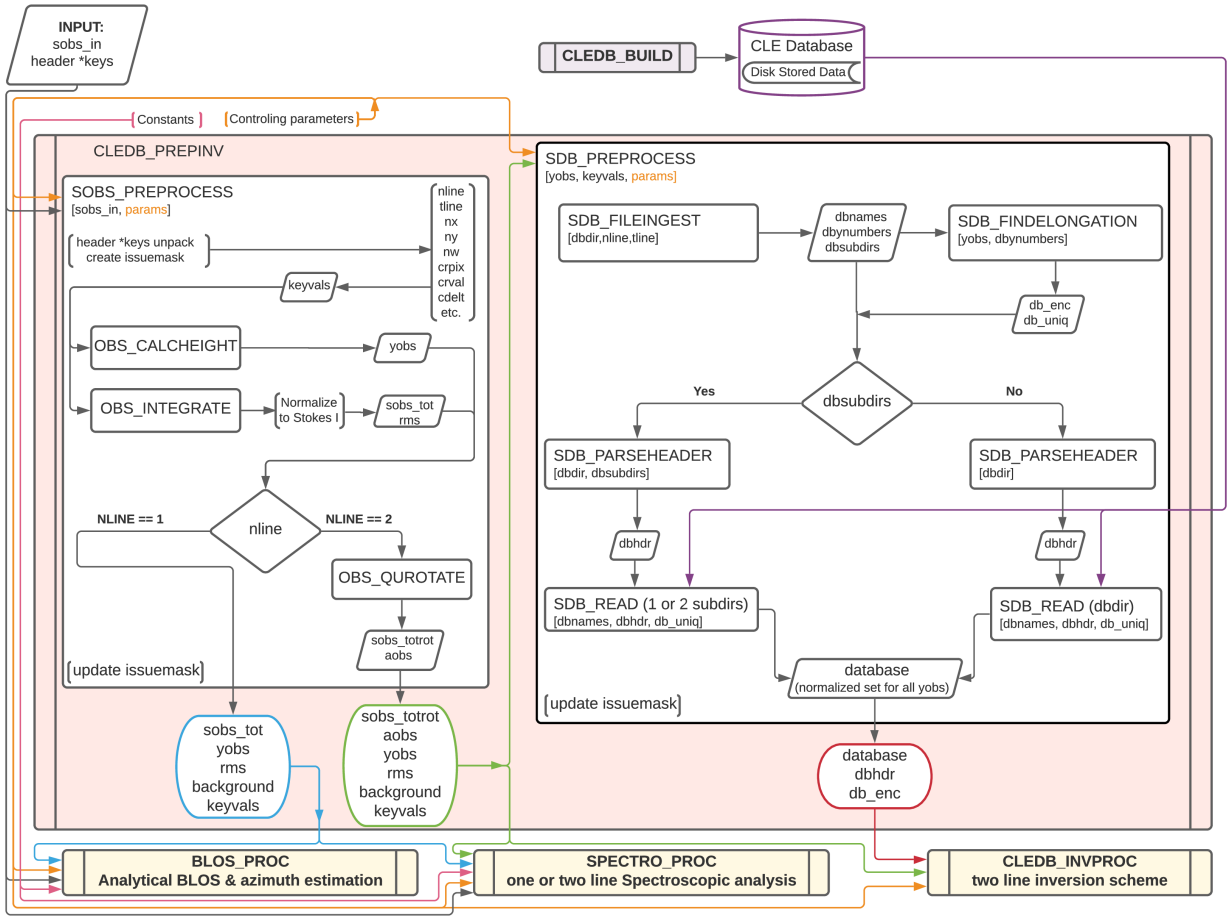
▷ **SDB_READ**

Reads all needed database files. This concludes all the disk I/O operations done during one run of the inversion.

> **Warning:** As databases are written as binary files, the variable type fed to the *np.fromfile* reader needs to match the Fortran datatype CLE *dbe.f* uses to write the calculations. Currently these are set as single precision floats of np.float32 and REAL types respectively.

The `CLEDB_PREPINV` module is not fully compatible with Numba non-python mode, due to disk I/O operations. All non-python compatible functions are enabled in non-python mode while the rest are compiled in object-mode via the hard-coded "forcedobj=True" flag in the *@jit* decorators. *The Python Modules* section provides more details on the differences between the two Numba modes. The algorithm flow is described in the below diagram.

DATA PRE-PROCESSING DIAGRAM (CLEDB_PREPINV)



6.2 CLEDB_PREPINV Main Variables

`sobs_tot [xs,ys,nline*4] float array`
Contains the background subtracted, integrated, and normalized Stokes IQUV spectra for 1-line ([xs,ys,4]) or 2-line ([xs,ys,8]) observations.

`sobs_totrot [xs,ys,nline*4] float array`
Derived from `sobs_tot`. The Stokes Q and U components *are rotated* along the center of the Sun to match the reference direction for linear polarization (the reference in which the database is created by `CLEDB_BUILD`). In inner functions of `CLEDB_PROC` only one pixel is passed at a time as *sobs_1pix*. The variable is initialized as a "zero" array that is returned in the case of 1-line observations to keep a standardized function input/output needed for Numba vectorization.

`background [xs,ys,nline*4] float array`
Returns averaged background counts for each observed voxel and each Stokes component.

`rms [xs,ys,nline*4] float array`
Returns the root mean square (*RMS*) of the total counts in each Stokes profile. The rms calculation is correspondent to the ratio between intensity in the line core and background counts (the variance). This measurement shows the quality in the signal for a particular observed voxel.

`yobs [xs,ys] float array`
The header keyword input is used to construct a height projection for each observed voxel in units or $R_\odot$. In

inner functions of CLEDB_PROC only one pixel is passed at a time as *yobs_1pix*.

aobs [xs,ys] float array
: Stores the linear polarization angle transformation performed by the **OBS_QUROTATE** *function*. This information will be used to derotate the matched database profiles found by the **CLEDB_INVPROC** *2-line inversion function* for comparison. In inner functions of CLEDB_PROC only one pixel is passed at a time as *aobs_1pix*. The variable is initialized and returned as a "zero" array in the case of 1-line observations due to Numba vectorization requirements.

dbsubdirs [string] or [string list]
: Contains the directory structure formatted as described in the *CLEDB_BUILD Output* section.

database [ned,nx,nbphi,nbtheta,nline*4] list of float arrays
: The list is the minimal subset of databases that are compatible with the observation taken from the set of *ny* entries of the database.

dbhdr [ints, floats and strings] list
: Database header information containing the ranges and *physical parameters* used to generate the database.

db_enc [xs,ys] float array
: Keeps an encoding of which of the memory loaded databases (elements in list of databases) to use for matching in each pixel.

issuemask [xs,ys] float array
: An array that encodes issues appearing during processing. This array will be updated across all modules. The tentative *issuemask implementation* is described separately.

Note: Input variables, e.g. header *keys, sobs_in, *ctrlparams*, *constants*, etc. that are described in the *Input Variables and Parameters* section are not repeated in this section.

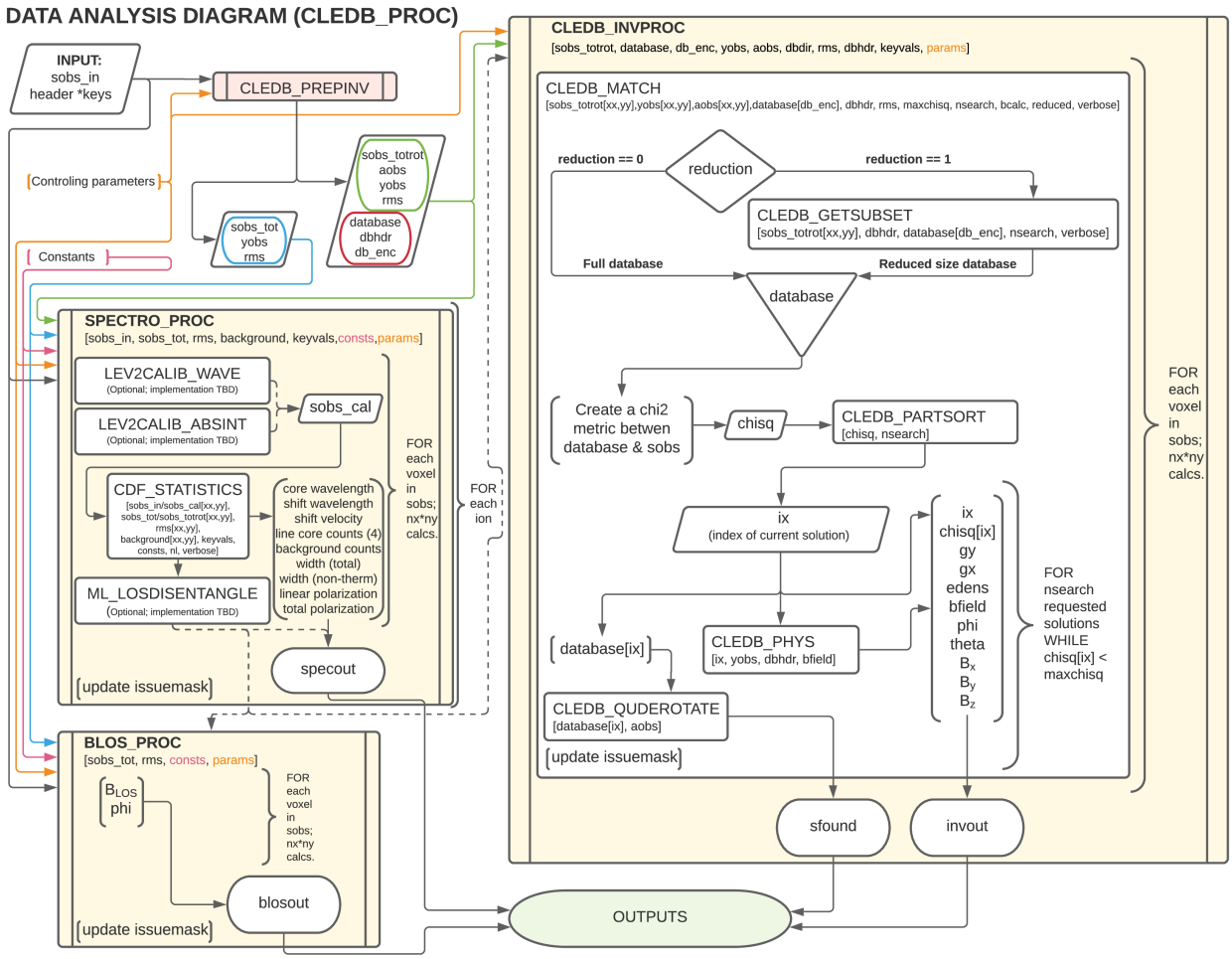
CHAPTER

SEVEN

CLEDB_PROC - ANALYSIS AND INVERSION

Purpose:

Three main functions, **SPECTRO_PROC**, **BLOS_PROC**, and **CLEDB_INVPROC** are grouped under the CLEDB_PROC data analysis and inversion module. Based on the 1-line or 2-line input data, two or three modules are called. Line of sight or full vector magnetic field outputs along with plasma, geometric and spectroscopic outputs are inverted here. The algorithm flow and a data processing overview is described in the flowchart.



7.1 The SPECTRO_PROC Function

Purpose:

Ingests the fully prepped data from *sobs_preprocess* and produces spectroscopic outputs for each input line. Part of the outputs are used downstream in **BLOS_PROC** or **CLEDB_INVPROC**. This module requires data in the formats as resulting from the `CLEDB_PREPINV` module. Optional sub-modules are envisioned to be integrated into this processing based on upstream instrument processing and retrieved data quality. This is a computationally demanding and time consuming function.

Note: The ◇, ▷, and ▷▷ symbols respectively denote main, secondary, and tertiary (helper) level functions. Main functions are called by the example scripts. Secondary functions are called by the main functions, and tertiary from either main or secondary functions.

7.1.1 SPECTRO_PROC Main Functions

◇ **SPECTRO_PROC**

▷ **CDF_STATISTICS**

Performs pixelwise analysis on the stokes IQUV spectra for each line and computes relevant spectroscopic outputs (see *specout*) by using via a *ctrlparams gaussfit key*. By default a gaussian fitting coupled with non-parametric approaches, namely the analysis of *CDF* functions is utilized.

▷▷ **OBS_CDF and ` OBS_GAUSSFIT**

These are two helper routines used by CDF_STATISTICS to perform parameter fits and estimations.

Hint: The *ctrlparams gaussfit key* == 2 represents the slowest component of the entire CDF_STATISTICS block. On the other hand, it is the most accurate and reliable profile fitting method of the three options.

▷ **ML_LOSDISENTANGLE (Opt.)**

Provisioned to be implemented at a later time. If observations permit, uses Machine Learning techniques for population distributions to help disentangling multiple emitting structures along the LOS in situations where the single point assumption might fail.

▷ **LEV2CALIB_WAVE (Opt.)**

Provisioned to be implemented at a later time. Higher order wavelength calibration using the spectroscopic profiles. See Ali, Paraschiv, Reardon, & Judge, ApJ, 2022 for additional details. This function can couple if the upstream wavelength accuracy of the input observation is lower than 0.005 nm.

Important: Upstream Level-1 calibration for DKIST is provisioned to match or exceed this accuracy requirement. Implementation is of low priority.

▷ **LEV2CALIB_ABSINT (Opt.)**

To be implemented at a later time, if feasible. Absolute intensity calibration function that produces an additional output, the calibrated intensity in *physical units*. The approach is not easily automated as it requires a more convoluted and specific planning of the observations to gather the necessary input data.

Important: This functions was provisioned in the incipient stages of the pipeline design. Subsequently, it was found that CLEDB can utilize only normalized Stokes profiles such that absolute calibrations are not required (see Paraschiv & Judge, SolPhys, 2022). Implementation is halted at this time.

7.1.2 SPECTRO_PROC Main Variables

`sobs_cal [nx,ny,sn,4] float array (opt.)`
Optional calibrated level-2 data in intensity and or wavelength units. This array would be used by the CDF_STATISTICS function instead of `sobs_in`.

Note: As LEV2CALIB_ABSINT and LEV2CALIB_WAVE are not currently implemented, `sobs_cal` is currently just a placeholder.

`specout [nx,ny,nline,12] output float array`

Returns 12 spectroscopic output products, for each `nline` input line and for every pixel location.

- **specout[:, :, :, 0]**
  Wavelength position of the line core. Units are [nm].
- **specout[:, :, :, 1]**
  Doppler shift with respect to the theoretical line core defined in the *constants* class *line_ref key*. Units are [nm].
- **specout[:, :, :, 2]**
  Doppler shift with respect to the theoretical line core defined in the *constants* class *line_ref key*. Units are [km s^{-1}].
- **specout[:, :, :, 3:6]**
  Intensity at computed line center wavelength (`specout[:, :, :, 0]`) for *Stokes I*, *Stokes Q and U*. Units are *ADU* or calibrated *physical units* if LEV2CALIB_ABSINT is utilized.
- **specout[:, :, :, 6]**
  Intensity at lobe maximum for *Stokes V*. The signed "core" counts are measured in the core of the absolute strongest lobe. Thus, the Stokes V measurement will not match the wavelength position of the Stokes IQU intensities. Units are *ADU* or calibrated *physical units* if LEV2CALIB_ABSINT is utilized.

  Attention: If the *ctrlparams* class *iqud key* == True, this dimension will be returned implicitly as 0.

- **specout[:, :, :, 7]**
  Averaged background intensity outside the line profile for the *Stokes I* component. Since background counts are in theory independent of the Stokes measurement, we utilize just this one realization. Units are *ADU* or calibrated *physical units* if LEV2CALIB_ABSINT is used.
- **specout[:, :, :, 8]**
  Total line *FWHM*. Units are [nm].
- **specout[:, :, :, 9]**
  Non-thermal component of the FWHM line width. A measure or estimation of the instrumental line broadening/width will significantly increase the accuracy of this determination. Units are [nm].

- **specout[:, :, :, 10]**
  Fraction of linear polarization P_l with respect to the total *Stokes I* counts. Dimensionless.
- **specout[:, :, :, 11]**
  Fraction of total polarization (linear + circular) P_v with respect to the total *Stokes I* counts. Dimensionless.

Attention: Regardless if solving for 1-line or 2-line observations, `specout` will return both `nline` dimensions. In the case of 1-line observations, the `nline` = 1 dimension corresponding to the hypothetical second line is returned as 0 for all pixel locations. The unused dimension can be removed from the upstream example script, if needed. This behavior is known and enforced to keep output casting static, making the codebase compatible with Numba and speeding up execution.

7.2 The BLOS_PROC Function

Error: Stokes V observations are required for this analytical method. Thus, BLOS_PROC is incompatible with the IQUD *setup*.

Purpose:

Implements the *analytical solutions* of Casini & Judge, ApJ, 1999 and Dima & Schad, ApJ, 2020 to calculate the *LOS* projected magnetic field strength and magnetic azimuth angle. The module returns two degenerate constrained magnetograph solutions, where the one that matches the sign of the atomic alignment is more precise. The less precise "classic" magnetograph formulation is also returned.

Attention: There is not enough information in 1-line observations to deduce which of the two degenerate solution is "more precise". The "classic" magnetograph estimation is less precise than the optimal degenerate constrained magnetograph solution, but more precise than the other. The differences will vary from insignificant to tens of percents of the magnetic field strength based on observation and magnetic geometry, and degree of linear polarization. The choice of what product to use remains the prerogative of the user.

This branch requires only 1-line observations (4 stokes profiles). The setup is used to get as much magnetic information as possible (the field strength and *LOS* projection) in the absence of a second line. For a *sobs_tot* input of 2-lines, the module will produce independent products for each input line observation.

Hint: Observations of Si X 1430.10 nm will benefit from an additional alignemnt correction due to the non-zero F factor of this transition. Additional details in Dima & Schad, ApJ, 2020.

7.2.1 BLOS_PROC Main Functions

◇ **BLOS_PROC**

7.2.2 BLOS_PROC Main Variables

`blosout [nx,ny,4*nline] output float array`

The array returns 4 or 8 products containing *LOS* projected magnetic field estimations and magnetic azimuth angle in G units at each pixel location.

- **blosout[:, :, 0] and/or blosout[:, :, 4]**
  First degenerate constrained magnetograph solution for each respective line.
- **blosout[:, :, 1] and/or blosout[:, :, 5]**
  Second degenerate constrained magnetograph solution for each respective line.
- **blosout[:, :, 2] and/or blosout[:, :, 6]**
  "Classic" magnetograph solution for each respective line. Values lie in between the two above degenerate solutions.
- **blosout[:, :, 3] and/or blosout[:, :, 7]**
  Magnetic field azimuth angle derived from the Q and U linear polarization components of the respective line; -π to π range.

> **Warning:** A $\frac{\pi}{2}$ *degeneracy* will manifest due to using arctan functions to derive the angle.

7.3 The CLEDB_INVPROC Function

Purpose:

Main 2-line inversion function. **CLEDB_INVPROC** compares the preprocessed observations with the selected databases by performing a χ^2 goodness of fit measurement between each independent voxel and the complete set of calculations in the matched database. If **CLEDB_GETSUBSET** is enabled via *ctrlparams* class *getsubset key*, a presorting of the database entries to those that match the direction of observer linear polarization azimuth is performed. After the main sorting is performed, the best database solutions are then queried with respect to the physical parameters that gave the matched profiles. **CLEDB_INVPROC** acts like a pixel iterator and variable ingestion setup for either CLEDB_MATCHIQUV or CLEDB_MATCHIQUD.

> **Caution:** The *reduced* presorting will slightly change the final ordering of solutions in certain cases.

7.3.1 CLEDB_INVPROC Main Functions

◇ **CLEDB_INVPROC**

◇ **CLEDB_MATCHIQUV**

Matches a set of two full Stokes IQUV observations with a model observation of the same Stokes quantities. Solutions are 2 times degenerate with respect to the *LOS*. Matching is done individually for one pixel in the input array. This is a computationally demanding and time consuming function.

◇ **CLEDB_MATCHIQUD**

Matches a set of two partial Stokes IQU observations with a model observation of the same Stokes quantities. The matched solutions are initially more degenerate than **CLEDB_MATCHIQUV**, usually 4 timee with respect to LOS and signed field strength combinations. Additional information from Doppler oscillation tracking are brought-in to recover field strengths and reduce degeneracies (to 2 times). Matching is done individually for one pixel in the input array. This is a computationally demanding and time consuming function.

Note: Based on the *ctrlparams iqud key* only one of CLEDB_MATCHIQUV or CLEDB_MATCHIQUD setups is selected and utilized.

▷ **CLEDB_GETSUBSET**

When *enabled* via *ctrlparams*, the information encoded in the Stokes Q and U magnetic azimuth is used to reduce the matched database by approximately 1 order of magnitude in terms of observation-comparable calculations.

Important: If the subset calculation is *enabled* via *ctrlparams*, execution time in the case of large databases is significantly decreased.

▷ **CLEDB_PARTSORT**

A custom function that performs a **fast** partial sort of the input array because only a small subset of *ctrlparams nsearch key* solutions are requested via the *ctrlparams nsearch key*. This increases execution times by a few factors when requesting just few `nsearch` solutions (< 100 on 10^8 entries databases). CLEDB_PARTSORT is used by CLEDB_MATCHIQUV, CLEDB_MATCHIQUD, and CLEDB_GETSUBSET functions. In CLEDB_MATCH, CLEDB_PARTSORT performs a < `nsearch` sorting of database entries based on the χ^2 metric. In CLEDB_GETSUBSET, CLEDB_PARTSORT selects for each φ angle orientation only the most compatible ϑ directions based on the Φ_B azimuth given by the linear polarization Q and U measurements.

▷ **CLEDB_PHYS**

Returns 9 physical and geometrical parameters corresponding to each selected database index following the *ctrlparams nsearch* and *maxchisq* constraints. These products are returned as dimensions of the *invout* output variable.

▷▷ **CLEDB_PARAMS, CLEDB_INVPARAMS, CLEDB_ELECDENS, and CLEDB_PHYSCLE**

These are helper functions that prop CLEDB_PHYS by providing interfaces with the parameters encoded in selected databases and helping transform quantities between different geometrical systems.

▷ **CLEDB_QUDEROTATE**

The inverse function of *OBS_QUROTATE*. Derotates the Q and U components from each selected database entry, in order to make the set of fitted solutions directly comparable with the original integrated input *sobs_tot* observation.

7.3.2 CLEDB_INVPROC Main Variables

`database [ned,nx,nbphi,nbtheta,nline*4] list of float arrays`

Individual entries from the database list are fed to the **CLEDB_MATCHIQUV** or **CLEDB_MATCHIQUD** functions. From the database list, only the best matching height entry via *db_enc* variable is passed via the *database_sel* internal variable.

`database_sel [ned,nx,nbphi,nbtheta,nline*4] float array`

Subset index of the database list that is fed to CLEDB_MATCHIQUV or CLEDB_MATCHIQUD for matching the observation in one pixel. This alleviates memory shuffling and array slicing operations. The array is then reshaped into a 2D [ned*nx*nbphi*nbtheta,nline*4] form (e.g. [index,nline*4]). In the case where *ctrlparams*

reduction key is enabled, *database_sel* is additionally reduced with respect to the number of potential indexes to match.

sobs_totrot
Input variable to CLEDB_INVPROC described *here*.

sobs_dopp
Doppler oscillation magnetic field strength and POS orientation resulting from wave tracking.

Caution: sobs_dopp is only used as input to CLEDB_MATCHIQUD when *ctrlparams iqud* is enabled. For Numba consistency, an empty array is also passed to CLEDB_INVPROC when performing full IQUV inversions, but it is never used.

chisq [ned*nx*nbphi*nbtheta,nline*4] float array
Computes the squared difference between the voxel IQUV measurements [nline*4] and each index element of the database [index,nline*4].

sfound [nx,ny,nsearch,nline*4] output float array;
Returns the first nsearch de-rotated and matched Stokes IQUV sets from the database.

Warning: Solutions are skipped if the χ^2 fitting residuals are greater than the limit set by the *ctrlparams maxchisq key*. Thus, it is possible and even expected that less than requested *ctrlparams nsearch* solutions to be returned for one observed voxel.

invout [nx,ny,nsearch,11] output float array
Main 2-line inversion output products. invout contains the matched database index, the χ^2 fitting residuals, and 9 inverted physical parameters, for all *nsearch* closest matching solutions with respect to the input observation. The 11 parameters follow with individual descriptions.

- **invout[:,:,:,0]**
  The index of the database entry that was matched at the *nsearch* rank. The index is used to retrieve the encoded physics that match the observations.
- **invout[:,:,:,1]**
  The χ^2 residual of the matched database entry.
- **invout[:,:,:,2]**
  Plasma density computed via the database. This output is applicable for the Fe XIII 1074.68/1079.79 line ratio (same ion). Other line combinations will produce less accurate results due to the relative abundance ratios, that are varying dynamically. For a real-life observation, we do not consider trustworthy the implicit static relative abundance ratios of different ions, resulted from the *CHIANTI* tabular data implicitly ingested via the *ATOM files* when build databases. Units are logarithm of number electron density in cm^{-3}.
- **invout[:,:,:,3]**
  The apparent height of the observation. Analogous to the *yobs* variable. Units are $R_\odot$.
- **invout[:,:,:,4]**
  Position of the dominant emitting plasma along the *LOS*. Units are $R_\odot$.
- **invout[:,:,:,5]**
  Magnetic field strength recovered via the ratio of observed stokes V to database Stokes V (computed for B = 1 G); Uses *ctrlparams* class *bcalc key*. Units are [G].

> **Warning:** Due to how the problem is posed, **CLEDB_MATCHIQUV** can only use bcalc = 0, 1, or 2 while **CLEDB_MATCHIQUD** can only use *bcalc* = 3.

- **invout[:,:,:,6]**
  Magnetic field *LOS* angle in CLE frame. Range is 0 to 2π.
- **invout[:,:,:,7]**
  Magnetic field azimuth angle in CLE frame. Range is 0 to π.
- **invout[:,:,:,8]**
  B_x cartesian projected magnetic field depth/*LOS* component. Units are [G].
- **invout[:,:,:,9]**
  B_y cartesian projected magnetic field horizontal component. Units are [G].
- **invout[:,:,:,10]**
  B_z cartesian projected magnetic field vertical component. Units are [G].

> **Attention:** Regardless of the number of solutions (if any) that are found inside the *ctrlparams* *maxchisq* and *nsearch* constraints, the `invout` output array will keep its dimensions fixed and return "0" value fields to keep output data shapes consistent. This is a Numba requirement. Only the index is set to "-1" to notify the user that no result was outputted.

CHAPTER

EIGHT

OUTPUT PRODUCTS

8.1 Output Variable Overview

The main CLEDB inversion algorithm outputs are stored in the following variables:

- **specout**
  12 **SPECTRO_PROC** output products. These are described *here*.
- **blosout**
  4 **BLOS_PROC** output products. These are described *here*.
- **invout**
  11 **CLEDB_INVPROC** output products. These are described *here*.
- **issuemask**
  Records any issues that arise in processing for each pixel (to be implemented). The issuemask will be updated by both modules.

Note: The global process followed to produce these outputs is sketched in *The CLEDB Modules*.

8.2 Tentative Issuemask Implementation

The inversion will implement a confidence/issue map of size [nx,ny] for all spatial pixels in an input observation that will be returned along with the main output products.

Important: Issuemask encoding not currently active. Final form to be decided and implemented.

Example of issuemask coding:

Code 0
No apparent problem in pixel.

Code 1
One or more of Stokes I, Q, U are lower than noise RMS threshold.

Code 2
Stokes V is lower than noise RMS threshold.

Code 4
Linear polarization is close to Van-Vleck ambiguity (warning).

Code 8
B_{LOS} or Φ_B is lower than noise threshold (for 1-line observations).

Code 16
Database fit failed to converge reliably (for 2-line obs).

Code 32
One or more of B, Φ_B, Θ_B are lower than noise threshold (for 2-line observations).

Code 64
-————————TBD————————-

Code 128
-————————TBD————————-

Encoding the information is done sequentially when progressing through the different modules. This will be done by using powers of 2. The issuemask values thus become cumulative. Following the sketch map encoding from above, we take for example a pixel from a 1-line observation with unreliable Stokes V signal. The uncertainty in Stokes V will also lead to compromised B_{LOS} information. Thus, the *issuemask* will encode a value of 10 for that respective pixel.

CHAPTER

NINE

FOCUSED README FILES

Warning:

- The sections below are dinamically linked to standalone readme markdown files. A documentation rebuild will capture changes in any of the files.
- Consequentially, some dynamic links to functions will not work when displayed here as they would from the direct rendering of the readmes. This is because relative paths can not be kept consistent.

9.1 MAIN README

9.1.1 CLEDB Coronal Field Database Inversion

solar-coronal-inversion repository on github

Main repository for **CLEDB**, the Coronal Line Emission DataBase inversion code distribution.

Authors: Alin Paraschiv & Philip Judge. High Altitude Observatory & National Solar Observatory

Contact: arparaschiv “at” ucar.edu; paraschiv.alinrazvan+cledb “at” gmail.com

Main aim:

Invert coronal vector magnetic field products from observations of polarized light. The algorithm takes arrays of one or two sets of spectro-polarimetric Stokes IQUV observations to derive line of sight and/or full vector magnetic field products.

Applications:

Inverting magnetic field information from spectro-polarimetric solar coronal observations from instruments like DKIST Cryo-NIRSP; DL-NIRSP; COMP/UCOMP.

Documentation

1. Extensive documentation, **including installation instruction, dependencies, algorithm schematics and much more** is available in a dedicated documentation write-up. README-CODEDOC.pdf.
2. In-depth documentation for the Bash & Fortran parallel database generation module is provided in README-RUNDB.md.
3. Installation and usage on RC systems is described in README-SLURM.md.
4. This is a beta-level release. Not all functionality is implemented. TODO.md documents updates, current issues, and functions to be implemented in the near future.

System platform compatibility

1. Debian+derivatives Linux x64 – all inversion modules are fully working.
2. RC system CentOS linux x64 – all inversion modules are fully working. Additional binary executable is provided. May require local compiling.
3. OSX (Darwin x64) Catalina and Big Sur – all inversion modules are fully working; One additional homebrew package required. See README-CODEDOC.pdf.
4. Windows platform – not tested.

Examples

Install the CLEDB distribution, generate databases, and update the database save location in the *ctrlparams.py* class, as described in the README-CODEDOC. Afterwards, both 1-line and 2-line implementations of CLEDB can be tested with synthetic data using the two provided Jupyter notebook examples

1. test_1line.ipynb
2. test_2line.ipynb

The synthetic CLE test data is hosted separately here.

For terminal only compute systems the test data can be downloaded via the shell interface with the following method:

i. Load the following gdrive wrapper script into your bash window directly, or introduce it in your .bash_alias setup.

```
function gdrive_download () {   CONFIRM=$(wget --quiet --save-cookies /tmp/cookies.txt --
↪keep-session-cookies --no-check-certificate "https://docs.google.com/uc?
↪export=download&id=$1" -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*/\1\n/p');   wget --
↪load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=
↪$CONFIRM&id=$1" -O $2;   rm -rf /tmp/cookies.txt; }
```

ii. Download the file using its gdrive FILE_ID from the download link (*test data FILE_ID = 1XpBxEwU-UyaqYy1NjbVKyCHJhMUKzoV_m*):

```
gdrive_download FILE_ID local_path/local_name   (sometimes needs to be run two times to␣
↪set cookies correctly!)
```

Note: The script versions of both tests *test_1line.py* and *test_2line.py* together with the *test_cledb_slurm.sh* are slurm enabled to be used for headless RC system runs. These offer the same functionality as the notebooks (from which they are directly generated from). See the dedicated README-SLURM for additional information.

Both test examples are expected to fully execute with parallel job spawning via Numba/JIT in a correct installation.

Scholarly works supporting the CLEDB inversion

1. Paraschiv & Judge, SolPhys, 2022 covered the scientific justification of the algorithm, and the setup of the CLEDB inversion.
2. Judge, Casini, & Paraschiv, ApJ, 2021 discussed the importance of scattering geometry when solving for coronal magnetic fields.
3. Ali, Paraschiv, Reardon, & Judge, ApJ, 2022 performed a spectroscopic exploration of the infrared regions of emission lines available for inversion with CLEDB.
4. Dima & Schad, ApJ, 2020 discussed potential degeneracies in using certain line combinations. The one-line CLEDB inversion utilizes the methods and results described in this work.
5. Schiffmann, Brage, Judge, Paraschiv & Wang, ApJ, 2021 performed large-scale Lande g factor calculations for ions of interest and discusses degeneracies in context of their results.
6. Casini & Judge, ApJ, 1999 and Judge & Casini, ASP proc., 2001 described the theoretical line formation process implemented in CLE, the coronal forward-synthesis code that is currently utilized by CLEDB.

9.2 README-RUNDB

9.2.1 CLEDB Parallel Database Generator

README for running CLE database calculations on multiple CPU threads.

solar-coronal-inversion repository on github

Contact: Alin Paraschiv (arparaschiv@ucar.edu)

History for BUILD module:

- ARP: 20210617 - initial release.
- ARP: 20210827 - Added a Slurm enabled version of the script for batch jobs on RC systems.
- ARP: 20210915 - Rewrote the thread scaling to allocate tasks uniformly across threads; Both interactive and batch scripts now can utilize RC Slurm capabilities. The interactive version can only use Slurm allocated resources inside interactive jobs. The batch dedicated version can utilize scratch directories; It copies final outputs in a user's project directory after finalizing tasks.
- ARP: 20221222 - Updated both scripts to fix an error with calculating the optimal heights that are scaled across available nodes.

SCOPE:

This is a simple bash script implementation that launches separate parallel processes for building Stokes IQUV databases as part of the CLEDB_BUILD module. Two versions are provisioned:

1. *rundb_1line.sh* (For local **interactive** runs; can be utilized inside slurm interactive environments too.)
2. *rundb_1line_slurm.sh* (For **batch** and/or headless runs.)

INSTALL and USAGE:

- make sure the scripts are executable:

```
chmod u+x rundb_1line.sh
chmod u+x rundb_1line_slurm.sh
```

- (Only on OSX) Install gnu-sed (See notes below):

```
brew install gnu-sed
```

- (Optional if needed) OSX might have issues with running executables ("cannot execute binary file"). To fix try:

```
xattr -d com.apple.quarantine /path/to/file
```

- (Optional) for interactive jobs on RC systems, the correct modules may need to be preloaded in order for scripts to execute.

```
module load slurm/blanca
module load gcc/10.2.0                (gcc is preloaded automatically in the␣
↪batch version of the script.)
```

- run interactive jobs with (after starting the interactive node; see README_SLURM):

```
./rundb_1line.sh
```

- run batch/headless jobs with:

```
sbatch rundb_1line_slurm.sh
```

NOTES:

- The interactive *rundb_1line.sh* script requires two manual keyboard user inputs.

  i. select how many CPU threads to use;

```
Hi
You have xx CPU threads available.
How many to use?
```

  ii. which ion/line to compute. Each ion/line will create its own subfolder in the directory structure to store computations.

```
Please indicate the line to generate. Options are:
1:    FE XIII 1074.7nm
2:    FE XIII 1079.8nm
3:    Si X    1430.1nm
4:    Si IX   3934.3nm
```

- The batch *rundb_1line_slurm.sh* script has no keyboard inputs, but has manually defined variables that control the ions to generate and system paths.
- Most directory and file pointers are dynamically linked to the CLEDB distribution directory. Local runs should run without interference. Some directory/system containing variables are defined to be compatible with the CURC system (scratch, project, etc. dirs). These may need to be updated for different systems.
- ** NEWLY COMPLETED RUNS WILL DELETE/OVERWRITE PREVIOUSLY COMPUTED CALCULATIONS AND LOGS IN THE CORRESPONDENT SUBFOLDER**
- The scripts are configured to produce one line database outputs. All atomic data for the four ions of interest along with the configuration files are available in the *config* directory. This setup selects the relevant inputs automatically.
- Outside of the two batch scripts, the only user editable file is the config/DB.INPUT that configures the database number of calculations (parameter resolution).
- Database output, header, and logs will be written in the correspondent ion sub-directory. Intermediary folders and files will be deleted upon completion. The logs are dynamically written and calculation status can be checked anytime with *tail*; e.g.

```
tail BASHJOB_0.LOG
```

- The ./rundb scripts will wait for all thread tasks to finish before exiting. Due to limitation in CPU process ID (PID) tracking, the user is not notified in order of threads finalizing, but in the order they were scheduled. e.g. if thread 2 finishes before thread 0, the user will find out only after thread 0 and thread 1 finish. A bug might manifest if a new unrelated task is scheduled with the same PID as one of the runs, but this should not occur in normal circumstances. If such a case occurs, a tail of the logs will verify that everything went well and scripts can be exited manually.
- The number of Y-heights to calculate between the ymin and ymax ranges are not always a multiple of the number of CPU threads. The scripts will efficiently scale the tasks on the available threads. If you request less tasks (via DB.INPUT) than threads (via keyboard or sbatch), the script will not utilize all pre-allocated resources.
- The script heavily relies on the SED function. SED has different implementations on Linux (GNU) vs mac (BSD) which makes commands not be directly correspondent. A function wrapper *SEDI* that disentangles GNU vs BSD syntax is provided in the scripts. OSX users need to install a gnu implementation of sed (gnu-sed) for the script to be portable between systems (via the gsed command).

```
brew install gnu-sed
```

- The script cuts and appends midline on the DB.INPUT file, to set the ymin and ymax ranges for each CPU thread. The number of decimals for all variables and 3 spaces in between them need to be kept in the configuration file in order to not introduce bugs.
- Executables (dbxxx) need to be build (from CLE) on the current architecture: ELF(linux) or Mach-O(OSX) If non-correct executables are called a "cannot execute binary file" error is produced. Architecture can be checked with the *file* command. The configuration deduces the OS in use and selects and uses the proper dbxxx executable in each case, where both Darwin and LINUX executables exist. The linux executable has a CURC cross compiled executable compiled with gcc/10.2.0 for use in RC systems.

9.3 README-SLURM

9.3.1 CLEDB Research Computing Runs

solar-coronal-inversion repository on github

Contact: arparaschiv "at" ucar.edu; paraschiv.alinrazvan+cledb "at" gmail.com

SLURM ENABLED RESEARCH COMPUTING INTERACTIVE OR HEADLESS RUNS

Detailed instructions for setting up and running the CLEDB inversion distribution on research computing (RC) systems.

1. Slurm enabled test scripts

- test_cledb_slurm.sh
- test_1line.py
- test_2line.py

Note: the *test_1line.py* and *test_2line.py* scripts are plain script versions of the test notebooks. These are directly exported from the Jupyter .ipynb notebooks. All changes to the notebooks should be exported to the scripts.

2. Installation and run instructions for RC systems

These instructions are following the CURC system guidelines and scripts are provisioned to be compatible with the blanca-nso compute nodes.

- Activate the slurm/blanca module with:

```
module load slurm/blanca
```

2.a Interactive runs

- Start an interactive job:

```
sinteractive --partition=blanca-nso --time=01:00:00 --ntasks=2 --nodes=1 --m=12gb
```

- Install **CLEBD** via git clone in the /projects/$USER/ directory following the instructions in README-codedoc.PDF.
- Create or update a .condarc file with the following contents so that anaconda environments and packages install to your /projects/$USER/ directory instead of /home/$USER/ directory due to lack of storage space.

```
pkgs_dirs:
- /projects/$USER/.conda_pkgs
envs_dirs:
- /projects/$USER/software/anaconda/envs
```

- Anaconda install/enable. This step needs to be run at **each** sinteractive login to enable Anaconda.

```
source /curc/sw/anaconda3/latest
```

- Install the **CLEDBenv** anaconda environment using the CLEDBenv.yml file. Detailed instructions in README-codedoc.PDF. Note: Install inside the sinteractive run or a compile node following the CURC guidelines. Don't perform the installation from the login node.
- Activate your new environment

```
conda activate CLEDBenv
```

- Generate a database:

```
module load gcc/10.2.0
./CLEDB_BUILD/rundb_1line.sh
```

- Note: A Fortran executable cross compiled on the CURC system with gcc/10.2.0 is provided and will be automatically used by the script. If libraries are missing, and runs are not executing, please contact us for the CLE source code distribution. The most current CLE distribution is **not yet** publicly hosted, but available upon request.
- Update the database save location in the *ctrlparams.py* class, and then run any of the two .py test scripts.

```
python3 test_1line.py
python3 test_2line.py
```

- Everything should work (remember to download the test data to the main CLEDB root dir) with the exception of remotely connecting to a Jupyter notebook server spawned inside an sinteractive session (which on CURC refuses to connect). CURC offers dedicated Jupyter notebook/lab compute nodes, but beware of how the low resource allocation (usually 1 thread) might interact negatively with the Numba/JIT parallel enabled functions.

2.b Batch/headless runs

- The database generating scripts in CLEDB_BUILD directory have a dedicated headless run script *rundb_1line_slurm.sh* which has slurm headers and where all user inputs are disabled. RC resources are requested via the sbatch commands in the script header. The ion to generate the database along with some path variables need to be manually edited in the script before running. This version of the database generation script will perform disk I/O on $SCRATCH partitions, and not on local directories. Databases will be moved back to the /projects/$USER/ directories after computations are finished.
- Call it using sbatch after editing for the ion and paths to generate for each ion (multiple sbatch commands can be run concurrently if resources are available):

```
sbatch rundb_1line_slurm.sh
```

- The bash *test_cledb_slurm.sh* wrapper script is a starting point for running test/production headless runs via the sbatch command. It provisionally calls one of the two above mentioned .py scripts based on a decision tree.
- The script is to be *updated/finalized* when production runs are ready and data and header ingestion procedures are known.

CHAPTER

TEN

LIST OF CHANGES AND TODO TASKS

Warning: The changelog is dinamically linked to the standalone markdown files. A documentation rebuild will capture changes in any of these files.

10.1 CLEDB coronal inversion UPDATES and TODO list

Github update history

Commit Tag	Date	Description
initial	20210802	Self explanatory.
update-slurm	20210828	- Implemented initial slurm-enabled and batch versions for both jupyter and database build scripts. - Updated and extended documentation.
update-numbaproc	20210906	- Implemented a parallel enable/disable keyword for numba enhanced functions. - Functions that don't benefit from parallel splitting are hard-coded to parallel=False.
update-tidbits	20210908	- Small updates for optional plotting; save/load datacubes, etc. - Implemented slurm tidbits; standard scripts can now be run inside interactive sessions on RC systems. - Revised how calculations are computed in the CLEDB_BLOS module. - Updated constants and ctrlparams classes.
update-CLEDB_BUILD	20210915	- Updated database building scripts. - Fully implemented the batch run version with $scratch partition use. - Updated documentation accordingly.
update-databugfixes	20220818	- Bug fixes update after running comp inversions. - Fixed bugs in *obs_qurotate* (PREPINV module) and *cledb_quderotate* (PROC module). - Fixed the *cledb_obsderotate* (PROC module) function. - Added angle correction (azimuth derotate) to bphi calculation in *cledb_phys* (PROC module). - *cledb_getsubset* routine (PROC module) is optimized and re-written. - Introduced the "integrated" control parameter. - Updated the conda distribuition and python 3.10, numpy, numba, etc. - Bugfix update to CLE 2.0.3 executables in BUILD module.
update-iqud	20230115	- Major update. - CURRENTLY INCOMPLETE. If any problems arice. DOWNLOAD previous commit tag. - CLE is updated to 2.0.4. A number of issues with the forward-synthesis are resolved as outlined in the CLE README. - CLE incurred a small fix to height and depth calculations. Databases need to be recomputed with CLE >=2.0.4 Small bugfix updates to the task scheduling scripts in BUILD. - The database compression routine *sdb_dcompress* introduced numerical instability at particular field geometries. It is now disabled. - Databases are now uncompressed and of np.float32 type. Consequentially, double precision throughout CLEDB is unnecesary and removed. - The PREPINV *obs_integrate* routine is rewritten to improve execution time. Noise statistics and observation rms are enabled. - Partially implemented the IQUD (IQU+Doppler) functionality to invert vector fields in the absence of Stokes V. - Introduced the additional *ctrlparams* "iqud" keyword that works alongside the "integrated" control parameter. - (not ready)Modified functions in the PROC module that perform matches for IQUD setups. New *cledb_matchiqud* function is added. - Keyword ingestion partially implemented. A new *obs_headstructproc* function is added to PREPINV. - PREPINV module can now recognize CoMP/uCoMP keywords at input. - (not ready) A new CoMP test example notebook and script *test_2line_iuqd* is available. - Numba activate/deactivate and caching enabled/disabled keywords are implemented in the *ctrlparams* class. - Updated the numba, numpy, and scipy package versions under the currently stable Python 3.10 conda env. - (not ready) Updated the documentation. The IQUD setup for line-integrated observations is described. - (not ready) Numba optimization/improvement of several functions (*obs_integrate*, *cdf_statistics*, *cledb_matchiquv*, *cledb_partsort*).
update-readthedocs	20230126	- Transfered the code documentation from a latex based distribution to a sphinx webpage format build in the current code distribution and hosted on readthedocs.io

TODO list last update: 20230106

1. Add the ISSUEMASK setup as outlined in the documentation.
2. ~~Needed observation Keywords are currently manually implemented, where a synthetic CLE observation is used. Information from observation keywords will need to be ingested after observations will become available.~~*update-iqud*: Keywords are now ingested with data.
3. Develop and use a public test case for more convoluted MURAM data.
4. ~~Implement additional numba compiler flags and options. Make numba active/disabled with a ctrlparam. Implement numba caching.~~*update-iqud*

5. ~~After more information on input data is obtained, implement the LEV2CALIB_WAVE and LEV2CALIB_ABSINT functions as outlined in the documentation.~~*update-iqud*: DKIST lev-1 data will contain the needed corrections.
6. implement ML_LOSDISENTANGLE Disentangling multiple LOS contributions in observations, as outlined in the documentation.
7. ~~Implement a noise uncertainty quantification method (RMS computations and relevance against signal).~~*update-iqud*

CHAPTER

ELEVEN

GLOSSARY

ADU
Arbitrary Data Units; detector calibrated counts when no absolute intensity calibration exists.

Analytical solutions
Frame an inverse problem in a well-understood and reasonably posed mathematical form and approximates a solution.

CDF
Cumulative Distribution Function. Statistical method for interpreting normal distributions.

CLE
Coronal Line Emission FORTRAN spectral synthesis code. It is hosted on Github.

CLEDB
Coronal Line Emission DataBase Inversion PYTHON algorithm that matches spectropolarimetric observations with CLE generated databases.

CHIANTI
atomic database for spectroscopic diagnostics of astrophysical plasmas. See the documentation.

χ^2 fitting solution
Statistical hypothesis to determine whether a variable is likely to come from a specified distribution. The χ^2 residual is used to find the closest match to a discrete distribution point.

Degeneracy
When performing an inversion, the degrees of freedom of the problem might not allow to recover an exact mathematical solution. Sets of equivalent solutions inside an inversion metric are called degenerate. e.g., disentangling an angle value knowing that sin a = $\frac{1}{2}$, a is degenerate to either $\frac{\pi}{6}$ or $\frac{5\pi}{6}$.

FWHM
Full Width at Half Maximum. Measurement of a standard width of a normal distribution.

Glob
This is a Python library to process and manipulate os pathnames.

Header
Sets of input metadata that accompanies an observation datafile.

Inversion
Mathematical process that starts from the output of a physical process and backtraces to recover one or more input variables. In our particular case, we start from output Stokes IQUV profiles and attempt at recovering coronal magnetic fields responsible for producing said profiles.

JIT
Just In Time compilation decorator from the Numba library package.

LOS
Line Of Sight.

Normal distribution
A Gaussian function, or a bell curve. Probability distribution that is symmetric around a mean value, in which data near the mean are more frequent in occurrence than data far from the mean.

Numba
An open source JIT compiler that translates a subset of Python and NumPy code into fast machine code. Serial task parallelization and loop-lifting is also available. See documentation.

Numpy
Open source library for fast numeric operations.

Physical parameters
A set of observable parameters like density, magnetic field strength, magnetic geometry components, temperature, 3D coordinate position, etc.

Physical units
Definition of measurement that is calibrated to physically etalonated constants; e.g. intensity in [erg cm^{-2} s^{-1} nm^{-1} sr^{-1}]

Pixel
A 2D representation for a signal integrating area. This is equivalent to a LOS integration of a voxel. This is also the fundamental storage datatype for Python/Numpy arrays. In this document we refer to pixels when discussing data/array elements.

Radiative transfer
Transfer of electromagnetic radiation through a medium.

RMS
Root Mean Square. The square root of the arithmetic mean of the squares in a set of discrete realizations.

Slurm
A computation worload manager used predominantly by research computing clusters.

Spectroscopic data
Electromagnetic radiation flux spread in individual bins inside an electromagnetic spectral range.

Spectroscopic emission line
Excess flux exceeding background counts at determined spectral positions, occurring when the electrons of an excited atom or molecule move between energy levels.

Stokes IQUV
A set of values or spectra that describe the polarization state of electromagnetic radiation.

Stokes I
Total intensity of spectroscopic line emission.

Stokes Q and U
Linear polarization components of spectroscopic line emission.

Stokes V
Circular polarization component of spectroscopic line emission.

Voxel
A generalized concept of a pixel. In our case, by voxel we envision 2D projection of a volume inside a square area that contains information about the integral emission along the line of sight. Voxel is used in this document instead of *pixel* when refering to the physical counts recorded inside a spatial integration area of the size of a pixel.

INDEX