The GRASP Package. An overview.

Generalized Retrieval of Atmosphere and Surface Properties

Oleg Dubovik, Laboratoire d'Optique Atmosphérique <oleg.dubovik@univ-lille.fr>
Fabrice Ducos, Laboratoire d'Optique Atmosphérique <fabrice.ducos@univ-lille.fr>
David Fuertes, GRASP SAS <david.fuertes@grasp-sas.com>

The GRASP Package. An overview. : Generalized Retrieval of Atmosphere and Surface Properties

by Oleg Dubovik, Fabrice Ducos, and David Fuertes

A browsable, and possibly more up-to-date, version of this document can be obtained here:

`http://www.grasp-open.com/doc`

This document describes an overview of the GRASP project, its goals and its architecture. If you want to contribute to the development, you may be more interested in the technical documentation [http://www.grasp-open.com/tech-doc].

The source code of this documentation is in the same repository as GRASP open algorithm. If you want to contribute with your corrections, please check how to do it in User documentation chapter [http://www.grasp-open.com/tech-doc/chap02.php#chap020201].

Publication date 01 February 2024

Table of Contents

List of Figures

List of Tables

List of Equations

Foreword

1. Caveat

This document is under development. To write a documentation is a hard task and usually the code evolves quicker than the documentation. We are doing a big effort to document all details of the software. While the project will mature, new chapters will be draft and old ones will be revised in order to stay as close as possible in sync with the actual realization. Please, keep tuned and check regularly the news in this document. Also, remember that the source code of the documentation is in same repository that GRASP software allowing everybody to contribute to this document. If you are interesting in contribute making corrections or adding new text we will be very glad of receive your suggestions. In that case, please, have a look to the technical documentation to know how to contribute (www.grasp-open.com/tech-doc [http://www.grasp-open.com/tech-doc/]).

2. What you will find in this document

- A general vision of scientific algorithm
- A general vision of software arquitecure
- How to compile the code including description of prerequisites
- How to run the code including a general vision about how it works
- References to other documents, especially to the detailed documentation of the major components of the system when it is available or scientific papers.

3. What you won't find in this document

- A detailed description of the scientific algorithm, one refers to Dubovik et al. 2011 [http://www.atmos-meas-tech.net/4/975/2011/amt-4-975-2011.html] for more information.
- A detailed description of each module, routine or data structure being part of the production system. For developers, technical documentation is provided as another documentation. Please, have a look to: www.grasp-open.com/tech-doc [http://www.grasp-open.com/tech-doc/]
- A documentation of all extensions of the software. Each extension has to have its own documentation. Here it is described only main GRASP package and only if an extension provide a very important feature which is widely used it can be described here exceptionally.

4. Versioning

GRASP software born many years ago. In 2013, the developement team started the tag releases with following format vXX.YY.ZZ where XX is the number of major version, YY is the number of minor version and ZZ the revision. This nomenclature system is valid from first stable version (v1.0.0) that at the moment of write this lines it does not exist yet. Beta versions (v0.YY.ZZ) will follow more flexible rules of naming and it each name will be decided by the developer team taking into account if a version introduce new important features, break the compatibility with previous versions, etc. For most curious people, first GRASP tag was v0.2.0 and it was not public yet. The code started to be publicly available from the version v0.7.0 and regularly, developer team releases the latest developments with really innovative features.

Chapter 1. Introduction

This document provides an introductive description of GRASP software package for general users. At the same time, it is expected that the users interested in the package are familiar with the field of atmospheric remote sensing and have, at least, basic of understanding of the problematic. The document does not include the details description of the scientific algorithm. The relevant scientific content can be found in publication following the citations provided in the text below. This document does not include the elaborated technical details either (for developers). That information can be found on GRASP technical documentation, provided on line (www.grasp-open.com/tech-doc [http://www.grasp-open.com/tech-doc/]). This description is aimed to explain how to handle the software package and orient the qualified user regarding actions needed for adapting the software to a specific application and for contributing into the evolution and the improvement of the GRASP software and the overall concept. Additionally, for extra services such as code adaptations, specific developments or code optimization for specfic purposes, GRASP SAS company can be contacted (www.grasp-sas.com/tech-doc [http://www.grasp-sas.com]).

1.1. Scientific background and heritage

GRASP (Generalized Retrieval of Atmosphere and Surface Properties), introduced by Dubovik et al. (2014), is the first unified algorithm and a software package developed for retrieving atmospheric properties from wide variety of remote sensing observations including satellite, ground-based and airborne passive and active measurements of atmospheric radiation and their combinations.

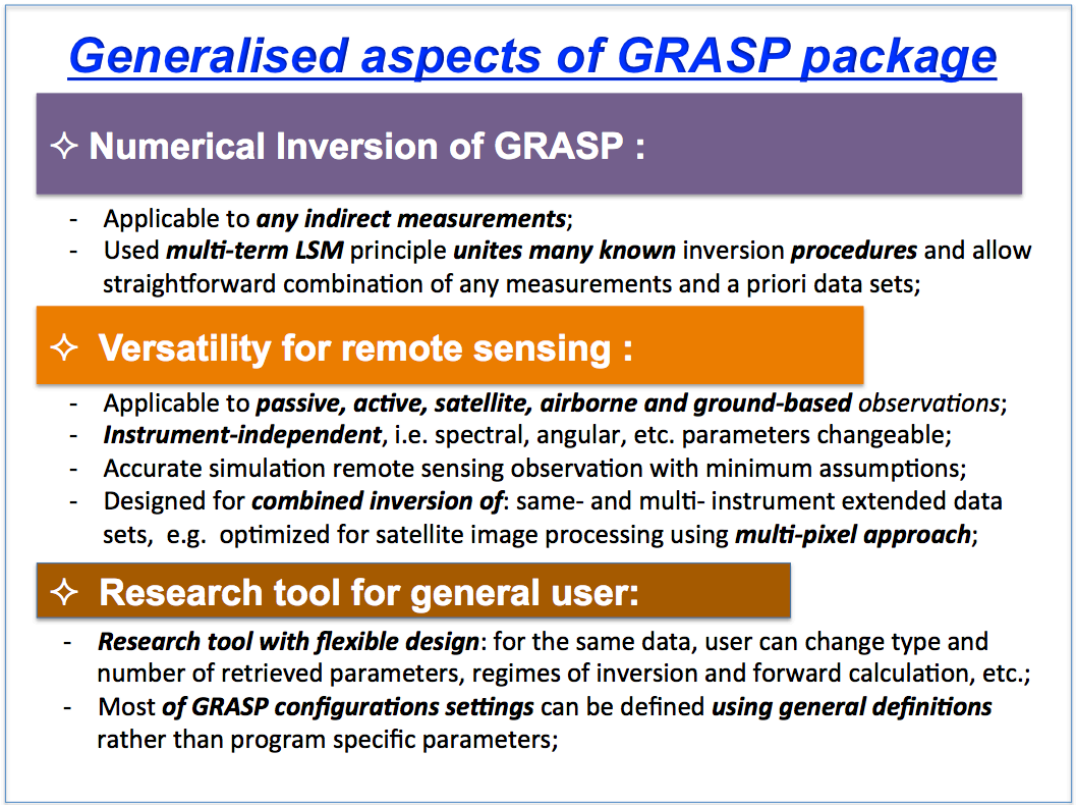
GRASP relies on the heritage of retrieval advances [Dubovik and King, 2000, Dubovik et al. 2000, 2002a,b. 2009] implemented for AERONET (see Holben et al., 1998) a worldwide network of over 300 radiometer sites that generate the data used to validate nearly all satellite observations of atmospheric aerosols. The AERONET retrievals derive detailed aerosol properties (Dubovik et al. 2002a) including absorption, providing information of vital importance for reducing uncertainty in assessments of climate change. The concept of GRASP was proposed in the recent efforts by Dubovik et al. (2011) to develop the algorithm for improved aerosol retrieval from the French Space Agency's PARASOL imager (see Tanre et al. 2011) over bright surfaces like deserts where high surface reflectance dwarfs the signal from aerosols. In these efforts several principles used in AERONET retrieval concept were used and applied for PARASOL satellite retrieval. Then, in return, several new features of newly developed PARASOL retrieval by Dubovik et al. (2011) appeared to be useful for improving ground-based measurements interpretation by combining observations of radiometer and lidar [Lopatin et al., 2013]. Finally, the structure of the algorithm was adapted to convenient and efficient application with diverse remote sensing observations and their combinations.

1.2. Generalized aspects of GRASP algorithm and package

The important feature of the GRASP is that both the core scientific algorithms and the whole package are based on several generalization principles with the idea of developing a scientifically rigorous, versatile, practically efficient, transparent, and accessible algorithm.

There are several "layers" of complimentary generalizations used in the GRASP designs, that are outlined in the diagram shown in Figure 1.1, "Structure of the GRASP software package"

Figure 1.1. Structure of the GRASP software package



1.2.1. Generalized approach of numerical inversion

The corner stone of the GRASP is the used approach for implementing numerical inversion that is highly elaborated and is highly abstract in the sense that it is not linked in any way to specific measurement type. Here are the key elements of the inversion approach employed at the GRASP package:

- The main part of the inversion concept is formulated for inverting abstract "indirect measurements" independent of the physical nature;

- The used mathematical inversion formalism (Dubovik 2004, Dubovik et al. 2011, etc.) complementarily unites advantages of a variety of practical inversion approaches of known mathematical inversion procedures;

- The specific numerical procedure is defined as Multi-Term LSM (Least Square Method) statistically optimized fitting of "positively redundant [1] set of observations". The Multi-Term LSM follows general Least Square concept, however it explicitly considers all inverted data (both the actual observations and used a priori constraints) as "observations" known with different levels of accuracies. As a result, the concept is highly practical:

  a. it benefits from numerous known fundamental developments relevant to LSM (i.e. no need to invent new fundamental principles of optimized inversion);

  b. the methodology is highly suitable for inverting combined data (both observations and a priori constraints).

[1] i.e. formally redundant in the sense that the number of inverted observations exceeds the number of retrieved parameters

1.2.2. Practical generalization of the algorithm for atmospheric remote sensing

Though the numerical inversion is highly abstract of the measurement type, and can be applied to any indirect measurements, the GRASP package was developed for application in the field of atmospheric remote sensing with pursuing the following generalization ideas:

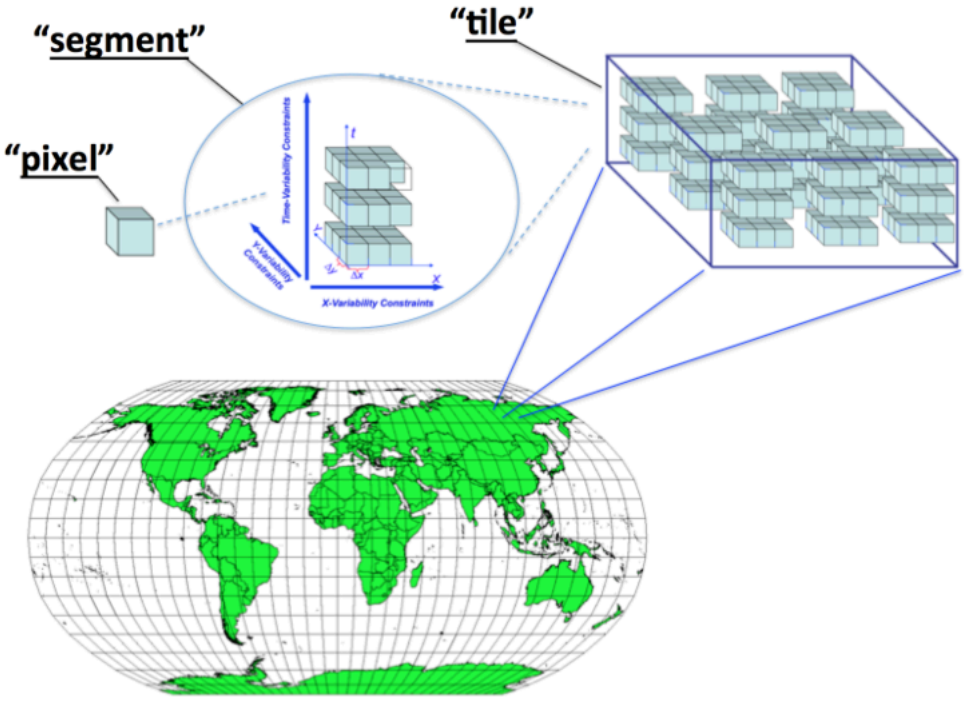
- Making GRASP instrument-independent algorithm, as a result GRASP can be applied to ground-based, satellite and airborne, passive and active measurements and the spectral, angular, polarization, etc. specifications can be changed flexibility within applicability of GRASP "forward model";

- Implementing "forward modeling" simulation of the measurements using accurate approach with minimum dependence of the algorithm on the a priori assumptions (atmospheric radiation is calculated on-line without using look-up-tables);

- Applicable to the combined data from the same instrument: i.e., the data can be obtained both in the exactly same location at different observation times or/and at different (e.g. neighboring) locations at the same or different time moments;

- Applicable to multi-instrument retrievals, i.e. both single observations and extended data sets of observations by different instruments can be processed simultaneously (in highly synergetic way). Since the remote sensing observations (especially from space) are often composing images, Dubovik et al. (2011) proposed improving retrieval using **multi-pixel** principle when a group of pixels is inverted simultaneously under additional inter-pixel constraints. This principle was realized using rigorous approach of inversion optimization (see more explanations below and related articles) and may significantly improve the retrieval results. However, applying multi-pixel retrieval requires specific elaborated data preparation. Therefore, the additional "service" pieces of software were developed as part of GRASP package that significantly simplified practical application of the GRASP multi-pixel approach to real observations. Thus, GRASP realizes processing of global (or regional) time series of instrument observations as illustrated in Figure 1.2, “Structure of the GRASP software package”. The entire image is divided by the user into geo-located grid composed by rather large data sets that are called "tiles", each tile is composed by "segments" groups of adjacent segments of the observation "pixels". Correspondingly, GRASP can be set to process: (i) only one pixel, (ii) only one segment, (iii) tiles, i.e group of segments.

Figure 1.2. Structure of the GRASP software package



1.2.3. Adaptation of GRASP for general user

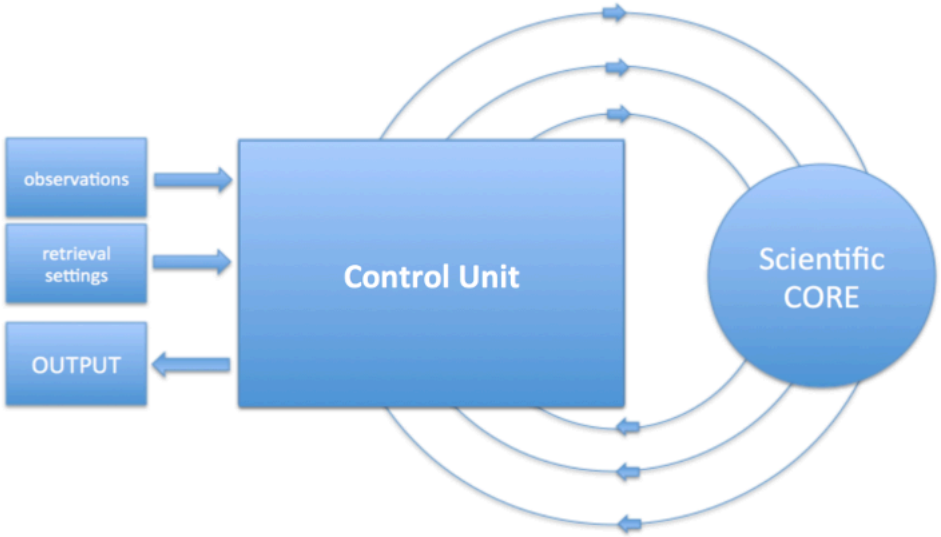
In addition to elaborated theoretical concepts (numerical inversion approach, modeling of atmospheric radiation and image processing approach) GRASP was designed with the idea of making convenient "research tool" that can be used by a user possessing general knowledge in remote sensing retrieval but not familiar with the details of source code routines. Therefore, the following user-oriented principles were realized in GRASP:

- Possibility for user to construct custom retrieval using GRASP, i.e. user set up the inversion of the same observations in many different ways by choosing:
  - different sets (in terms of number and type) of the retrieved and a priori known (and/or even fixed) parameters;
  - different approaches in making forward simulations;
  - different assumption for noise distribution in the inverted data;
  - different sets of a priori constraints for retrieved parameters;
  - different standard procedures used in numerical inversion (i.e. for solving linear systems);
  - different approaches for data processing: inverting each pixel independently, inverting large images (see Fig.2), combining available independent co-incident and/or co-located observations;
  - etc.
- User independence of specific program realization, because the management of GRASP inputs is done using command lines, not symbols, that represent the "names" of parameters or procedures and can be understood by a user that has general knowledge in the atmospheric retrieval.

1.3. Concept of GRASP software package

The key component of the software package is the **scientific GRASP core**. This is a code that implements actual inversion of remote sensing observations following the retrieval procedure assumed by scientific algorithm. Initially, **the scientific GRASP core** was used directly for processing observations by reading the input data from files and providing retrieval output on the screen or/and in the file. However, in order to achieve the highly optimized processing of large volumes of data, such as satellite observations, the scientific GRASP core has been complemented by the development of **the control unit** – software package that manages the preparations of observations, implementation of actual retrievals by scientific code and the output of results. The utilization of the control unit allows an implementation of the retrieval without generating intermediate input/output files and a number of other optimizations of applying scientific core **.** In other words the communication of user or processing routing with the scientific core passes via the control unit as illustrated in Figure 1.3, “Structure of the GRASP software package”.

Figure 1.3. Structure of the GRASP software package



Such set up of the scientific package was designed for simplifying the processing of large satellite images by GRASP. Thus, while original GRASP scientific core could provide the retrieval for only one pixel or segment (see Figure 1.2, “Structure of the GRASP software package”), using the control unit manages application of retrieval to the groups of segments (a tile). This approach provides a number of conveniences in employing GRASP for processing the actual observations. Indeed, now the GRASP can read directly a raw data archive and perform data preparation on the fly and without generating intermediate files. The code can manage large volumes of data: the control unit organizes input data, implements multiple calls of the scientific core, obtains the output for all archives, may manage the display of the results, etc.

In addition, the development of the control unit resulted in many convenient features of managing scientific core, not only at the level of the tiles processing of observation data, but also in implementing the retrievals of the segments and even single pixels. For example, the input texts files were replaced by settings files in YAML format that significantly improves the user-interface:

- all inputs can be provided in a standard format instead of a specific one developed only for GRASP;
- the settings are driven by the text commands that can be organized in any order (only the content is important);
- information redundancy was decreased because structure and size of arrays are automatically adapted to the input defined by the user.

- All settings are auto-documented: every parameter has a description directly in the code that can be viewed calling help command.

It should be noted that the current version of the GRASP software conserves the possibility of running scientific core without using the control unit. While this option may be of some interest for the GRASP developers, the utilization of the entire package (including control unit) is the recommended approach for general GRASP users.

Chapter 2. GRASP software package

2.1. GRASP architecture

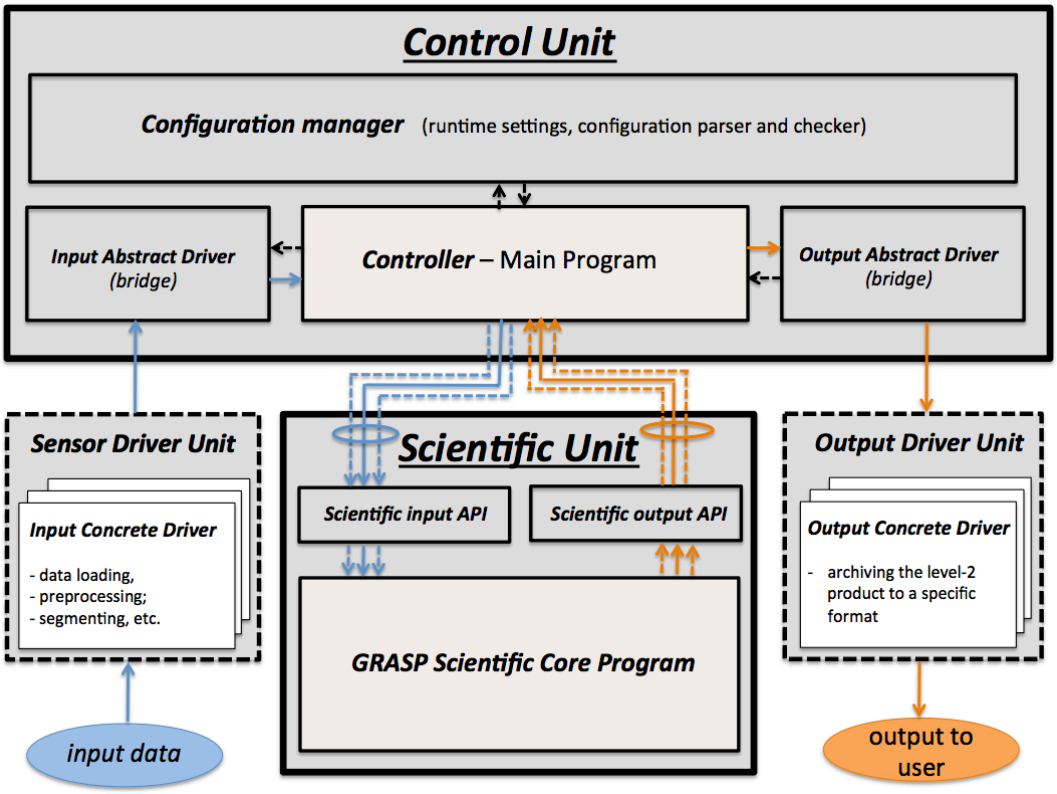
Figure 2.1, “The architecture of the GRASP software package” illustrates the architecture of the GRASP software package organization. This architecture deploys decoupled independent modules, such as configuration (settings) module, scientific core and the controller module, which communicates between them. The dashed boxes show the application-specific modules (input / output drivers) that can be optionally added to the GRASP software package.

This design by modules of the package aims at minimizing dependencies between developed subsystems and enabling its extensibility. As a result, the two most valuable aspects realized in the GRASP architecture are:

i. the common interfaces were defined for the replaceable elements;

ii. evolution of the scientific core without modifications of the whole package.

Figure 2.1. The architecture of the GRASP software package



2.2. GRASP input and retrieved data

2.2.1. Measurements and retrieved parameters

The list of measurements and the retrieved parameters can present a variety of possibilities. This list can change strongly depending on the selected application and the inversion strategy. An example of the measurements and the retrieved parameters configuration for the PARASOL space observation application can be found in Dubovik et al. (2011). Therefore, this document refers to the inverted observations and retrieved properties using the terms: "input measurements/observations", "retrieved parameters/characteristics". The details of the application of the GRASP algorithm to specific

measurements are expected to be clarified to users from the comments included in the input and the text of the source files. Examples and relevant scientific discussion can be found in referred articles. Additionally, the software package is distributed together with some examples. Once the code is compiled and ready to be executed, the users are encouraged to consult and run the examples.

2.2.2. GRASP inputs

The inputs of GRASP contain the following information:

- **measurements;**
- **definition of unknowns** (and the employed forward model);
- **retrieval setting and a priori constraints.**

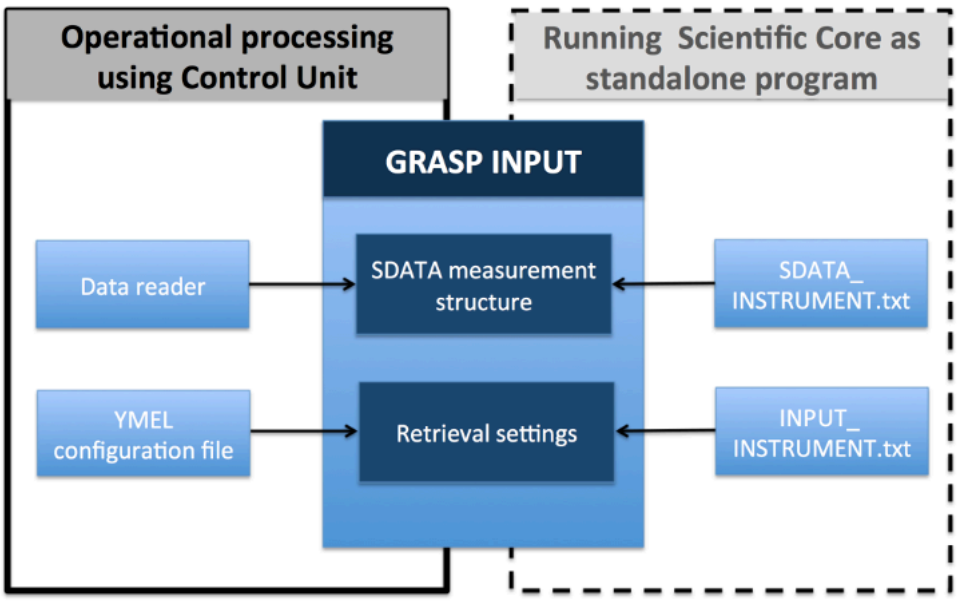
It should be noted that the inputs include not only the actual observations needed to be inverted, but also an ancillary information that drives many aspects of the retrieval. For instance, the employed "exact forward model" and the assumptions, the variety of a priori constraints, the mathematical and logical procedures, etc. Though, such information is generally expected in the input for any retrieval, the GRASP stands out from most of existing retrieval methods/codes by the flexibility of the retrieval and the versatility of its applicability.

The GRASP input is separated into two groups:

- **"measurements"** – includes the actual values of measurements and some information of their configurations;
- **"retrieval settings"** – includes all information about retrieval implementation (description of the retrieved characteristics, all settings for forward simulations and numerical inversions, etc.)

As shown in Figure 2.2, “Illustration of managing input data for GRASP software package.”, when the GRASP package is employed for operational processing, the observation are provided by the control unit from the *data reader* and the user defines the retrieval using **YAML configuration file** . If the scientific core is running as a standalone code, the next input text files are used: **SDATA_INSTRUMENT.dat** – the file containing the observation data **INPUT_ INSTRUMENT.txt** – the file containing the retrieval configuration information. The **SDATA_INSTRUMENT.dat** input file can also be used with the control unit. This is a useful option while applying GRASP to a new type of data aiming for the functionality and sensitivity tests.

Figure 2.2. Illustration of managing input data for GRASP software package.



2.2.3. GRASP input data structures

The control unit provides the measurements to the retrieval via "SDATA measurement structure", prepared using a specific data reader. The description of the structure is provided in Section 4.2.1, “The SDATA format”. The configuration information is provided by the control unit from the YAML configuration files, described in Section 4.1.1, “Settings file”. The list of parameters and their explanations are provided in Section 4.1.1.1, “HELP argument”. This information can be directly assessed by typing "help" command.

2.2.4. Input text files for running the Scientific Core alone

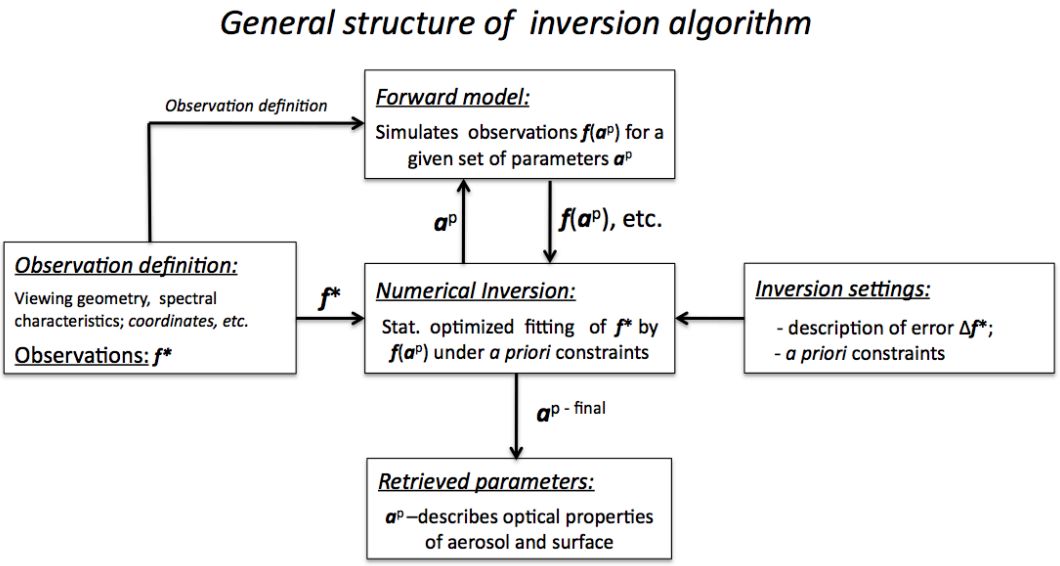
Retrieval library keeps its old capability of running as a stand alone application. In this case, the measurements description has to be provided in sdata format (see Section 4.2.1, “The SDATA format”). In fact, this format also works when the entire system is executed because it was ported from the scientific library to the entire system. This format is an easier one for the scientific community. When the entire system is executed, a settings file in YAML format defines the inversion strategy. In the case of using the scientific core as a stand alone application, a settings file in ascii format has to be provided. Each parameter in this file has a specific fixed location that should be respected. The curent guide, however, does not describe this file structure. Please note that the scientific module can be isolated and tested independently, but it is only for development purposes. If the reader is interested in knowing more details about that, the technical documentation can be reviewed (www.grasp-open.com/tech-doc [http://www.grasp-open.com/tech-doc/]). Note also that, for a general user, it is not recommended to run separately the scientific library.

2.3. GRASP Scientific Core algorithm

2.3.1. Overall structure

The structure of the scientific GRASP code is shown in Figure 2.3, “General structure of the GRASP scientific algorithm (Fig.3 in Dubovik et al. 2011).”. The code and retrieval are organized as an interaction of the two main functionally different modules: "Numerical Inversion" and "Forward model". The "Numerical Inversion" is the module that drives the whole retrieval, therefore it can be considered as hierarchically the main part of the core algorithm program that determines the retrieval data flow. The "Forward model" implements simulations of the inverted observations. The overall GRASP development concept emphasizes the **generalized** structure of the algorithm and the retrieval. This assumes that the algorithm should be versatile, i.e. applicable to variety of remote sensing observations, and enable some flexibility in choosing retrieval approaches. For instance, choosing of different: assumptions of overall retrieval; mathematical procedures; physical models for simulating observations; presentations of obtained results, etc. Therefore, both "Numerical Inversion" and "Forward model" modules are adapted for implementing varieties of different procedures. At the same time, the data flow interaction between these modules and a high tolerance of overall code to the modifications inside of each module are implemented. The information transmitted from the input "Observation definition" and the "Inversion settings" modules determines the actual regime of the retrieval execution.

Figure 2.3. General structure of the GRASP scientific algorithm (Fig.3 in Dubovik et al. 2011).



The data flow exchange between the "Numerical Inversion" and the "Forward model" modules, as illustrated in Fig. 6, includes the information about the following values:

f * vector of inverted measurements,

f (a ^{p} **)** vector of measurement fit at p-th iterations,

a ^{p} vector of unknowns at p-th iteration (retrieved parameters).

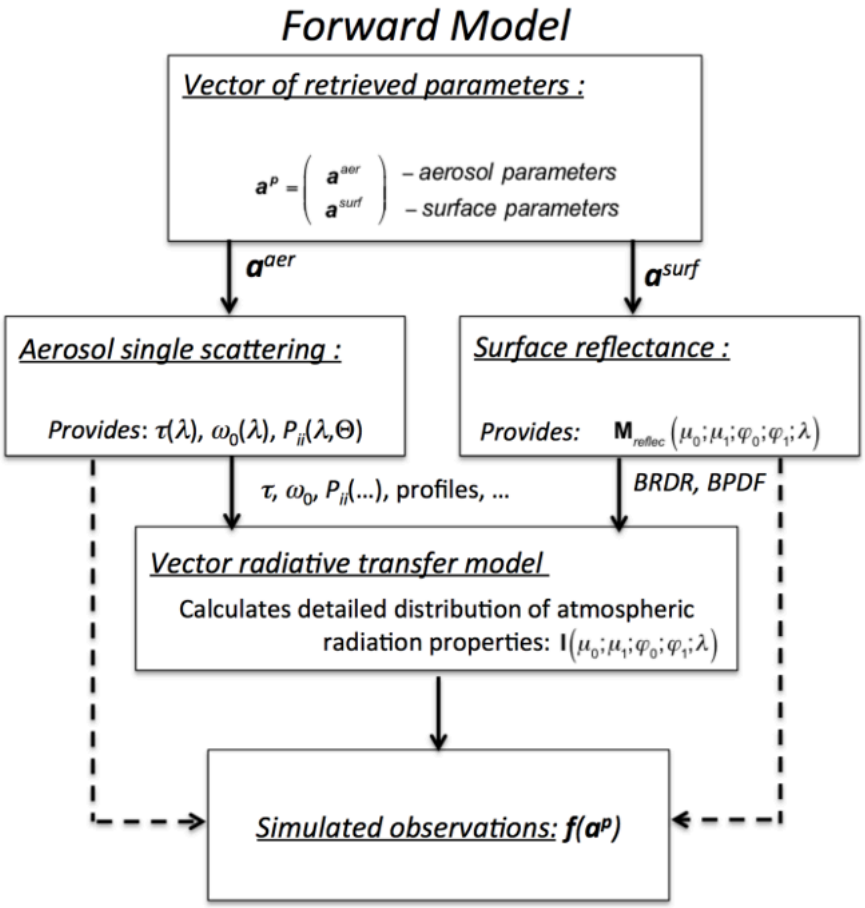
The content of these vectors was denoted in the describing input section.

The scientific GRASP code is written in Fortran 90. In the technical documentation [http://www.grasp-open.com/tech-doc/] there are descriptions of the structure of data flows, the source file structure and the locations.

2.3.2. Forward model

The "forward model" module implements simulations of the inverted remote sensing observations. The GRASP "forward model" is rather a universal one, i.e. can simulate large variety of remote sensing observations (passive and active observations obtained from ground and space). Also it consists from several distinct blocks (Figure 2.1, “The architecture of the GRASP software package”): aerosol single scattering; surface reflectance; and radiative transfer calculations. These blocks are semi-independent in the sense that each block can be changed or entirely replaced with no effect or minimal effect on other parts of the "forward model" routine. For example, GRASP "forward model" allows to choose physical approaches/models for simulating surface reflectance.

Figure 2.4. General organization of Forward modeling in the algorithm



Depending on the inverted data, only a part of the "forward model" can be used. The dashed lines in Figure 2.4, “General organization of Forward modeling in the algorithm” indicate that only single scattering or surface reflectance calculations can be used by the code, if accounting for multiple scattering is not needed, as in the cases when measurements of spectral AOD, phase matrices or lidar data are inverted. Moreover, the design assumes a possibility for users to add to the GRASP "forward model" other routines implementing similar simulations. For example, the subroutine implementing radiative transfer calculations can be replaced by a subroutine implementing another method to account for multiple scattering. In the future, several new modules are planed to be included in the "forward model", such as the module for accurate modeling of the gaseous absorption, the module for radiative transfer calculation for thermal infra red spectral range, etc.

In the GRASP code, the "Forward model" is driven by a single subroutine "forward_model_pixel_PHMX" located in the file "forw_model.f90" (see technical documentation [http://www.grasp-open.com/tech-doc/]). **Aerosol single scattering** properties are simulated assuming aerosol as mixture of randomly oriented spheroids using of DLS spheroid package (Dubovik et al. 2006). This package can be provided as an independent program with some descriptive documentation. **Surface reflectance BRDF and BPDF** can be calculated using a variety of subroutines representing different models (see scientific description in Dubovik et al. 2011 and directly in technical description included in the GRASP code settings file). **Radiative transfer calculation** accounting for multiple scattering effects in GRASP is implemented by on-line radiative transfer calculations using Successive Order of Scattering method using the program developed by M. Herman (the method is documented in the paper by Lenoble et al. 2007). The modules for **aerosol single scattering** and **BRDF, BPDF** are easily extractable from the program and can be easily used with other radiative transfer codes if needed. In addition, some **input** parameters in the configuration file define the regimes of **the radiative transfer calculation** implementations. Specifically, a number of trade-offs between accuracy and speed can be used including the possibilities of changing the number of terms M used in the expansion of the phase matrix into Legendre polynomials, the number of

terms N used in Gaussian quadrature for zenithal integration, number of numerical layers in vertical atmosphere properties integrations, etc.

2.3.3. Numerical Inversion

The "numerical inversion" is a main and most complex part of the code from the functional and the logistical point of view that governs the flow of the data. The description of the algorithm and the details of the approach are given in the scientific papers listed in Section 1. Here we provide only a short description sufficient for understanding the structure and the organization of **the GRASP Scientific Core**.

The program includes two main "layers" (parts): **Single-pixel inversion** and **Multi-pixel inversion**.

2.3.3.1. Single-pixel inversion

The structure of the single-pixel inversion is illustrated in Figure 2.5, “Organization of GRASP Numerical Inversion: Single-Pixel Scenario”. It includes the following main operations:

i. *Modeling observations* $\mathbf{f}(\mathbf{a}^p)$ *for state vector p-th approximation (for p=0, initial guess is used);*

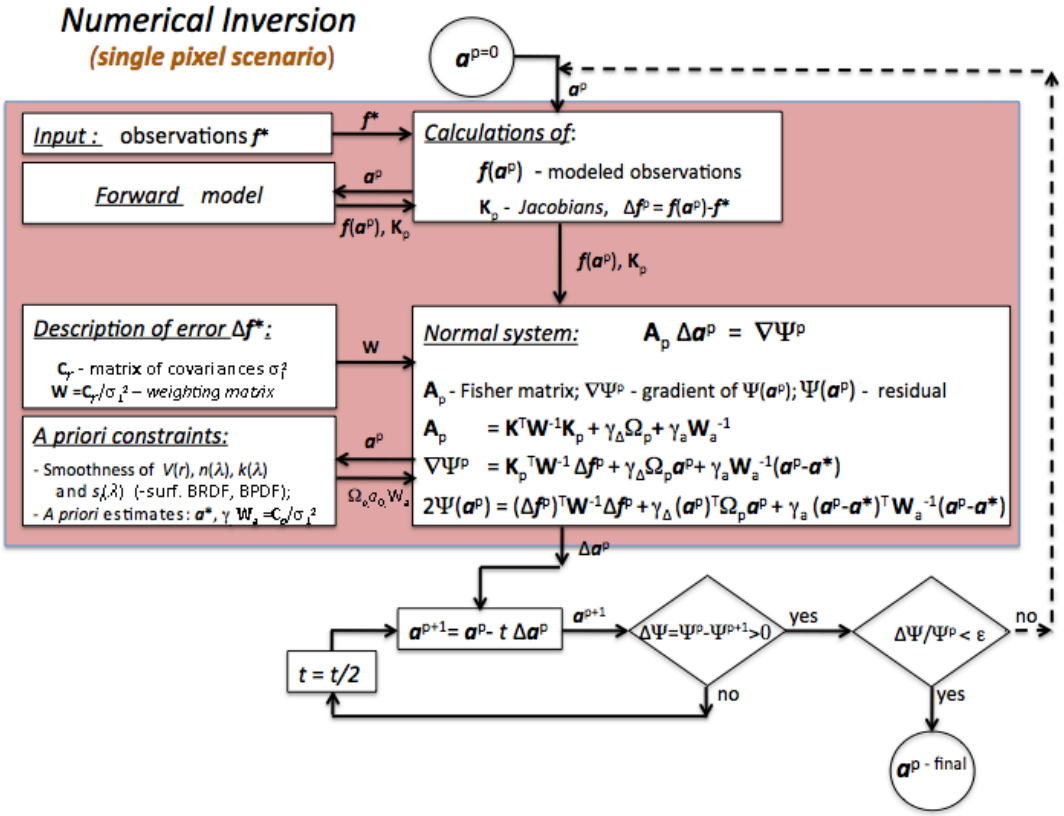
ii. *Calculation of matrices of first derivatives* $\mathbf{K}_p$ *Jacobians;*

iii. Forming p-th Normal System: $\mathbf{A}_p \Delta \mathbf{a}^p = \nabla \Psi^p$, where $\mathbf{A}_p$ Fisher matrix; $\Psi(\mathbf{a}^p)$ residual; $\nabla \Psi^p$ gradient of $\Psi(\mathbf{a}^p)$.

iv. Solving Normal System to determine $\Delta \mathbf{a}^p$, and correcting the solution approximation $\mathbf{a}^{p+1} = \mathbf{a}^p - t\,\Delta \mathbf{a}^p$ so that: $\Psi^p - \Psi^{p+1} > 0$;

v. Repeating steps **i - iv** until $\Delta\Psi = \Psi^p - \Psi^{p+1}$ changes significantly i.e. until $\Delta\Psi / \Psi^p < \varepsilon$

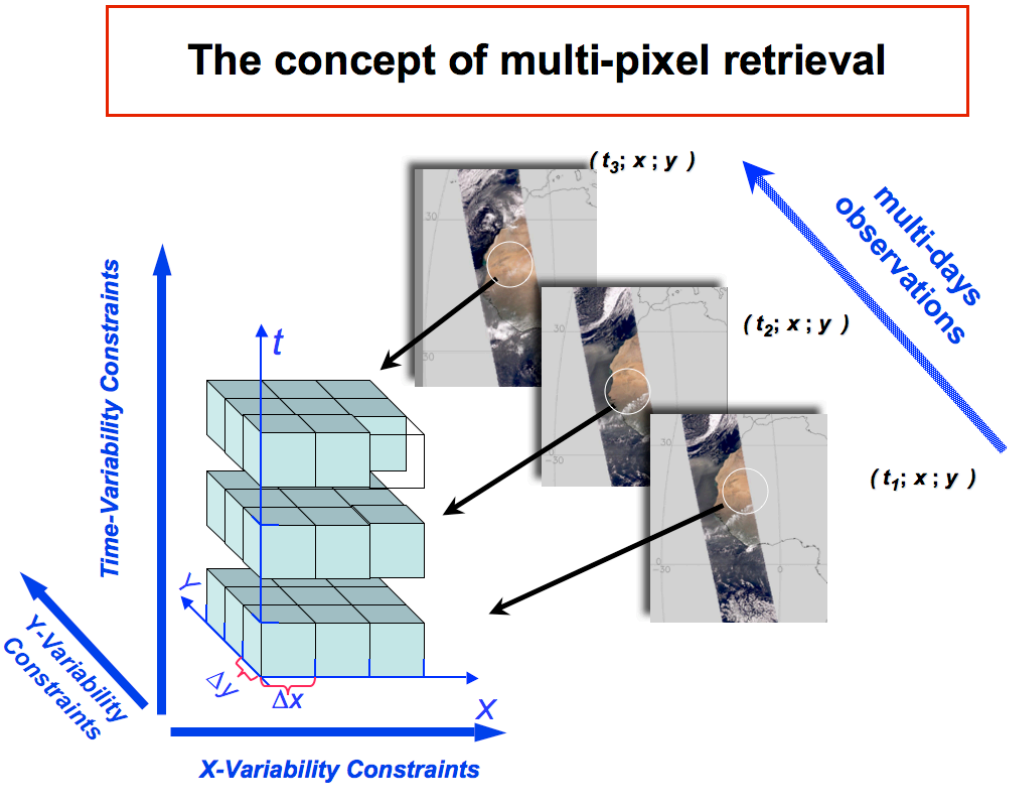
Figure 2.5. Organization of GRASP Numerical Inversion: Single-Pixel Scenario



2.3.3.2. Multi-pixel inversion

The multi-pixel retrieval approach proposed by Dubovik et al. (2011) is illustrated in Figure 2.6, “Organization of GRASP Numerical Inversion: Single-Pixel Scenario”. This is a new and very promising retrieval concept when a large group of "pixels" (instantaneous set of satellite data over one location) is inverted simultaneously. This approach allows a significant enhancement of atmospheric properties retrievals from remote sensing imagery by using additional a priori information on "correlation" between characteristics in different pixels of the inverted group. In addition, this principle allows a combination of different sets of coordinated observations, even when they are not perfectly co-incident and co-located (see Dubovik et al. 2014).

Figure 2.6. Organization of GRASP Numerical Inversion: Single-Pixel Scenario



The multi-pixel scenario retrieval was implemented in the code with the idea of achieving maximum benefits from the similarities in the mathematical and logistical operations between the single and multi-pixel retrievals. As a result, the multi-pixel retrieval, which is a more complex procedure compared to conventional single-pixel retrieval, was realized by implementing only limited modifications of the program. This approach practically does not increase calculation time (per pixel) and does not change (complicate) the code organization.

The structure of multi-pixel inversion is illustrated in Figure 2.6, “Organization of GRASP Numerical Inversion: Single-Pixel Scenario” for a *segment, i.e. a group of N inverted pixels* . It includes the following operations in addition to those realized for single-pixel retrieval scenario:

i. A loop implementing steps **i – iii** (of single-pixel procedure) for N pixels and forming N single-pixel Normal Systems $\mathbf{A}_{i,p}\,\Delta\,\mathbf{a}_i^p = \nabla\Psi_i^p$;

ii. Forming single Normal System for the *tile* of N pixels by arranging N single-pixel Normal Systems into a sparse diagonal matrix structure and adding the matrix Ω_{inter} defined using a priori inter-pixel smoothness constraints;

iii. Forming p-th Normal System:

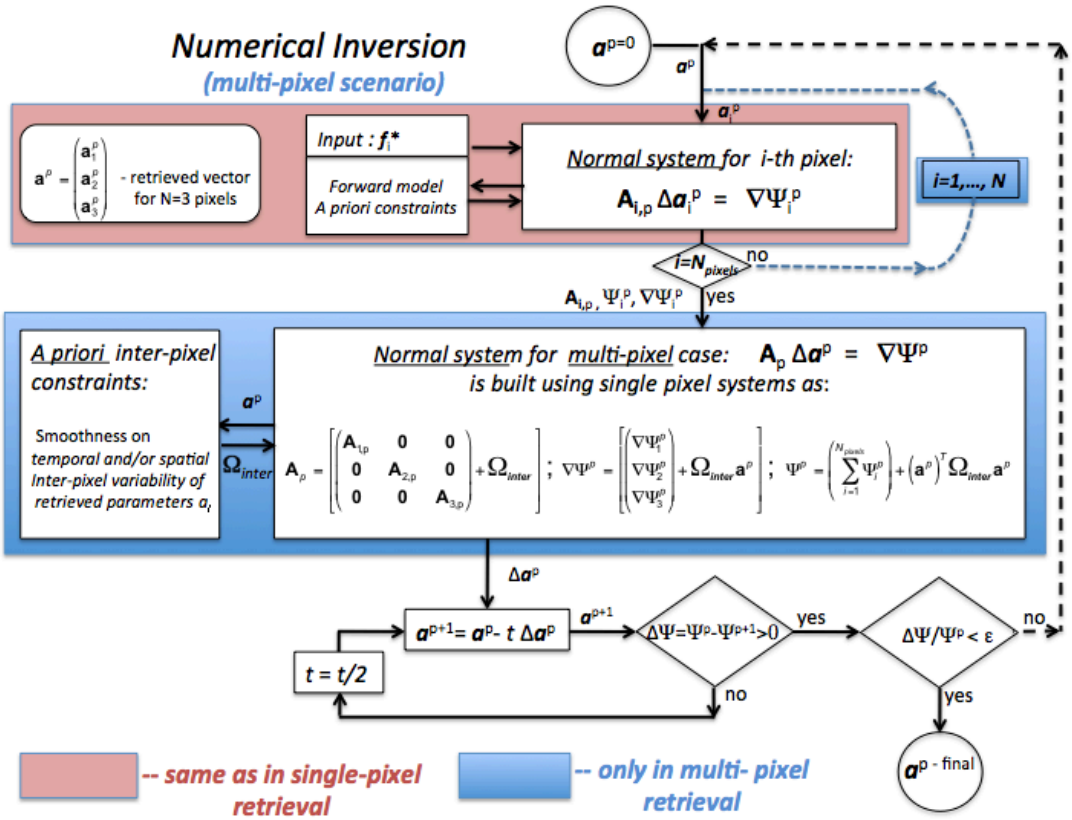
$\mathbf{A}_p \Delta \mathbf{a}^p = \nabla \Psi^p$,

where $\mathbf{A}_p$ Fisher matrix; $\Psi(\mathbf{a}^p)$ residual; $\nabla \Psi^p$ gradient of $\Psi(\mathbf{a}^p)$.

iv. Solving Normal System for the *tile* of N pixels to determine $\Delta \mathbf{a}^p$, and correcting the solution approximation $\mathbf{a}^{p+1} = \mathbf{a}^p t \Delta \mathbf{a}^p$ so that: $\Psi^p - \Psi^{p+1} > 0$;

v. Repeating steps **i – iv** until change of residual $\Delta \Psi = \Psi^p - \Psi^{p+1}$ is significant i.e. until $\Delta \Psi / \Psi^p < \varepsilon$

Figure 2.7. Organization of GRASP Numerical Inversion: Multi-Pixel Scenario



2.4. GRASP Control Unit

The control unit is a set of "service" programs that brings the application of the scientific GRASP algorithm to the operational level, first of all in the context of the processing of the data from satellite missions, such as PARASOL. It also provides a number of convenient for user features for applying GRASP to the observation and significantly reduces and simplifies the efforts in the development of new GRASP applications.

The control unit addresses a number of practical aspects:

- The original **GRASP scientific** core has been designed as a standalone application for processing a limited amount of observations both in spatial and temporal extent. However, integration of this original program to operational processing of remote sensing observation, such as global satellite observations, requires significant efforts on refactoring the scientific module and adapting it to operational data production environment.

- The data preparation for GRASP multi-pixel retrieval in processing satellite images is more complex than for classic operational retrievals, since the number of level-1 inputs needed for one level-2 output may range from a few days to several weeks. Correspondingly, the system must be able to

load the significant volume of data without exhausting the available memory. Also, a compromise between the spatial and temporal extent of multi-pixel retrieval application has to be found in order to satisfy the available memory constraints and processing time requirements.

- Though performance of GRASP algorithm is under constant improvement, the GRASP is a more complex and generally slower code than most of the conventional retrieval approaches. Therefore, a possibility of simultaneous retrievals is desirable for benefiting from parallelization of observation processing.

- The level of input data preprocessing for GRASP multi-pixel retrieval is significantly higher because inverted tails of (satellite) observations to be composed from observations acquired at different times should characterize the same grid of geo locations. Therefore, some kind of regridding is generally required in addition to common data preprocessing (application of cloud mask, gas corrections, etc.).

- The GRASP is versatile algorithm that has the potential to perform retrievals from diverse remote sensing observations and their combinations sensors, ranging from the ground-based photometers, radiometers and lidars to imagers onboard satellites. Therefore, adaptation of the GRASP algorithm for diverse observations should be always foreseeing. One of the main objectives of the control unit is to split the operations of the scientific algorithm and those of the data preparation.

The **control unit** manages all the system interactions with the processing environment. It loads the configuration settings. It is also responsible for receiving events from the system and provides the control commands for the application (the connection with the user interface). The control unit consists of the following unites (see Figure 2.1, “The architecture of the GRASP software package”):

2.4.1. Configuration manager

One of the first responsibilities of the controller is to load the configuration settings for the processing (production settings and scientific settings, such as initial guesses, number of parameters for the forward model etc). The *configuration manager* provides the possibility to deal with all the settings, including both the production and the scientific ones, in one unique way. In the development of the control unit this approach was considered as a strategic one, even though the production settings and the scientific settings are of entirely different nature, since they do not intervene at the same levels.

The configuration management is a key part of the developed system because the user usage experience depends on it. This module describes the usage interface, how to work with the code to achieve results. In addition, for developing this module it was necessary to understand, to document and to organize all the possible options (different behaviors) of the complex retrieval code. Moreover, some refactoring of the scientific package has been done to realize the configuration management concept.

The configuration manager controls the behavior of the control unit, as well as the peripheral elements such as the scientific input settings. Therefore, a change in the interface of the subsystems (especially for the scientific package as it evolves) can occur.

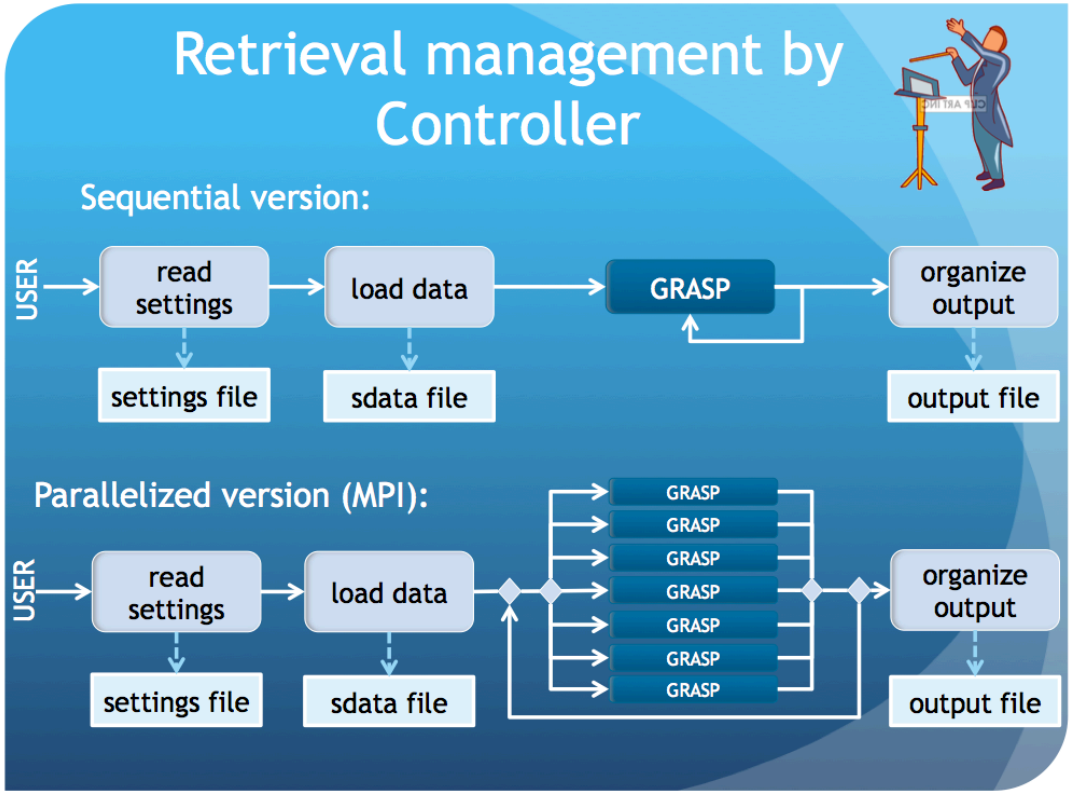
2.4.2. Controller Module

The GRASP executable results issued from the compilation of the controller module contains the main routine of the system. As illustrated in Figure 2.8, “Illustration of the data processing by the Controller”, the controller governs the data processing:

- gets orders and other events from the runtime interface

- performs actions in response to the events

Figure 2.8. Illustration of the data processing by the Controller



The controller is responsible for making all the parts of the control unit work together. While it receives events from the runtime interface, it takes actions and delegates most of its work to other modules of the control unit, such as the input and output drivers, and certainly to the scientific package.

There are two main workflows implemented in the controller. In the sequential version, the controller will retrieve a tile (a block of data that can be decomposed in many segments - a minimum instrument data treated inside the retrieval) and will work segment by segment, sequentially. In the parallel version of the controller, it can retrieve many segments at the same time, using MPI technology. The parallelization technology allows the controller to send jobs to different cores in the system, obtaining a lower total processed time.

2.4.3. Abstract input and output drivers

These sub-systems are responsible for preparing the input data for the scientific module and gathering output data in the unified "abstract" format. This procedure is not dependent on the particular application and is managed in unified manner by the GRASP scientific core. The creation of these sub-systems within the control unit assure the versatile and "generalized" character of GRAPS algorithm, allowing the system to be extended for specific purposes.

2.4.4. Concrete input and output data drivers

These sub-systems can be considered as peripheral sub-systems since they can be replaced in the context of every specific application. The **concrete input data drivers** are responsible for the satellite (e.g. PARASOL, MERIS) or ground-based (e.g. photometer or lidar) data loading. The rest of the system should never communicate directly with the loading driver but always with the abstract input bridge. The GRASP multi-pixel retrieval scenario uses multi-temporal data organized in the so-called segments, while the native formats of the input data may be in the form of many independent files (orbits for a given period, ancillary data, etc). Therefore, it is the role of the concrete input data drivers to obtain the data in the native format, gather them in a single, easy-to-use object tile, and to present

them as if they came from a single data source. Also, the input drivers may include some preprocessing of the data, such as atmospheric gaseous correction for satellite data, application of calibration, etc.

The **concrete output data drivers** are responsible for the scientific retrieval output products storage. They can be declined in several output formats, depending on the needs of the users and of the applications, and also on the requirements of the data centers: HDF, NetCDF, GIS databases, etc. The design of the control unit assures that the rest of the system does not interact directly with a concrete output driver, but with an abstract output bridge that delegates the action of writing to a concrete driver. This is because all storage formats are not adapted to all data sources and to all applications. In addition, the control unit system allows a straightforward replacement of the storage module by another one, if the GRASP retrieval is adapted to a new application or if an instrument is changed in the developed application.

2.4.5. GRASP file organization

The following list shows the GRASP source files organization. The code is classified into folders. The folders are represented by bold letters, followed by an explanation of their content.

- **build:** Compiled executable. It appears after the code compilation.
- **doc:** Technical and user documentation of the software package. The lines that appear for reading are stored in a raw format in this folder.
- **examples:** Some examples of retrieving instrument data
- **libs:** Bridges to certain libraries (facades)
- **src** Source code of the GRASP software
  - **controller:** contains the source files used by the controller main program, responsible for organizing the calls to all the modules of the system.
  - **global:** contains the source files of some functionalities that can be used by different submodules of GRASP. This code is GRASP dependent, thus can not be located in the "libs" folder, but is general enough to be used by the entire system.
  - **input:** contains the source files used by input abstract driver - the module is responsible for handling input data and injecting them into the scientific unit functions (the retrieval algorithm). This module can be extended by adding an input concrete driver that can include two additional kinds of functions: i) specific instrument drivers and ii) "transformers". i) are the functions called for loading data from specific instrument and ii) are the functions called after reading the input, call the scientific unit and transform the input data to scientific core GRASP algorithm.
  - **output:** contains the source files used by the output abstract driver - a module responsible for handling retrieval output. For example, the module creates a tile output based on single-segment outputs. This module can be extended by the output concrete driver that may includes different functions: 1) **output segment functions -** the functions that receive the output from a segment (provided by the output abstract driver) and can use it for extracting and printing target information; 2) **output tile functions -** these functions are called at the end of the process (once the retrieval information was received by the output abstract driver) in order to print the output for the entire tile; 3) **output current functions -** these functions can be called after processing a segment (once the retrieval information was received by the output abstract driver). Yet, the retrieval results for the entire tile will receive the tile output information as an argument. This approach can be used for printing a current status of retrieval for a tile before finishing the complete retrieval process.
  - **retrieval:** source files used by the scientific unit
    - **constants_set:** different sets of constants which define main array sizes used in the code. The use of this constants allows to optimize the memory used by GRASP for different applications.

- **inversion:** fortran functions related to numerical inversion.
- **forward_model:** fortran files for computation of modeled measurements (forward model)
- **interfaces:** routines that provide data preparation, validation and exchange between different submodules of the scientific module
- **external_interfaces:** definition of connections of the code with some external softwares (mainly superLU solver).
- **utilities:** general routines used in many different submodules of the scientific code such as print routines.
- **internal_files:** kernels used for computing particle single scattering properties by the forward model part of the code

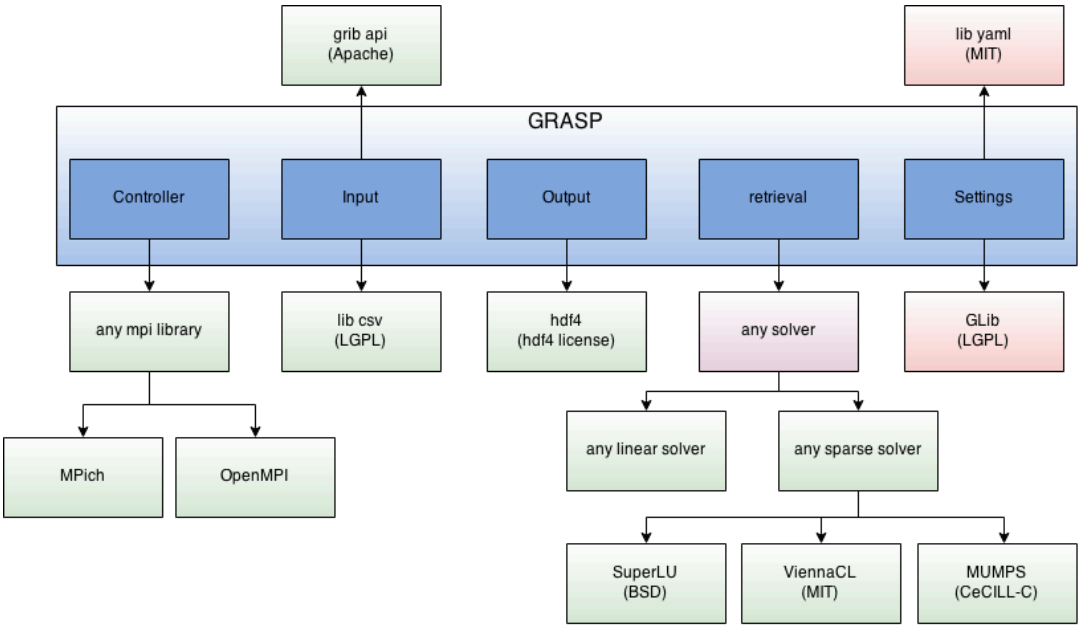
- **settings:** contains the source files used by the configuration unit that defines the settings for the calls to all the modules of the system.

2.4.6. External Libraries used by GRASP code

The GRASP software package allows the performance optimization of both the scientific retrieval and the control unit by utilising the external standard libraries that are not distributed as part of the GRAPS Open Code, but can provide some performance improvement of the code. These software packages are available on the Internet open access and can be downloaded by the users directly with no charge. Figure 2.9, “Structure of the utilization of public standard libraries in the GRASP code” shows the utilization of the standard software libraries in GRASP.

Figure 2.9. Structure of the utilization of public standard libraries in the GRASP code

(green color indicates the optional libraries, violet color indicates the optional but highly desirable libraries, the reddish color indicates that which is currently mandatory for the control unit, but that will be separated from the code before GRASP open is released). The licenses of each library are indicated in parenthesis.



The following main libraries are used by GRASP:

mpi library: the control unit has the optional feature of parallelizing segment process using mph technology. The various mpi libraries can be used (each one with different licenses). Thus, the user can choose the implementation of the mpi technology, using the selected software that may have different performance and license.

lib csv: this library helps to parse the databases prepared in CSV (Comma-Separated Values) format that is used in some input concrete drivers. This library is not needed if a specific compilation is used (that depends on the concrete data and driver used).

grib api: this library is needed to read the grib format that is used for reading climatology information in concrete satellite data drivers.

hdf4: this library is used to read/write files in hdf4 format. It is used in some output optional GRASP functions. Using a specific compilation (removing these output functions) the code can be run without using these libraries.

solver : the software package is optimized for solving linear systems. Such solver can significantly improve the performance of GRASP in certain situations since GRASP scientific core performs retrieval sequentially solving a number of linear systems. For example, when the multi-pixel retrieval is performed, the GRASP scientific core solves linear systems that can be of very large dimension and have pronounced sparse structure. The code was adapted and tested for using libraries such as SuperLU, ViennaCL, and MUMPS. It is applicable for solution of any linear system in the GRASP internal routine.

GLib: this is the GNU C library that contains a set of tools for programing in C. Specifically, it is used by yaml settings library (which source code is part of GRASP settings module) helping to read YAML files (using lib yaml dependency) and translate them into C structures. In that process GLib is used to define internal tree structures.

lib yaml: this is low level library to parse yaml format files.

Chapter 3. Installation

3.1. Introduction

This section describes how to download, compile and install the GRASP Open software from the user's perspective. Basic knowledge of the terminal and some tools such as GIT are necessary to complete this process.

3.2. Hardware requirements

The system has been tested on modern Intel architectures only (PC compatible, Macintoshes). It should work on any 32-bit and 64-bit platform. A minimum of 2 GB of RAM is recommended. GRASP is a very hardware demanding software because math calculations are very time-consuming. In a multicore scenarios GRASP is able to parallelize the entire process using MPI technology. Additionally, GRASP offers a GPGPU module (as extension, not installed by default) that allows to parallelize a single segment retrieval using graphics cards.

3.3. Operating Systems

The code has been widely tested under Linux machines and MacOSX™ systems. The installation process in old operating systems is usually more complex so we suggest to always use the lastest version of operating system. Ubuntu systems have shown the easiest installation process so we recommend it for regular users.

For windows users some test under Microsoft Windows™ has been also performed successfully using Cygwin, but we do not offer official support for this solution. Some manipulations are needed depending on the specific version of Windows and Cygwin, so even if a solution is possible, it is not straightforward and we do not recommend it. There is one exception: the last version of Microsoft Windows 10™ offers the feature **Windows Subsystem for Linux** which allows the users to set up an Ubuntu environment over Windows. This unestable but very promissing feature can really help to install GRASP on Windows. You can start with this solution following this article of MSDN [https://msdn.microsoft.com/commandline/wsl/about]. Once you have your Ubuntu environment configured, you can follow this guide as a Linux user.

3.4. Access to GRASP Open repository

To access the code, the users have to register their account on GRASP Open web page. There are different ways to download the code (direct download link, clone over HTTP...) but we recommend to clone the code over GIT protocol. This method has the benefit of being manageable by the grasp-manager (see Section 3.7.1, “GRASP Manager”) which is the easiest way to keep the code up-to-date and manage GRASP extensions. The following steps show the process of getting access to the repository and setting up your access via GIT protocol.

1. Go to GRASP-Open web page code section [https://www.grasp-open.com/products/]. And click in the "GITLAB REPOSITORY" bottom.

2. Fill all the fields of the registration form (last section of that page), accept the conditions and press SIGNUP button.

3. If the registration was successful, you will get a confirmation message. Click on the "Initialize password" button.

4. The GitLab system, which is the system used to manage the code, will ask you for your email to reset the password (initialize it). Introduce your email and click on the "Reset Password" button.

5. You'll receive an email for resetting the password. Follow the link in the email and set up your new password.

6. The next window will show the main page of the GitLab system. You can sign in using your email and your password. Remember that you can access that web page at anytime, using the button "GITLAB REPOSITORY" in GRASP-Open web page code section [https://www.grasp-open.com/products/].

7. On the main page of the GitLab system you can see different repositories you can access. Go to the GRASP repository or use this link [http://code.grasp-open.com/open/grasp]

8. There you can explore many things: you can see the code, see the changes, open an issue to get in touch with the developer team ... To download the code, there are three alternatives: a) direct download, b) cloning repository over http or c) cloning repository over git protocol. The last solution is recommended so we'll continue explaining this process.

9. Set up your ssh-key in GitLab. It allows you to access the code via GIT protocol whitout a need of typing passwords. To set up a ssh-key you have to go to you-profile>edit>ssh-keys section or using this link [http://code.grasp-open.com/profile/keys].

10. Follow this guide [http://code.grasp-open.com/help/ssh/README] to create a ssh-key if you don't have one. If you already have a ssh-key you can follow the same guide but skip the first steps of a key creation, just copy it to the clipboard.

11. Paste your ssh-key in the GitLab system and click the button "Add key"

12. Now your system is properly configured and you can download the code over the git protocol. The section Section 3.5.2, “Basic installation of GRASP” explains how to download the code, compile and install it.

3.5. Building and installing GRASP

The GRASP software makes a heavy use of a number of libraries for data preparation and numerical computations. As for the GRASP software itself, it relies on the following libraries FOSS [http://en.wikipedia.org/wiki/Free_and_open-source_software] or belong to the public domain.

We are very focused on keeping the whole system free and depending only on non-commercial libraries. That does not mean that non-open source software can't be linked to GRASP (certainly closed-source, or even open-source, although restricted [1] solutions are sometimes better than the free ones for certain purposes), but the system should always be able to run only with free, open-source alternatives.

3.5.1. Dependencies

The following list shows the GRASP core dependencies. Some extensions can require extra dependencies, in that case, please follow the documentation of the extension to know the installation process.

- a C compiler (recommended gcc).
- a Fortran compiler (known to work with gfortran [http://gcc.gnu.org/wiki/GFortranBinaries] and ifort).
- a make command (provided on any POSIX system).
- the cmake [http://www.cmake.org] building software.
- One of the four numerical packages:
  - SuperLU [http://crd-legacy.lbl.gov/~xiaoye/SuperLU/]
  - SuperLU_MT [http://crd-legacy.lbl.gov/~xiaoye/SuperLU/] (not included in the framework yet)

[1] For the distinction between the *free* and *open source* movements, see http://www.gnu.org/philosophy/free-software-for-freedom.en.html.

  - MUMPS [http://graal.ens-lyon.fr/MUMPS/]
  - ViennaCL [http://viennacl.sourceforge.net]
- a BLAS library (Netlib BLAS [http://www.netlib.org/blas/], ATLAS [http://math-atlas.sourceforge.net], GotoBLAS [http://www.tacc.utexas.edu/tacc-projects/gotoblas2/]). Actually, BLAS is not used directly by GRASP, but some numerical packages (SuperLU, MUMPS), which GRASP relies on, are built on the BLAS. BLAS will be necessary only if you build GRASP with these packages. Currently, Netlib BLAS is the default BLAS library for the GRASP but this may change over time. The GRASP code has currently been tested with the Netlib, BLAS and ATLAS.
- The LAPACK [http://www.netlib.org/lapack/] library. As for the ATLAS, LAPACK is only necessary with the SuperLU and MUMPS packages, not with ViennaCL. Please note that ATLAS ships with a partial LAPACK implementation for its own purposes, but that is not sufficient for the numerical packages on which GRASP relies. You must install the full LAPACK package.
- ScaLAPACK [http://www.netlib.org/scalapack/] (a requirement for the SuperLU and MUMPS numerical packages only).
- ParMETIS [http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview] (a requirement for the MUMPS numerical package only).
- OpenMPI [http://www.open-mpi.org] (optional). This package is needed to work with the MUMPS. The GRASP software can also be compiled using MPI by adding it the capability to process many segments at the same time.
- The Gnome Library GLib [https://developer.gnome.org/glib/] (mostly used by the configuration manager).
- The LibYAML [http://pyyaml.org/wiki/LibYAML] library (YAML is the chosen format for GRASP configuration).
- The C Unit Testing Framework: CUnit [http://cunit.sourceforge.net].

These dependencies can be installed in an Ubuntu system with the following command:

```
# All deps except superlu:
sudo apt-get install build-essential cmake git gfortran libyaml-dev libglib2.0-dev libcunit1-dev libsuperlu-dev
```

3.5.2. Basic installation of GRASP

Your system should be ready now for the installation. If it is not the case, please refer to the previous section. Also, it is assumed that the user has access to the git repository, otherwise please check Section 3.4, “Access to GRASP Open repository”.

The steps to download and install GRASP are the same for all platforms (Windows/Cygwin, MacOSX, Linux). Depending on your system, you may or may not have the **sudo** command. It is used for running a command with administrative rights. In that case, you can try without **sudo** (e.g. on Cygwin), or use the **su** instead for logging as administrator. You can also perform a custom installation (see the next section) so you don't need to be administrator. If none of this makes sense for you, ask your local Unix guru.

```
$ git clone git@code.grasp-open.com:open/grasp.git
$ cd grasp
$ # you should now be in the master branch of the project
$ # (developers of the project may need to checkout the dev branch)
$ make # build the project using the default build settings
$ sudo make install # install grasp. Administrative privileges are needed.
```

3.5.3. Advanced compilation

GRASP uses cmake system to compile the code. You can compile the code using CMAKE following these steps:

```
$ # Place a terminal in GRASP root folder
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Release -DADDITIONAL_DEPENDENCIES_PATH=/usr/local/grasp-deps -DCONSTANTS_SET=generic
$ make -j12
$ sudo make install
$ grasp # test the command
```

You can use a different compilation configuration. All configuration options are defined in Section 3.5.3.2, “Custom installation using cmake”

For users that are not familiar with cmake there is a Makefile which wraps cmake system and is placed in root folder. This Makefile simplifies the use of cmake via make script. Internally, the Makefile creates cmake structure and call it. Thanks to this operation, you can compile using this system like it is defined in Section 3.5.2, “Basic installation of GRASP”. Addionally, this Makefile also allows you to use some extra configuration parameters that are explained in Section 3.5.3.1, “Custom installation using make”.

Finally, the last way to compile the code is via grasp-manager. Grasp-manager is described in the next section: Section 3.7.1, “GRASP Manager”
.

3.5.3.1. Custom installation using make

If you have compiled the code following the rules explained in Section 3.5.2, “Basic installation of GRASP”, you have compiled the code with default options. The options and their possible values are listed below (the default value is written in italic):

- **CONSTANTS_SET:** *generic* or see Section 3.5.3.3, “Constants sets”
- **BUILD:** *Release* or Debug, RelWithDebInfo or Fast
- **MPI:** *off* or on
- **DEBUG_MPI:** *off* or on
- **F90:** *gfortran* or ifort
- **PREFIX:** */usr/local* or other valid path where dependencies are available
- **BUILD_DIR:** *build* or other name but then build is ignored by git
- **CC:** *cc* or another valid c compiler
- **CCX:** *c++* or another valid c++ compiler

For example:

```
$ make CONSTANTS_SET=polder MPI=on
```

3.5.3.2. Custom installation using cmake

By default, the resources and dependencies will be installed in the following directories:

`/usr/local/share/grasp` (resources that are internal databases or files used by GRASP)
`/usr/local/grasp-deps` (for general-purpose, utility libraries)

If one does not wish (or may not be able) to install GRASP under default system directories, it is possible, and very easy to change these paths of installation with the `PREFIX` variable.

Please find below the way to install the project under your HOME directory instead of `/usr/local` (now you don't need administrative rights anymore). In this case, we will use cmake compilation system instead of the Makefile placed in the root folder, which wraps it.

```
$ cd ~/grasp/dependencies
$ sudo make PREFIX=$HOME/local install
$ #The command above builds and installs the third-parties dependencies under /home/your_name/local,
$ #instead of /usr/local
$ cd ..
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Release -DADDITIONAL_DEPENDENCIES_PATH=$HOME/local -DCONSTANTS_SET=generic
$ #The command above generates a Makefile with custom parameters
$ make -j12 # build the project
$ sudo make install # install grasp under $HOME/local/bin
$ $HOME/local/bin/grasp
$ # test the command (of course, it is then recommended to add $HOME/local/bin to your PATH)
```

Another possible customization is to change the numerical solver for GRASP. By default, it is set to SUPERLU [http://crd-legacy.lbl.gov/~xiaoye/SuperLU/]. If you wish to try another solver [2], you may use:

cmake SPARSE_SOLVER= *your_chosen_solver*

Table 3.1. SPARSE_SOLVER's valid values

SUPERLU (default)
SUPERLU_MT (not supported yet)
VIENNA_CL
MUMPS

You may also want to link the project to the BLAS implementation of your choice. This is possible with:

cmake BLAS= *your_chosen_blas_library*

Table 3.2. BLAS valid values

netlib-blas (default)
atlas

The CONSTANTS_SET setting can be used for performing memory optimizations with a specific set of data, e.g. **cmake CONSTANTS_SET=polder**. If one is not sure, one may stick to the default. See Section 3.5.3.3, “Constants sets”

Table 3.3. Main CONSTANTS_SET values (installed by default)

generic (default)
valgrind
(you can install external constants set and use them)

The `BUILD` setting can be used to switch between debug (non optimized, with debug information), dev (partially optimized) and prod (optimized) modes at build time, e.g. **cmake BUILD=prod**. The default is `dev`.

[2] That it is possible doesn't mean that it is recommended. In the current status of the project, GRASP has not been tested extensively with other solvers than SuperLU and the support for other solvers is still quite sloppy.

Table 3.4. BUILD valid values

Debug
Release (default)
RelWithDebInfo
Fast

Of course, there is no limitation to the number of settings that the **cmake** command can accept:

cmake PREFIX=$HOME/local BUILD=prod SPARSE_SOLVER=SUPERLU CONSTANTS_SET=generic

3.5.3.3. Constants sets

GRASP is a highly memory- and time-consuming software, which is strongly optimized. Usually in software development, time can be optimized using more memory and memory can be saved using more calculation time. In software development, there should be a balance between the time consumption and the memory cost. GRASP has found this balance using many static arrays to optimize time performance and defining the size of these arrays like constant values. This method permits to choose during compilation time, the size of these arrays, allowing the user to optimize the software, depending on the use. For example, GRASP will take more memory if it is compiled to use a maximum of eight wavelengths instead of four. We have created some standard constants sets for the most common uses and in general you can use "generic" CONSTANTS_SET as one valid to test all instruments. If it is not valid you can check whether your application fits with another existing constants set or create your own. Constants sets are managed by grasp-manager as an extension of GRASP. It means that you can install an external definition of the constants set depending on your necessities or create your own. Constants set are placed in `src/retrieval/constants_set/{CONSTANTS_SET_NAME}` and consist of four files:

- mod_par_DLS.inc
- mod_par_OS.inc
- mod_par_DLS_bin.inc
- mod_par_inv.inc

Where {CONSTANTS_SET_NAME} is the name of the constants set, which also can be defined by the user, for code compilation.

3.6. Running the code

Once the code is installed, you can test it right away. In the following examples, it is assumed that the user has installed GRASP as a wide system executable. Otherwise, the GRASP binary is placed in `./build/bin/grasp`. When GRASP is executed without arguments, it prints some general information about the version and how it was compiled:

```
$ grasp

GRASP core version: v0.7.0 (commit: c0bd56a ; branch_name: HEAD)
Compiled on 2016-06-02 10:21:03 +0200 commit of 2016-05-24 18:17:11 +0200
With C compiler: Apple LLVM version 7.3.0 (clang-703.0.31)
With FORTRAN compiler: GNU Fortran (Homebrew gcc 5.3.0 --with-all-languages) 5.3.0
Using generic constant set and build type Release
Maximum segment size: nx=2 ; ny=2 ; nt=30
Input drivers loaded: sdata
Input transformers loaded: none
Output segment functions loaded: ascii classic classic_plot none
Output tile functions loaded: ascii none
Output current functions loaded: none
Path to resources: /usr/local/share/grasp/
```

```
Sparse solver used: SuperLU
Build System: Darwin-15.5.0
Executable path: undefined absolute path (./build/bin/grasp_app)

usage: grasp [OPTIONS] {settings_file.yml}|help

OPTIONS:
-V       use valgrind
```

In the examples folder there are some examples that the user can apply to verify that the whole system is working properly. To run an example you have to call GRASP followed by the settings file as the first argument:

```
$grasp examples/sunphotometer/settings_example_sunphotometer_inversion.yml
Config file read successfully ❶
The tile is divided in segments with 1 rows, 1 cols and 1 times. 1 inversions will be performed (sequential version) ❷
Retrieval #0 (1/1): 100.00%: 1 pixel will be processed ❸
```

... ❹

```
826.69409  1:   0.76332E+00   84.82804 %    2:   0.47585E+00    119.16735 %    pixel # 1      Residual using INITIAL GUESS
277.77368  1:   0.29931E+00   30.63890 %    2:   0.33364E+00    30.73591 %   pixel # 1    Residual after iteration # 1
22.29646  1:   0.71556E-01    5.62210 %    2:   0.21300E-01     2.00627 %    pixel # 1    Residual after iteration # 2
8.99939  1:   0.19544E-01    1.39876 %    2:   0.86748E-02      1.84197 %    pixel # 1    Residual after iteration # 3
1.26664  1:   0.34922E-02    1.21845 %    2:   0.25801E-03      0.03841 %    pixel # 1    Residual after iteration # 4
0.91952  1:   0.27402E-02    0.83636 %    2:   0.34244E-03      0.05860 %    pixel # 1    Residual after iteration # 5
0.49750  1:   0.27118E-02    0.39311 %    2:   0.28940E-03      0.09854 %    pixel # 1    Residual after iteration # 6
0.49593  1:   0.22867E-02    0.25629 %    2:   0.41266E-03      0.10136 %    pixel # 1    Residual after iteration # 7
0.37954  1:   0.83430E-03    0.12501 %    2:   0.34994E-03      0.03372 %    pixel # 1    Residual after iteration # 8
0.19446  1:   0.67640E-03    0.09489 %    2:   0.16508E-03      0.04304 %    pixel # 1    Residual after iteration # 9
0.11306  1:   0.59844E-03    0.07205 %    2:   0.84066E-04      0.02038 %    pixel # 1    Residual after iteration # 10
0.11299  1:   0.58013E-03    0.07008 %    2:   0.85618E-04      0.01960 %    pixel # 1    Residual after iteration # 11
Retrieval #0 (1/1): 100.00%: 1 pixels processed in 15.478408 seconds (cpu time: 15.433357).
Average per pixel: 15.478408 (cpu time: 15.433357)
Retrieval #0 (1/1): 100.00%: finished
```

... ❺

```
Size Distribution dV/dlnr (normalized to 1) for 1 - fraction
0.50000E-01   0.18937E-03
0.65604E-01   0.25878E-01
0.86077E-01   0.31692E+00
0.11294E+00   0.46703E+00
0.14818E+00   0.49857E+00
0.19443E+00   0.31519E+00
0.25510E+00   0.13060E+00
0.33472E+00   0.37423E-01
0.43917E+00   0.15058E-01
0.57623E+00   0.21979E-01
0.75605E+00   0.60639E-01
0.99200E+00   0.12603E+00
0.13016E+01   0.20535E+00
0.17078E+01   0.31518E+00
0.22407E+01   0.32588E+00
0.29400E+01   0.32922E+00
0.38575E+01   0.22504E+00
0.50613E+01   0.15226E+00
0.66407E+01   0.84118E-01
0.87131E+01   0.25702E-01
0.11432E+02   0.33366E-02
0.15000E+02   0.15965E-03
```

... ❻

```
Total Time: 1 pixels processed in 15.573089 seconds (cpu time: 15.515442).
Average per pixel: 15.573089 (cpu time: 15.515442)
Algorithm Time: 1 pixels processed in 15.478408 seconds (cpu time: 15.433357).
Average per pixel: 15.478408 (cpu time: 15.433357)
Control Unit Time: 1 pixels processed in 0.094681 seconds (cpu time: 0.082085).
Average per pixel: 0.094681 (cpu time: 0.139732)
```

❶ First step of execution is to parse settings file and validate it. If everything is OK this line will be printed, otherwise a list of errors will be produced. Please, pay attention to error messages because they should help you to understand what is going on and how to resolve the problem.

❷ This line informs you how the data are going to be organized for retrieval. Data are organized as segments that will run inside the retrieval algorithm. Control unit will organize these segments into a tile as a bigger group of pixels.

❸ The first retrieval is launched and the retrieval algorithm starts to process the first segment (group of pixels)

❹ The retrieval can be in a verbose mode or not. If it is in a verbose mode, a detailed information on the process will be printed. The most important information is in the next lines. The user can follow the fitting process and see how the errors decrease iteration by iteration.

❺ Finally, the results are processed. This example prints results on the screen. You can dump them to a file in different formats such as CSV, HDF ... Output functions are extensions that you can optionally install. In the settings file, it is defined which functions are used and how (set up).

❻ The process is finished with a small summary about how many data have been processed and how long it took.

It is also possible to add any number of arguments in the form of `setting=value`. They provide a quick and easy way to override the default settings in the configuration file, for experimenting without editing this file.

Table 3.5. Some common settings

Name	Type	Default	Description
`help`	`boolean` [a]	`false`	When set to `true`, displays an exhaustive list of settings with their significance.
`input.debug.print_clean_segment`	`boolean`	`false`	When set to `true`, prints a segment that has just been cleaned from its non-significant data. (see also print_raw_segment)
`input.debug.print_raw_segment`	`boolean`	`false`	When set to `true`, prints a segment that has just been loaded from a driver (with possible non-significant data). (see also print_clean_segment)
`input.sdata.dump`	`boolean`	`false`	When set to `true`, displays the input measurements in the form of a SDATA stream. Mostly useful for the maintainers of the scientific subsystem.
`input.sdata_driver.debug`	`boolean`	`false`	When set to `true`, displays the actions of the sdata driver when a SDATA file is being read. This can be used for validating a new SDATA file (the contents of this file will be displayed in a readable form), and less commonly for debugging the SDATA driver.
`retrieval.debug.print_segment_information`	`boolean`	`false`	When set to `true`, prints a segment with data that are actually passed to the retrieval library (unless there is a bug in the C/ Fortran interface, the data should be the same as those from input.debug.print_clean_segment).
`retrieval.debug.verbose`	`boolean`	`true`	When set to `true`, displays debugging information relative to the retrieval subsystem. Mostly useful for the maintainers of the scientific subsystem.

[a] Boolean variables (also known as *logical* in the FORTRAN community) can take the values `true` or `false` (that can also be abbreviated as `t` and `f`)

3.6.1. Usage of GRASP: The configuration file

The default behaviour of the system is defined in a configuration file, whose settings can be overridden by command line arguments, as described in the previous section. A configuration manager (also known as the settings module) is responsible for loading the configuration file and for taking care of the overriding mechanism for command line arguments, if necessary. It centralizes all the information needed for a processing and in that matter drives the actual behaviour of the framework.

When there is more than a few parameters to change, or when one wishes to perform more persistent corrections, it is easier to edit the configuration file than to set parameters in the command line.

The chosen format for new GRASP configuration files is YAML [http://en.wikipedia.org/wiki/YAML], a standard format perfectly adapted for complex configurations such as the one needed by the GRASP project. It provides support for simple values as well as for complex data structures, while maintaining a high level of readability. YAML format is based on fixed indentation (the spaces before an element define the level where it applies). The command line interface of GRASP proposes an easy system to overwrite the main settings file with a "dot syntax", where each level of indentation is replaced by a dot. For example, in GRASP, it is equivalent to be called with the argument `input.driver=sdata` or to be defined in the settings file, as follow:

```
input:
    driver: sdata
```

GRASP settings system allows to import external files into one. Note that the standard YAML format doesn't support file inclusion, but GRASP configuration files support this feature through the `import` keyword that expects a list of files to be included. It is, therefore, possible to split large configuration

files into smaller, easier to maintain independent files (this is especially important since several people will have to maintain different sections of the configuration).

Figure 3.1. Excerpt of configuration file

```
import: [ ] ❶

input: ❷
    driver: sdata_driver
    filename: bin/SDATA_NEW.dat

retrieval:
    # General retrieval parameters ❸
    general:
        minimization_convention: logarithm
        threshold_for_stopping: 1.0e-5
        number_layers: 50 ❹
        shift_for_applying_logarithm_to_negative_values: 0.2
        binning_method: logarithm
        maximun_iterations_of_Levenberg-Marquardt: 15
        stop_before_performing_retrieval: false
        internal_file_path: "../retrieval/internal_files/"
        external_file_path: "../../home/"
        reference_plane_for_polarization: meridian
        regime_of_measures_fitting: absolute_polarization_components
        linearization_threshold:  0.03
        IMQ:       2
        use_internal_initial_guess:     false
        threshold_for_length_corrections:     -1.0e-2
        threshold_for_stopping_Q_iterations:  1.0e-2
        scale_for_finite_difference:          1.0e-3
        irradiance_corrected: false # or no or 0
        coeff_corr: 0.96
        regime_of_multipixel_constraints:
            inversion_regime: multi_pixel_followed_by_single_pixel
            time-scale:   100.01
            x-scale:   100.05
            y-scale:   100.05
        error_estimation: true
        number_of_characteristics_retrieved: 7
...
    noises:
        noise[1]: ❺
            standard_deviation: 0.0
            error_type:  absolute
            variation:  0.01
            measure_type[1]:
                type: I
                wavelength_involved: [ 1, 2, 3, 4, 5, 6 ] ❻
        noise[2]:
            standard_deviation: 0.0
            error_type:  absolute
...
```

❶ The GRASP configuration file supports the file inclusion, with the `import` statement. The included files must be given in a comma-separated list, this can remain empty.

❷ The configuration is organized in sections. The structure is defined by the explicit indentation. The order of the sections and the elements in the sections doesn't matter, as long as the structure and indentation are respected.
❸ Comments are supported: anything after a # sign is concidered as a comment.
❹ Access to fields from the command line (for overriding a setting, say) is straightforward, using the common dotted notation: for instance, the fully qualified name of `number_layers` is `retrieval.general.number_layers`
❺ Section and property tags can be indexed, with a simple bracket notation. A feature that is not supported by the standard YAML format. This makes it easy to define arrays of complex data structures.
❻ Simple lists of data are supported (a standard feature of YAML). The number of elements can be determined at runtime by the configuration manager.

All these qualities make the new configuration files very easy to read, maintain and extend.

3.7. Code repository and extensions

The GRASP code is managed by GIT. If you have a version that doesn't use GIT (for example, by downloading it from a web server), we strongly recommend you to look for a version downloaded via GIT (see Section 3.4, “Access to GRASP Open repository”). It will allow you to be connected to the server to make updates of the code. Additionally, the GRASP code can be extended in different parts. You can extend the input module or the output module. Extensions in input module are classified as drivers and transformers. A driver is a module that is called to read input data. The most basic driver is the SDATA driver which reads a SDATA file (Section 4.2.1, “The SDATA format”) but many other drivers can be implemented to read raw databases and inject, without using intermediate text files, data directly in the GRASP algorithm. A good example is to process satellite data, where the performance is a keystone to be able to process this kind of huge archives. Satellite data is not transformed to sdata format, instead of this, a specific driver is developed to connect raw satellite archive with GRASP. Transformers are another type of input extensions which allow to modify input data after reading it. An example of a transformer could be to load a climatology database to optimize input parameters. This action can be shared between different drivers and it is called after loading data by a driver. Output module can be extended with output functions. There are three types of output functions:

- **segment output function:** It will be called after processing each segment of a tile
- **tile output function:** It will be called after processing entire tile
- **current output function:** It will be called after processing each segment, but it receives a partial tile as an argument. This function can print a tile with the current processed information

Each extension is distributed separately of the core code (except if an extension is considered as a core, for example, sdata driver is essential). To install a new extension you can place the source code in the specific place or you can use the grasp-manager, an extra tool which will help you to manage the code and its extensions. It is described below (Section 3.7.1, “GRASP Manager”), and here we will explain how to install manually an extension.

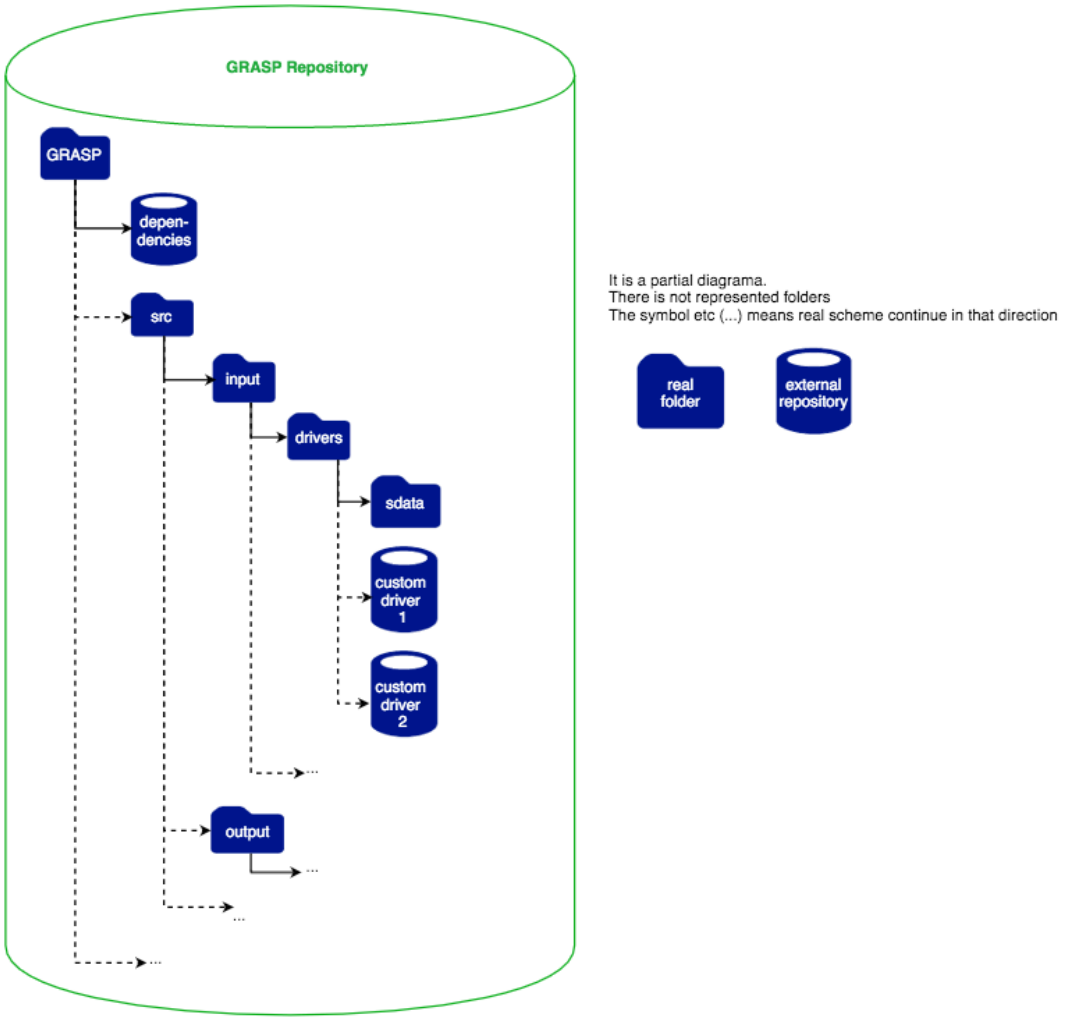
Extensions are the pieces of code that are detected and added during compilation. To install an extension, you have to place the code inside the core repository in a proper place and compile again. The corresponding places for each of the extensions are the following:

- For input
  - drivers have to be placed in `src/input/drivers`
  - transformers have to be placed in `src/input/transformers`
- For output
  - segment functions have to be placed in `src/output/segment_functions`

- tile functions have to be placed in `src/output/tile_functions`
- current functions have to be placed in `src/output/current_functions`

- Kernels: They have to be placed in `src/retrieval/internal_files`
- Constants sets: They have to be placed in `src/retrieval/src/constants_sets`

The following diagram shows how the GRASP repository integrates extension repositories inside itself.



Typing the command **grasp** without arguments, you will obtain information about how the software was compiled including available extensions. To know more details about the extensions and write one by your own, please go to technical documentation. [http://www.grasp-open.com/tech-doc]

Trick: Since the GRASP code files are tracked by GIT it is not recommended to modify them, except if you want to develop something. If you want to place your tests and examples near the code, use a folder called "home" (it is created during compilation but you can create it by yourself, if you wish). This folder will not be tracked by GIT, allowing you to have your tests with the code without having conflicts with GIT.

3.7.1. GRASP Manager

GRASP Manager is a script placed in the GRASP root folder which simplify (even more) the update process. This script wraps Makefile system and adds a mechanism to work with GIT and the multi-repository environment used by GRASP (see Section 3.7, “Code repository and extensions”). Compilation of GRASP is always based on cmake. The make system wraps cmake, helping with the

creation of a necessary folder structure, and grasp-manager script wraps make system, helping with the use of GIT, for users that don't feel comfortable using it. Also, it takes care of the extensions, downloading and installing them in the correct places. GRASP Manager script is configured via `grasp-manager.yml` file. Since it is a configuration file it is not tracked by GIT. Instead of this, a template called `grasp-manager.yml.dist` is offerred by the system. If there is no `grasp-manager.yml` file, it will be created as a copy of `grasp-manager.yml.dist` file, first time you run grasp-manager script.

To know the list of actions offered by this script, you can just type **./grasp-manager.sh**. A list of available actions will be printed. In these sections we will explain the most interesting actions for the users, but there are more actions that can be interesting if you are a developer. Please remember to have a look at the technical documentation to know more about this script.

The basic actions for regular users are **update-grasp** and **update-grasp-to-dev**. They allow you to update the code and the extensions to the newest version. With **update-grasp** you will get the last stable version and with **update-grasp-to-dev** you will get the last unstable version (next code to be released), which is not recommended to use except if you know what you are doing ;-). These commands accept an argument that is called "environment". Environments are defined in `grasp-manager.yml` and allow you to customize the way to compile the code and the list of extensions that you want to have available. The documentation about how to write a configuration file for grasp-manager is written in the file `grasp-manager.yml.dist`. Please check that file, to know all available options. Note: Remember that in YAML format, lines that start by # symbol are comments.

When you run grasp-manager script, a backup of your code is created. This backup contains the information about the previous GIT commit and unsaved changes in the code. Please take into account that the repository has to be "clear" before performing any update actions, otherwise the changes will be undone and saved in a backup. You can find these backups in the `home/grasp-manager` folder. They are organized by time (when grasp-manager was executed) and you can apply them via **rollback** (applies last backup) or **apply** actions.

Finally, the next code will show an example of the use of grasp manager:

```
$ # Following command will move the repository to dev branch using the environment defined
$ # as release in grasp-manager.yml
$ # It will install/uninstall extension in order to have exactly the extensions release environment specify
$ ./grasp-manager.sh checkout dev release
$ # Then, we can compile the code using the compilation settings defined in 'release' environment
$ # Since the release environment was the last used, it can be omitted.
$ # When the environment argument is omitted, we use the constants used
$ # in the last command, but we don't install/uninstall extensions (available extensions are already in use)
$ ./grasp-manager.sh make
$ # Finally we are going to install the code with the system in the last version.
$ # This is the most important action for regular users.
$ # A user of GRASP can regularly update the system via the following command:
$ ./grasp-manager.sh update-grasp
```

3.8. Known problems

Listed here, are common problems found during installations process.

- In MacOS system, some users receive fatal error: The remote end hung up unexpectedly; fatal: early EOF; fatal: index-pack failed
  It can be necessary to increase Git buffer size using the following command:git config --global http.postBuffer 1048576000

Chapter 4. How to use GRASP

4.1. How to run the code

After the installation of GRASP software, the **grasp** command will be available. As it is explained in a previous section (see Section 3.6, “Running the code”) just typing **grasp** will print some general information about how the software was compiled.

First arguments that **grasp** executable expects is a path to the settings file. The settings file describes the inversion strategy and general behaviour of the process: where is the input data, in which format is the input data, where to store the output results... Therefore, a deep knowledge of the settings parameters is the base to understand GRASP.

4.1.1. Settings file

Settings file is written in YAML [http://en.wikipedia.org/wiki/YAML] format and that brings many benefits: easy to write, clear to read, self-explanatory names, flexible and powerful. The concepts are organized in blocks that are translated to YAML thanks to the fixed indentation (we suggest 4 white-spaces). So, for example, the first level defines the different modules:

```
input:
    # Here settings related with input module
    segment: # It defines a description of the input segment
        x: 2 # It defines the maximum size of x dimension of the segment to 2.
output:
    # Settings linked with the output
    # ...
retrieval:
    # Definition of inversion strategy
    # ...
...
```

When GRASP is called, the first step is to read settings and the second is to prepare the environment for the settings defined in the main structures. If the settings are not valid, an explicit error message will be printed. Please read the first line of it carefully to understand the error.

The settings parameters can also be defined by the command line. In this case, after the first argument (settings file name), extra settings parameters can be defined with the syntax key=value where key is the parameter name in "dot syntax". For example, in GRASP, it is equivalent to be called with the argument `input.driver=sdata` or to have defined in the settings file the following content:

```
input:
    driver: sdata
```

It is important to define clearly how the relative paths have to be defined in the settings files. Relative paths are always relative to the file that defines it. In the next section, the reader will learn about how to include other settings file inside of a settings file, but this rule will stay valid: Relative paths are defined from settings file that define it. In the case of usage of command line, relative paths are relative to the current working directory. In the case of absolute paths, all this complexity disappears but the results are less portables.

4.1.1.1. HELP argument

The list of available parameters for GRASP is long. The "help" argument will help you to know the available parameters or to look for something specific. When **help** argument is present, GRASP is not executed normally, but instead, the help information will appear in the screen print. Additionally, help can be followed by a search string to filter the results. For example, **help=input** will print only the settings which contain "input" string in its definition and **help=input.segment** will print only the settings which will be under the block "segment" inside of the block "input" (because "dot" symbol defines block separator).

Table 4.1. List of GRASP options

Field Name	Field Content
help	Shows this help information
version	Shows GRASP code version information and stops
resources_path	Path to framework resources folder
retrieval. general. path_to_internal_files	Path to internal data files
retrieval. mode	Define how the code is run. Valid values 'inversion' or 'forward' (inversion=full inversion; forward=only forward model)
retrieval. inversion. convergence. minimization_convention	Minimization in absolute or logarithm space
retrieval. inversion. convergence. threshold_for_stopping_Q_iterations	Threshold for stopping q - iterations: once the change in the residual smaller than this parameter the iterations are stopped
retrieval. inversion. convergence. scale_for_finite_difference	Defines DL for calculations of derivatives: (f(x+DL)-f(x))/DL
retrieval. inversion. convergence. threshold_for_stopping	Threshold for stopping p - iterations: once the change in the residual smaller than this parameter the iterations stops
retrieval. inversion. convergence. normal_system_solver	Defines the method to solve normal system
retrieval. inversion. convergence. maximum_iterations_of_Levenberg-Marquardt	The maximum number of ip iterations where Levenberg-Marquardt correction is applied
retrieval. inversion. convergence. maximum_iterations_for_stopping	Maximum number of iterations performed in the retrieval before stop
retrieval. inversion. convergence. shift_for_applying_logarithm_to_negative_values	This value (usually 1) is added to negative parameters in order to be able to apply logarithmic transformations
retrieval. inversion. regime_of_multipixel_constraints. inversion_regime	Flag for single multi-pixel retrieval (VALUES)
retrieval. inversion. regime_of_measurement_fitting. polarization	1. -We fit: I, Q, U; 2. - We fit: I, Q/I, U/I; 3. - We fit: I, (Q^2+U^2)^(1/2) (Polarized measurements can be defined in input data as Q and U or P); 4- We fit: I, [(Q^2+U^2)^(1/2)]/I (Polarized measurements can be defined in input data as Q and U or P); 5- We fit: P/I (User has to provide P/I in input data)
retrieval. inversion. regime_of_measurement_fitting. scattering_angle_for_normalized_p11	For P11 retrieval. If P11 is given as absolute value, this option has to be -180. 0 (default value). If P11 is relative, indicate with this option the value of the angle that has to be used to divide all p11 values.
retrieval. inversion. noises. noise[]. standard_deviation_synthetic	Standard deviation of synthetic random noise added to the corresponding inverted data, if value is 0 no synthetic random noise is added
retrieval. inversion. noises. noise[]. error_type	Error type used for definition of covariance matrices used in measurement fitting and in modeling synthetic noise
retrieval. inversion. noises. noise[]. standard_deviation	Standard deviation of the random noise expected (this defines the covariance matrix used in fitting)
retrieval. inversion. noises. noise[]. measurement_type[]. type	Type of relevant measurements for applying the noise assumptions
retrieval. inversion. noises. noise[]. measurement_type[]. index_of_wavelength_involved	List of indices of relevant wavelengths for applying the noise assumptions
retrieval. forward_model. phase_matrix. size_binning_method_for_triangle_bins	Determining the scale used for the binning of size distribution
retrieval. forward_model. phase_matrix. number_of_elements	Number of phase matrix elements used in the calculations and retrieval
retrieval. forward_model. phase_matrix. kernels_folder	Path to kernels when we retrieve size distribution for triangle bins or lognormal size distribution
retrieval. forward_model. phase_matrix. radius. mode[]. bins	Number of triangle bins set for size distribution representation
retrieval. forward_model. phase_matrix. radius. mode[]. min	Minimum value of the radius used in the triangle bins
retrieval. forward_model. phase_matrix. radius. mode[]. max	Maximum value of the radius used in the triangle bins
retrieval. forward_model. phase_matrix. ratio. mode[]. value	Values of bins using in retrieved aspect or axis distributions

Field Name	Field Content
retrieval. forward_model. radiative_transfer. number_of_layers	Maximum number of vertical layers resolved in radiative transfer (minimum and maximum value. If only one number is assigned maximum is by default KNT-1)
retrieval. forward_model. radiative_transfer. molecular_profile_vertical_type	It defines which model will be used to describe vertical profile of molecular (Rayleigh) scattering. Values: 'Exponential' describes molecular density profile at altitude h as exp(h/8)/8, 'Standard_atmosphere' uses standard atmosphere model (pressure and temperature profiles) to calculate molecular density at each altitude.
retrieval. forward_model. radiative_transfer. aerosol_profile_vertical_type	It defines which model will be used to describe vertical profile of aerosol distribution. Values: 'Exponential' describes aerosol concentration profile at altitude h as exp(h/HM)/HM, 'gaussian' uses normal distribution exp(h-HM)/ (sqrt(rpi)*sqrt(2.)*sigma) with sigma and HM parameter taken from parameters.
retrieval. forward_model. radiative_transfer. phase_matrix_truncation	Switch on/off the truncation: the technique to calculate scattering effects from the sharp forward peak of the phase function separately from those of the rest of the phase function. It doesn't effect considerably the accuracy while provides much faster calculations
retrieval. forward_model. radiative_transfer. absolute_error_rt_calculations	Absolute value of truncation threshold of Fourier and order-of-scattering series expansions in radiative transfer calculations
retrieval. forward_model. radiative_transfer. reference_plane_for_polarization	Reference plane for polarization calculations
retrieval. forward_model. radiative_transfer. BOA_reflectance	Perform atmospheric correction calculation of surface reflectance and surface BRDF at the Bottom-Of-Atmosphere
retrieval. forward_model. radiative_transfer. rt_kernels. mode	LUT Mode for radiative transfer (disable by default)
retrieval. forward_model. radiative_transfer. rt_kernels. folder	Folder for aerosol look-up-table
retrieval. forward_model. radiative_transfer. simulating_observation. order_of_scattering	Regime of scattering used for modeling diffuse radiation observations
retrieval. forward_model. radiative_transfer. simulating_observation. number_of_gaussian_quadratures_for_expansion_coefficients	Number of Gaussian quadratures for calculating expansion coefficients used for multiple scattering simulations
retrieval. forward_model. radiative_transfer. simulating_observation. number_of_guassian_quadratures_for_fourier_expansion_coefficients	Number of Gaussian quadratures for calculating Fourier expansion coefficients used for multiple scattering simulations
retrieval. forward_model. radiative_transfer. simulating_observation. number_of_fourier_expansion_coefficients	Number of Fourier expansion coefficients used in multiple scattering simulations
retrieval. forward_model. radiative_transfer. simulating_derivatives. order_of_scattering	Regime of scattering used in calculation of radiance derivatives
retrieval. forward_model. radiative_transfer. simulating_derivatives. number_of_gaussian_quadratures_for_expansion_coefficients	Number of Gaussian quadratures for calculating expansion coefficients used in calculation of radiance derivatives
retrieval. forward_model. radiative_transfer. simulating_derivatives. number_of_guassian_quadratures_for_fourier_expansion_coefficients	Number of Gaussian quadratures used for calculating Fourier expansion coefficients used in calculation of radiance derivatives
retrieval. forward_model. radiative_transfer. simulating_derivatives. number_of_fourier_expansion_coefficients	Number of Fourier expansion coefficients used in calculation of radiance derivatives
retrieval. product_configuration. wavelength_indices_for_angstrom	Indices of wavelengths which will be used to calculate Angstrom exponent
retrieval. product_configuration. aerosol_particulate_matter_diameter	Diameters of aerosol particles in microns which will be used to calculate Particulate Matter (PM)
retrieval. product_configuration. wavelength_indices_for_ndvi	Indices of wavelengths which will be used to calculate NDVI if it is calculated
retrieval. product_configuration. wavelenght_indices_for_aod_error_estimation	Indices of wavelengths which will be used to estimate aod error
retrieval. product_configuration. wavelenght_indices_for_ssa_error_estimation	Indices of wavelengths which will be used to estimate ssa error
retrieval. product_configuration. wavelenght_indices_for_lidar_error_estimation	Indices of wavelengths which will be used to estimate lidar error
retrieval. products. aerosol. chemistry	Retrieve aerosol chemical composition (if retrieved)
retrieval. products. aerosol. lidar	Retrieve columnar lidar ratios (e. g. , if lidar data are inverted)
retrieval. products. aerosol. optical_properties	Provide aerosol optical properties
retrieval. products. aerosol. phase_matrix	Obtain aerosol phase matrix
retrieval. products. aerosol. refractive_index	Provide aerosol refractive index
retrieval. products. aerosol. theoretical_bimodal_extinction	Provide estimated aerosol extinction for fine and coarse modes

Field Name	Field Content
retrieval. products. aerosol. theoretical_bimodal_parameters	Provide estimated aerosol microphysical characteristics for fine and coarse modes
retrieval. products. aerosol. particulate_matter	Obtain aerosol particulate matter estimation at given particle diameters
retrieval. products. aerosol. type	Obtain aerosol type index i. e. 0-Complex mixture,1-Background,2-Maritime,3-Urbn. Polluted,4-Mixed,5-Urbn. clean,6-Smoke flam. ,7-Smoke. sold. ,8-Dust
retrieval. products. error_estimation. aerosol. lidar	Implement error estimation for aerosol lidar products
retrieval. products. error_estimation. aerosol. optical_properties	Implement error estimation for optical properties of aerosol products
retrieval. products. error_estimation. parameters	Implement error estimation for retrieved parameters
retrieval. products. forcing. broadband_flux	Provide upward and downward fluxes integrated over the solar spectrum at the user-defined levels for the scenarios with and without aerosols
retrieval. products. forcing. forcing	Provide aerosol radiative forcing values at the user-defined levels
retrieval. products. retrieval. fitting	Provide obtained measurement fitting
retrieval. products. retrieval. parameters	Provide retrieved parameters
retrieval. products. retrieval. residual	Provide values of obtained residuals
retrieval. products. surface	Provide surface reflectance products
retrieval. edges_size. x	Size of edges width in pixels (it have to be lower than KIEDGE compilation constant)
retrieval. edges_size. y	Size of edges height in pixels (it have to be lower than KIEDGE compilation constant)
retrieval. edges_size. t	Size of temporal dimension of edges (it have to be lower than KIEDGE compilation constant)
retrieval. debug. verbose	Retrieval prints progress information while it is performing the inversion
retrieval. debug. additional_information	Print some additional information
retrieval. debug. simulated_sdata_file	Filename where simulated observation data are to be printed. This option force retrieval. general. stop_before_performing_retrieval=true
retrieval. debug. path_to_extra_files	Path to folder with retrieval extra files: debug information, extra resources like image. dat files. . .
retrieval. debug. use_internal_initial_guess	Test option which allows to load different initial guess for each pixel (loading them from image. dat files)
retrieval. constraints. characteristic[]. type	Type of characteristic
retrieval. constraints. characteristic[]. retrieved	Specify if this characteristic will be retrieved or only used in forward model
retrieval. constraints. characteristic[]. mode[]. initial_guess. value	Initial values for a specific (determined by type) characteristic
retrieval. constraints. characteristic[]. mode[]. initial_guess. min	Minimum value for the specific characteristic
retrieval. constraints. characteristic[]. mode[]. initial_guess. max	Maximum value for the specific characteristic
retrieval. constraints. characteristic[]. mode[]. initial_guess. index_of_wavelength_involved	Indices of Wavelengths associated to the specific characteristic
retrieval. constraints. characteristic[]. mode[]. initial_guess. estimate_error	Flag to retrieve the error of specific parameter if retrieval. products. error_estimation. parameters is true
retrieval. constraints. characteristic[]. mode[]. single_pixel. a_priori_estimates. lagrange_multiplier	Value of the Lagrange multiplier associated to a priori estimate of the retrieved characteristics (applied in each single pixel)
retrieval. constraints. characteristic[]. mode[]. single_pixel. smoothness_constraints. difference_order	Order of the derivatives/differences used for applying a priori smoothness constrains for the retrieved characteristics
retrieval. constraints. characteristic[]. mode[]. single_pixel. smoothness_constraints. lagrange_multiplier	Value of the Lagrange multiplayer used for applying a priori smoothness constraints for the retrieved characteristics
retrieval. constraints. characteristic[]. mode[]. multi_pixel. smoothness_constraints. derivative_order_of_X_variability	Order of derivatives/differences used for applying a priori smoothness constraints on parameter inter-pixel X-variability in multi-pixel retrieval regime
retrieval. constraints. characteristic[]. mode[]. multi_pixel. smoothness_constraints. lagrange_multiplier_of_X_variability	Value of the Lagrange multiplier used for applying a priori smoothness constraints on parameter inter-pixel X-variability in multi-pixel retrieval regime
retrieval. constraints. characteristic[]. mode[]. multi_pixel. smoothness_constraints. derivative_order_of_Y_variability	Order of derivatives/differences used for applying a priori smoothness constraints on parameter inter-pixel Y-variability in multi-pixel retrieval regime
retrieval. constraints. characteristic[]. mode[]. multi_pixel. smoothness_constraints. lagrange_multiplier_of_Y_variability	Value of the Lagrange multiplier used for applying a priori smoothness constraints on parameter inter-pixel Y-variability in multi-pixel retrieval regime
retrieval. constraints. characteristic[]. mode[]. multi_pixel.	Order of derivatives/differences used for applying a priori smoothness constraints on parameter inter-pixel T-variability in multi-pixel retrieval regime

Field Name	Field Content
smoothness_constraints. derivative_order_of_T_variability	
retrieval. constraints. characteristic[]. mode[]. multi_pixel. smoothness_constraints. lagrange_multiplier_of_T_variability	Value of the Lagrange multiplier used for applying a priori smoothness constraints on parameter inter-pixel T-variability in multi-pixel retrieval regime
settings. debug	Shows settings read to run this program
settings. strict	Force GRASP to continue when there were parse or validation errors in the settings file
settings. dump	Stream to dump read settings in short format(experimental)
settings. long_dump	Stream to dump read settings in long format (experimental)
input. driver	The driver that will be called for inverting data
input. file	Name of file(s) which contain input observation
input. center. latitude	Latitude of the center of tile to invert
input. center. longitude	Latitude of the center of tile to invert
input. corner. row	The number of the row for input driver in the native coordinate system of the sensor
input. corner. column	The number of the column for input driver in the native coordinate system of the sensor
input. grid_offset. row	Information of the first row in the input grid that will be use for normalizing the output. If 0 is used the output coordinate reference will be the same than the input. This offset is for forcing the output row to start at 0 (e. g. if you put 1, output_row == input_row - 1)
input. grid_offset. column	Information of the first column in the input grid that will be used for normalizing the output. If 0 is used the output coordinate reference will be the same than the input. This offset is for forcing the output column to start at 0 (e. g. if you put 1, output_column == input_column - 1)
input. area. width	The width of the covered area in pixels. It has to be divisible by intput. segment. x value
input. area. height	The height of the covered area in pixels. It has to be divisible by intput. segment. y value
input. time. from	Initial date and time for data processing
input. time. to	Final date and time for data processing
input. segment. x	Size of segment width in pixels (it have to be lower than KIX compilation constant)
input. segment. y	Size of segment height in pixels (it have to be lower than KIY compilation constant)
input. segment. t	Size of segment temporal dimension (it have to be lower than KITIME compilation constant)
input. transformer	Name of input data transformer functions to be used after load data
input. debug. raw_segment	Stream to print raw segment data loaded
input. debug. clean_segment	Stream to print segment information after clean NaN values
input. debug. used_files	Stream to print names of the files that have the pixels for inverting
input. sdata. dump	Stream where to dump sdata information
input. sdata. dump_original	Stream where to dump sdata information just after being generated by the driver. Some transformers can modify sdata information, this setting is thought for debugging purposes where the user is interested in knowing sdata generated by the driver instead of data driving inside the inversion.
input. imagedat. dump	Stream where to dump initial guess information (image. dat format)
input. preload_segment. x	This parameter specifies how many segments in X dimension will be preloaded in each block
input. preload_segment. y	This parameter specifies how many segments in Y dimension will be preloaded in each block
input. preload_segment. t	This parameter specifies how many segments in T dimension will be preloaded in each block
input. driver_settings. sdata. debug	Print debug information from sdata reader subsystem
input. transformer_settings. segment_imagedat. file	File which contains initial guess information in classic input. dat format
output. segment. function	Driver to process output for every single retrieval (show information in screen, perform a map, plotting, . . .)
output. segment. stream	Stream to dump segment output data
output. iteration. function	Driver to process output for every single retrieval (show information in screen, perform a map, plotting, . . .)
output. iteration. stream	Stream to dump segment output data
output. tile. function	Driver to process output after processing complete tile (show information in screen, perform a map, plotting, . . .)
output. tile. stream	Stream to dump tile output data
output. current. function	Driver to process output after each retrieval (show information in screen, perform a map, plotting, . . .)
output. current. stream	Stream to dump current progress information about the retrieval conducted
output. sdata. simulated_file	Stream where to dump sdata fitting information. Fitting product has to be enabled, otherwise this is ignored. Fitting is dumped after retrieval process.
output. segment. function_settings. csv. delimiter	Separator between fields
output. segment. function_settings. csv. compression	If true output is compressed in GZ format (gz extension is automatically added)
output. segment. function_settings. csv. show_timing	If true 'time per pixel' information is added in the output (default). Hide this information is useful to compare results with diff command
output. tile. function_settings. csv. chemical_concentration	Calculate and print chemical concentration
output. tile. function_settings. csv. delimiter	Separator between fields

Field Name	Field Content
output. tile. function_settings. csv. compression	If true output is compressed in GZ format (gz extension is automatically added)
output. tile. function_settings. csv. show_timing	If true 'time per pixel' information is added in the output (default). Hide this information is useful to compare results with diff command
controller. segment_range	This parameter allows specifying a range of segments that will be inverted/processed. If it is a single number a specific retrieval will be processed. Use -1 as undefined. For example [15, -1] will process all retrievals starting by the segment #15
controller. debug. perform_retrieval	Allowing controller to call retrieval. If this parameter is false the framework will work without inverting the data and it will force to use only none output functions (results will not be printed). This parameter is useful for debug the framework or prepare input data
controller. debug. compilation_information	Print compilation information at the beginning of the process
controller. debug. tracking_memory_stream	Stream where printing all information about memory allocated during the execution (debugging information)
controller. stream	Stream to dump controller information
controller. mpi. maximum_job_time	Maximum number of seconds that the master node will wait for a working node for obtaining results. After this time (specified in seconds) if the job is not finished the controller will kill the task and the segment will be skipped
controller. mpi. polling_time	Number of seconds that the master node wait after checking the workers

4.1.1.2. Extending settings: command line, import and template statements

The first argument of GRASP has to be the settings file but this file can be modified by another mechanism proposed by the settings module. The main way is by the command line, which allows to replace every settings parameter with "dot" syntax (replacing indentation by "dot" symbol and colon symbol by equal). All parameters that have been defined before and being replaced in the command line will cause a "note" information during the execution of GRASP. That sentence is just to inform the user that command line arguments always have higher priority than parameters in settings files. The value from command line will be the value that will be used to run the code. The command line is a very powerful feature to be used in the production scripts.

But the command line is not the only way to modify GRASP settings files. Settings files accept "import" and "template" statement. These statements could look similar but theirs behaviour is a bit different. Both of them allow defining other settings files that are read before the current one, but in case of import, the settings can not be overwritten. The template statement allows loading other settings files and then, modifying some settings to customize the loaded file. It is necessary to take into account that these statements can be used in cascade, creating problems to debug the code. So please use these statements carefully.

4.1.1.3. Streams

Some of the settings parameters are defined as "streams". GRASP output streams allow users to create dynamic names avoiding overwriting files or having to change the filenames each time they execute GRASP. When a description of a parameter is defined as "output stream", the user can set up a regular output path, for example **./folder/file.extension** or use the "magic" behind the output streams by using a wildcard that will be replaced by dynamic values. For instance:

```
output:
    segment:
        function: hdf
        stream: "GRASP_Banizoumbou_20080101_20080331_2x2+3286+1376.hdf"
    tile:
        function: [ ascii, hdf ]
        stream: [ "GRASP_Banizoumbou_20080101_20080331.txt",
                "GRASP_Banizoumbou_20080101_20080331.hdf" ]
```

That definition is ok for many cases but if many tiles or segments are going to be processed, the fixed names will produce name collisions (the content of some files will be overwritten during the process). It would be tedious (and not always possible) for the user to change by himself the dates or other numeric substrings in the file names. For this reason, the configuration system provides some

wildcards that will be automatically replaced with the given values depending on the state of the processing. These wildcards are marked with curly braces and their names are quite self-explanatory. The previous example can be rewritten in a more generic way using the stream wildcards:

```
output:
    segment:
        function: hdf
        stream: "GRASP_Banizoumbou_{tile_from(%Y%m%d)}_{tile_to(%Y%m%d)}_{segment_nx}x{segment_ny}//
        +{segment_corner_column(4)}+{segment_corner_row(4)}.hdf"
    tile:
        function: [ ascii, hdf ]
        stream: [ "GRASP_Banizoumbou_{tile_from(%Y%m%d)}_{tile_to(%Y%m%d)}.txt",
                "GRASP_Banizoumbou_{tile_from(%Y%m%d)}_{tile_to(%Y%m%d)}.hdf" ]
```

In addition, wildcards will provide the user the capability to set some system streams. If you use the values "true", "screen", "t" or "1", the information will be printed in the terminal (stdout). If the stream is set to "false", "none", "null", "f" or 0, nothing will be printed (like redirect to /dev/null). Finally, using the value "stderr" or "-1", the output will be redirected to the standard error output.

The following list shows all available wildcards that can be used for creating dynamic output filenames:

- **auto(N):** itime x icol x irow with N zeros at the left
- **icol(N):** current column number with N zeros at the left
- **irow(N):** current row number with N zeros at the left
- **itime(N):** current time number with N zeros at the left
- **iinversion(N):** current inversion id with N zeros at the left
- **segment_nx(N):** number of X elements per segment with N zeros at the left
- **segment_ny(N):** number of Y elements per segment with N zeros at the left
- **segment_nt(N):** number of T elements per segment with N zeros at the left
- **tile_from(FORMAT):** start tile date in FORMAT. By default FORMAT is %FT%H_%M_%SZ
- **tile_to(FORMAT):** final tile date in FORMAT. By default FORMAT is %FT%H_%M_%SZ
- **tile_corner_column(N):** number of the corner (column) of the segment defined in settings file. Requirement: Input data have to be defined using input.corner instead of input.center
- **tile_corner_row(N):** number of the corner of (row) the segment defined in settings file. Requirement: Input data have to be defined using input.corner instead of input.center
- **tile_center_longitude(FORMAT):** longitude of the center of the tile defined in settings file. Requirement: Input data have to be defined using input.center instead of input.corner
- **tile_center_latitude(FORMAT):** latitude of the center of the tile defined in settings file. Requirement: Input data have to be defined using input.center instead of input.corner
- **tile_coordinate_x(I):** x input reference of center of the tile defined in settings file. It can be defined by corner or latitude. If is N in case it was defined by corner or 0.I in case it was defined like center
- **tile_coordinate_y(I):** y input reference of center of the tile defined in settings file. It can be defined by corner or latitude. If is N in case it was defined by corner or 0.I in case it was defined like center
- **tile_width(N):** Number of X elements in tile with N zeros at the left
- **tile_height(N):** Number of Y elements in tile with N zeros at the left
- **segment_corner_column(N):** number of column of the segment corner with N zeros at the left. Requirement: Input data have to be defined using input.corner instead of input.center

- **segment_corner_row(N):** number of row of the segment corner with N zeros at the left. Requirement: Input data have to be defined using input.corner instead of input.center
- **segment_first_date(FORMAT):** date of first pixel inside the segment in FORMAT. By default FORMAT is %FT%H_%M_%SZ
- **segment_last_date(FORMAT):** date of last pixel inside the segment in FORMAT. By default FORMAT is %FT%H_%M_%SZ
- **iteration(N):** Number of iterations with N zeros at the left. (Note. In case of single pixel it returns the number of iterations of first pixel)
- **settings_filename:** the name of settings file used to run the retrieval
- **version:** version of grasp if it is compiled with saving this information
- **branch:** git branch of grasp if it is compiled with saving this information
- **commit:** reference of git commit of grasp if it is compiled with saving this information
- **constants_set:** constants set used in compilation time
- **pwd:** this is replaced by current folder and it is only valid at the beginning of the stream definition
- **yml:** this is replaced by current folder of main configuration file and it is only valid at the beginning of the stream definition

4.1.2. Retrieved characteristics

The ensemble of characteristics available to be retrieved or simulated by GRASP is open to user selection in the settings file inside *retrieval.constraints.characteristic* section. An example of the general structure of them is showed below:

```
retrieval:
    constraints:
        characteristic[1]:
            type: characteristic_name
            retrieved: true
            mode[1]:
                initial_guess:
                    value:                          [0.0, 0.0, 0.0, 0.0, ...]
                    min:                            [0.0, 0.0, 0.0, 0.0, ...]
                    max:                            [0.0, 0.0, 0.0, 0.0, ...]
                    index_of_wavelength_involved:   [0.0, 0.0, 0.0, 0.0, ...]
                single_pixel:
                    smoothness_constraints:
                        difference_order: 0.0
                        lagrange_multiplier: 0.0
                multi_pixel:
                    smoothness_constraints:
                        derivative_order_of_X_variability:    0.0
                        lagrange_multiplier_of_X_variability: 0.0
                        derivative_order_of_Y_variability:    0.0
                        lagrange_multiplier_of_Y_variability: 0.0
                        derivative_order_of_T_variability:    0.0
                        lagrange_multiplier_of_T_variability: 0.0

            mode[2]:

                ...

            mode[3]:

                ...

            ...
```

The number assigned at each characteristic has no relevance at all while coherence is maintained. The *retrieved* field can be set to 'true' if the corresponding characteristic is going to be a retrieved parameter in the inversion; or to 'false' if it is just a fixed value for the forward simulation. The number of modes included in each characteristic depends on the nature of each one. For characteristics representing optical or mycrophisical aerosol properties (size distribution or refractive index for example), the number of modes corresponds to the number of aerosol modes selected for the retrieval/simulation (tipically one or two if fine/coarse distinction is made). For characteristics representing surface properties three modes will be needed to describe the associated model; except for polarization that only one is nedeed. The fields present in *initial_guess* part contain one element for each wavelength following the structure provided in the SDATA file in the case of optical wavelength dependent characteristics; one element for each bin for size distribution related characteristics; and one element for other mycrophysical magnitudes or non-wavelength dependent optical characteristics. In the former case, *index_of_wavelength_involved* should be filled with zeros for all the corresponding bins. The rest of the elements included in *single_pixel* or *multi_pixel* parts are always formed by one single element.

The list of available characteristics can be found below:

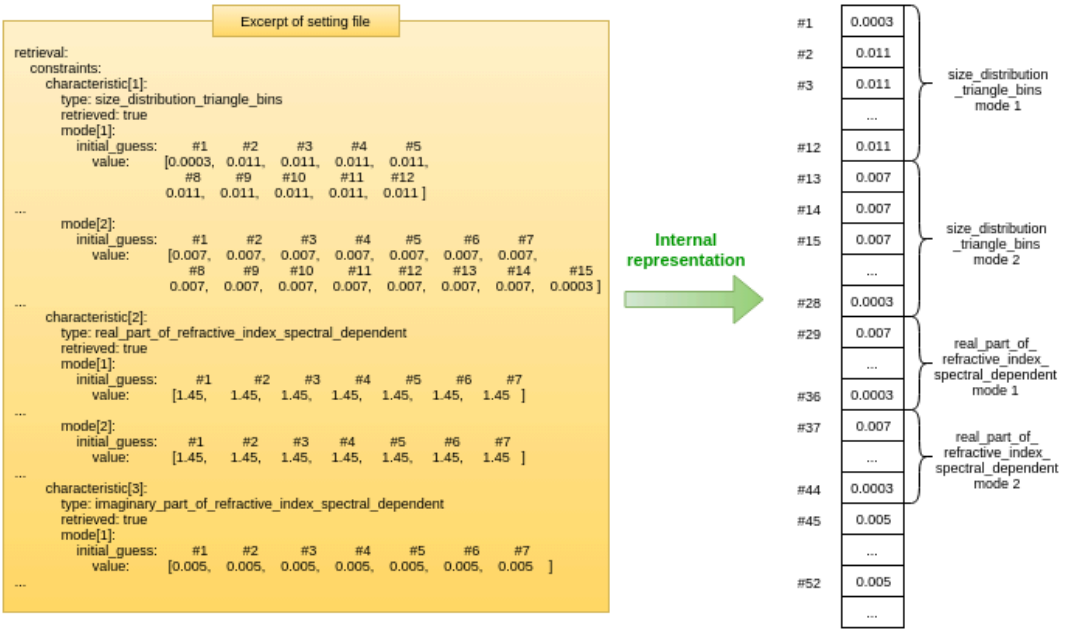
Table 4.2. Available GRASP characteristics

Characteristic Name	Description
size_distribution_triangle_bins	Normalized Size Distribution dV / dlnr at "triangle" bins
size_distribution_precalculated_lognormal	Normalized Size Distribution dV / dlnr for precalculated lognormal bins
size_distribution_lognormal	Parameters of bi - modal Lognormal Size Distribution dV / dlnr
aerosol_model_concentration	Aerosol model concentration
real_part_of_refractive_index_spectral_dependent	Spectral dependent Real part of complex refractive index
real_part_of_refractive_index_constant	Complex Refractive Index Real part is spectrally constant
particle_component_volume_fractions_linear_mixture	Real part of complex refractive index is mixture
particle_component_fractions_chemical_mixture	Chemistry, fraction of: water, fslbl, finslbl, soot, iron
imaginary_part_of_refractive_index_spectral_dependent	Spectral dependent Imaginary part of complex refractive index
imaginary_part_of_refractive_index_constant	Complex Refractive Index Imaginary part is spectrally constant
sphere_fraction	Fraction of spherical particles
aspect_ratio_distribution	Axis Ratio Distribution
vertical_profile_parameter_height	Scaling factor in case of exponential profile, mean height in case of gaussian distribution
vertical_profile_normalized	Aerosol normalized vertical profile
aerosol_concentration	Aerosol concentration
lidar_calibration_coefficient	Calibration coefficient for lidar
vertical_profile_parameter_standard_deviation	Standard deviation for aerosol vertical profile
surface_land_brdf_ross_li	BRDF Land normalized parameters according to Ross and Li model
surface_land_brdf_rpv	BRDF normalized parameters according to RPV model
surface_land_litvinov	BRDF Land normalized parameters according to Litvinov model
surface_land_litvinov_fast	BRDF Land normalized parameters according to Litvinov fast model
surface_land_polarized_maignan_breon	BRDF Land normalized parameters according to Maignan and Breon model
surface_land_polarized_litvinov	BPDF Land normalized parameters according to Litvinov model
surface_water_cox_munk_iso	BRDF Water normalized parameters according to Cox and Munk model
surface_water_cox_munk_ani	BRDF normalized parameters according to Maignan and Breon model
surface_water_litvinov	BRDF Water normalized parameters according to Litvinov model

4.1.2.1. Initial guess through the algorithm

Understanding GRASP inversion procedure means understanding how the algorithm is starting from an initial guess and obtains a results array. This is an iterative procedure explained in the literature and introduced in Section 2.3, “GRASP Scientific Core algorithm”. The purpose of this section is to explain how initial guess is represented inside the code as an array which evolves in each iteration until getting the result array. The following diagram shows how initial guess is read from settings file and translated into an internal array in the code. This detail could look very technical and related with the development but understanding of some internal concepts of the code helps to understand how it works. The following diagram shows this transformation:

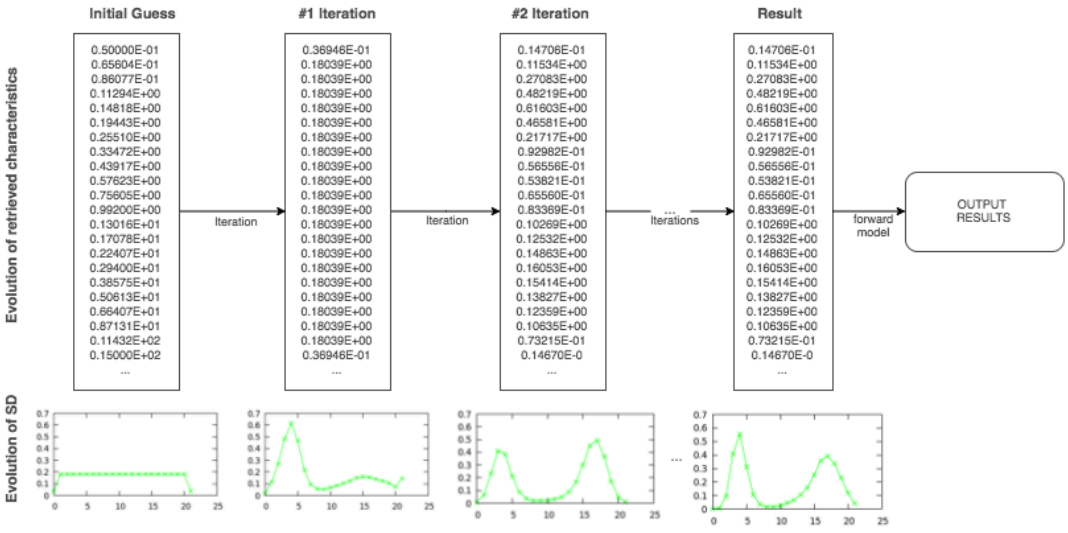
Figure 4.1. Translation of settings file into initial guess array



Since the initial guess is defined in the settings, it is defined once for the entire segment (one should remember that GRASP can work in a multi-pixel approach). Sometimes, it is desirable to have a different initial guess for each pixel. In that case the already described mechanisms to establish the initial guess are not enough, and GRASP proposes another useful tool to modify the initial guess of each segment. This information is considered as part of the input because it contains pixel dependent information and will be defined in detail in the Section 4.2.3, “Input information for characteristics”.

Once the initial guess is loaded, it is set as first array of characteristics to be retrieved. Retrieval process iterates over it until it gets the results. The results of GRASP retrieval is the array with the same shape as the initial guess, but containing the results of retrieval of these parameters to match them with SDATA file. Then, the GRASP forward model is called using this results array, GRASP obtains the rest of the derived results it provides. So, we can talk about two kinds of results: basic results and derived products. The following diagram shows this process:

Figure 4.2. Evolution of retrieved characteristics during GRASP processing



To know all the details about the products obtained by GRASP, please see Section 4.3.1, “The list of GRASP output parameters”.

4.1.3. Noise simulation

In order to harmonize all possibilities to add random noise to the measurements that are going to be retrieved in GRASP a new setting called *add_random_noise* in the *inversion.noises* group has been created. There are three possible values that this setting can take: *disable*, *measurement_fitting* (default) and *sdata*. These options will produce a different effect depending on which retrieval mode (forward or inversion) has been selected. Here it is a small description of the meaning of the options:

- *disable*: If this option is selected there is no addition of any noise even if *standard_deviation_synthetic* greater than 0. All the rest of the settings related to it will be ignored.
- *measurement_fitting*: This is the default value and this option maintains the same behaviour of GRASP random noise as before: for both *forward* and *inversion* retrieval modes random noise is added to FS measurement vector to be fitted.
  - If the *forward* option is selected in *retrieval.mode* any noise addition will be made.
  - If *inversion* is selected, the noise is added in the FS vector and it is totally transparent to the user. There is no possibility to see what are the noisy measurements that are really being used in the retrieval. This option will always work no matter if the measurements in the sdata do really correspond or not with the ones that are being inverted (this is especially referred to the polarization measurements).
- *sdata*: the sdata mode is used to enable the user to see what are the noises added to the measurements. This option will only work if the measurements in the sdata and the measurements that are really retrieved are exactly the same. In other case, meaning that the code is fitting different measurements than those in the input data, an error will arise and the code will not be executed.
  - If the *forward* option is selected in retrieval.mode, the added noise will be done over the FS vector. It will be seen in the column *fit_X* of the output file, or if *output.sdata.dump* setting is present in the yml file.
  - If *inversion* is selected, the noise will be added to the original sdata provided by the user independently of the forward model configuration. The noise measurements will be available for the user in the *meas_X* column of output file or via *input.sdata.dump*.

```
mode: inversion

   inversion:
       regime: single_pixel

       convergence:
           minimization_convention: logarithm
           maximum_iterations_for_stopping: 35
           maximum_iterations_of_Levenberg-Marquardt: 35
           threshold_for_stopping: 1.0e-3
           threshold_for_stopping_Q_iterations: 1e-12
           scale_for_finite_difference: 1.0e-5
           normal_system_solver: sparse_matrix_solver

       measurement_fitting:
           polarization: degree_of_polarization

       noises:
           add_random_noise: measurement_fitting # | disable | sdata
           noise[1]:
               standard_deviation_synthetic: 0.1
               error_type:  relative
               standard_deviation:  0.05
               measurement_type[1]:
                   type: I
                   index_of_wavelength_involved: [ 1, 2, 3, 4 ]
           noise[2]:
```

```
                    standard_deviation_synthetic: 0.01
                    error_type:  absolute
                    standard_deviation:  0.005
                    measurement_type[1]:
                        type: tod
                        index_of_wavelength_involved: [ 1, 2, 3, 4 ]
...
```

4.2. Input module

The input module is responsible for reading the data and for setting the internal input data structures. GRASP provides a very flexible way to inject the input data offering developers the capability to create the input drivers. An input driver is an extension of GRASP that is added during compilation and is selected in the settings file. These generic drivers allow the developers to create a custom way to read a specific database and load it into the GRASP scientific algorithm. Therefore, GRASP can read infinite kinds of input databases, as much as drivers exist. Additionally, drivers can take the responsibility of performing some pre-processing actions (such as calibration corrections) before retrieving the data. Finally, when GRASP is used for massive data processing, drivers are extremely important since they help to prepare input data and load it into scientific module without the use of any intermediate files, everything is performed in memory. For general use, GRASP proposes a generic driver called the SData driver, which reads files in SData format (Sensor-Data format). These files are not standard, they have a specific format proposed by GRASP to start working with the code. This format is described in Section 4.2.1, “The SDATA format”.

Transformers are the second kind of extensions that the input module offers to users (and developers). They are called after getting input data for a segment, it means after calling the driver. The purpose of transformers is to add the capability to modify the segment after obtaining the data. An example of transformation is to load a climatology database and to modify the initial guess of each pixel to optimize the number of iterations needed to retrieve the data. Performing this action after getting the data allows reusing it between different drivers. Default installation of GRASP does not offer any transformers. All of them are considered as optional extensions and have to be externally added manually or using the grasp-manager.

4.2.1. The SDATA format

The SDATA (sensor data) format is the original input data format of the GRASP code. It is a simple text format designed by the science team at the early stages of the development of the scientific code and it's the easiest way to create test data.

In the context of the GRASP project, the SDATA format has a number of pros:

- Designed by the scientific team at the origin of the GRASP project, it is well adapted to its needs.
- It is very simple to describe.
- It is a text format, and therefore portable, quite easy to check (for the accustomed eye!) and to edit and to make some quick experiments.
- It is a piece of cake to read in Fortran :-)

It has also a number of cons:

- It is not standard (a by-product of the "not designed by a committee" approach). No off-the-shelf library is available to parse it and validate it (meaning outside of the GRASP project).
- It lacks of flexibility: the order and the number of values are fixed for one version and any change in the format to meet new requirements is likely to break the compatibility with the former versions.
- It is fragile: a malformed file may easily make the code crash or produce mysterious bugs. The design of the format, while simple, makes it hard to develop a really reliable validator.

- Comments are expected only after values, on the same line (after a colon sign)
- While the format uses a text representation for the data, it contains lots of numeric values, with limited accuracy and with no comment. Large files are tedious to read and it's easy to make shift errors while reading and editing even for the experienced user.
- Being a text format, it becomes very inefficient for large data volumes. While it can be compressed for archiving, it must be uncompressed for processing, and only sequential access is possible. It is still possible to perform regional processings (several dozens of thousands of pixels that cover more than a few hundreds of kilometres in both directions) with this format (it was actually done for the sake of necessity), but it stresses the computing system a lot and can't be scaled up to the global processing.

Whatever the number and seriousness of the cons, one of the design objectives is to keep the code simple and flexible allowing the scientific community to play with the code. It does not make sense to implement a driver for a single user who wants to do some tests with GRASP. That's why this easy format is maintained by the developer team.

In the following description, the elements in `fixed-width font` are the snippets of content. The numeric values in these snippets (e.g. in `2 2 2 : NX NY NT`) are given only as examples.

Figure 4.3. An example of SDATA file

```
SDATA version 2.0
2   2   2  : NX NY NT

4  2008-01-04T13:15:00Z 70000.0  0  0 : NPIXELS TIMESTAMP HOBS_km NSURF IFGAS
1  1  1  3286  1377   2.599 13.528   252.0  100.0  6  0.443  0.490  0.565 ...
2  1  1  3287  1377   2.657 13.528   242.0  100.0  6  0.443  0.490  0.565 ...
1  2  1  3286  1376   2.601 13.583   241.0  100.0  6  0.443  0.490  0.565 ...
2  2  1  3287  1376   2.658 13.583   239.0  100.0  6  0.443  0.490  0.565 ...


4  2008-01-06T13:02:41Z 70000.0  0  0 : NPIXELS TIMESTAMP HOBS_km NSURF IFGAS
1  1  1  3286  1377   2.599 13.528   252.0  100.0  6  0.443  0.490  0.565 ...
2  1  1  3287  1377   2.657 13.528   242.0  100.0  6  0.443  0.490  0.565 ...
1  2  1  3286  1376   2.601 13.583   241.0  100.0  6  0.443  0.490  0.565 ...
2  2  1  3287  1376   2.658 13.583   239.0  100.0  6  0.443  0.490  0.565 ...
```

SDATA files have a simple structure:

1. The first line is the FILE HEADER. It contains a magic identifier `SDATA` followed by a version number.
2. The second line is the SEGMENT HEADER. It contains three numbers, `NX`, `NY` and `NT`, the spatial and temporal dimensions of the segment that this SDATA file represents. You can notice a colon and names of fields after the values. This is the way how comments are written in the SDATA files. Everything starting from the colon will be ignored by the SDATA parser.
3. An empty line follows the SEGMENT HEADER.
4. Then comes the first CELL (group of neighbouring pixels) of the SEGMENT. Each CELL has at most NX*NY pixels (but it may have less, for various reasons: cloudy pixels that have been filtered, missing pixels, etc.). The number of CELLs in the SEGMENT is given by the NT number provided in the SEGMENT HEADER.

Table 4.3. The SDATA main structure

Field Name	Field Content
FILE HEADER	`SDATA version 2.0`

Field Name	Field Content
SEGMENT HEADER	`2 2 2 : NX NY NT`
empty line	
CELL 1	cell content, look for CELL structure
empty line	
CELL 2	cell content, look for CELL structure
empty line	
...	
CELL it	cell content, look for CELL structure
empty line	
...	
CELL `NT`	cell content, look for CELL structure
empty line	

A CELL is a set of neighbouring pixels, that form the base of a SEGMENT. Each CELL has a HEADER and a number of PIXELs, supposedly acquired at the same time.

The CELL HEADER contains:

1. The number of pixels in the CELL (`NPIXELS`). It may not be larger than `NX*NY`

2. The timestamp of acquisition of the pixels, in the ISO8601 [http://en.wikipedia.org/wiki/ISO_8601] time format.

3. A "height" of observation, in metres. The value here is a bit weird (`70000`), and doesn't correspond to the satellite altitude (that is at least 10 times larger). Actually, the value doesn't really matter as long as it is large. Historically, the scientific team has used this value of `70000` in many SDATA files.

4. Two values for the number of surface and gas parameters. These two values are currently not documented and can be set to 0 for the moment.

5. Comments starting with a colon.

Table 4.4. The CELL structure

Field Name	Field Content
CELL HEADER	`4 2008-01-04T13:15:00Z 70000.0 0 0 : NPIXELS ...` `... TIMESTAMP HOBS NSURF IFGAS`
PIXEL 1	a line of values, look for PIXEL structure
PIXEL 2	a line of values, look for PIXEL structure
...	
PIXEL `NPIXELS`	a line of values, look for PIXEL structure

Each line of data after the CELL HEADER represents exactly one pixel, with all its fields. The Table 4.5, “The PIXEL structure” describes the order and type of these fields. For the types, the Fortran notation is used: array types are described with the dimensions of arrays between parentheses, and the ordering is such that the first index increases faster. Indices start from 1, not from 0 like in C. For instance, when one reads `real(nwl)` for wavelengths, that means that one has to read a list of `nwl` real values that represent wavelengths.

Table 4.5. The PIXEL structure

Field Type	Variable Name (in source code)	Field Content
integer	pixel[`ipix`].ix	coordinate x in the current cell, starting at 1 (in the direction EW)
integer	pixel[`ipix`].iy	coordinate y in the current cell, starting at 1 (in the direction NS)
integer	pixel[`ipix`].icloudy	cloud flag: 0 = cloud, 1 = clear [a]
integer	pixel[`ipix`].icol	column of the pixel in its original grid or database (can be set to 0 when not relevant)[b]
integer	pixel[`ipix`].irow	line of the pixel in its original grid or database (can be set to 0 when not relevant)[b]

Field Type	Variable Name (in source code)	Field Content
real	pixel[ipix].x	longitude of the pixel, in decimal degrees, in the range [-180..180] 0: Greenwich meridian, east of Greenwich: positive, west of Greenwich: negative
real	pixel[ipix].y	latitude of the pixel, in decimal degrees, in the range [-90..90]
real	pixel[ipix].MASL	altitude of the ground, in metres (MASL: metres above sea level)
real	pixel[ipix].land_percent	percentage of land, in the range [0 (sea) .. 100 (land)]. Intermediate values correspond to coastal pixels
integer	pixel[ipix].nwl	number of available wavelengths (nwl)
real(nwl)	pixel[ipix].meas[nwl].wl	list of wavelengths, in micrometers
integer(nwl)	pixel[ipix].meas[nwl].nip	number of types of measurements for each wavelength (nip)
integer(nip, nwl)	pixel[ipix].meas[nwl].meas_type[nip]	list of types of measurements meas_type (see Table 4.6, "Types of measurements") The ordering is as follows: meas_type(1, wl_1) meas_type(2, wl_1) ... meas_type(nip, wl_1) meas_type(1, wl_2) ... meas_type(nip, wl_n)
integer(nip, nwl)	pixel[ipix].meas[nwl].nbvm[nip]	number of valid measurements (nbvm), for each type of measurement and for each wavelength The ordering is as follows: nbvm(1, wl_1) nbvm(2, wl_1) ... nbvm(nip, wl_1) nbvm(1, wl_2) ... meas_type(nip, wl_n)
real(nwl)	pixel[ipix].meas[nwl].sza	solar/sounding zenith angle (sza or θ_s) in decimal degrees ([0..90]), for each wavelength
real(nbvm, nip, nwl)	pixel[ipix].meas[nwl].thetav[nip][nbvm]	viewing zenith angle (vza or θ_v) in decimal degrees ([0..90] for PARASOL, TBD for PHOTOMETERS) The ordering is as follows: θ_v(1, 1, wl_1) vza(2, 1, wl_1) ... θ_v(nbvm, 1, wl_1) θ_v(1, 2, wl_1) ... θ_v(nbvm, nip, wl_n) In case of lidar or vertical observations this field contains altitudes or ranges of the observations in meters.
real(nbvm, nip, nwl)	pixel[ipix].meas[nwl].phi[nip][nbvm]	relative azimuth angle (raa or $\Delta\varphi$) in decimal degrees ([-180..180] or [0..360]) The ordering is as follows: $\Delta\varphi$(1, 1, wl_1) $\Delta\varphi$(2, 1, wl_1) ... $\Delta\varphi$(nbvm, 1, wl_1) $\Delta\varphi$(1, 2, wl_1) ... $\Delta\varphi$(nbvm, nip, wl_n)
real(nbvm, nip, nwl)	pixel[ipix].meas[nwl] .tau[nbvm];...;.P[nbvm]	measurements (depending on meas_type), for each wavelength The ordering is as follows: meas(1, 1, wl_1) meas(2, 1, wl_1) ... meas(nbvm, 1, wl_1) meas(1, 2, wl_1) ... meas(nbvm, nip, wl_n) Where meas can be: tau ... P
real(nsurf, nwl)	pixel[ipix].meas[nwl].groundpar[nsurf]	ground parameters This part can be ignored for now. The ordering is as follows: groundpar(1, wl_1) groundpar(2, wl_1) ... groundpar(nsurf, wl_1) groundpar(1, wl_2) ... groundpar(nsurf, wl_n)
real(nwl)	pixel[ipix].meas[nwl].gaspar	gas absorption (tau gases) or molecular depolarization ratio. This parameter has to be provided only if the setting IFGAS (in the CELL HEADER) is set to 1. The ordering is as follows: gaspar(wl_1) gaspar(wl_2) ... gaspar(wl_n)
integer(nip, nwl)	pixel[ipix].meas[nwl].ifcov[nip]	ifcov (1 if a covariance matrix is available, 0 otherwise) The ordering is as follows: ifcov(1, wl_1) ifcov(2, wl_1) ... ifcov(nip, wl_1) ifcov(1, wl_2) ... ifcov(nip, wl_n)
real(nbvm, nip, nwl)	pixel[ipix].meas[nwl].cmtrx[nip][nbvm]	cmtrx (diagonal of covariance matrix, also known as Ω). These values have to be skipped if ifcov=0 The ordering is as follows: cmtrx(1, 1, wl_1) cmtrx(2, 1, wl_1) ... cmtrx(nbvm[c], 1, wl_1) cmtrx(1, 2, wl_2) ... cmtrx(nbvm[c], nip, wl_n)
integer(nip, nwl)	pixel[ipix].meas[nwl].ifmp[nip]	ifmp (1 if a vertical profile (mprof) is available, 0 otherwise) The ordering is as follows: ifmp(1, wl_1) ifmp(2, wl_1) ... ifmp(nip, wl_1) ifmp(1, wl_2) ... ifmp(nip, wl_n)
real(nbvm, nip, nwl)	pixel[ipix].meas[nwl].mprof[nip][nbvm]	mprof (vertical profile of Rayleigh backscattering). These values have to be skipped if ifmp=0 The ordering is as follows: mprof(1, 1, wl_1) mprof(2, 1, wl_1) ... mprof(nbvm[d], 1, wl_1) mprof(1, 2, wl_2) ... mprof(nbvm[d], nip, wl_n)

[a] This fairly counter-intuitive coding has a reason: the cloud flag was at first intended to be a general processing flag (0 = pixel not to be processed, 1 = to be processed), cloud contamination is only one particular case. Now the flag is limited to cloud screening, but unfortunately the coding couldn't be changed right away. Since the framework is still in development, it is planned to correct this unnatural feature in the near future.

[b] These fields are actually not used by the processing and therefore the SDATA implementer is free to put whatever he or she likes here (e.g. 0 for non-gridded data). They are intended mainly for documentation and debugging. For satellite data, they make it possible to retrieve the pixel original information in the original database.

[c] nbvm is actually to be multiplied by ifcov(ip, iwl). If this last number equals 0, the array reduces to an empty set and no value is to be read.

[d] nbvm is actually to be multiplied by ifmp(ip, iwl). If this last number equals 0, the array reduces to an empty set and no value is to be read.

The field **pixel[ipix].meas[nwl].meas_type[nip]** of pixel structure is a special code which defines the type of measure. The following table describes the valid codes and their interpretation:

Table 4.6. Types of measurements

Constant Name (used in source code)	Value (SDATA 2.0)	Meaning
MEAS_TYPE_UNKNOWN	0	The measurement type is invalid or not yet implemented
MEAS_TYPE_TOD	11	Total Optical Depth
MEAS_TYPE_AOD	12	Aerosol Optical Depth
MEAS_TYPE_ABS	13	Aerosol absorption optical depth
MEAS_TYPE_P11	21	Phase Matrix Element P11
MEAS_TYPE_P12	22	Phase Matrix Element P12
MEAS_TYPE_P22	23	Phase Matrix Element P22
MEAS_TYPE_P33	24	Phase Matrix Element P33
MEAS_TYPE_P34	25	Phase Matrix Element P34
MEAS_TYPE_P44	26	Phase Matrix Element P44
MEAS_TYPE_P11_rel_ang	27	p11/p11(given_angle) phase matrix element
MEAS_TYPE_P12_rel	28	-p12/p11 phase matrix element
MEAS_TYPE_LS	31	Lidar Signal
MEAS_TYPE_RL	32	Raman Lidar Signal
MEAS_TYPE_DP	35	Volume Depolarization Ratio
MEAS_TYPE_VEXT	36	Vertical Extinction profile
MEAS_TYPE_VBS	39	Vertical Backscatter profile
MEAS_TYPE_I	41	Normalized Radiance I [a]
MEAS_TYPE_Q	42	Polarized radiance Q[a]
MEAS_TYPE_U	43	Polarized Radiance U[a]
MEAS_TYPE_P	44	Polarization Rate: P = sqrt(Q*Q + U*U)/I
MEAS_TYPE_P	44	Polarization Rate: P = sqrt(Q*Q + U*U)
MEAS_TYPE_I_rel_sum	45	Relative Stokes parameter I/ sum(I(1:NBVM)), where NBVM is the total number of provided angles
MEAS_TYPE_P_rel	46	Linear polarization sqrt(Q*Q+U*U)/I

[a] All the Stokes Parameters are to be expressed as reduced quantities, without dimension

Equation 4.1. Conversion from absolute radiances to normalized, reduced radiances

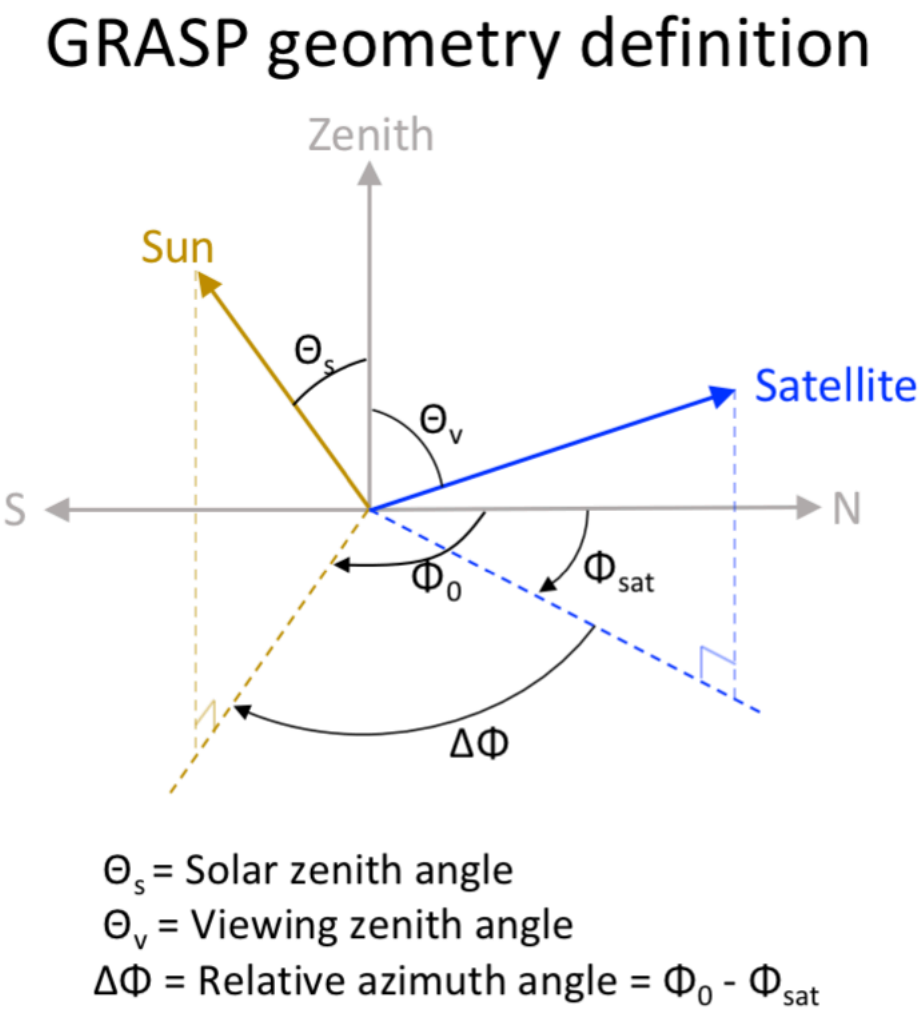
I = radiance * π / E0

where radiance is the radiance of the instrument, and E0 the solar spectral flux, that may be both in mW / (m^2 * sr * nm) or equivalent units

4.2.2. Angle definition

This chapter main goal is to describe how the angles should be defined to be used inside of GRASP code. The universal spirit of GRASP, where many different instruments coexist (from satellite to ground based measurements), creates challenges to define a homogeneous way of defining the angles, keeping a unique geometry. As it is shown in the figure Figure 4.4, “Definition of GRASP geometry” GRASP angles are defined to be considered as "normal" for satellite reference.

Figure 4.4. Definition of GRASP geometry



Since GRASP angles are defined using a satellite reference, it provokes some problems to define what we could call as an intuitive "ground based" reference system. That is why we are going to put special emphasis on the definition of the angles to these less intuitive applications. The intuitive reference for "ground based" measurements, in spherical geometry, is given as follows:

- θ_{gb} zenith angle: with the zero established in the zenith
- φ_{gb} azimuth angle: with the zero considered in the sun position

where the sub index "gb" makes reference to "ground based".

The conversion to the GRASP geometry is done as follows:

Equation 4.2. Conversion from θ_{gb} (ground based) to θ_G (GRASP)

$$\theta_G = 180° - \theta_{gb}$$

Equation 4.3. Conversion from φ_{gb} (ground based) to φ_G (GRASP)

$$\varphi_G = 180° + \varphi_{gb}$$

Here we propose some examples for better understanding of the process. Before defining the measurement angles introduced in the code, both "intuitive" and "GRASP", we need first to consider the instrument viewing angle for each scenario (θ_v, φ_v). The following figure and table will provide some examples of angles defined for the ground based applications.

Figure 4.5. Ground based angles definition example

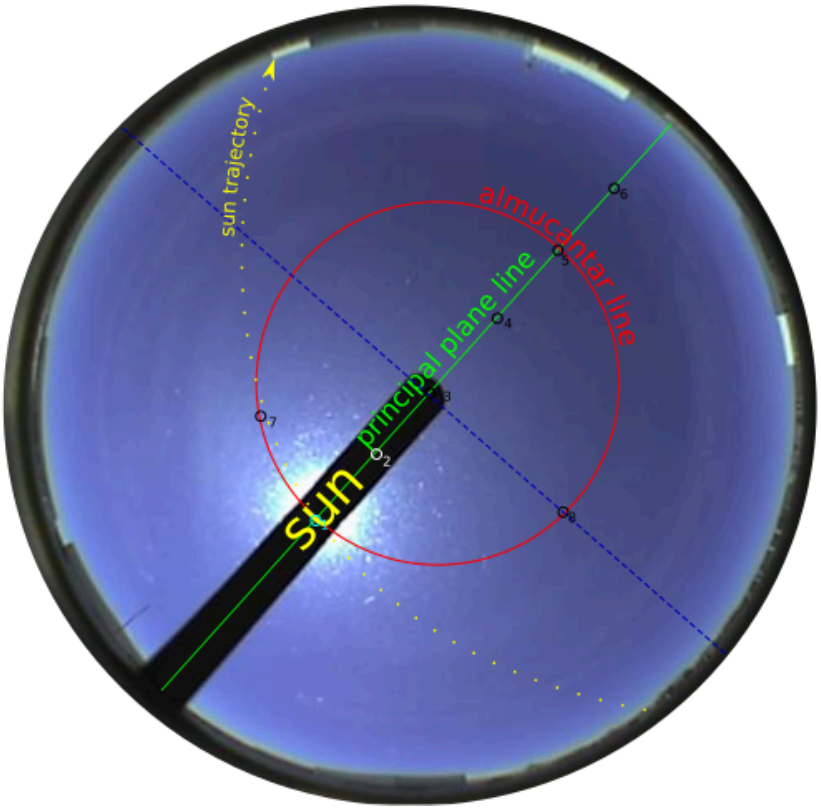


Table 4.7. Specific examples in ground based angle definition example

Example	θ_s	θ_v	φ_v	θ_{gb}	φ_{gb}	θ_G	φ_G
1	25°	0°	0°	25°	0°	155°	180°
2	25°	12.5°	0°	12.5°	0°	167.5°	180°
3	25°	25°	0°	0°	0°	180°	180°
4	25°	37.5°	0°	12.5°	180°	167.5°	0°
5	25°	50°	0°	25°	180°	155°	0°
6	25°	90°	0°	65°	180°	115°	0°
7	25°	0°	30°	25°	30°	155°	210°
8	25°	0°	90°	25°	90°	155°	270°

Since sunphptometers are widely used with GRASP, the following table provides information specifically about these instruments, considering the instrument viewing angle for each scenario (θ_v, φ_v). They can be understood as the "movements of the motors". The process will be as follow:

instrument viewing angle -> angle in (intuitive) ground based -> angle in GRASP

Table 4.8. Sunphotometer angle description

Measure type	Angle	Inst. View.	Range	Gr. Based	Range	GRASP	Range
Direct sun	θ	$\theta_v = 0°$	[0°]	$\theta_{gb} = \theta_s$	[0° -- 90°][a]	$\theta_G = 180° - \theta_s$	[180° -- 90°][a]
	φ	$\varphi_v = 0°$	[0°]	$\varphi_{gb} = 0°$	[0°]	$\varphi_G = 180°$	[180°]
Almucantar	θ	$\theta_v = 0°$	[0°]	$\theta_{gb} = \theta_s$	[θ_s]	$\theta_G = 180° - \theta_s$	[180° -- θ_s]
	φ	φ_v = 3°, 3.5°, 4°, 5°, ..., 90°, ... 180°	[0° - 180°]	φ_{gb} = 3°, 3.5°, 4°, 5°, ..., 90°, ... 180°	[0° -- 180°]	φ_G = 183°, 183.5°, 184°, 185°, ... 270°, ... 360°	[180° -- 360°]
Principal plane measurement: Before the zenith	θ	θ_v = -6°, ... , -3°, 3°, $\theta^{max} < \theta_s$	[-6° -- ($\theta^{max} < \theta_s$)]	$\theta_{gb} = \theta_s + 6°$ ... $\theta_s + 3°$, $\theta_s - 3°$, ... $\theta_s - 6°$, $\theta_s - \theta^{max}$	[($\theta_s + 6°$) -- 0][bc]	$\theta_G = 180° - \theta_s - 6°$, ... 180° - $\theta_s - 3°$, 180° - $\theta_s + 3°$, ... 180° - $\theta_s + \theta^{max}$	[(180° - θ_s - 6°) -- 180°]
	φ	$\varphi_v = 0°$	[0°]	$\varphi_{gb} = 0°$	[0°]	$\varphi_G = 180°$	[180°]

Measure type	Angle	Inst. View.	Range	Gr. Based	Range	GRASP	Range
Principal plane measurement: After the zenith	θ	$\theta_v = \theta^{min} < \theta_s$, ... 140° or $\theta^{max} - \theta_s > 90°$	[(θ^{min} < θ_s) -- 140°]	$\theta_{gb} = \theta^{min} - \theta_s$ ... 140° - θ_s or $\theta^{max} - \theta_s$	[0° -- 90°][bc]	$\theta_G = 180° + \theta_s - \theta^{min}$, ... 180° + θ_s - 140° or 180° + $\theta_s - \theta^{max}$	[180° -- 90°][bc]
	φ	$\varphi_v = 0°$	[180°]	$\varphi_{gb} = 180°$	[180°]	$\varphi_G = 0°$	[0°]
Polarized principal plane measurement: before the zenith[d]	θ			θ_{gb} = 85°, 80°, 75°, ... 10°, 5°, 0°		θ_G = 95°, 100°, 105°, ... 170°, 175°, 180°	
	φ			$\varphi_{gb} = 0°$		$\varphi_G = 180°$	
Polarized principal plane measurement: after the zenith[d]	θ			θ_{gb} = 0°, 5°, 10°, ... 75°, 80°, 85°		θ_G = 180°, 175°, 170°, ... 105°, 100°, 95°	
	φ			$\varphi_{gb} = 180°$		$\varphi_G = 0°$	

[a] θ_s refers to the solar zenith angle (for different measurements)

[b] decreasing values

[c] decreasing values

[d] The data of the polarized principal plane correspond always to fixed points in the sky and it is given for the instrument in so-called ground based coordinates.

In the case of nephelometer data angle definition, it is a bit different since only θ angle has to be defined, the rest of the angles will be ignored. To provide nephelometer data (phase matrix), the conversion to the GRASP geometry is done as follows:

Equation 4.4. Conversion from θ_n (nephelometer scattering angle) to θ_G (GRASP)

$$\theta_G = 180° - \theta_n$$

4.2.3. Input information for characteristics

While in many cases the input data is just represented by SDATA information, sometimes it is necessary to provide extra information for each pixel. The way to inject extra information to the algorithm is to do it through the initial guess. Sometimes, this is just because the user wants to provide different initial guesses for each pixel, or sometimes it is just input information. In the first case, this can be easily done by the imagedat files that will be described below. On the other hand, when a charactacteristic is defined as “retrieved=false”, it is taken as input information.

The “segment_imagedat” tool enables the user to provide an ASCII file containing the complete set of initial guesses for all the pixels. Following illustration is an example of how the imagedat files looks like:

Figure 4.6. Example of a possible use of imagedat

```
1   -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
2   -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
3   -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
4   -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
5   -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
6   0.058018    0.030021    0.050812    0.030991    0.058421    0.030278    0.051478    0.031338
7   0.076549    0.044353    0.068028    0.045371    0.077004    0.044621    0.068740    0.045729
8   0.113291    0.077165    0.102823    0.077981    0.113800    0.077354    0.103514    0.078226
9   0.124698    0.084711    0.112765    0.085605    0.125224    0.084923    0.113407    0.085874
10  0.159424    0.107566    0.142857    0.108697    0.159978    0.107849    0.143296    0.109041
11  0.175329    0.111182    0.155521    0.112296    0.176969    0.112061    0.156522    0.113161
12  0.220433    0.151935    0.206052    0.155613    0.221507    0.152876    0.206771    0.156512
13  0.309332    0.247467    0.315043    0.259119    0.308149    0.247979    0.314397    0.259505
14  0.320031    0.259707    0.324584    0.273104    0.319534    0.260553    0.324632    0.273732
15  0.449005    0.584904    0.560287    0.596099    0.440385    0.581362    0.560533    0.594357
16  0.186116    0.188525    0.179675    0.200471    0.185942    0.188563    0.179126    0.200096
17  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
18  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
19  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
20  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
21  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
22  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
23  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
24  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
25  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
26  -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000 -999.000000
```

The columns in this ASCII file correspond to the pixels, and each row is associated with a different characteristic. The first column is just the enumeration of characteristics starting by 1. The total number of both columns and rows has to be consistent with the input file. The value “-999” can be assigned to the initial guesses which the user does not need to modify, in this case the value defined in the characteristics section of the settings file will be taken by default. This mechanism provides a very versatile tool to inject pixel-dependent data to the retrieval code, for adjusting the initial guess or just for providing external input information from climatologies, models, etc.

4.2.4. How to prepare the photometer data

Sunphotometers are widely used with GRASP. They take measurements of sky radiance and direct sun. Many inversion strategies can be used to retrieve sunphotometer data, but this section will explain how to define input data. The information that runs inside of GRASP has to be pre-processed in order to screen clouds, calibrate and normalize the data.

Following Section 4.2.1, “The SDATA format”, the direct sun measurements can be described as AOD or TOD defined in Table 4.6, “Types of measurements” as measurements of type 11 or 12. At this point it is needed to take into account that if "ifgas" field is defined as 1 in the case of AOD, no gaseous absorption optical depth will be accounted, but in the case of TOD they will be subtracted. The gases also affect the radiance measurements but in lower magnitude. In the case of TOD + radiances with ifgas=1, the same model will be applied to all measurements. If AOD is used, some (minor) incongruences could come from the use of different models to calculate gases for AOD and for radiances.

Radiance measurements are defined with the constant MEAS_TYPE_I(41). Polarized measurements can be defined as Q,U (42, 43) or as polarization rate (44). It is also important to check how polarized data is going to be manipulated in the retrieval code based on inversion strategy defined in the settings file.

4.2.5. How to prepare the lidar data

Note that all processing will be considering range corrected profile for one wavelength. The procedure for other profiles from different wavelengths are exactly the same. Range corrected profile implies that at least the background noise was subtracted and altitude corrections were applied to the raw signal, but if it's possible to consider all other corrections (electrical noise and overlap correction, dead time correction, gluing analog and photon-counting signals and all others that your system may have), you should apply them.

Step 1. Background noise subtraction and range correction.

Let **B'** be the estimation of the background noise. Usually **B'** is estimated as $\mathbf{P(Z_B)}$, accumulated and averaged around selected altitude $\mathbf{Z_B Z}$, much higher than the maximum altitude of lidar extraction in step 2 ($\mathbf{Z_{max}}$). For example with maximum altitude $\mathbf{Z_{max}=15km}$, it is averaged around 30 km and accumulated for the whole period of lidar observation. The noise and range corrected signal will be:

$$\mathbf{S(Z_i) = (P(Z_i)-B')*Z_i^2}$$

Step 2. Altitude range selection and signal cropping.

The minimum $\mathbf{Z_{min}}$ and maximum $\mathbf{Z_{max}}$ altitudes are selected and the signal is cropped, so $\mathbf{Z_{min}<Z_i<Z_{max}}$. The minimum altitude should be selected as low as possible, preferably in the region where the overlap correction could be correctly applied. The $\mathbf{Z_{max}}$ should be selected from the following considerations: maximum altitude where the noise levels of lidar measurement are acceptable and the amount of atmospheric aerosols is still noticeable.

In case if the lidar used is inclined The sounding zenith angle should be provided for the corresponding wavelenghts. The angle value should be placed in the position of the SZA corresponding to this wavelength. All SZA's for all vertical profile measurements should be the same. Vertically pointed

lidar should have this value set to 0. *Note: Keep in mind that retrieved aerosol vertical profiles will be retrieved for vertically projected altitudes, i.e. if your lidar is inclined, the maximum altitude of the profile will be equivalent to* $Z_{max}*cos(SZA)$.

Step 3. Backscatter of molecular profile

Backscatter vertical profile is calculated inside of GRASP based on standard atmosphere model for each lidar wavelength[1]:

$$\beta_{mol}(Z_i,\lambda) = N * \sigma(\lambda) * (P(Z_i)/P_{SA}) * (T_{SA}/T(Z_i))$$

where:

- **N** molecular number density
- $\sigma(\lambda)$ total Rayleigh cross section per molecule, which analytical formula can be written like $\sigma(\lambda) = A * \lambda^{-B-C*\#-D/\lambda}$
- P_{sa} pressure of standard atmosphere model
- T_{sa} temperature of standard atmosphere model
- $P(Z_i)$ pressure profile of atmosphere
- $T(Z_i)$ temperature profile of atmosphere

Step 4. Reducing the number of points in profiles

GRASP/GARRLiC can use arbitrary altitude/range scale. However, to keep number of retrieved parameters reasonable and to fight higher noise contamination of lidar signals at higher altitudes, it is recommended to use a logarithmical altitude/range scale with N_Z points to represent aerosol profiles in the atmosphere. For that, we have to present all vectors (altitude/range vector and profiles of lidar signals) in logarithmically equidistant manner.

1. Move to logarithmic scale and find altitude/range step:

logarithmic scale: $Z_i^{lg} = lg(Z_i)$

step in log scale: $\#Z = (Z_{max}^{lg} - Z_{min}^{lg})/N_Z$

logarithmic altitude ranges (from h_k to h_{k+1}) for averaging data in logarithmically equidistant manner, k = 1 .. N_Z: $h_k = Z_0^{lg} + (k - 1) * \#Z$

2. Average the data profiles

$$A_k = (\sum_{j=1}^{n} A_j(h_k, h_{k+1})) / n$$

where:

- **A** altitude vector or profile of lidar signal or molecular backscatter
- **n** number of points inside logarithmic altitude ranges

After such procedure, the number of points in three main vectors reduced to N_Z points in logarithmically equidistant manner. These values should be placed in places corresponding to the zenith wieving (`vza` or θ_v) angles for the vavelenths corresponding to lidar or vertical measurements.

Hint: the altitude/range vectors for measurements at all wavelengths have to be the same.

Step 5. Profile normalization

Values of lidar signals (except for volume depolarization profiles) vary from instrument to instrument, from detector to detector, that is why GRASP/GARRLiC requires normalized lidar signal. For consistency with the molecular optical depth, the profile of the molecular backscatter inside the code has to be normalized as well. Normalized lidar and backscatter profiles:

$$\mathbf{A'_k = A_k / \int_{Zmin}^{Zmax} A_k \, dZ}$$

where **A** represents profile of lidar signal or molecular backscatter.

Caution: integration have to be done using meters in altitudes.

At the end, for each wavelength you have to have normalized lidar profiles and altitude vector. Volume depolarization profiles don't need to be normalized, the only requirement for such observations is to be presented in the percentage range i.e. [1.0e-9, 100]

References: Anthony Bucholtz, Rayleigh-scattering calculations for the terrestrial atmosphere, Optical Society of America, 1995.

4.2.6. How to prepare nephelometer data

GRASP is able to retrieve and simulate nephelometer measurements. This section is devoted to the definition of the convention to properly account the units and measurement geometry of these instruments within GRASP standards.

The zero reference of scattering angles in GRASP is 180 °. Thus, in order to introduce the nephelometer geometry in GRASP sdata, only θ angle (zenith angle) has to be defined, the rest of the angles will be ignored. The conversion to the GRASP geometry is done as follows:
$\mathbf{\theta_G = 180° - \theta_n}$

Where θ_G corresponds to sdata angles and θ_n to the scattering angles of the nephelometer.

GRASP measurement input corresponds to the normalized Phase Matrix element $P_{1,1}$, and the relation with scattering function is described below.

GRASP scattering phase function units are:
$\mathbf{[F_{1,1}(\lambda, \Theta)] = 1/\mu m}$

GRASP scattering phase function units are:
$[\sigma]$ **= 1/µm**

The relationship between both is represented by the following expression:
$\mathbf{\sigma = (1/2) * \int_0^{\pi} F_{1,1}(\lambda, \Theta) * \sin(\Theta) \, d\Theta}$

Which means that the normalized Phase Matrix element $P_{1,1}$ is defined as:
$\mathbf{(1/2) * \int_0^{\pi} P_{1,1}(\lambda, \Theta) * \sin(\Theta) \, d\Theta = 1 \;\; P_{1,1}(\lambda, \Theta) = F_{1,1}(\lambda, \Theta)/\sigma}$

If polar neph measurements are used, an additional factor of $4*\pi$ is necessary to convert into the units in GRASP: µm-1Str-1 to µm-1.

4.3. Output module

The output module is responsible for managing results for each segment or entire tile and driving them to the correspond destination (file or screen). There are two kinds of main output structure into GRASP: tile and segment. Segment output structure represents the output results obtained from the retrieval library. Then, core unit compacts it and stores it in a tile output structure, which contains the results of the entire process.

Output module can be extended in the same way as input module. In the case of the output there are three kinds of extensions:

- **output segment function:** it is called after retrieving a segment and is called with the results of that single segment.

- **output current function:** it is called after retrieving each segment but it is called with the partial tile processed until that moment. In each call to this function, it gets a partial tile nearest to completion. Last call to that function will send the entire tile results.

- **output tile function:** at the end of the process, a function is called sending to it the entire tile results. It can print a complete map of the process.

As the user can see, GRASP is very flexible in the way to work with the output. In a high optimized process, it can be adapted directly to the format of the output database. By default, GRASP comes with some ASCII output functions, which allows to print the results in a readable way (ASCII or specific GRASP format), into a file (using stream library) or on the screen. Additional extensions can be added to get the output in different formats such us HDF, NetCDF, png plots...

4.3.1. The list of GRASP output parameters

The output from GRASP is quite complex and strongly dependent on the settings used. As it was discussed in Section 4.1.2, “Retrieved characteristics”, there are two kinds of output products: direct and derived. Direct results are the values directly inverted in the initial guess array, then after obtaining them, forward model is called one more time to obtain the derived products. The list of direct and derived products obtained depends on the data and the inversion strategy selected (defined in the settings file). This chapter contains a complete list of products that can be obtained with GRASP, but it does not mean that all of them can be obtained with all inversion strategies.

In the internal GRASP output structures there are some products that are repeated between direct products structure (array with the same shape as initial guess) and derived products. The reason is that for some applications they are direct information, for others they are derived results. This depends on input data and inversion strategy, defined in the settings file.

Before defining the output, we are going to define the size of some arrays. This information is needed to understand the list of the products. For example, when a product such as AOD is defined as many times as wavelengths, it is because the output will be wavelength dependent (a value for each wavelength):

- **NW:** Number of wavelengths

- **NSD:** Number of aerosol components

- **NKNOISE:** Number of noises defined

- **NPARS:** Number of parameters to be retrieved

The following list includes all products that can be obtained using GRASP:

- Number of iterations (niter)

- Total final measurement fitting residual for multi-pixel retrieval (rest)

- Detailed (i.e., separated by the type of observation) final absolute measurement fitting residuals for segment (resat)

- Detailed (i.e., separated by the type of observation) final relative measurement residuals for segment (resrt)

- If SD (size distribution) retrieved in form of binned SD, the number of grid radii for SD (used to print output) (radius)

- If SD retrieved in form of binned SD, grid radii

- If the pre-calculated lognormal bins where used, the function describing each lognormal SD bin (used to print output) (SDL)
- Main output for each single pixel (for both single- and multiple-pixel retrievals):
  - Single pixel total residual (meas. + smoothness constraints) (res)
  - Detailed absolute measurement residuals (resa[NKNOISE])
  - Detailed relative measurement residuals (resr[NKNOISE])
  - Retrieved aerosol and surface reflectance parameters (par[NPARS])
  - Angstrom exponent (Aexp)
  - For each wavelength:
    - Spectral total aerosol extinction (extt)
    - Spectral total aerosol single scattering albedo (ssat)
    - Spectral total aerosol absorption extinction (aext)

    If retrieved aerosol consist of several components:
    - Spectral extinction for each component (ext[NSD])
    - Spectral single scattering albedo for each component (ssa[NSD])
    - Real part of refractive index for each aerosol component (mreal[NSD])
    - Imaginary part of refractive index for each component(mimag[NSD])
- If SD retrieved in form of binned SD, the optical properties can be calculated for fine and coarse modes, separated using chosen inflection radii:
  - Volume median radius (rv)
  - Standard deviation (std)
  - Concentration (cv)
  - Effective radius (reff)
  - Spectral extinction for each wavelength (ext[NW])
- Phase matrix parameters:
  - Number of scattering angles (nangle)
  - Values of scattering angles (angles)
  - For each pixel and for all the wavelengths:
    - Phase matrix elements for each aerosol component ph11, ph12, ph22, ph33, ph34, ph44
    - Total aerosol phase matrix elements pht11, pht12, pht22, pht33, pht34, pht44
    - Lidar and depolarization ratios for each aerosol component (lr, dlpr)
    - Total aerosol lidar and depolarization ratios (lrt, dlprt)
  - Phase matrix norm. The analytical expresion used to calculate the norm is:
    Norm = $(1/2) * \int_0^{\pi} P_{1,1}(\lambda, \Theta) * \sin(\Theta)\, d\Theta$

However, in order to achieve an optimal degree of accuracy it is recomended to perform this integration trought the Simpson rule with 721 bins in the logaritmic space.

- Asymmetry parameter. The analytical expresion used to calculate this magnitude is:
  $< \cos(\Theta) > = (1/2) * \int_0^{\pi} P_{1,1}(\lambda, \Theta) * \sin(\Theta) * \cos(\Theta) \, d\Theta$
  It is recomended to perform the numerical calculation of the integral analogously to the case of the phase matrix norm.

- If chemical composition is retrieved, then for each pixel and each aerosol component:
  - Relative humidity (rh(NSD))
  - Fraction of soluble (fslbl(NSD))
  - Insoluble fractions of soot (fsoot(NSD))
  - Insoluble fractions of iron (firon(NSD))
  - Insoluble fractions of “quartz” (fslbl(NSD))
  - Water fraction (fwtr(NSD))
- Surface reflectance parameters for each pixel and for each wavelength:
  - All parameters of BRDF
  - All parameters of BPRDF
  - Surface Albedo
- Lidar characteristics for each pixel:
  - Levels for vertical profiles
  - Vertical profiles for retrieved aerosol components (avp(NSD))
- Lidar optical characteristics for each pixel and for each wavelength:
  - Aerosol extinction profiles $(\sigma^{aer}(\lambda, h))$. In order to obtain this magnitude from the variables in the standard GRASP output file, the column aerosol optical depth just needs to be weighted by the normalized aerosol vertical profile for each mode.

    $\sigma^{aer,i}(\lambda, h) = \tau^i * avp_i(h)$

    If two or more aerosol modes are included, the total extinction profile can be obtained adding up all profiles.

    $\sigma^{aer}(\lambda, h) = \sum_{i=1}^{n} \tau^i * avp_i(h)$
  - Aerosol backscatter profiles $(\beta^{aer}(\lambda, h))$. To obtain this magnitude it is only necessary to divide the aerosol extinction profile of each mode by its corresponding Lidar ratio.

    $\beta^{aer}(\lambda, h) = \sum_{i=1}^{n} \sigma^{aer,i}(\lambda, h) / S_i(\lambda)$
  - Aerosol absorption profiles $(\sigma^{aer}{}^{abs}(\lambda, h))$. In order to obtain this magnitude from the variables in the standard GRASP output file, the column aerosol absorption optical depth $(\tau^{i\,abs})$ just needs to be weighted by the normalized aerosol vertical profile for each mode.

    $\sigma^{aer}{}^{abs}(\lambda, h) = \sum_{i=1}^{n} \tau^{i\,abs} * avp_i(h)$
  - SSA profiles $(\omega^0(\lambda, h))$. Once all the previous described magnitudes has been calculated the calculation of SSA profiles is inmidiate:

$\omega^0(\lambda, h) = \sigma_{aer}^{scat}(\lambda, h) / \sigma_{aer}(\lambda, h) = (\sigma_{aer}(\lambda, h) - \sigma_{aer}^{abs}(\lambda, h)) / \sigma_{aer}(\lambda, h)$

- Lidar ratio profiles (S(#, h)):

  $S_{aer}(\lambda, h) = \sigma_{aer}(\lambda, h) / \beta_{aer}(\lambda, h)$

- Phase matrix ($P_{j,k}(\lambda, \Theta, h)$):

  $P_{j,k}(\lambda, \Theta, h) = \Sigma_{i=1}^{n} P_{j,k}^{i}(\lambda, \Theta) \sigma_{aer,i}^{scat}(\lambda, h) / \sigma_{aer}^{scat}(\lambda, h)$

- Lidar depolarization profiles ($\delta(\lambda, h)$):

  $\delta(\lambda,\#) = (P_{1,1}(\lambda, 180°, h) - P_{2,2}(\lambda, 180°, h)) / (P_{1,1}(\lambda, 180°, h) + P_{2,2}(\lambda, 180°, h))$

- Retrieved lidar calibration coefficients (for lidar wavelength only)

- Fit of every measured characteristic for each pixel and for each wavelength
- Error estimation for each pixel:
  - Standard deviations of the random errors of the retrieved parameter logarithms (~relative errors) (ERRP)
  - Standard deviation of systematic errors of the retrieved parameter logarithms (BIASP)
  - Standard deviations of the random errors of the retrieved extinction for each aerosol component (~relative errors) (ERR_ext)
  - Standard deviations of systematic errors of the retrieved extinction for each aerosol component (BIAS_ext)
  - Standard deviations of the random errors of the retrieved total extinction (~relative errors) (ERR_extt)
  - Standard deviations of systematic errors of retrieved total extinction (BIAS_extt)
  - Standard deviations of the random errors of the retrieved single scattering albedo for each aerosol component (~relative errors) (ERR_ssa)
  - Standard deviations of systematic errors of retrieved single scattering albedo for each aerosol component (BIAS_ssa)
  - Standard deviations of the random errors of the of retrieved total single scattering albedo (~relative errors) (ERR_ssat)
  - Standard deviations of systematic errors of the retrieved total single scattering albedo (BIAS_ssat)
  - Standard deviations of the random errors of the of retrieved lidar ratio for each aerosol component (ERR_lr)
  - Standard deviations of systematic errors of the lidar ratio for each aerosol component (BIAS_lr)
  - Standard deviations of the random errors of the of retrieved depolarization ratio for each aerosol component (ERR_dr)
  - Standard deviations of systematic errors of the depolarization ratio for each aerosol component (BIAS_dr)
- Radiative forcing for each pixel:
  - The heights for forcing output (HLV)

- Broad band up-ward flux without aerosol at each height (BBUFX0)
- Broad band down-ward flux without aerosol at each height (BBDFX0)
- Broad band up-ward flux with aerosol at each height (BBUFXA)
- Broad band down-ward flux with aerosol at each height (BBDFXA)

- Estimations of aerosol particulate matter at the ground level (PM)
- Aerosol type for each pixel (requires that the optical properties for fine and coarse modes are included in the calculated output)

4.3.2. GRASP classic output description

In this section the GRASP classic output format is going to be described for both if the output.segment.stream setting parameter has been set to "screen", in this case all output information will be printed on terminal; or alternatively if a path to an ascii file is provided. However, note that there are more possibilities of GRASP output formatting which can differ from what is going to be shown here.

The GRASP classic output is divided in three main sections:

- Information of the residuals.
  This information is place in the head of the classic output. It contains one line per pixel with information about convergence and residuals after the last iteration. The first float number in each of these lines corresponds to the absolute value of the total inversion residual of the corresponding pixel. Then, the absolute and relative residuals for each noise type defined in settings can be found.

Figure 4.7. An example of the residual information in GRASP classic output

```
      noise  abs         rel      noise  abs         rel
0.358 1:     0.777E-03   0.262 %  2:     0.317E-03  0.115 %   pixel # 1  Residual after iteration # 21
0.288 1:     0.124E-03   0.425 %  2:     0.341E-03  0.155 %   pixel # 2  Residual after iteration # 17
```

After the residuals, the information of date, time, longitude and latitude of each pixel can be found.

The next part of the output is a list of all the retrieved parameters as they are calculated in the inversion matrix, each column is associated with a different pixel.

Figure 4.8. An example of the vector of retrieved parameters in GRASP classic output

```
Parameter #, Vector of retrieved parameters
1    0.31000E-03    0.21000E-03
2    0.38939E-02    0.89239E-02
3    0.19772E-01    0.13772E-01
4    0.45246E-01    0.55232E-01
5    0.50813E-01    0.68823E-01
6    0.35267E-01    0.23481E-01
7    0.16086E-01    0.24567E-01
8    0.41738E-02    0.67799E-02
9    0.58516E-03    0.24514E-03
10   0.44740E-04    0.44740E-04
11   0.50000E-05    0.43456E-05
12   0.13710E-04    0.19340E-04
13   0.50331E-04    0.548370-04
14   0.18693E-03    0.20019E-03
15   0.67248E-03    0.23535E-03
16   0.22029E-02    0.11244E-02
17   0.59918E-02    0.60022E-02
18   0.12137E-01    0.23731E-01
19   0.17703E-01    0.19822E-01
20   0.18786E-01    0.12441E-01
21   0.14473E-01    0.29249E-01
22   0.79026E-02    0.08708E-02
23   0.29037E-02    0.33077E-02
24   0.69835E-03    0.54197E-03
25   0.11000E-03    0.21002E-03
```

- Detailed parameters.
  In the next part of the output different atmospheric and surface parameters can be found. Which of them are shown and which are not is defined by the settings parameters in retrieval.products part. Each product is identified by a header where the name and some information is provided. The different columns corresponds to the different pixels analougously to the case of the vector of retrieved parameters.

  If there are more than one atmospheric component in the retrieval the information corresponding to each mode is expressed differently depending on if it is an optical or microphysical product. In the case of microphysical products the information of the different components is expressed in different lines which start with an identificative integer.

Figure 4.9. An example of the aerosol volume concentration in GRASP classic output for two aerosol modes

```
Aerosol volume concentration (um^3/um^2 or um^3/um^3)
    1   0.47855E-01 0.68932E-01
    2   0.22771E-01 0.13793E-01
```

However, in the case of optical products, the corresponding mode of the product is indicated in the product header name, and each line is associated with a wavalength which is also indicated in the beginning of it.

Figure 4.10. An example of the Aerosol Optical Depth in GRASP classic output for two aerosol modes

```
Wavelength (um), AOD_Total (unitless or 1/um)
       0.44000   0.45107E+00   0.354561E+00
       0.67500   0.18630E+00   0.223511E+00
       0.87000   0.99709E-01   0.782688E-01
       1.02000   0.66909E-01   0.456723E-01
Wavelength (um), AOD_Particle_mode_1 (unitless or 1/um)
       0.44000   0.43880E+00   0.30345E+00
       0.67500   0.17344E+00   0.20230E+00
       0.87000   0.86553E-01   0.701544-01
       1.02000   0.53769E-01   0.405612E-01
Wavelength (um), AOD_Particle_mode_2 (unitless or 1/um)
       0.44000   0.12274E-01   0.54561E-01
       0.67500   0.12856E-01   0.23462E-01
       0.87000   0.13156E-01   0.82655E-02
       1.02000   0.13140E-01   0.65231E-02
```

- Information of the fitting.
  In the final part of the GRASP classic output the information is organized in different blocks. In each of these blocks the measurements associated to each wavelength and pixel of the SDATA and the fitted measurements after the final iteration can be found. Each measurement is separated by a header where the measurement in the sdata is indicated as meas_+"measurement name", and the fitted as fit_+"measurement name". If the measurement is defined by a specific geometry, this geometry is also included here. As for example the Solar Zenith Angle (sza), the zenith angle of the measurement (vis), the azimuth angle of the measurement (fis) or the corresponding scattering angle (sca_ang).

Figure 4.11. An example of the fitting information in GRASP classic output for one wavelenght of one pixel with TOD and irradiance measurements

```
-------------------------------------------------------------------------
pixel #    1   wavelength #   1     0.440 (um)
-------------------------------------------------------------------------
       meas_tod        fit_tod
    0.70302E+00    0.70290E+00
   #      sza      vis      fis  sca_ang         meas_I          fit_I
   1    70.00   110.00   183.50     3.29    0.67436E+00    0.67369E+00
   2    70.00   110.00   184.00     3.76    0.62857E+00    0.62765E+00
   3    70.00   110.00   185.00     4.70    0.57672E+00    0.57630E+00
   4    70.00   110.00   186.00     5.64    0.55140E+00    0.55151E+00
   5    70.00   110.00   187.00     6.58    0.53713E+00    0.53745E+00
   6    70.00   110.00   188.00     7.52    0.52757E+00    0.52796E+00
   7    70.00   110.00   190.00     9.40    0.51327E+00    0.51358E+00
   8    70.00   110.00   192.00    11.27    0.50043E+00    0.50059E+00
   9    70.00   110.00   194.00    13.15    0.48706E+00    0.48712E+00
  10    70.00   110.00   196.00    15.03    0.47341E+00    0.47335E+00
  11    70.00   110.00   198.00    16.91    0.45892E+00    0.45876E+00
  12    70.00   110.00   200.00    18.78    0.44384E+00    0.44361E+00
  13    70.00   110.00   205.00    23.47    0.40481E+00    0.40449E+00
  14    70.00   110.00   210.00    28.15    0.36583E+00    0.36555E+00
  15    70.00   110.00   215.00    32.83    0.32882E+00    0.32864E+00
  16    70.00   110.00   220.00    37.49    0.29500E+00    0.29489E+00
  17    70.00   110.00   225.00    42.15    0.26492E+00    0.26483E+00
  18    70.00   110.00   230.00    46.80    0.23864E+00    0.23853E+00
  19    70.00   110.00   240.00    56.05    0.19656E+00    0.19638E+00
  20    70.00   110.00   250.00    65.23    0.16617E+00    0.16595E+00
  21    70.00   110.00   260.00    74.32    0.14471E+00    0.14453E+00
  22    70.00   110.00   270.00    83.28    0.13010E+00    0.13000E+00
  23    70.00   110.00   280.00    92.08    0.12075E+00    0.12071E+00
  24    70.00   110.00   300.00   108.94    0.11330E+00    0.11323E+00
  25    70.00   110.00   320.00   124.02    0.11489E+00    0.11461E+00
  26    70.00   110.00   340.00   135.46    0.11897E+00    0.11849E+00
```

4.4. Forward model

GRASP has several forward models and each of them are used (or not) depending on the application. For example, to retrieve nephelometer data, just single scattering (particle properties) will be used. For other applications, GRASP has also a multiple scattering module (radiative transfer) and a lidar signal module.

4.4.1. How to use the forward model: Derived products and reprocesing data

As it was explained in Section 4.1.2, “Retrieved characteristics” section, the retrieval algorithm works iteratively over an array of parameters (in its first definition it is called the initial guess), until it represents the best solution. This solution array has same shape as the initial guess (the same parameters and defined in the same position). Once it is obtained, a final call of the forward model with the resulting array provides a complete list of output products.

For some applications, it can be useful to use the forward model without inverting any data. It can be done easily in GRASP with the use of the setting parameter retrieval.mode=forward. When no retrieval is performed, just one call of forward model is performed. If in the initial guess array the user has set an aerosol model, it will be used inside of the forward model obtaining therefore an entire output structure, with information in all fields.

This procedure can be used also to reprocess some data. If output parameters of a retrieval are stored, then they can be set as initial guess and then, running GRASP with the same settings, except for retrieval.mode=forward the entire output can be obtained again. This procedure can be used to reprocess data with many objectives such as saving storage space (just save the output array of grasp and reprocess to obtain the rest, if it is needed) or obtaining extra products in the future.

4.4.2. Synthetic data

The previous procedure can also help to simulate the input data. It is useful because a valid geometry is necessary in the input data to set an SDATA file. In this case, an aerosol model is set as an initial guess and the code works just for the forward run. Then, by using retrieval.debug.simulated_sdata_file parameter, the user can set a path to the simulated files. A SDATA file will be dump to that path where the geometry is the same and the measurements are filled with the output results of the forward run. Then, this SDATA file can be used to self-consistency tests, where synthetic data is retrieved.

4.5. Aerosol modeling in GRASP

Due to the versatility and the high degree of generalisation of GRASP, there are different approaches to model aerosols in order to maximize the possibilities of the different retrieval schemes. Each of the approaches described below presents different advantages depending on the information available in the input measurements, the desired output products or the available computation time.

4.5.1. Kernels

In this approach the retrieved characteristics which determine the optical properties of aerosols are the size distribution, the real and imaginary refractive indexes, and the percentage of spherical particles (the non-spherical part is modeled as spheroids). The calculation of the radiative properties (phase matrix, scattering and absorption cross-sections) from these initial characteristics is made through four-dimensional look-up-tables called "Kernels".

The four dimensions of the Kernels are: the size parameter, sphericity and the real and the imaginary refractive indices. Due to the extensive nature of these kernels, this approach represents the less restricted methodology, because any possible combination of the represented characteristics can be obtained as the solution.

The lack of intrinsic restrictions of this approach (note that as in all GRASP retrievals, apriori constraints can be set for any characteristic) presents obvious advantages. However, the general nature of Kernels methodology normally requires input measurements containing a higher amount of information in comparison with other approaches.

There are three different ways to represent the aerosol size distribution, from the most general to the simplest: triangle bins, lognormal bins and precalculated lognormal bins. The corresponding characteristics names in the GRASP YAML settings file are: “size_distribution_triangle_bins”, “size_distribution_lognormal” and “size_distribution_precalculated_lognormal”. Independently of the selected representation of the aerosol size distribution, the rest of the steps of this methodology to obtain aerosol radiative properties are the same.

In the triangle bins representation the value of each bin is retrieved independently from the others, and it corresponds to the integrated value following the trapezoidal rule between the provided size limits. The lognormal bins approach is a simplified version of the former, where each aerosol size distribution mode is modeled as a gaussian function only represented by three parameters: the center, the standard deviation and the norm. The former value is the aerosol concentration of the corresponding mode. The precalculated lognormal bins size distribution represents a step further in the simplification of this characteristic. In this case each aerosol mode is also represented by a Gaussian function. However, the center and the standard deviation are fixed to a predefined value and only the norm (aerosol concentration) of each mode is retrieved.

4.5.2. Models

The main retrieved aerosol products of this approach are the fractions of the total aerosol concentration of precalculated aerosol models. These precalculated models correspond to the main aerosol types established by all our previous experiences (Ex: smoke, urban, oceanic and dust). Each of these models correspond to a fixed particle size distribution and refractive indices, containing the already calculated phase matrix, and the extinction and absorption cross-sections. The total aerosol retrieved characteristics can be obtained by weighting the characteristics used to calculate each of these models by its corresponding volume concentration.

The significant reduction of retrieved parameters makes this approach very suitable for the retrieval of input measurements with a reduced amount of information. The absence of Kernels in the whole process makes this option by far the fastest of all. One of the main drawbacks of this methodology is the fact that the inversion is intrinsically constrained by the selected models. However, these models have been carefully selected to be suitable to cover almost all atmospheric situations. Moreover, they can be recalculated or extended in order to cover specific situations.

An example of the necessary settings to use this approach can be found below. Where in the phase matrix section the bin of each mode no longer represents the limit radius but the accounted aerosol models. Particle size distribution, refractive indexes and sphericity characteristics are substituted by another characteristic called "aerosol_model_concentration".

```
forward_model:

      phase_matrix:
          size_binning_method_for_triangle_bins: logarithm
          number_of_elements: 4
          kernels_folder: "models_ang35_wl22_optimized"
          radius:
              mode[1]:
                  bins: [ 1.,  2., 3., 4., 5. ]

  .
  .
  .


  constraints:
      characteristic[1]:
          type: aerosol_model_concentration
          retrieved: true
          mode[1]:
```

```
            initial_guess:                        #smoke       #urban    #oceanic   #dust
                value:                           [0.9,           0.4,      0.4,     0.4.,      0.4]
                min:                             [0.000005, 0.000005, 0.000005, 0.000005, 0.000005]
                max:                             [1.0,           1.0,      1.0,      1.0,      1.0]
                index_of_wavelength_involved:    [0,               0,        0,        0,        0]
            single_pixel:
            .
            .
            .

    characteristic[2]:
        type: aerosol_concentration
        retrieved: true
        mode[1]:
            initial_guess:
                value:                           [0.05   ]
                min:                             [0.0001 ]
                max:                             [5.0    ]
                index_of_wavelength_involved:    [0      ]
            single_pixel:
            .
            .
            .
```

4.5.3. Chemistry

The chemistry approach can be seen somehow as an intermediate approach between Kernels and Models. In this case size distribution and sphericity percentage are directly retrieved as in the case of Kernels. However, instead of the refractive indexes, here the fractions of the total aerosol concentration corresponding to the different chemical components are retrieved. The refractive indices corresponding to each of these components are predefined. Thus, the weighted refractive indexes of all the fractions in combination with the particle size distribution and the sphericity parameter are used as input for the Kernels look-up-tables to calculate the radiative properties.

In comparison with the Models approach, where the radiative properties can only be linearly weighted, the refractive index presents an extra layer of complexity when they are mixed. Because the weighting methodology used to obtain the total refractive index retains information about the aerosol internal structure. Two mixing possibilities are available now in GRASP: an internal mixture, where a linear mixture of the different components is performed; but also a Maxwell-Garnett mixture is available, where one element is considered as the main host and the rest of the chemical elements are taken as inclusions inside of it.

In this case the necessary settings to use this approach are very similar to Kernels. The size distribution can be represented using the three already described possibilities in the Kernels approach. However, the refractive index characteristics are substituted by "particle_component_volume_fractions_linear_mixture" in the case of the linear mixture, or by "particle_component_fractions_chemical_mixture" in the case of a Maxwell-Garnett mixture. An extra section called "chemistry" has to be added in the Forward model part, where it is provided: the names of each chemical component accounted in the inversion, the soluble component (if necessary) and the path to the directory where these predefined refractive index look-up-tables are located.

```
forward_model:

    aerosol:
        chemistry:
            folder: "chemistry_refractive_indexes/"
            soluble: "ammnm_ntrt"
            species:
                mode[1]: [ 'black_carbon', 'mix_dust','iron_oxide','water']

    phase_matrix:
        size_binning_method_for_triangle_bins: logarithm
        number_of_elements: 1
        kernels_folder: "KERNELS_BASE/"
        radius:
            mode[1]:
                min: 0.05
                max: 15.0
```

```
.
.
.

constraints:
    characteristic[1]:
        type: size_distribution_triangle_bins
        retrieved: true
        mode[1]:
            initial_guess:
                value:                       [1.4715e-05, 1.4715e-03, ..., 1.4715e-03, 1.4715e-05]
                min:                         [0.00001,  0.00001,  ...,  0.00001,  0.00001]
                max:                         [1.0,      1.0,      ...,      1.0,      1.0]
                index_of_wavelength_involved: [0,        0,       ...,        0,        0]
        .
        .
        .

    characteristic[2]:
        type: particle_component_volume_fractions_linear_mixture
        retrieved: true
        mode[1]:
            initial_guess:             #1         #2        #3         #4
                value:                 [0.00001,       0.0001,      0.0001,      0.0001      ]
                min:                   [0.000001,    0.000001,   0.000001,    0.000001   ]
                max:                   [0.2,        1.0,       1.0,        1.0      ]
                index_of_wavelength_involved:   [0,        0,      0,     0      ]
            .
            .
            .
```

4.5.4. Transport models

The transport models approach constitutes the most complex methodology to model atmospheric aerosols. This approach has been designed to facilitate the interface between sophisticated aerosol transport models and GRASP code, both for forward modeling and retrieval. Normally, these complex transport models work simultaneously with a high number of different aerosol particles with specific size distributions, shape, optical characteristics, vertical distributions… In the rest of the GRASP approaches for aerosol modeling only one or two aerosol modes are considered which have all these aforementioned characteristics totally independent between each other. However, the GRASP transport model approach is not only an additional interface that enables the possibility to work with multiple independent aerosol modes. But it is also a tool that allows the conversion between the characteristics that define the particles in the aerosol transport models (as mass mixing ratio) to the normal microphysical and optical characteristics that are normally used in GRASP.

GRASP transport model approach consists of 5 main aerosol components similar to MERRA-2 and CAMS aerosol models: sulphate (SU), desert dust (DU), sea salt (SS), organic (OC) and black carbon (BC). Each component can exhibit hydrophobic or hydrophilic properties resulting in 15 aerosol tracers: hydrophilic sulphate (SU), five size bins for hydrophobic dust (DU) and hydrophilic sea salt (SS), hydrophobic and hydrophilic modes of organic (OC) and black carbon (BC) aerosol . The concentration and optical depth of each tracer in the model is refined through the mass mixing ratio. Furthermore, each of these aerosol modes count with its own vertical profile, particle size distribution, sphericity parameter and all the rest of optical characteristics. Note that the different aerosol tracers are externally mixed with vertically dependent mass mixing ratios to obtain the corresponding total aerosol optical properties.

An example of the necessary settings to define the additional aerosol parameters of the transport models can be found below:

```
forward_model:
    phase_matrix:
        size_binning_method_for_triangle_bins: logarithm
        number_of_elements: 4
        use_transport_model: true
        number_of_bins_for_lognormal_size_distribution: [ ^repeat(43;15) ]
```

```
transport_model:
    vertical_profile: column_average  #tracer_average
    tracers:     ['du1', 'du2', 'du3', 'du4', 'du5','bc1','bc2','oc1','oc2','ss1', 'ss2',
    hydrophilic: [  0,     0,     0,     0,     0,    0,    1,    0,    1,    1,     1,
    #hydrophilic: [  0,     0,     0,     0,     0,    0,    0,    0,    0,    0,     0,
    density:     [2500., 2650., 2650., 2650., 2650.,1000.,1000.,1800.,1800., 2200., 2200.
kernels_folder:  'KERNELS_BASE' # 'merra2_data_base'
radius:
    mode[1]: # du1
        min: 0.08
        max: 20.0
    mode[2]: # du2
        min: 0.08
        max: 20.0
    mode[3]: # du3
        min: 0.08
        max: 20.0
    mode[4]: # du4
        min: 0.08
        max: 20.0
    mode[5]: # du5
        min: 0.08
        max: 20.0
    mode[6]: # bc1
        min: 0.0001 #0.002
        max: 5.0
    mode[7]: # bc2
        min: 0.0001 #0.002
        max: 5.0
    .
    .
    .
```

Where “forward_model.phase_matrix.transport_model.tracers” refers to the 15 aerosol tracers. “forward_model.phase_matrix.transport_model.density” and “forward_model.phase_matrix.transport_model.hyfrophilic” mark correspondingly the density of dry tracers and their hygroscopic properties (“0” corresponds to hydrophobic and “1” corresponds to hydrophilic aerosol).

The rest of the aerosol characteristics are defined following the standard GRASP settings convention for different aerosol modes:

```
constraints:
    characteristic[1]: #1
        type: size_distribution_lognormal
        retrieved: true
        mode[1]:
            initial_guess:
                value:                        [0.6576, 0.2828]
                min:                          [0.1,   0.1]
                max:                          [3.0,   0.9]
                index_of_wavelength_involved: [^repeat(0;2)]
            single_pixel:
                a_priori_estimates:
                    lagrange_multiplier:      [1.0e-03,  1.0e-03]
                smoothness_constraints:
                    difference_order: 0
                    lagrange_multiplier: 0.0
        mode[2]:
            initial_guess:
                value:                        [1.2436, 0.25]
                min:                          [0.1,   0.1]
                max:                          [3.0,   0.9]
                index_of_wavelength_involved: [^repeat(0;2)]
            single_pixel:
                a_priori_estimates:
                    lagrange_multiplier:      [1.0e-03,  1.0e-03]
                smoothness_constraints:
                    difference_order: 0
                    lagrange_multiplier: 0.0
        mode[3]:
            initial_guess:
                value:                        [2.2969, 0.344]
                min:                          [0.1,   0.1]
                max:                          [3.0,   0.9]
                index_of_wavelength_involved: [^repeat(0;2)]
            single_pixel:
```

```
            a_priori_estimates:
                lagrange_multiplier:      [1.0e-03,  1.0e-03]
            smoothness_constraints:
                difference_order: 0
                lagrange_multiplier: 0.0
        mode[4]:
            initial_guess:
                value:                        [5.3645, 0.707]
                min:                          [0.1,   0.1]
                max:                          [6.0,   0.9]
                index_of_wavelength_involved: [^repeat(0;2)]
            single_pixel:
                a_priori_estimates:
                    lagrange_multiplier:      [1.0e-03,  1.0e-03]
                smoothness_constraints:
                    difference_order: 0
                    lagrange_multiplier: 0.0
        .
        .
        .


    characteristic[2]:
            type: tracer_level_concentration
            retrieved: true
            mode[1]:
                initial_guess:
                    value:                        [^repeat(0.007;72)]
                    min:                          [^repeat(1e-15;72)]
                    max:                          [^repeat(0.500;72)]
                    index_of_wavelength_involved: [^repeat(0;72)]
                single_pixel:
                    a_priori_estimates:
                        lagrange_multiplier:      [^repeat(0;72)]
                    smoothness_constraints:
                        difference_order: 0
                        lagrange_multiplier: 0.0
            mode[2]:
                initial_guess:
                    value:                        [^repeat(0.007;72)]
                    min:                          [^repeat(1e-15;72)]
                    max:                          [^repeat(0.500;72)]
                    index_of_wavelength_involved: [^repeat(0;72)]
                single_pixel:
                    a_priori_estimates:
                        lagrange_multiplier:      [^repeat(0;72)]
                    smoothness_constraints:
                        difference_order: 0
                        lagrange_multiplier: 0.0
            mode[3]:
                initial_guess:
                    value:                        [^repeat(0.007;72)]
                    min:                          [^repeat(1e-15;72)]
                    max:                          [^repeat(0.500;72)]
                    index_of_wavelength_involved: [^repeat(0;72)]
                single_pixel:
                    a_priori_estimates:
                        lagrange_multiplier:      [^repeat(0;72)]
                    smoothness_constraints:
                        difference_order: 0
                        lagrange_multiplier: 0.0
            mode[4]:
                initial_guess:
                    value:                        [^repeat(0.007;72)]
                    min:                          [^repeat(1e-15;72)]
                    max:                          [^repeat(0.500;72)]
                    index_of_wavelength_involved: [^repeat(0;72)]
                single_pixel:
                    a_priori_estimates:
                        lagrange_multiplier:      [^repeat(0;72)]
                    smoothness_constraints:
                        difference_order: 0
                        lagrange_multiplier: 0.0

    characteristic[3]:
            type: real_part_of_refractive_index_spectral_dependent
            retrieved: true
            mode[1]:
                initial_guess:                       #1        #2        #3        #4
                    value:                        [^repeat(1.53;7)]
                    min:                          [^repeat(1.33;7)]
                    max:                          [^repeat(1.6;7)]
                    index_of_wavelength_involved: [^repeat(0;7)]
```

```
            single_pixel:
                smoothness_constraints:
                    difference_order: 1
                    lagrange_multiplier: 1.0e+1
            multi_pixel:
                smoothness_constraints:
                    derivative_order_of_X_variability:    1
                    lagrange_multiplier_of_X_variability: 1.0e-1
                    derivative_order_of_Y_variability:    1
                    lagrange_multiplier_of_Y_variability: 1.0e-1
                    derivative_order_of_T_variability:    1
                    lagrange_multiplier_of_T_variability: 2.0e-2
        mode[2]:
            initial_guess:                      #1        #2        #3        #4
                value:                       [^repeat(1.53;7)]
                min:                         [^repeat(1.33;7)]
                max:                         [^repeat(1.6;7)]
                index_of_wavelength_involved: [^repeat(0;7)]
            single_pixel:
                smoothness_constraints:
                    difference_order: 1
                    lagrange_multiplier: 1.0e+1
            multi_pixel:
                smoothness_constraints:
                    derivative_order_of_X_variability:    1
                    lagrange_multiplier_of_X_variability: 1.0e-1
                    derivative_order_of_Y_variability:    1
                    lagrange_multiplier_of_Y_variability: 1.0e-1
                    derivative_order_of_T_variability:    1
                    lagrange_multiplier_of_T_variability: 2.0e-2
        mode[3]:
            initial_guess:                      #1        #2        #3        #4
                value:                       [^repeat(1.53;7)]
                min:                         [^repeat(1.33;7)]
                max:                         [^repeat(1.6;7)]
                index_of_wavelength_involved: [^repeat(0;7)]
            single_pixel:
                smoothness_constraints:
                    difference_order: 1
                    lagrange_multiplier: 1.0e+1
            multi_pixel:
                smoothness_constraints:
                    derivative_order_of_X_variability:    1
                    lagrange_multiplier_of_X_variability: 1.0e-1
                    derivative_order_of_Y_variability:    1
                    lagrange_multiplier_of_Y_variability: 1.0e-1
                    derivative_order_of_T_variability:    1
                    lagrange_multiplier_of_T_variability: 2.0e-2

        .
        .
        .
```

4.6. Error estimation

GRASP provides rigorous dynamic error estimates of the retrieved characteristics, but also these calculations are available for some of the derived parameters that GRASP calculates internally.

In order to perform the calculations some extra information for each measurement has to be provided in the GRASP settings file. Two different kinds of bias can be assumed: “bias_measurements_synthetic” and “bias_equation”. The units of these bias are the same as the corresponding measurement and the “error_type” option establishes if they are taken as an absolute or relative value to the measurement. An example of the definition of these settings can be found below:

```
noises:
    noise[1]:
        standard_deviation_synthetic: 0.05
        bias_measurements_synthetic: 0.05
        bias_equation: 0.05
        error_type:  relative
        standard_deviation:  0.03
        measurement_type[1]:
            type: I
            index_of_wavelength_involved: [ 1, 2, 3, 4 ]
    noise[2]:
```

```
        standard_deviation_synthetic: 0.01
        bias_measurements_synthetic: 0.01
        bias_equation: 0.01
        error_type:  absolute
        standard_deviation:  0.01
        measurement_type[1]:
            type: aod
            index_of_wavelength_involved: [ 1, 2, 3, 4 ]
```

In a similar way to other products provided by GRASP, in the “products” section the “error_estimation” group it is possible to configure how these error estimates are made with the option “using_Levenberg-Marquardt”. In order to select to what magnitudes are applied, the setting “retrieved” establishes to provide the error estimates for all retrieved characteristics, and in the “derived” group the user can select to what of the derived products (ex.: AOD, Angstrom Exponent, SSA…) these calculations will be applied:

```
products:
    error_estimation:
        using_Levenberg-Marquardt: true
        derived:
                aerosol:
                    lidar: true
                    optical_properties: true
        retrieved: true
```

The error estimation of the selected magnitudes, retrieved and derived, can be found in the end of the classic GRASP output file separated in three: “Total standard deviations”, “Standard deviations” and “BIAS - Standard deviation”. An example of an output of the error estimates can be seen below:

```
-------------------------------------------------------------------------
Total standard deviations of retrieved parameter logarithms (~relative errors) :
-------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
            1   0.84496E+00
            2   0.59295E+00
            3   0.32039E+00
            4   0.12886E+00
            5   0.10403E+00
    .
    .
    .
-------------------------------------------------------------------------
Standard deviations of retrieved parameter logarithms (~relative errors) :
-------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
            1   0.84343E+00
            2   0.35544E+00
            3   0.15784E+00
            4   0.11046E+00
            5   0.84152E-01
        .
    .
    .


-----------------------------------------------------------------------------
BIAS - Standard deviation of systematic errors of retrieved parameter logarithms :
-----------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
            1   0.50815E-01
            2   0.47461E+00
            3   0.27880E+00
            4   0.66360E-01
            5  -0.61162E-01
                .
    .
    .

INVSING = 0
```

```
-------------------------------------------------------------------------------------------
Total standard deviations of retrieved optical characteristic logarithms (~relative errors) :
-------------------------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
Wavelength (um), Aerosol Optical Depth (Random) for Particle component 1
        0.4400   0.17618E-01
        0.6750   0.68020E-01
        0.8700   0.86484E-01
        1.0200   0.82026E-01
Wavelength (um), Single Scattering Albedo (Random) for Particle component 1
        0.4400   0.59979E-01
        0.6750   0.49679E-01
        0.8700   0.25789E-01
        1.0200   0.22196E-01
-------------------------------------------------------------------------------------
Standard deviations of retrieved optical characteristic logarithms (~relative errors) :
-------------------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
Wavelength (um), Aerosol Optical Depth (Random) for Particle component   1
        0.4400   0.65241E-02
        0.6750   0.13940E-01
        0.8700   0.19455E-01
        1.0200   0.25821E-01
Wavelength (um), Single Scattering Albedo (Random) for Particle component   1
        0.4400   0.12061E-01
        0.6750   0.99402E-02
        0.8700   0.13855E-01
        1.0200   0.18953E-01
-----------------------------------------------------------------------------------------------
BIAS - Standard deviations of systematic errors of retrieved optical characteristic logarithms :
-----------------------------------------------------------------------------------------------
Date:             2014-08-22
Time:               14:58:12
Wavelength (um), Aerosol Optical Depth (Bias) for Particle mode 1
        0.4400   0.16370E-01
        0.6750   0.66575E-01
        0.8700   0.84366E-01
        1.0200   0.77988E-01
Wavelength (um), Single Scattering Albedo (Bias) for Particle mode 1
        0.4400   0.58718E-01
        0.6750   0.48694E-01
        0.8700   0.21934E-01
        1.0200   0.13280E-01
```

In order to make a proper interpretation of the error estimates provided by GRASP two considerations have to be considered. First, the random (standard deviation) and the bias components for each parameter have to be added quadratically to obtain the total error:

Secondly, GRASP operates in the logarithmic space, which includes de error estimates. Thus, some calculations are needed in order to represent together the value of the different magnitudes and the corresponding error estimates:

$\ln(a^*) = \ln(a) \pm \sigma_a$

Thus:

$a \exp(\sigma_a) = a^*_{high}$

$a \exp(-\sigma_a) = a^*_{low}$

Bibliography

[Dubovik 2011] *Atmospheric Measurement Techniques*. O. Dubovik, M. Herman, A. Holdak, T. Lapyonok, D. Tanre, J. L. Deuze, F. Ducos, A. Sinyuk, and A. Lopatin. *Statistically optimized inversion algorithm for enhanced retrieval of aerosol properties from spectral multi-angle polarimetric satellite observations*. “Statistically optimized inversion algorithm for enhanced retrieval of aerosol properties from spectral multi-angle polarimetric satellite observations”. 975-1018. 4. 5. 10.5194/amt-4-975-2011. http://www.atmos-meas-tech.net/4/975/2011/. 2011.

[Tanré 2011] *Atmospheric Measurement Techniques*. D. Tanre, F. M. Breon, J. L. Deuze, O. Dubovik, F. Ducos, and Fran. *Remote sensing of aerosols by using polarized, directional and spectral measurements within the A-Train: the PARASOL mission*. “Remote sensing of aerosols by using polarized, directional and spectral measurements within the A-Train: the PARASOL mission”. 1383-1395. 4. 7. 10.5194/amt-4-1383-2011. http://www.atmos-meas-tech.net/4/1383/2011/. 2011.

[Kokhanovsky 2010] *Atmospheric Measurement Techniques*. A. A. Kokhanovsky, J. L. Deuze, D. J. Diner, O. Dubovik, F. Ducos, C. Emde, M. J. Garay, R. G. Grainger, A. Heckel, M. Herman, I. L. Katsev, J. Keller, R. Levy, P. R. J. North, A. S. Prikhach, V. V. Rozanov, A. M. Sayer, Y. Ota, D. Tanre, G. E. Thomas, and E. P. Zege. *The inter-comparison of major satellite aerosol retrieval algorithms using simulated intensity and polarization characteristics of reflected light*. “The inter-comparison of major satellite aerosol retrieval algorithms using simulated intensity and polarization characteristics of reflected light”. 909-932. 3. 4. 10.5194/amt-3-909-2010. http://www.atmos-meas-tech.net/3/909/2010/. 2010.

Glossary

C

Cell — A spatial extent of neighbouring pixels (e.g. 2x2 or 5x5 pixels). The base of a segment.
See Also Segment, Pixel, Tile.

P

Pixel — A picture element. The smallest unit of acquisition.
See Also Cell, Segment, Tile.

S

Segment — A temporal stack of cells. The processing unit of the GRASP algorithm.
See Also Cell, Pixel, Tile.

T

Tile — A set of neighbouring segments, that can be loaded together in memory. The processing unit of the GRASP framework.
See Also Cell, Pixel, Segment.