THE SwING LIBRARY
(SoftWares for Investigating Nebulae and Galaxies)

Frédéric Galliano

AIM, CEA, CNRS, Université Paris-Saclay, Université Paris Diderot, Sorbonne Paris Cité, F-91191 Gif-sur-Yvette, France

Created on June 14, 2019; updated on July 12, 2019

Contents

Chapter 1

INTRODUCTION

The `SwING` library, standing for *SoftWares for Investigating Nebulae Galaxies*, is the best part of the library I have developed over the years to pursue my research interests. It is a combination of Fortran codes with IDL (and soon Python) wrap-ups. The present manual is written both for (motivated) users and developers wanting to use some of the functions in the library.

The library contains six directories:

Documentation/ contains this manual, as well as specific notes on technically challenging aspects (computation of grain properties, instrument filters, *etc.*). It also contains scripts, data and figures used by these various documents.

Fortran/ contains all the Fortran modules, programs and tests. These routines are the heart of the library. They implement the numerically intensive tasks.

IDL/ contains the IDL routines used for reading input/output and plotting results. This library will become obsolete when it will have been completely translated in Python. This IDL library also contains many other routines, part of my historical library, but not particularly relevant to `SwING`.

Python/ contains the Python routines for reading input/output, plotting results and performing simple pre- and post-processing. This library is still in development.

Model_templates/ contains the pre-computed model templates used to speed-up the calculation.

Demo contains several scripts demonstrating the use of the library. This directory is not mandatory but can be useful.

The Fortran library has been developed in a modular fashion, in order to have a small number of well-tested routines, with interface overloading. We read and write data using ASCII files for small files (input files, setting files, *etc.*) and HDF5 files for large data sets.

Chapter 2

INSTALLATION

2.1 Prerequisites

Before installing the SwING library, you need the following prerequisites.

1. You need a computer. If you do not have one, go purchase one.
2. A Fortran compiler and a makefile. The free compiler gfortran is a good solution.
3. The HDF5 library.
4. IDL (version 7.0 or higher).
5. ImageMagick.

2.1.1 Installation with the Python Script

The script install_fitSED.py performs the iteration. On the cluster, in a qlogin session, simply type:

```
> module add Python
> module add hdf5
> install_fitSED.py --help
```

It adds the python module to run the script. It also adds the HDF5 Fortran compiler (h5fc) that will be used to compile the codes. Finally, the last line displays the help. For a general installation, type:

```
> install_fitSED.py --install
```

and follow the instruction printed by the script to update your paths.

2.1.2 Installation by Hand

If, for a reason or another, you do not want to use the Python script, then follow the following steps.

Unpacking

You need the following directories. Put them in a directory SwING/ where you want.

1. The Fortran/ directory contains the Fortran library. It can be put anywhere. It is not a work directory.
2. The IDL/ directory contains the IDL library. Same as for Fortran/.
3. The large ($\simeq$ 30 Gb) Model_templates/ directory containing the pre-computed model templates.
4. The Demo/ directory contains several demonstration scripts. This is a work directory, where one will run codes.

Updating the paths and variables

Now, you need to update the following variables.

1. In `Fortran/Tools/utilities.f90`, you need to edit the path of the directory at the line:

   ```
   CHARACTER(*), PARAMETER, PUBLIC :: LibF = "/usr/data/Fortran/"
   ```

   In the same file, for cosmetic reasons, you can also edit the variable `programrunner` at the line:

   ```
   CHARACTER(*), PARAMETER, PUBLIC :: programrunner = "F. Galliano"
   ```

2. In `IDL/idl_init.pro`, edit the path:

   ```
   DEFSYSV, "!IDLFRED", "/usr/data/IDL/", 1
   ```

   and, in the same file, for cosmetic reasons, edit:

   ```
   DEFSYSV, "!PROGRAMRUNNER", "F. Galliano", 1
   ```

3. Make sure that your Unix `$path` variable points to the directory `Fortran/Programs/`.

HDF5 Installation

If this library is not installed:

1. Download the latest library package from: http://www.hdfgroup.org/ftp/HDF5/releases.
2. Untar the package in a directory.
3. Make sure that the mask will let these files be installed with the proper access rights (*e.g.* umask 022).
4. In the `hdf5-X.Y.Z/` directory, type:

   ```
   > sudo configure --prefix=/usr/local/hdf5 --enable-fortran --enable-cxx
   > sudo make
   > sudo make check
   > sudo make install
   > sudo make check-install
   ```

   If you want to select a particular Fortan compiler then, before executing the commands above, you must:

   ```
   > bash
   > set FC={alternate compiler}
   ```

5. Make sure `h5fc` is in the path (`/usr/local/hdf5/bin/` to be added to `$path`, if necesssary). Make sure `h5fc` has the correct rights. Make sure texttt/usr/local/hdf5/include/hdf5.mod has the correct rights.
6. To compile fortran programs with the HDF5 libraries, simply use `h5fc` instead of `gfortran`.

Compiling

Once the directories are organized and the paths are updated, you can compile the different codes. In the `Fortran/Programs` directory, type `make`. This will compile all the Fortran codes. Then, to set up the IDL library, simply create a Unix alias in your `.bashrc` file:

```
alias iidl="idl ${your_path}/IDL/idl_init.pro"
```

and call IDL with the `iidl` command.

2.1.3 At CEA: Existing Installation on the Iclust cluster

A working version of each code is currently installed on the `iclust` cluster in the directory `/dsm/herschel10/nuages/SwING/`. To use the IDL interface, you simply need to have an alias:

```
alias iidl="idl /dsm/herschel10/nuages/SwING/IDL/idl_init.pro"
```

and to use the Fortran code, make sure your `$path` points to:

```
/dsm/herschel10/nuages/SwING/Fortran/Programs/
```

Then, you should be able to run it from your own account (to be tested).

2.2 RUNNING

2.2.1 Testing the Codes with the Example Scripts

The directory `Demo/` contains several scripts aimed at testing the four main codes:

1. `compute_grain_spectrum` computes the grain properties (out-of-equilibrium emission, extinction, etc.) of different grains and different grain mixtures;

2. `simulate_SED` simulates samples of sources and their SED; this code is used to test the two following fitting codes;

3. `fitSED_chi2` performs least-squares fitting of a sample of observed SEDs with a wide variety of model components;

4. `fitSED_HB` performs hierarchical Bayesian inference on a sample of observed SEDs with the same model components as `fitSED_chi2`.

The name of the IDL interface is the same as the Fortran code, except for `fitSED.pro` which calls `fitSED_chi2.f90`, `fitSED_HB.f90` and `fitSED_HB_analysis.f90`. The IDL demonstration scripts can be run in IDL simply by typing: `.RUN demo_X.pro`, or by cutting and pasting in the command line.

2.2.2 Running the Code Without the IDL Wrap-Up

All the Fortran codes can be run without the IDL wrap-up. It will be useful mainly for the Bayesian one, which takes a long time to run, and that you might want to run on the cluster.

1. To do so, you first need to prepare the input files and the output directory by calling in IIDL the `fitSED` routine with the all the desired keywords, plus `/ONLYPREPARE`. You can easily do it on your laptop. This will create the data file `observations_fitSED.h5` containing all the necessary data, as well as the `input_*.txt` containing the settings of the run, the directory `Results_fitSED_*` where the results will be stored, and the directory `Figures_fitSED_HB/` where the figures will be created. This step is very fast and does not require a lot of memory.

2. Then, exit IDL. You can eventually move the files and directory created to another place (or to the cluster). Let's call this final directory `{path}/Work/`.

3. Either in Unix or in a `qsub` PBS script (after the option commands), type:

   ```
   cd {path}/Work/
   fitSED_HB
   fitSED_HB_analysis
   ```

   This will launch the Fortran code and the subsequent MCMC analysis.

4. Once everything is done, you can eventually move all the `{path}/Work/` directory back on your laptop. Then in IIDL, call again the `fitSED` routine with all the required keywords plus `/ONLYPLOT`. This last step will create the figures in the `Figures_fitSED_HB/` directory.

2.2.3 Analyzing the Results While the Code is Still Running

You may want to analyze the results while the code is still running to inspect the preliminary trends, find problems, anticipates convergence, *etc.*To do that, `cd` to the work directory (where `observations_fitSED.h5` is located), and type `fitSED_HB_analysis`. This code may take several hours to run depending on your sample an settings. It will provide useful statistics on screen and duplicated in `log_fitSED_analysis.txt`. Once it is done, you can eventually create the figures, by typing in IIDL:

```
IIDL> plot_fitSED_HB
```

Add the `/OVERCHI2` option to this call if you want the results to be overlaid on the previously computed least-squares results (the directory `Results_fitSED_chi2/` has to be present in the same work directory).

2.2.4 Restarting a Run that Was Interrupted

If a run was interrupted due to various reasons (electric cut, *etc.*), you can restart it where it stopped.

1. `cd` to the work directory (where `observations_fitSED.h5` is).
2. Edit the file `input_fitSED_master.txt`, by setting to `T` the value of the variable `resume`. It should look like:

   ```
   resume =                T   ! restart the run where it stopped
   ```

3. Then start the code again, simply by typing `fitSED_HB` or follow the instructions starting at item 3 of Sect. 2.2.2.

Alternatively, you can simply call `fitSED` again in IIDL, with all the required keywords, plus `/RESUME`.

2.2.5 Generating New Model Templates

In the directory `Fortran/Programs/` there are 3 codes generating large grids of templates.

1. `generate_grain_cross_section` generates the grids of grain optical properties ($Q_{\rm abs}$, $Q_{\rm sca}$, $\langle\cos\theta\rangle$, *etc.*) for all the chemical species in the library. This code is supposed to be run from the directory `Fortran/Programs/`. The results are written in `Model_templates/Cross_sections/`.
2. `generate_grain_spectra` computes the stochastic heating of individual grains of various species, for a given radiation field. It has to be run in a directory where the proper `input.txt` file exists. Usually these directories are in `Model_templates/Grain_spectra/`. If you want to generate grids with a different radiation field shape or with collisionnal heating, you have to run this code.
3. `generate_dust_models` makes large grids of ready-to-fit templates: modified black bodies, dust, stellar continuum and radio templates. If you have run `generate_grain_spectra`, you should use this code to integrate your individual grain spectra over the size distribution and perform synthetic photometry. You may also want to run this code, if you want to include new photometric filters.

Chapter 3

THE CODES

3.1 Program synthetic_photometry.f90 (BASIE)

The code synthetic_photometry.f90, nicknamed BASIE (Broadband Astronomical Spectrum Integration Emulator) for public relations, can be used with its IDL interface synthetic_photometry.pro. This interface is called the following way.

```
Fnu_synth[Nx,Ny,Nfilt] = SYNTHETIC_PHOTOMETRY(w[Nw],Fnu[Nx,Ny,Nw],
                      FILTERS=filt[Nfilt],WCEN=[Nfilt],
                      VMAT=[Nfilt,Nfilt],RMAT=[Nfilt,Nfilt],
                      SMAT=[Nfilt,Nfilt],/CALIB)
```

Performs the synthetic photometry, Fnu_synth, of a spectral cube, Fnu, mapped on the wavelength grid wave, through the list of photometric filter labels, filt, accounting for color corrections. Optionally, it can also preturn the covariance matrix of calibration uncertainties, VMAT, as well as its correlation matrix, R, and its standard deviation matrix, S. The currently available filter labels are: IRAC1, IRAC2, IRAC3, IRAC4, MIPS1, MIPS2, MIPS3, PACS1, PACS2, PACS3, SPIRE1, SPIRE2, SPIRE3, HFI1, HFI2, HFI3, HFI4, HFI5, HFI6, WISE1, WISE2, WISE3, WISE4, AKARI_IRC1, AKARI_IRC2, AKARI_IRC3, AKARI_IRC4, AKARI_IRC5, AKARI_IRC6, AKARI_IRC7, AKARI_IRC8, AKARI_IRC9, AKARI_FIS1, AKARI_FIS2, AKARI_FIS3, AKARI_FIS4, 2MASS1, 2MASS2, 2MASS3, MSX1, MSX2, MSX3, MSX4, DIRBE1, DIRBE2, DIRBE3, DIRBE4, DIRBE5, DIRBE6, DIRBE7, DIRBE8, DIRBE9 or DIRBE10.

This code is mainly a wrap-up for the functions of the module instrument_filters, synthetic_photometry and read_caliberr. More details can be found in Documentation/instrument_filters.pdf.

3.2 Program compute_grain_spectrum (BIrD)

The code compute_grain_spectrum.f90, nicknamed BIrD (Broadband and spectral InfraRed emission by Dust) for public relations, can be used with its IDL interface compute_grain_spectrum.pro. This code can compute the stochastic heating of either a single grain, or a full dust mixture. It is essentially a wrap-up for the procedure grain_stochastheat of module grain_spectrum. For the single grain case, use:

```
COMPUTE_GRAIN_SPECTRUM, /SINGLE_GRAIN, LABQABS='', LABH='', RADIUS=rad,
```

where labQabs is the label of the grain absorption efficiency, to choose among: Sil_LD93, Sil_WD01, Sil_LD01, Sil_D03, SiC_LD93, Gra_D03, Gra_LD93, ACAR_Z96, ACH2_Z96, BE_Z96, PAHi_LD01, PAHn_LD01, PAHi_DL07, PAHn_DL07, PAHi_DL07_G11, PAHn_DL07_G11, PAHi_DL07_C11, PAHn_DL07_C11, a-C_man20nm_big, a-Forst_Fe_man5nm, a-Enst_Fe_man5nm, Sil_Mg07_J03, Sil_Mg10_J03, Sil_Mg15_J03, Sil_Mg20_J03, Sil_Mg24_J03 or FeO_H95; labH is the enthalpy label, to choose among: PAH_D97, Gra_D97, Sil_DA85, PAH_DL01, Gra_DL01, Sil_DL01, a-C_man20nm, a-Sil_mix_Fe_man5nm; rad is the grain radius in μm. See Documentation/grain_properties.pdf, for more details.

For the dust mixture case, use:

```
COMPUTE_GRAIN_SPECTRUM, DUST_MIXTURE=, /ADAPTRAD, ACCRAD=, DLNR=,
```

where the choice of the dust mixture, dust_mixture, can be Z04 (Zubko et al., 2004, BARE-GR-S), G11_AC (Galliano et al., 2011, AC), C11 (Compiègne et al., 2011) or J17 (Jones et al., 2017, THEMIS). If keyword /ADARTRAD is set, then the radius sampling is adapted to ensure accuracy of the SED, accrad. Otherwise, DlnR denforces a constant logarithmic step for the radius grid.

The dust heating sources can be set with the keywords:

```
ISRF='', U=, /NOIONIZATION, T_BB={K}, Z=,
/COLLISIONS, N_EL={cm-3}, T_EL={K}, /NOSUBLIMATION,
```

where the `ISRF` label can be `Mathis83` (the Mathis et al., 1983, ISRF) or `BB`, a black body at temperature `T_BB`. Both ISRF include a CMB component, which is a black body at temperature $2.7 \times (1 + \mathtt{z})$ K. The `/NOION` suppresses the ionizing photons. If `/COLLISIONS` is set, then collisional heating by electrons, with density `n_el` (in cm^{-3}), and temperature `T_el`, in K, is included. If `/NOSUBLIMATION` is set, then grains which were supposed to sublimate are kept in the SED.

The rest of the parameters control which quantities we want to compute:

/NOSPECTRUM tells the routine to not compute the stochastic heating; it will simply compute the optical properties.

/NOOPTICAL tells the routine to not compute the optical properties.

/NOINDIVIDUAL tells the routine to not display the quantities relative to individual grains, but only the size-distribution integrated quantities.

/ONLYSPECTRUM tells the routine to output only the emitted grain spectrum.

/ONLYOPTICAL tells the routine to output only the optical properties.

FILTERS gives the list of photometric filter in which one wants the SED to be integrated; the list of currently available filter is the same as in Sect. 3.1.

Q_PAH is the PAH mass fration, before sublimation.

F_SG is the mass fraction of small grains, before sublimation, but only relative to non-PAHs grains ($f_{\mathrm{SG}} = q_{\mathrm{SG}}/(1 - q_{\mathrm{PAH}})$).

F_SIL is the mass fraction of silicate-to-non-PAHs grains ($f_{\mathrm{sil}} = q_{\mathrm{sil}}/(1 - q_{\mathrm{PAH}})$).

fionPAH is the charge fraction of PAHs before sublimation is $0 < \mathtt{fionPAH} < 1$.

/ONLYPREPARE tells the routine to simply write the input files to the Fortran code.

/ONLYPLOT tells the routine to simply plot the output of the Fortran code.

/NOPREPARE tells the routine to not write the input files.

/NOPLOT tells the routine to not produce the figures.

/NORUN tells the routine to not launch the Fortran code.

3.3 Program `simulate_SED` (TRANE)

The code `simulate_SED.f90`, nicknamed TRANE (Testing Routines of Analysis on Noised Emission models) for public relations, can be used with its IDL interface `simulate_SED.pro`. It simulates SEDs, using th pre-computed templates in `Model_templates/Dust_models/`, add noise and instrumental effects, in order to provide data sample to test fitting methods. The simulations presented by Galliano (2018) were all performed with this code.

The properties of the observational sample can be defined with:

```
SIMULATE_SED, NPIX=, /NOCALIBERR, /ROBUST_CAL, /ROBUST_RMS,
              /ADDITIVE_NOISE, /MULTIPLICATIVE_NOISE,
```

where `Npix` is the number of source in the sample, `/NOCALIBERR` suppresses calibration uncertainties, `/ROBUST_CAL` and `ROBUST_RMS` uses Student's t distribution instead of gaussian for calibration and noise uncertainties. The noise level in each pixel is randomly distributed assuming a median signal-to-noise ratio per waveband, `MEDSOVN`, and the corresponding skewness, `SKEWNESS_RMS`. The SED model components are selected with:

```
/MBB1, /MBB2, BBQ=['',], /BEMBB, /DELTAU, /POWERU, /RADIO, /STARBB,
/AGN, DIRTEMP_{component name}='', LIMITSWMOD=[wmin,wmax],
DUSTMIXT_DELTAU='', DUSTMIXT_POWERU='', FILTERS=['''],
```

which are self-explanatory. The parameter names currently available can be lnM_MBB1, lnT_MBB1, beta_MBB1, lnM_MBB2, lnT_MBB2, beta_MBB2, lnM_BBQ, lnT_BBQ, lnM_BEMBB, lnT_BEMBB, beta1_BEMBB, beta2_BEMBB, lnwb_BEMBB, lnM_deltaU, lnU_deltaU, lnq_PAH_deltaU, fionPAH_deltaU, fSG_deltaU, fsil_DeltaU, lnM_powerU, lnUm_powerU, lnDU_powerU, alpha_powerU, lnq_PAH_powerU, fionPAH_powerU, fSG_powerU, fsil_PowerU, lnL0_radio, alphas_radio, fFF_radio, lnL_starBB, lnL_AGN, th_AGN, lnR_AGN, lnVc_AGN, lnAc_AGN, lnAc_AGN and lnAd_AGN. See Galliano (2018, Sect. 2) for more details. FSG and FSIL are assumed to be fixed to their default values, unless otherwise specified. The current choices for *BBQ* components can be Sil_D03, SiC_LD93, Gra_D03, ACAR_Z96, ACH2_Z96, BE_Z96, a-C_man20nm, a-Forst_Fe_man5nm, a-Enst_Fe_man5nm, Sil_Mg07_J03, Sil_Mg10_J03, Sil_Mg15_J03, Sil_Mg20_J03, Sil_Mg24_J03 or FeO_H95 (see Documentation/grain_properties.pdf, for more details). The possible grain mixtures are the same as in Sect. 3.2. If FILTERS is not set, then a *Spitzer/Herschel* list is used. If we use /FILTERS, then all database filters are used.

The statistical distributions of the component parameters can be set through the keywords DIST{parameter name}_{component name}={av:,sig:}, where the field of the structure contains the average, av, and the standard-deviation, sig. If DIST*_*.sig=0, then the parameter is considered fixed. Parameters, external to the SED model, can also be simulated. Their name will be the vector EXTRANAME and the noise will be the vectors RMS_EXTRA and SKEWNESS_EXTRA. The correlations between the parameters can be set with:

```
CORRELATION_STRUCTURE=[{par:['',''],val:0.D}]
```

where the field par gives the name of the two parameters and val, the value of the correlation coefficient. If CORRELATION_STRUCTURE[i].val=1, then the two parameters are considered tied. If NPIX is an array, then the parameters will be drawn from several distributions. In particular, if NPIX is a vector of size Nsamp, CORRELATION_STRUCTURE.val can also be a vector of size Nsamp.

Finally, if /NEWSEED is set, a new random generator seed is used. By default SED units are in $L_\odot/Hz$, and masses in $M_\odot$. If /FLUX is set, then SED units are $W/m^2/Hz$, masses are in kg/m^2 and powers are in W/m^2.

3.4 Program fitSED_chi2 (LestER)

The code fitSED_chi2.f90, nicknamed LestER (LEaST-squares fitting of dust Emission Routine) for public relations, can be used with its IDL interface fitSED.pro, with keyword /CHI2. Its core method is presented in Appendix C of Galliano (2018).

The properties of the observational sample can be defined with:

```
FITSED, Lnu[Nx,Nw]or[Nx,Ny,Nw]or'', dLnuRMS[Nx,Nw]or[Nx,Ny,Nw],
        filters[Nw], MASK=[Nx]or[Nx,Ny]or[Nx,Nw]or[Nx,Ny,Nw], HEADER=,
        SKEWNESS_RMS=[Nx,Ny,Nw], /CHI2, /FLUX, /NOCALIB,
```

where:

Lnu is are the fluxes of the SED in $L_\odot/Hz$; it can be either a Nw vector containing a single SED as a function of wavelength, or a Nx×Nw array containing a list of Nx SEDs, or a Nx×Ny×Nw containing a Nx×Ny image for each wavelength;

dLnuRMS contains the noise; it must have the same units and size as Lnu;

mask is the mask flagging SEDs of wavelengths to be ignored during the fitting; a value of 0 means the flux will be fit, a value of 1 means the flux will not be fit; mask must have the same size as Lnu;

SKEWNESS_RMS is the optional skewness of the noise; it must have the same size as Lnu;

filters contains the list of photometric filters (*cf* Sect. 3.1 for the list of currently available labels);

HEADER contains the header of the spectral cube (an array of strings), in case the input SED is a 3D array; it is used only to produce FITS file maps of the physical parameters.

If /FLUX is set, then SED units are $W/m^2/Hz$, masses are in kg/m^2 and powers are in W/m^2. If NOCALIB is set, no calibration uncertainties are taken into account.

Several general keywords can be used:

NINIMC is the number of Monte-Carlo iterations on the initial parameter values (12 by default); initial conditions are chosen uniformly through the parameter space, the least-squares fitter then run starting from these different values, and only the best overall chi-squared value is kept;

NOINIMC forces the code to not randomly draw the initial guesses of the parameters

INITPARVAL is a `Nx×Ny×Npar` array containing the initial values of the parameters; by default these values are automatically estimated;

INITPARNAME is the corresponding list of parameter labels;

ONLYPREPARE, if set, the procedure will only write the input files for the Fortran code;

ONLYPLOT, if set, the procedure will only read the outputs of the Fortran code and make the figures.

The choice of model components and the constraints on the parameters (limiting, fixing, tying) are set the same way as in Sect. 3.3. The main analysis keywords are the following:

OVERSIMU overplotted results on the true values, in case the data have been simulated with `simulare_SED` (Sect. 3.3), and if the corresponding `Simulate_SED` dicrectory is present in the working directory;

SURFACEUNIT contains the string of character of the surface unit if the SEDs are expressed per unit area; it can be `pc2`, `m2`, *etc.*;

FIR tells the code to also compute the FIR for each dust component.

3.5 Program `fitSED_HB` (HerBIE)

The code `fitSED_HB.f90`, nicknamed HerBIE (HiERarchical Bayesian Inference for dust Emission; Galliano, 2018) for public relations, can be used with its IDL interface `fitSED.pro`, with keyword `/HB`. Technically, `fitSED_HB.f90` computes the MCMC and can run several weeks. Its results are then analyzed by `fitSED_HB_analysis.f90`, which runs faster but can use a lot of memory, depending on the size of the sample.

The Hierarchical-Bayesian specific keywords of the general `fitSED.pro` procedure are the following:

NMCMC is the length of the MCMC (default: 1 million);

NOHYPER, if set, the run is non-hierarchical (*i.e.* standard Bayesian);

NOASIS, if set, tells the code to not use the Ancillarity-Sufficiency Interweaving Strategy (Yu & Meng, 2011);

NEWSEED, if set, then the random generator will use a new seed;

ROBUST_RMS, if set, the noise will follow a Student's t distribution (default: gaussian), only if `SKEWNESS_RMS` is not present;

ROBUST_CAL, if set, the calibration uncertainties will follow a Student's t distribution (default: gaussian);

INITCHI2, if set, the MCMC will start from the best chi-squared parameters;

OVERCHI2, if set, the Bayesian results will be overplotted on the best chi-squared results.

NHDF5FILE is the number of fractionated HDF5 files in which the MCMC is written. This keyword is recommended for long runs, as a power cut during an HDF5 file writing corrupts it, and the /RESUME mechanism will not work. With this keyword, one lose only the last file in case of corruption.

PAREXTRA is a `Nx×Ny×Nextra` array; it contains the values of the external parameters;

ERREXTRA is the corresponding uncertainty on the extra parameters; it must have the same size and units as `PAREXTRA`;

SKEWNESSEXTRA is the skewness of the uncertainties on the external parameters; it must have the same size as `PAREXTRA`;

EXTRAMASK is the mask on the external parameters, with the same convention as the mask on the SED (Sect. 3.4); it must have the same size as `PAREXTRA`;

EXTRANAME contains the labels of each external parameter;

T_BURNIN is the burn-in time, excluded in the analysis (by default it is the first 10 % of the MCMC);

T_END is the last index one wants to analyze the MCMC (by default, it is the whole MCMC);

NINDIVIDUAL is number of individual SEDs to be analyzed in details;

INDIVIDUAL_SOURCES is a `Nind`×2 array containing the coordinates of the pixels one wants to analyze in detail.

3.6 Program `fitMIR` (MILES)

The code, nicknamed MILES (Mid-Infrared Line Extraction Software) for public relations, is still in development.

Chapter 4

MODULES OF GENERAL TOOLS

4.1 BASIC UTILITIES: MODULE `Tools/utilities.f90`

4.1.1 Public Variables

LI `KIND` for long integers.

DP `KIND` for double precision floats.

CDP `KIND` for double precision complex floats.

lenstrnum maximum length of the string when a number is converted to a string.

ustd default standard output unit number.

lenmax

tinyDP `TINY` for `DP` kind.

epsDP `EPSILON` for `DP` kind.

hugeDP `HUGE` for `DP` kind.

verbatim boolean setting the default amount of information printed on screen during execution (can be overwritten by specific functions).

warnings boolean setting the default amount of warnings.

programrunner name of the program runner to stamp written data.

time_type `TYPE` for printing CPU clock run times.

4.1.2 Public Routines for Code Formatting

Function `libF`

```
dir = libF()
```

Returns the machine-dependent directory where the Fortran library is.

Function `temproot`

```
dir = temproot()
```

Returns the machine-dependent directory where the model templates are.

Procedure `banner_program`

```
CALL BANNER_PROGRAM('name',unit,SWING=T/F)
```

Prints a generic banner at the start of a program. If `unit` is not present then it is `ustd`. If `SWING` is true, then the SwING banner is printed.

Procedure strike

```
CALL STRIKE(proc_name,comment)
```

Prints the error message `comment` and stops the code, `proc_name` being the name of the function where the procedure is called, for backtracking.

Procedure warning

```
CALL WARNING(proc_name,comment)
```

Prints the error message `comment`, without stopping the code, `proc_name` being the name of the function where the procedure is called, for backtracking.

Function timestring

```
"" = TIMESTRING(time)
```

Returns a formatted string printing the time in h, m, s.

Function timinfo

```
"" = TIMINFO(time0)
```

Prints the CPU time since `time0`, in a formatted string. Simply initiate the `time0` variable at the beginning of the code with `CALL CPU_TIME(time0)`. Successive calls to `TIMINFO` will print the difference between `time0` and the moment where the `TIMINFO` instruction is reached.

Function today

```
"" = TODAY()
```

Prints the date of the day in a formatted string.

4.1.3 Public Routines for String Formatting

Function trimLR

```
string = TRIMLR(char)
```

Removes the leading and trailing blanks in the chain of characters `char`.

Function trimEQ

```
bool[N] = TRIMEQ(charray[N],char)
```

Tells where an array of strings, `charray` is equal to a particular string `char`, not accounting for leading and ending blanks. If `charray` is a scalar then `bool` is a scalar.

Function pring

```
string = PRING(number,Ndec)
```

Creates a string from a given `number` (integer or double). `NDEC` is the number of decimal digits.

Functions strupcase and strlowcase

```
"" = STRUPCASE(string)
"" = STRLOWCASE(string)
```

Converts a string to all upper/lower case letters.

Function strreplace

```
news[N] = STRREPLACE(s[N],text[M],repl[M])
```

Finds every occurence of `text` in a string or array of strings (up to 3 dimensions), `s`, and replaces them by `repl`. Both `text` and `repl` can be either scalars or arrays of the same dimension.

4.1.4 Public Elementary Routines

Procedure swap

```
CALL SWAP(v1,v2)
```

Swaps the values of two variables, `v1` and `v2`, of the same types: integers, double or complex. These variables can be scalar or arrays of the same size (1 or 2 dimensions).

Function flEQ

```
bool[N] = flEQ(v1[N,M],v2[N],TOL=1.E-5)
```

Determines equality of two real numbers, `v1` and `v2`, within a tolerance interval `TOL`, true if:

$$|\mathtt{V1} - \mathtt{V2}| \leq \mathtt{TOL} \times \min(|\mathtt{V1}|, |\mathtt{V2}|). \tag{4.1}$$

If `V2` is scalar, then `V1` can be scalar, vector or matrice. Alternatively, `V1` and `V2` can be both be vectors of the same size.

Procedure incr

```
CALL INCR(x[,val])
```

Increments `X` by `VAL`, if present, else by 1. `X` can be integer or double. It can be a scalar or an array up to 3 dimensions.

Procedure scl

```
CALL SCL(x,fact)
```

Scales `x` by a factor `fact`. `X` can be integer or double. It can be a scalar or an array up to 3 dimensions.

Function outerprod

```
x[N,M] = OUTERPROD(a[N],b[M])
```

Computes the matrix `X[N,M]` as the outer product of vectors `A[N]`, `B[M]`. All the variables are doubles.

Function outerand

```
x[N,M] = OUTERAND(a[N],b[M])
```

Computes the matrix `X[N,M]` as the outer logical product of vectors `A[N]` and `B[M]`.

Function reallocate_pointer

```
pointer2 = REALLOCATE_PT(pointer1,N1[,N2])
```

Allocates or reallocates the pointer `pointer2` of size `N1` or `N1,N2` and assigns it the values of the pointer `pointer1`, of the same size. These pointers can be integers or doubles. It comes from Press et al. (2007).

Function arth

```
x[N] = ARTH(first,increment,N)
```

Returns the array `x` containing an arithmetic progression of `N` elements starting at `first` and spaced by `increment`. The series can be integer or double.

Function cumsum

```
y[N] = CUMSUM(x[N],seed)
```

Returns an array `y` containing the cumulative sum of an array `x`:

$$\mathtt{y[i]} = \sum_{j=1}^{i} \mathtt{x[j]}. \tag{4.2}$$

If `seed` is present, then the `y` is the cumulative sum of `x` plus `seed`. The arrays can be integers or doubles.

Function cumprod

```
y[N] = CUMPROD(x[N],seed)
```

Returns an array y containing the cumulative product of an array x:

$$y[i] = \prod_{j=1}^{i} x[j]. \tag{4.3}$$

If seed is present, then the y is the cumulative product of x times seed. The arrays can be integers or doubles.

Function zroots_unity

```
r = ZROOTS_UNITY(N,NN)
```

Complex function returning NN powers of the Nth root of unity.

Function adjustc

```
centered_text = ADJUSTC(text,strlen)
```

Centers a string, containing leading or trailing blanks. If strlen is present, then centered_text is of length strlen.

Function isNaN

```
bool[N] = ISNAN(val[N])
```

Decides if a double scalar or array up to 4 dimensions, val, is a NaN.

Function NaN

```
NaN = NAN(val)
```

Returns a NaN with the same type as val.

Function IsInf

```
bool[N] = ISINF(val[N])
```

Decides if a double scalar or 1 dimension array, val, is Infinity.

4.2 SIMPLE ARRAY MANIPULATION: MODULE Tools/arrays.f90

4.2.1 Public Routines for Modifying Arrays

Procedure reallocate

```
CALL REALLOCATE(array,N1,N2,N3,N4)
```

Deallocates (if already allocated) and reallocates an array of integers, doubles, complexes, booleans or strings, up to 4 dimensions.

Function ramp

```
r[N] = RAMP(N,Rinf,Rsup,XLOG=T/F,POWIND=)
```

Generates an array r of N values increasing regularly from Rinf to Rsup either linearly (default), logarithmically (XLOG=T) or following a power-law of index powerind.

Procedure ramp_step

```
CALL RAMP_STEP(Xinf,Xsup,X[N],DX,DlnX,N)
```

Generates an array X of increasing values from Xinf to Xsup, with a step DX. If DlNX is present instead of DX then the grid is logarithmic. The number of points, N is computed to ensure the minimum number of points with an actual step smaller or equal to the required one.

Function reverse

```
x[N] = REVERSE(x[N])
```

Returns an array containing the elements of double `x`, in reversed order.

Procedure incrarr

```
CALL INCRARR (arr(1D or 2D),val(scal or 1D))
```

General procedures adding elements `val` to an allocatalbe array `arr`.

- If `ARR` is a vector of size `N` and `VAL` is scalar, then the returned array `ARR` has size `N+1`, and its new element `ARR[N+1]` is `VAL`.
- If `ARR` is a vector of size `N` and `VAL` is a vector of size `M`, then the returned array `ARR` has size `N+M`, and its new elements `ARR[N+1:N+M]` are `VAL[1:M]`.
- If `ARR` is a matrix of size `N`×`M`, and `VAL` is a vector of size `M`, then the returned `ARR` has size `N+1`×`M` and its new elements `ARR[N+1,1:M]` are `VAL[1:M]`.

The arrays must have the same type, either integer, double, boolean or character.

4.2.2 Public Routines for Searching Arrays

Function closest

```
i = CLOSEST(x[N],val)
```

Returns the index `i` corresponding to the closest `x[i]` value of `x` to `val`. It works with integers and doubles. If `val` is a vector, then `i` is an array of the same size. If several values are equidistant, then the returned index is the one of the first encountered element.

Function uniq_sorted

```
bool = UNIQ_SORTED(x[N],Nsort(OUT),FIRST=T/F,LAST=T/F)
```

Returns the mask of the non repeated elements `x[N]`, previously sorted. `Nsort` is the number of unique elements. Flags `FIRST/LAST` decide if the first or last occurence is kept. Array `x` can be integer or double.

Function sort

```
list_sorted = SORT(list,IND=indices)
```

Routine of quick sorting using the Haore algorithm, from: Brainerd, W.S., Goldberg, C.H. & Adams, J.C. (1990) "Programmer's Guide to Fortran 90", pages 149-150, modified by Alan Miller. The index of the sorted list are returned in `IND`, if present. Warning: this routine crashes if there are NaNs in the list.

Procedure print_vec

```
CALL PRINT_VEC(v1[N1],v2[N2],v3[N3],v4[N4],PRECISION=2)
```

Prints one or several vectors in column, for visualizing data.

Procedure iwhere

```
CALL IWHERE(bool[N],ind[M])
```

Return the indices where vector `BOOL` is true, in `IND`. If we pass `IND` as a vector then it contains all the occurences, otherwise, if it is passed as a scalar, then the first occurence is returned.

4.3 INPUT/OUTPUT: MODULE `Tools/inout.f90`

4.3.1 Public Variables

ascext is the default extension for ASCII files.

binext is the default extension for binary files.

h5ext is the default extension for HDF5 files.

textwid is the default line length for ASCII files.

unitdata is the default output unit for ASCII files.

Ndecim_def is the default number of decimals to display float numbers is ASCII files.

lenpar is the default length of the string of characters to print parameter values in the special formatted input files used by `read_input_line`.

lenline is the default length of the lines in the special formatted input files used by `read_input_line`.

lenpath is the default length of the string of characters to print paths in the special formatted input files used by `read_input_line`.

4.3.2 Public Routines for Reading and Writing ASCII Files

Procedure `write_ASCII`

```
CALL WRITE_ASCII(file,vec1[N],vec2[N],vec3[N],vec4[N],vec5[N], &
                 mat1[N,M],mat2[N,M],mat3[N,M],mat4[N,M], &
                 comgen="", &
                 comvec1="",comvec2="",comvec3="",comvec4="", &
                 comvec5="", &
                 commat1="",commat2="",commat3="",commat4="", &
                 IDL=T/F)
```

Writes vectors and matrices to a customised ASCII file with a header and data blocks. `FILE` is the name of the ouput ASCII file. `VEC1` to `VEC5` are vectors of size `N`. `MAT1` to `MAT3` are matrices of size `N×M`. `Comgen` is a string that will be written as a general comment, the other `com*` are comments specific to each array. The boolean `IDL`, if true, prevents exponents larger than 99 to be printed (NaN instead).

Procedure `read_ASCII`

```
CALL READ_ASCII (file,vec1,vec2,vec3,vec4,vec5,mat1,mat2,mat3,mat4)
```

Reads the customised ASCII files with a header and data blocks, written by `write_ASCII`.

4.3.3 Public Routines for Reading and Writing Single HDF5 Files

Subroutine `write_HDF5`

```
CALL WRITE_HDF5(DBLARR{1-6}D or STRARR1D or INTARR{1-3}D,FILE="", &
                COMPRESS=T/F,INITDBLARR=[],INITINTARR=[], &
                APPEND=T/F,NAME="",IND1=[idim1_inf,idim1_sup], &
                IND2=[idim2_inf,idm2_sup],IND3=[idim3_inf,idm3_sup], &
                IND4=[idim4_inf,idm4_sup],IND5=[idim5_inf,idm5_sup], &
                IND6=[idim5_inf,idim6_sup],UNIT=)
```

Writes a 1D to 6D double precision array, `DBLARR{1-6}D`, or a 1D string array, `STRARR1D`, or a 1D to 3D integer array, `INTARR{1-3}D`, to an HDF5 file, `FILE`, with or without compression (optional boolean `COMPRESS`). If `APPEND` is False (default) then a new file is created, otherwise the array is appended to the existing file. If the corresponding `INIT*` is set, then the array is just initialized, not filled. It can be filled by a subsequent call, using the `IND*` keywords, corresponding to the initial and final indices in each dimension of an existing array in the file where the input array will be written. The value of `INIT*` is the list of dimensions (profile of the array). `NAME` is the name of the field identifying the array in the HDF5 file and should not contain any "/", "[" or "]".

Procedure read_HDF5

```
CALL READ_HDF5(DBLARR{1-6}D or STRARR1D or
               INTARR{1-3}D,FILE="",NAME="",
               IND1=[idim1_inf,idim1_sup], &
               IND2=[idim2_inf,idim2_sup],IND3=[idim3_inf,idim3_sup], &
               IND4=[idim4_inf,idim4_sup],IND5=[idim5_inf,idim5_sup], &
               IND6=[idim6_inf,idim6_sup],N1=,N2=,N3=,N4=,N5=,N6=)
```

Reads a 1D to 6D double precision array, `DBLARR{1-6}D`, or a 1D string array, `STRARR1D`, or a 1D to 3D integer array, `INTARR{1-3}D`, from an HDF5 file, `FILE`. `NAME` is the name of the field identifying the array in the HDF5 file. A sub-array can be read if the `IND*` keywords are set to the initial and final indices in each dimension. The size of the array in each dimension can be retrieved using the `N*` keywords.

Procedure getdim_HDF5

```
CALL GETDIM_HDF5(FILE="",NAME="",N1=,N2=,N3=,N4=,N5=)
```

Reads the dimensions of an array, identified by `NAME`, in an HDF5 file, `FILE`. The size in each dimension is returned by the `N*` variables.

Procedure check_HDF5

```
bool = CHECK_HDF5(file)
```

Returns False is the HDF5 file, `file`, is corrupted.

4.3.4 Public Routines for Managing a List of Fractionated HDF5 Files

When arrays are very big, writing them to a single HDF5 file is not a good idea, as if the file is corrupted by improper code stop or during file transfer, all the data can be lost. We have come up with a simple protocol to write single large arrays to a list of multiple *fractionated* HDF5 files.

Procedure ind_HDF5_frac

```
CALL IND_HDF5_FRAC(N,Nfile,IFIRST=[Nfile],ILAST=[Nfile])
```

Simple function to set the list of first and last indices of fractionated files, to be used by `WRITE_HDF5_FRAC` and `READ_HDF5_FRAC`. The input is the number of files, `Nfiles`, we want the output to be spread over. The indices are sorted and contiguous: `IFIRST[1]`=1, `IFIRST[i]`$\leq$`ILAST[i]`, `IFIRST[i]`=`ILAST[i-1]`+1 and `ILAST[Nfiles]`=`N`.

Procedure write_HDF5_frac

```
CALL WRITE_HDF5_FRAC(IFIRST=[Nfile],ILAST=[Nfile], &
                     DBLARR{1-5}D or STRARR1D or
                     INTARR{1-3}D,FILE=[Nfile], &
                     COMPRESS=T/F,INITDBLARR=[],INITINTARR=[], &
                     APPEND=T/F,NAME=[""],IND1=[idim1_inf,idim1_sup], &
                     IND2=[idim2_inf,idm2_sup],IND3=[idim3_inf,idm3_sup],&
                     IND4=[idim4_inf,idm4_sup],IND5=[idim5_inf,idm5_sup],&
                     UNIT=)
```

Writes a 1D to 5D double array, `DBLARR{1-5}D`, or a 1D string array, `STRARR1D`, or a 1D to 3D integer array, `INTARR{1-3}D` to a fractionated list of HDF5 files, `FILE`. `IFIRST` and `ILAST` are the first and last indices of the array in each file (call `IND_HDF5_FRAC` to set them). It is assumed that the dimension to be fractionated is the last dimension the array (size `N`, corresponding the value entered when intially calling `ind_HDF5_frac`. An array written with this function must have been initialized first with the `INIT*` keywords. Other keywords are similar to `write_HDF5_frac`.

Procedure read_HDF5_frac

```
CALL READ_HDF5_FRAC(DBLARR{1-5}D or STRARR1D or
                    INTARR{1-3}D,FILE=[Nfile], &
                    IFIRST=[Nfile],ILAST=[Nfile],NAME="", &
                    IND1=[idim1_inf,idim1_sup], &
                    IND2=[idim2_inf,idm2_sup], &
                    IND3=[idim3_inf,idm3_sup], &
                    IND4=[idim4_inf,idm4_sup], &
                    IND5=[idim5_inf,idm5_sup], &
                    N1=,N2=,N3=,N4=,N5=)
```

Reads an array from a list of fractionated HDF5 files. Keywords are similar to `write_HDF5_frac`.

4.3.5 Public Routines for Reading and Writing Binary Files

Procedure write_binary

```
CALL WRITE_BINARY(array,FILE=file)
```

Writes a 1D to 4D double array, `array` to a binary file, `file`. There is only one array per file. This binary files are meant to be used for fast data exchange in the same platform, but are not portable. For portable binary format, use HDF5.

Procedure read_binary

```
CALL READ_BINARY(array,FILE=file)
```

Reads a 1D to 4D double array, `array` to a binary file, `file`.

4.3.6 Read and Edit Formatted Input Files

We have designed procedures to read and edit files containing single parameter values. These files are used as input for codes. They are also readable and can be edited by hand, for that reason.

Procedure read_input_line

```
CALL READ_INPUT_LINE(unit,parname,parval,iostat)
```

Reads the lines of an input file and returns the name of the parameter, `parname`, and its value (a string for all types), as `parval`.

Procedure edit_input_line

```
CALL EDIT_INPUT_LINE(parname,inputfile,outputfile, &
                     parval_DP,parval_Int,parval_Bool,parval_Str, &
                     verbose)
```

Edits a line in a formatted parameter file, `inputfile`, and overwrites it unless `outputfile` is present. The line in question corresponds to the parameter `parname`. Its value is changed to `paraval_*`, depending on its type.

Chapter 5

MODULES FOR MATHEMATICAL OPERATIONS

5.1 INTERPOLATION: MODULE Math/interpolation.f90

5.1.1 Public Function interp_lin_sorted

```
y_new[M] = INTERP_LIN_SORTED(y_old[N],x_old[N],x_new[M], &
                             XLOG=T/F,YLOG=T/F,FORCE=T/F)
```

Performs fast logarithmic or linear (default) interpolation (xlog and ylog keywords) of a tabulated function, y_old, at new points x_new, where x_old must have been previously sorted. If x/ylog is used, the corresponding array must be positive; no control is done. Keyword FORCE forces to zero the values that are NaN (usually LOG(0)). The value where to interpolate, x_new can be either scalar or vector and the output, y_new will have the same dimension.

5.1.2 Public Function locate_sorted

```
index = LOCATE_SORTED(xx[N],x)
```

Finds the index position of the x double value in the xx double array, by bisection, provided that the array has been previously sorted.

5.1.3 Public Function interp_poly

```
y_new[M] = INTERP_POLY(y_old[N],x_old[N],x_new[M],dy_new[M],DEGREE, &
                       XLOG=T/F,YLOG=T/F)
```

Evaluates the tabulated function (x_old,y_old) into the new array x_new and returns its values into y_new and error into err_y, using the Neville's algorithm. The degree of the polynomial can be set if degree is present (default is 2). The interpolation can be performed in any combination of linear/logarithmic space, via booleans XLOG and YLOG, both flase by default. The value where to interpolate, x_new can be either scalar or vector and the output, y_new will have the same dimension.

5.1.4 Public Function interp_spline

```
y_new[M] = INTERP_SPLINE(y_old[N],x_old[N],x_new[M],XLOG=T/F,YLOG=T/F)
```

Evaluates the tabulated function (x_old,y_old) into the new array x_new and returns its values into y_new, using cubic spline interpolation. x_new must be sorted and in ascending order. The interpolation can be performed in any combination of linear/logarithmic space, via booleans XLOG and YLOG, both flase by default.

5.2 INTEGRATION: MODULE Math/integration.f90

5.2.1 Public Function dPrimitive

```
dF[N] = DPRIMITIVE(x,f,XLOG,YLOG,lnx,lnf)
```

Returns the differential primitive of a function, f tabulated on a given grid, x, for various combinations of linear/logarithmic via keywords xlog and ylog. Log(x), lnx, and LOG(f), lnf, can eventually be passed as argument to avoid having to comput them twice.

5.2.2 Public Function integ_tabulated

```
I = INTEG_TABULATED(x[N],f[N],xrange=[x0,x1],PRIMITIVE[N],XLOG,YLOG, &
                    FORCE,RESCALE)
```

Computes the integral of a function, f, on a tabulated grid, x, using the trapezium method in various combinations of xlog, ylog. If xrange is not present, then the integral s evaluated between $\min(x)$ and $\max(x)$. X must be SORTED without any duplicates. Keyword force forces the NaNs to 0. If rescale is true (default), all computations are done on $f/\max(f)$ and the results are then remultiplied by $\max(f)$, to avoid numerical issues, in case of extremelly small or large values. If present, primitive returns the actual primitive function tabulated on x.

5.2.3 Public Function integ_romb

```
I = INTEG_ROMB(func(x),a,b)
```

Returns the integration of a user-defined function f(x) from a to b>a, using the Romberg method of order k=5.

5.3 ADAPTATIVE GRID: MODULE Math/adaptative_grid.f90

5.3.1 Public Function gridadapt1D, simple case

```
CALL GRIDADAPT1D(xcoarse[N0],xfine[N],yfine[N],yfunc,accuracy,&
                 ABSACC=T/F,INTERP=T/F,INTEG=T/F,primitive[N], &
                 XLOG=T/F,YLOG=T/F,REEVALUATE=T/F,MINSTEP=, &
                 LNFUNC=T/F,RESCALE=T/F,ADJUSTXLIM=[T/F,T/F], &
                 SCALING=,NMAX=)
```

Designs an adaptative grid, xfine of size N. This grid is adaptated to user-defined function, yfunc:

```
  FUNCTION yfunc(x)
    REAL(DP), DIMENSION(:), INTENT(IN) :: x
    REAL(DP), DIMENSION(SIZE(x)) :: yfunc
  END FUNCTION yfunc
```

An coarse initial grid, xcoarse, must be provided. The fine grid is refined in the range defined by xcoarse. The function values at the refined grid points, yfine, is returned. If INTERP is true or INTEG is flase, the grid is adapted so that the accuracy on the linear interpretation between two values is better than ACCURACY. If ABSACC is true, then it is an absolute accuracy (default is relative). If INTEG is true or INTERP is false, the accuracy is on the integral and not on the interpolation. If keyword REEVALUATE is set, then the function is evaluated on the entire grid at each iteration, while it is evaluated only at mid-points if it is not set. REEVALUATE is used when the function needs the entire grid to be evaluated properly (normalization, *etc.*). If keyword SLIM is set, then we keep only the points needed to achieve the required accuracy, and not necessary the last midpoints. This keyword is useful when one want use the computed grid later. This way the grid has the minimal size. If LNFUNC is true, then YFUNC is supposed to return ln(Y) instead of Y. If RESCALE, then YFUNC is rescaled at each iteration so that its maximum sampled is 1. In the end the results YFINE and PRIMITIVE are affected by this rescaling. This scaling factor can be recovered through the SCALING argument. This function is useful when sampling a non normalized probability distribution.

5.3.2 Public Function gridadapt1D, extra parameter case

```
CALL GRIDADAPT1D(xcoarse[N0],zgrid[Nz],xfine[N],yfine[N,Nz], &
                 yfunc,accuracy,ABSACC=T/F,INTERP=T/F, &
                 INTEG=T/F,REEVALUATE=T/F,primitive[N],slim)
```

Same function as above, but in the case where the user-defined function depends of an additional parameters, zgrid:

```
  FUNCTION yfunc (x,z)
    REAL(DP), DIMENSION(:), INTENT(IN) :: x, z
    REAL(DP), DIMENSION(SIZE(x),SIZE(z)) :: yfunc
  END FUNCTION yfunc
```

In this case, the accuracy criterion at a given x, must be true for all the zgrid values. It can correspond *e.g.* to the sampling in wavelength (X) of a black body spectrum, that has to be accurate for all temperatures (Z). All other keywords are similar as above.

5.4 DERIVATIVES: MODULE Math/derivatives.f90

5.4.1 Public Function gradient

```
dFdx[N] = GRADIENT(func,x,UP=T/F,DOWN=T/F,BOTHSIDES=T/F,EPSFCN=)
```

The returned dFdx, vector or scalar, is the gradient of the user defined function FUNC, with respect to X (vector or scalar):

```
FUNCTION func (x)
  REAL(DP), DIMENSION(:), INTENT(IN) :: x
  REAL(DP) :: func
END FUNCTION func
```

It is computed using first order finite differences:

UP: $dF/dx = (F(x+h) - F(x))/h$;

DOWN: $dF/dx = (F(x) - F(x-h))/h$;

BOTHSIDES (default): $dF/dx = (F(x+h) - F(x-h))/2h$.

5.5 DISTRIBUTIONS: MODULE Math/distributions.f90

5.5.1 Public Functions Implementing Common Distributions

Function dist_power

```
f[N] = DIST_POWER(x[N],alpha,xinf,xsup)
```

Returns a normalised power-law distribution of index alpha, between xinf and xsup, for the input array x. It is singular in alpha $= -1$.

Function dist_gauss

```
f[N] = DIST_GAUSS(x[N],xmean,sigma)
f[N] = DIST_GAUSS(x[N,M],xmean[M],invcov[M,M],detcov)
```

Returns a normalised Gauss distribution. If xmean is a scalar, then it returns a 1D distribution centered on xmean (default 0), with a standard deviation sigma (default 1). In this case, x can either be a scalar or a vector; the result will have the same dimension. If xmean is a vector, then it returns a 2D distribution with mean vector, xmean, and inverse covariance matrix, invcov. If detcov is provided, then the determinant of the covariance matrix is not recomputed. In this case, x can either be a vector with the same size as xmean, or a matrix where the second dimension has the same size as xmean.

Function dist_skewnorm

```
f[N] = DIST_SKEWNORM(x[N],ksi,omega,alpha)
```

Returns a normalised skew-normal distribution with position parameter ksi, scale parameter omega and shape parameter alpha. Both x and f must have the same size and can be either scalars or vectors.

Function param_skewnorm

```
CALL PARAM_SKEWNORM(ksi,omega,alpha,mean,sigma,skewness,param2moment, &
                    moment2param)
```

Conversion between parameters of the skew-normal distribution and its moments. You need to input either the three parameters, ksi, omega and alpha, or the three moments, mean, sigma and skewness. The booleans param2moment and moment2param determine in which sense the conversion is performed. All parameters and moments must have the same size and can be either scalars or up to 3 dimension arrays.

Function dist_splitnorm

```
f[N] = DIST_SPLITNORM(x[N],ksi,omega,alpha)
```

Returns a normalised split-normal distribution with position parameter mu, scale parameter lambda and shape parameter tau. Both x and f must have the same size and can be either scalars or vectors.

Function param_splitnorm

```
CALL PARAM_SPLITNORM(mu,lambda,tau,mean,sigma,skewness,param2moment, &
                     moment2param)
```

Conversion between parameters of the split-normal distribution and its moments. You need to input either the three parameters, `mu`, `lambda` and `tau`, or the three moments, `mean`, `sigma` and `skewness`. The booleans `param2moment` and `moment2param` determine in which sense the conversion is performed. All parameters and moments must have the same size and can be either scalars or up to 3 dimension arrays.

Function dist_lognormal

```
f[N] = DIST_LOGNORMAL(x[N],lnxmean,sigma)
```

Returns a normalised log-normal distribution centered on `lnxmean`, with a standard deviation `sigma`. Careful, the variable `lnxmean` and `sigma` are the gaussian parameter of the distribution of `LOG(x)`, but the input variable `x` is mapped in the linear space. Both `x` and `f` must have the same size and can be either scalars or vectors.

Function dist_lorentz

```
f[N] = DIST_LORENTZ(x[N],xmean,width)
```

Returns a normalised Lorentz distribution centered in `xmean`, with a width `width`. Warning, the standard-deviation of a Lorentzian is undefined. Both `x` and `f` must have the same size and can be either scalars or vectors.

Function dist_splitlorentz

```
f[N] = DIST_SPLITLORENTZ(x[N],mu,lambda,width)
```

Returns a normalised split-Lorentz distribution, with position parameter, `mu`, scale parameter, `lambda` and shape parameter, `tau`. Both `x` and `f` must have the same size and can be either scalars or vectors.

5.5.2 Public Functions for Binning Data

Procedure histogram1D

```
CALL HISTOGRAM1D (x[N](IN),xbin[Nb](OUT),pbin[Nb](OUT),xlimits, &
                  Nperbinmax(IN)])
```

Makes an histogram of the array `X` of size N. `XBIN` contains the center of each bin and `PBIN` their probability distribution (number-per-bin/N). The grid is regular and depends on the parameter `Nperbinmax` (30 by default). The limits of the binned range can be enforced with `xlimits`. If some values are outside `xlimits`, the histogram will ignore them.

Procedure histogram1Dmulti

```
CALL HISTOGRAM1DMULTI(x[N,M](IN),xbin[Nb](OUT),pbin[Nb,M](OUT),xlimits,
                      Nperbinmax(IN)])
```

Make an histogram of the 2D array array `X`, binning only along its first dimension, `N`, and keep the second dimension, `M`, unchanged. As a results, it produces `M` histograms, all binned in the same grid, `xbin` The other keywords are the same as for `histogram1D`.

Procedure histogram2D

```
CALL HISTOGRAM2D(x[N](IN),y[N](IN),xbin[Nb](OUT),ybin[Nb](OUT),
              pbin[Nb,Nb](OUT),Nperbinmax(IN),xlimits(IN),ylimits(IN)])
```

Makes a 2D histogram (joint density distribution) of the arrays `X` and `Y`. The returned distribution, `pbin` is a function of the binned parameters, `xbin` and `ybin`. The other keywords are the same as for `histogram1D`.

5.6 COMMON FUNCTIONS: MODULE Math/special_functions.f90

5.6.1 Public Function expm1

```
EXP(X) - 1 = EXPM1(x)
```

Numerically accurate development of exp(x)-1 around 0, using the expansion of Abramowitz & Stegun (4.2.45). It is suposed to be accurate at 2×10^{-10}. X can be a scalar or a vector.

5.6.2 Public Function factorial_small

```
n! = FACTORIAL_SMALL(n)
```

Determines the factorial of a small integer as an integer.

5.6.3 Public Function factorial

```
n! = FACTORIAL(n)
```

Determines the approximate factorial of an integer as a real number, using the Gamma function. N can be either scalar or vector.

5.6.4 Public Function lngamma

```
y = LOG(GAMMA(x))
```

Log of gamma function for scalar or vector x> 0.

5.6.5 Public Function igamma

```
IGAMMA(a,x) = 1/Gamma(a) * int(EXP(-t)*t^(a-1),{t=[0,x]}), a>0
```

Incomplete gamma function of a scalar or vector x. We must have x≥ 0 and a> 0. The preliminary private functions GSER et GCF return the two forms of the incomplete gamma function $P(a,x)$ and $Q(a,x) = 1 - P(a,x)$, with different numerical methods, depending on the parameter values.

5.6.6 Public Function igammac

```
IGAMMAC(a,x) = 1/Gamma(a) * int(EXP(-t)*t^(a-1),{t=[x,infty]}), a>0
```

Complementary incomplete gamma function, $Q(a,x) = 1 - P(a,x)$. We must have x≥ 0 and $a > 0$.

5.6.7 Public Function betacf

```
y[N] = BETACF(a[N],b[N],x[N])
```

Evaluates the continued fraction for the incomplete beta function by modified Lenz's method. The variables a, b and x must have the same size, either scalars or vectors.

5.6.8 Public Function ibeta

```
y[N] = IBETA(a[N],b[N],x[N])
```

Evaluates the incomplete beta function. The variables a, b and x must have the same size, either scalars or vectors.

5.7 RANDOM VARIABLES: MODULE Math/random.f90

5.7.1 Public Procedure generate_newseed

```
CALL GENERATE_NEWSEED(fileIN,fileOUT)
```

Generates a new seed for the built-in random number generator, random_number and all the functions using, including the ones below. If FILEIN and/or FILEOUT are present, then the sequence of integers constituting the seed is read/written, and can therefore be save/re-used. To ensure the seed is different at each run, we initialize it using the date and time of the excution.

5.7.2 Public Function for Univariate Distribution

Function rand_exp

```
x = RAND_EXP(N,tau)
```

Generates random variables following an exponential law with a timescale `tau`. If `N` is present, the returned `x` is an array of size `N`, otherwise it is a scalar. Taken from http://users.bigpond.net.au/amiller/.

Function rand_norm

```
x = RAND_NORM(N,M,center,sigma)
```

Generates random variables following a normal law with mean `center` and standard-deviation `sigma`. If `N` and `M` are present, the returned `x` is an array of size `N`×`M`, if only `N` is present, `x` is an array of size `N`, otherwise `x` is a scalar. Taken from http://people.sc.fsu.edu/~jburkardt/f_src/normal/normal.f90, tested and vectorized.

Function rand_skewnorm

```
x = RAND_SKEWNORM(N,M,ksi,omega,alpha)
```

Generates random variables following a skew-normal law with position parameter `ksi`, scale parameter `omega` and shape parameter `alpha`. If `N` and `M` are present, the returned `x` is an array of size `N`×`M`, if only `N` is present, `x` is an array of size `N`, otherwise `x` is a scalar. Extracted from the MATLAB library.

Function rand_splitnorm

```
x = RAND_SPLITNORM(N,M,mu,lambda,tau)
```

Generates random variables following a split-normal law with position parameter `mu`, scale parameter `lambda` and shape parameter `tau`. If `N` and `M` are present, the returned `x` is an array of size `N`×`M`, if only `N` is present, `x` is an array of size `N`, otherwise `x` is a scalar.

Function rand_student

```
x = RAND_STUDENT(N,M,Df,center,sigma)
```

Returns random variables following a Student's t law with mean `center` and standard-deviation `sigma`, with `Df` degrees of freedom. Taken from http://www.netlib.org/random/. If `N` and `M` are present, the returned `x` is an array of size `N`×`M`, if only `N` is present, `x` is an array of size `N`, otherwise `x` is a scalar.

Function rand_poisson

```
x = RAND_POISSON(N,tau[,first])
```

Returns an array of size `N` containing random variables following a Poisson law with timescale `tau`. Taken from http://users.bigpond.net.au/amiller/.

Function rand_general

```
x[N] = RAND_GENERAL(N,func,limits,xlog,ylog,lnfunc,verbose,limited, &
                    accuracy,Nmax,Nsamp,xgrid,pgrid,Fgrid,proper)
```

Draw a random sample from a user defined distribution, `func`:

```
  FUNCTION func (x)
    REAL(DP), DIMENSION(:), INTENT(IN) :: x
    REAL(DP), DIMENSION(SIZE(x)) :: func
  END FUNCTION func
```

It returns an array of size `N`. The range can be truncated if `limited`, a 2 elements boolean array, and `limits`, a 2 elements double array, are present. The first element of these arrays corresponds to the lower limit and the second one, to the upper limit. If `limited[1/2]` is True, then the distribution is truncated down/up to `limits[1/2]`. It means the distribution is renormalized in the in the truncation interval, there is no accumulation of data at the edges. If `N` is not present, the returned value is a scalar.

This function uses `gridadapt1D`, and several of its keywords are available. The numerical efficiency can be improved by playing with the following booleans:

x/ylog: if true, the adaptative grid is sampled in the log space;

lnfunc: if true, then it is assumed that `func` is actually the natural logarithm of the function one wants to sample from.

The default relative accuracy is 10^{-3}; it can be overwritten with `accuracy`. The maximum size of the grid on which the distribution is mapped can be set with `Nmax`.

Optional returned values

5.7.3 Public Functions for Multivariate Distributions

Function rand_multinorm

```
x = RAND_MULTINORM(N,covar[M,M],mu)
```

Returns random variables following a multivariate normal distribution of covariance matrix `covar` and average vector `mu` (of size `M`). If `N` is present, then the returned variables are of size `M×N`, otherwise, they are of size `M`.

Function rand_multistudent

```
x = RAND_MULTISTUDENT(N,Df,covar[M,M],mu)
```

Returns random variables following a multivariate Student's t distribution of covariance matrix `covar` and average vector `mu` (of size `M`), with `Df` degrees of freedom. If `N` is present, then the returned variables are of size `M×N`, otherwise, they are of size `M`.

5.8 STATISTICS: MODULE Math/statistics.f90

5.8.1 Public Functions for Estimating Moments

Function mean

```
m(X) = MEAN(X[],weight[],mask[],dim)
```

Computes the mean of the variable `X`, with optional weights `weight`. An optional mask, the boolean `mask`, can be supplied to exclude some values in `X`. The arrays `X`, `weight` and `mask` must have all the same size (1 to 4D arrays). If `weight` is not present, the distribution is assumed to be uniform.

If `dim` is present, then the mean is estimated only along the dimension `dim`. The result is thus not a scalar, but a (N-1)D array, N being the number of dimensions of `X`.

Function sigma

```
s(X) = SIGMA(X,dim,mask)
     or SIGMA(X[N],weight[N])
```

Computes the standard-deviation of the variable `X`, with optional weights `weight`. An optional mask, the boolean `mask`, can be supplied to exclude some values in `X`. The arrays `X`, `weight` and `mask` must have all the same size (1 to 4D arrays). If `weight` is not present, the distribution is assumed to be uniform.

If `dim` is present, then the mean is estimated only along the dimension `dim`. The result is thus not a scalar, but a (N-1)D array, N being the number of dimensions of `X`. Currently, keywords `dim` and `weight` are incompatible.

Function moment_data

```
m(X) = MOMENT_DATA(X[N],order,weight[N])
```

Computes the moment of order `order` of the vector `X`, with optional weights, `weight`. If `weight` is not supplied then the distribution is assumed to be uniform.

Function median_data

```
m = MEDIAN_DATA(X[1D to 4D],dim=,mask)
    MEDIAN_DATA(X[N],weight[N],mask)
```

Computes the mean of the variable `X`, with optional weights `weight`, using part of the Hoare sorting method. An optional mask, the boolean `mask`, can be supplied to exclude some values in `X`. The arrays `X`, `weight` and `mask` must have all the same size (1 to 4D arrays). If `weight` is not present, the distribution is assumed to be uniform.

If `dim` is present, then the mean is estimated only along the dimension `dim`. The result is thus not a scalar, but a (N-1)D array, N being the number of dimensions of `X`.

Procedure median_conf

```
CALL MEDIAN_CONF(X[1-4D],CONF=0.99,x0,dxinf,dxsup)
```

Computes the median of a array X, up to four dimensions, returned in x0, and the lower and upper limits corresponding to a confidence inter val of CONF (default 99 %), returned in dxinf and dxsup. In other words, the array X has order×100 % of its points in the interval $x0^{+dxsup}_{-dxinf}$.

Function correlate

```
r = CORRELATE(x[N],y[N],mask[N])
```

Computes the Pearson correlation coefficient of two arrays, X and Y, up to 4 dimensions. The optional argument, mask must have the same size as X and Y.

5.8.2 Public Routines for Managing Correlation Matrices

Procedure N_corr

```
Ncorr = N_CORR(Npar)
```

Number of unique correlations for Npar parameters.

Function Rmat2corr

```
corr[Ncorr] = RMAT2CORR(Rmat[Npar,Npar],Npar,Ncorr)
```

Extract the correlation coefficients, corr, from the correlation matrix Rmat (doubles, booleans or strings). The number of parameters or size of the matrix, Npar, as well as the number of correlations, Ncorr= Npar!/2(Npar−2)!, have to be passed, in order to avoid recomputing them.

Function corr2Rmat

```
Rmat[Npar,Npar] = CORR2RMAT(corr[Ncorr],Npar)
```

Build a correlation matrix, Rmat, from its correlation coefficients, corr (doubles, booleans ort strings). The number of parameters or size of the matrix, Npar, has to be passed, in order to avoid recomputing it.

Function Vmat2corr

```
corr[Ncorr] = VMAT2CORR(Vmat[Npar,Npar],Npar,Ncorr)
```

Extract the correlation coefficients, corr, from the covariance matrix Vmat (doubles, booleans or strings). The number of parameters or size of the matrix, Npar, as well as the number of correlations, Ncorr= Npar!/2(Npar−2)!, have to be passed, in order to avoid recomputing them.

Procedure correl_index

```
CALL CORREL_INDEX(Npar,Ncorr,icorr2ij)
```

Returns the pair of indices, icorr2ij, corresponding to unique correlation coefficients of a Npar×Npar correlation matrix. The number of correlations, Ncorr= Npar!/2(Npar − 2)!, has to be passed, in order to avoid recomputing them. The array of indices has the size Ncorr×2.

Procedure correl_parlist

```
CALL CORREL_PARLIST(x[N,Npar]IN,rho[N_CORR(Npar)]OUT)
```

Computes all the non-trivial correlation coefficients, returned in rho, among a list of parameters, x, if double. If the x is a CHARACTER of size Npar, then the correlation name is built by adding a "-" between the two elements.

5.8.3 Public Functions for Time Series

Procedure autocorrel

```
CALL AUTOCORREL(data[N],rho[Nlag],Nlag)
```

Computes the reduced autocorrelation function, texttrho, of a time serie, data, as a a function of positive lag [1,Nlag].

Function intautocorrtime

```
t_int = INTAUTOCORRTIME(acf[Nlag],Nmcmc,Neff(OUT,OPT),low(OPT), &
                        high(OPT),step(OPT),c(OPT),FORCE=T/F)
```

Computes the integrated autocorrelation time, following the method by Sokal (http://www.stat.unc.edu/faculty/-cji/Sokal.pdf), as implemented in Emcee (https://github.com/dfm/emcee/blob/master/emcee/autocorr.py). The input ACF, `acf`, and the length of the series, `Nmcmc`, must be provided, both after burn-in. `LOW` is the minimum window size to test (default = 10). `HIGH` is the maximum window size to test (default = `Nmcmc`/2`C`). `STEP` is the step size for the window search (default = 1). `C` is the minimum number of autocorrelation times needed to trust the estimate (default = 10).

5.8.4 Other Public Routines

Procedure info_data

```
CALL INFO_DATA(X[N],WEIGHT[N],name)
```

Prints the basic statistical quantities of a set of an array `X`, up to two dimensions, with optional weight, `weight`, and variable name, `name` (string).

Procedure fwhm

```
fwhm = FWHM_DATA(x[N],F[N])
```

Computes the full width at half maximum of a distribution `F`, tabulated on a grid `X`. The distribution is supposed to be monomodal and well sampled.

5.9 LEAST-SQUARES: MODULE Math/chi2_minimization.f90

5.9.1 Public Procedure chi2min_LM

```
CALL CHI2MIN_LM (funcresidual,Nobs,par,tol,resid,status,verbose, &
                 limited,limits,fixed,itied,chi2,chi2red,parerr, &
                 covar,Niter,step,relstep,twoside)
```

Modified Levenberg-Marquardt chi2 minimization routine. Largey based on MINPACK's function `LMDIF1` and dependencies, painfully converted from F77, and subsequently, but still painfully, partly vectorized. The Jacobian is computed using finite difference approximation. Finally, I implemented several functionalities of C. Markwardt's IDL `MPFIT` (limiting, fixing parameters). `Npar` is the number of parameters `par`. `Nobs` is the number of observations. `PAR` is the parameter vector. `TOL` is the termination tolerance. `RESID` is the result of the user-defined function. `FUNCRESIDUAL` is the user-defined function returning the weighted residuals between the model and the observations. It should have the following interface:

```
FUNCTION funcresidual(par,Nobs)
  INTEGER, INTENT(IN) :: Nobs
  REAL(DP), DIMENSION(:), INTENT(IN) :: par
  REAL(DP), DIMENSION(Nobs) :: funcresidual
END FUNCTION funcresidual
```

The parameters can be limited through the `limited` and `limits`, `Npar`×2 arrays. They can also be fixed, via the boolean vector `fixed`. Finally, some parameters can be tied, via the `Npar` integer array `itied`, giving, for each parameter, the index of the parameter with which it is correlated, -1 if the parameter is independent.

`STATUS` is an integer output variable. If the user has terminated execution, `STATUS` is set to the (negative) value of iflag. Otherwise, `STATUS` is set as follows:

status=0: improper input parameters;

status=1: algorithm estimates that the relative error in the sum of squares is at most `TOL`;

status=2: algorithm estimates that the relative error between par and the solution is at most `TOL`;

status=3: conditions for `status`=1 and `status`=2 both hold;

status=4: `resid` is orthogonal to the columns of the jacobian to machine precision;

status=5: number of calls to funcresidual has reached or exceeded (`Niter`+1; 200 by default);

status=6: `tol` is too small; no further reduction in the sum of squares is possible;

status=7: `tol` is too small; no further improvement in the approximate solution par is possible.

The best absolute χ^2 ad its reduced value are returned in `chi2` and `chi2red`. The estimated parameter uncertainty is returned in `parerr` and the covariance matrix of the parameters in `covar`.

5.10 MATRICES: MODULE `Math/matrices.f90`

5.10.1 Public Routines for Computing Determinants

Function `determinant_Cholesky`

```
detA = DETERMINANT_CHOLESKY(A[N,N][,NODECOMPOSITION,NOPOSDEF])
```

Compute the determinant, `DETA` of a positive definite matrix `A`, using the Cholesky decomposition method. The keyword `NODECOMPOSITION` has to be used if the input matrix has already been decomposed. If `NOPOSDEF` is defined, then matrices which are not positive definite will return a NaN determinant.

Function `determinant_matrix`

```
detA = DETERMINANT_MATRIX(A[N,N][,LU,Cholesky])
```

Compute the determinant, `DETA` of a square matrix, `A`, using the LU decomposition method if `LU` is True (default). Otherwise, if `cholesky` is present and true, the Cholesky decomposition method is used.

5.10.2 Public Routines for Inverting Matrices

Function `invert_Cholesky`

```
invA[N,N] = INVERT_CHOLESKY(A[N,N][,determinant,noposdef])
```

Computes the inverse `INVA` of a positive definite matrix `A`, with the Cholesky decomposition method. If `NOPOSDEF` is defined, then matrices which are not positive definite will return a NaN filled inverse matrix. The determinant can be optionally returned, via `determinant`.

Function `invert_matrix`

```
invA[N,N] = INVERT_MATRIX(A[N,N][,gauss,lu,cholesky,determinant])
```

Computes the inverse `INVA` of a positive definite matrix `A`, with different methods, depending on the booleans `gauss`, `LU` and `Cholesky` (default Gauss-Jordan elimination).

5.11 FOURIER TRANSFORM: MODULE `Math/fft_specials.f90`

5.11.1 Public Procedure `realFT`

```
CALL REALFT(data,isign,zdata)
```

If `isign`=1, calculates the Fourier transform of a set of N real-valued data points, input in the array `data`. If the optional argument `zdata` is not present, the data are replaced by the positive frequency half of its complex Fourier transform. The real-valued first and last components of the complex transform are returned as elements `data(1)` and `data(2)`, respectively. If the complex array `zdata` of length N/2 is present, `data` is unchanged and the transform is returned in `zdata`. N must be a power of 2. If `isign`=1, this routine calculates the inverse transform of a complex data array if it is the transform of real data (result in this case must be multiplied by 2/N). The data can be supplied either in `data`, with `zdata` absent, or in `zdata`.

5.11.2 Public Function `correl`

```
f[N] = CORREL(data1[N],data2[N])
```

Computes the correlation of two vectors, `data1` and `data2`, for a lag: f(N) to f(N/2+1) are negative lags, f(1) is lag 0, and f(1) to f(N/2) are positive lags.

5.12 LINEAR SYSTEMS: MODULE Math/linear_system.f90

5.12.1 Public Routines for Matrix Decomposition

Function LU_decomp

```
LU[N,N] = LU_DECOMP(A[N,N],indx[N],d)
```

L.U. decomposition of square matrix A, following Press et al. (2007, Chap. 2.3).

Function Cholesky_decomp

```
L[N,N] = CHOLESKY_DECOMP(A[N,N],NOPOSDEF)
```

Compute the Cholesky decomposition of the symmetric positive definite matrix `A[N,N]`. `L[N,N]` is the lower triangle resulting from the decomposition. In the end, $A = L.L^T$.

5.12.2 Public Functions for Solving Linear Systems

Function linsyst_Gauss

```
X[N,M] = LINSYST_GAUSS_2D(A[N,N],B[N,M][,INVERSE[N,N],ONLY_INVERSE])
```

Solves the linear systems `A`[N,N].`X`[N,M]=`B`[N,M], using the Gauss-Jordan elimination with full pivot's algorithm (Press et al., 2007, p. 44). In other words, it simultaneously solves the M linear systems defined by:

$$\texttt{A(:,:).X(:,1)} = \texttt{B(:,1)},\ldots,\texttt{A(:,:).X(:,M)} = \texttt{B(:,M)}. \tag{5.1}$$

A natural product of the operation is the inverse matrix of `A` (`INVERSE`). If `ONLY_INVERSE` is true, then only the inverse matrix is solved, the solution is not.

If `B` is or rank 1, then only one system is solved and `X` is also of rank 1.

Function tridag

```
X[N] = TRIDAG(A[N-1],B[N],C[N-1],R[N])
```

Solves a tridiagonal matrix system with diagonal `B[N]`, off-diaognal elements `A[N-1]` and `C[N-1]`, and right-hand side `R[N]`.

Function linsyst_Cholesky

```
X[N] = LINSYST_CHOLESKY(L[N,N],b[N])
```

Solves the equation `A[N,N].X[N]=b[N]`, given `A` and `b`, using the Cholesky decomposition of $A = L.L^T$.

5.13 NON-LINEARITY: MODULE Math/nonlinear_equation.f90

5.13.1 Public Function nonolineq_dicho

```
x0 = NONLINEQ_DICHO(f(x),xinf,xsup,status,TOL=1.E-5)
```

Returns the value x_0 where $f(x_0) = 0$ within $[\texttt{xinf}, \texttt{xsup}]$, by dichotomy.

Chapter 6

MODULES FOR GENERAL PHYSICS

6.1 CONSTANTS: MODULE `Physics/constants.f90`

6.1.1 Public Parameters of Mathematical Constants

pi $= \pi$.

twopi $= 2\pi$.

oneoversqrt2pi $= 1/\sqrt{2\pi}$.

6.1.2 Public Physical Constants

Type MKS: fundamental constants

MKS%hplanck: Planck constant in $\mathrm{J.s}$.

MKS%clight: speed of light in vacuum in $\mathrm{m.s^{-1}}$.

MKS%grav: gravitation const in $\mathrm{N.m^{-2}.kg^{-1}}$.

MKS%kboltz: Boltzman constant in $\mathrm{J.K^{-1}}$.

MKS%stefan: Stefan-Boltzman in $\mathrm{W.m^{-2}.K^{-4}}$.

Type MKS: atomic constants

MKS%u: atomic mass unit in kg.

MKS%mp: proton mass in kg.

MKS%mn: neutron mass in kg.

MKS%me: electron mass in kg.

MKS%mH: Hydrogen atom mass in kg.

MKS%a0: Bohr radius in m.

MKS%Ryd: Rydberg constant in $\mathrm{m^{-1}}$.

MKS%cRyd: $c\times$Rydberg constant in Hz.

MKS%txsec: Thomson cross section in $\mathrm{m^2}$.

MKS%eV: electron volt in $\mathrm{J.eV^{-1}}$.

MKS%eV2wave: conversion of 1 eV to wavelength in m.

MKS%eV2nu: conversion of 1 eV to frequency in Hz.

MKS%eV2temp: conversion of 1 eV to temperature in K.

Type MKS: astronomical constants

MKS%Lsun: Solar luminosity in $\mathrm{J.s^{-1}}$.

MKS%Msun: Solar mass in kg.

MKS%Rsun: Solar radius in m.

MKS%pc: 1 parsec in m.

MKS%kpc: 1 kiloparsec in m.

MKS%Mpc: 1 megaparsec in m.

MKS%year: 1 year in s.

MKS%AU: Sun-earth distance in m.

MKS%TCMB: CMB temperature in K.

Type MKS: conversion between units

MKS%micron: 1 micron in m.

MKS%angstrom: 1 Angstrom in m.

MKS%erg: 1 erg in J.

MKS%mJy: 1 mJy in $\mathrm{W\,m^{-2}\,Hz^{-1}}$.

MKS%Jy: 1 Jy in $\mathrm{W\,m^{-2}\,Hz^{-1}}$.

Type CGS: fundamental constants

CGS%hplanck: Planck constant in $\mathrm{J.s}$.

CGS%clight: speed of light in vacuum in $\mathrm{m.s^{-1}}$.

CGS%grav: gravitation const in $\mathrm{N.m^{-2}.kg^{-1}}$.

CGS%kboltz: Boltzman constant in $\mathrm{J.K^{-1}}$.

CGS%stefan: Stefan-Boltzman in $\mathrm{W.m^{-2}.K^{-4}}$.

Type CGS: atomic constants

CGS%u: atomic mass unit in g.

CGS%mp: proton mass in g.

CGS%mn: neutron mass in g.

CGS%me: electron mass in g.

CGS%mH: Hydrogen atom mass in g.

CGS%a0: Bohr radius in cm.

CGS%Ryd: Rydberg constant in $\mathrm{cm^{-1}}$.

CGS%cRyd: $c\times$Rydberg constant in Hz.

CGS%txsec: Thomson cross section in $\mathrm{cm^2}$.

CGS%eV: electron volt in $\mathrm{erg.eV^{-1}}$.

CGS%eV2wave: conversion of 1 eV to wavelength in cm.

CGS%eV2nu: conversion of 1 eV to frequency in Hz.

CGS%eV2temp: conversion of 1 eV to temperature in K.

Type CGS: astronomical constants

CGS%Lsun: Solar luminosity in $\mathrm{erg.s^{-1}}$.

CGS%Msun: Solar mass in g.

CGS%Rsun: Solar radius in cm.

CGS%pc: 1 parsec in cm.

CGS%kpc: 1 kiloparsec in cm.

CGS%Mpc: 1 megaparsec in cm.

CGS%year: 1 year in s.

CGS%AU: Sun-earth distance in cm.

CGS%TCMB: CMB temperature in K.

Type CGS: conversion between units

CGS%micron: 1 micron in cm.

CGS%angstrom: 1 Angstrom in cm.

CGS%joule: 1 Joule in erg.

CGS%mJy: 1 mJy in $\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}}$.

CGS%Jy: 1 Jy in $\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}}$.

6.1.3 Public Atomic Periodic Table

If X is an atom's symbol (*e.g.* H, He, Li, Be, *etc.*), then:

ATOM%X_name is the full name (string);

ATOM%X_number is the atomic number (integer);

ATOM%X_weight is the atomic weight (double).

6.2 PHYSICS: MODULE `Physics/statistical_physics.f90`

6.2.1 Public Constants

wT_nu: $\lambda \times T$ at the maximum of a black body in m/K, expressed in frequency density (B_ν).

wT_w: $\lambda \times T$ at the maximum of a black body in m/K, expressed in wavelength density (B_λ).

6.2.2 Public Routines for Statistical Distributions

Function `black_body`

```
Bnu[N,M] = BLACKBODY(nu[N],Temperature[M])
```

Evaluates the Planck function at wavelengths `nu` [Hz], in $\mathrm{W/m^2/sr/Hz}$ and temperature, `temperature` [K]. If `temperature` is a scalar, then `Bnu` is a vector. If `nu` is a vector, then `Bnu` is a matrix.

Function `maxwell_boltzmann`

```
f(E)[N] = MAXWELL_BOLTZMANN(E[N],T)
```

Returns the Maxwell-Boltzmann distribution in energy [J-1], at temperature `T`, as a function of energy `E`. `E` and `f` must have the same size, either scalar or vector.

Chapter 7

DUST SPECIFIC MODULES

7.1 ISRF: MODULE Dust/isrf.f90

7.1.1 Public Function ISRF_Mathis83

```
Unu[N] = ISRF_MATHIS83(nu[N],chi,z)
```

Evaluates the Solar neighborhood ISRF in $\mathrm{J/m^3/Hz}$, for a scaling parameter, `chi`, as a function of frequency, `nu`. `Unu` is an isotropic monochromatic radiation density integrated over $4 \times \pi$ steradians ($U_\nu = 4\pi \times J_\nu/c$, where J_ν is the mean intensity as defined in chapter 1 of Rybicky et al. (1982).

- UV part from Mezger et al. (1982);
- visible part from Mathis et al. (1983).

If `z` is present then it is the redshift of the CMB component.

7.1.2 Public Function ISRF_BB

```
Unu[N] = ISRF_BB(nu[N],TBB,chi,z,NOIONIZING=T/F)
```

Evaluates the ISRF in $\mathrm{J/m^3/Hz}$, for a given scale parameter `chi`, which has the shape of a black body with T=`TBB`, as a function of frequency `nu`. `Unu` is an isotropic monochromatic radiation density integrated over 4π steradians ($U_\nu = 4\pi \times J_\nu/c$, where J_ν is the mean intensity as defined in chapter 1 of Rybicky et al. (1982). If `NOIONIZING` is true, the ionizing photons are suppressed (truncated ISRF).

7.1.3 Public Function Unu_ISRF

```
Unu[M,N] = UNU_ISRF(nu[N],U[M],labISRF,z,TBB,noion)
```

Returns the monochromatic energy density of the ISRF in $\mathrm{J/m^3/Hz}$, as a function of frequencies, `nu`. It is a interface for the other ISRF functions above, and admits the same keywords. The choice of ISRF is controlled by `labISRF` which can be either "Mathis83" or "BB".

7.2 SCATTERING: MODULE Dust/phase_functions.f90

7.2.1 Public Function phase_HG41

```
phase_func = phase_HG41(mu,g)
```

Calculte the Henyey & Greenstein (1941) phase function, for a given asymmetry parameter, `g`, and the cosine of the scattering angle, `mu`. Both `mu` and `phase_func` must have the same size, either scalar or vector.

7.2.2 Public Function invphase_HG41

```
phi = invphase_HG41(xi,g)
```

Calculte the Henyey & Greenstein (1941) cosine angle for a given asymmetry parameter, `g`, by inverting the repartition function of the phase function, `xi`. Both `xi` and `phi` must have the same size, either scalar or vector.

7.3 OPTICAL PROPERTIES: MODULE `Dust/grain_optics.f90`

7.3.1 Public Variables

lendustQ: length of the string to represent the dust species label.

dirQabs: directory where the optical properties are stored.

Current cross section labels:

Sil_LD93: silicates from Laor & Draine (1993).

finish

7.3.2 Public Functions for Computing Optical Properties

Procedure `Mie_scattering`

```
CALL MIE_SCATTERING (x,refrel,Nang,Qext,Qsca,Qabs,Qback,gsca, &
                     S1[2*Nang-1],S2[2*Nang-1])
```

Returns scattering and absorption by a homogeneous isotropic sphere, as a function of $\mathtt{X} = 2*\pi*a/lambda$, given `REFREL`, its relative refractive index (complex refractive index of the sphere divided by the real part index of the surrounding medium). `Nang`> 1 is the number of scattering angles between 0 and $\pi/2$ (will calculate $2 \times \mathtt{Nang} - 1$ directions from 0 to $\pi/2$).

- $m = \sqrt{\epsilon_1 + i.\epsilon_2}$;
- $Q_{\text{ext}} = C_{\text{ext}}/(\pi a^2)$ = efficiency factor for extinction;
- $Q_{\text{sca}} = C_{\text{sca}}/(\pi a^2)$ = efficiency factor for scattering;
- $Q_{\text{abs}} = Q_{\text{ext}} - Q_{\text{sca}}$ = efficiency for absorption;
- $Q_{\text{back}} = 4\pi \times (dC_{\text{sca}}/d\Omega)/(\pi a^2)$ = backscattering efficiency;
- $g_{\text{sca}} =< \cos(theta) >$ = asymetry parameter for scattering.

`S1` and `S2` are diagonal elements of the amplitude scattering matrix:

- `S1(1:2*Nang-1)`$= -i \times f_{22}$ (incident $\vec{E}$ perpendicular to scattering plane, scattered $\vec{E}$ perpendicular to scattering plane);
- `S2(1:2*NANG-1)`$= -i \times f_{11}$ (incident $\vec{E}$ parrallel to scattering plane, scattered E parrallel to scattering plane).

Cf. the book by Bohren & Huffman (1983, Appendix A).

Taken from Bruce Draine's website and translated from F77 (ftp://ftp.astro.princeton.edu/draine/scat/-bhmie/bhmie.f).

Procedure `RayleighGans_scattering`

```
CALL RayleighGans_Scattering(x,refrel,Qext,Qsca,Qabs,gsca)
```

Rayleigh-Gans Regime approximation. From Laor & Draine (1993, Sect. 2.2.1).

Procedure `Geometric_scattering`

```
CALL Geometric_Scattering(x,refrel,Qext,Qsca,Qabs,gsca)
```

Geometrical Optics Regime approximation. From Laor & Draine (1993, Sect. 2.2.2).

Procedure `grain_cross_section`

```
CALL GRAIN_CROSS_SECTION(radius,wave,refrel,Qext,Qsca,Qabs,gsca)
```

Compute the wavelength dependent cross-sections, for a set of sizes, using the method of Laor & Draine (1993), *i.e.* the three different regimes above.

Function coll_cross_section

```
sigma = COLL_CROSS_SECTION(E,a)
```

Collision cross-section for electrons, following Eq. (4) of Dwek (1986). `E` must be in J, `a` in microns, `sigma` is returned in m^2.

7.3.3 Public Routines for Managing Optical Properties

Function rho_grain

```
rho = rho_grain(species,name,reference)
```

Returns the mass per unit volume of material for grain species in $\mathrm{kg/m^3}$. The label of the species, `species`, should be the same as for the cross-sections. Optional returned argument are the full name of the species, `name`, as well as its bibliographic reference, `reference`.

Procedure read_optics

```
CALL READ_OPTICS(lab,Nwall,Nrall,NTall,Nwmax,Nrmax,NTmax, &
                 waveall,radiusall,tempall, &
                 Qabsall,Qscaall,gscall,Qavall, &
                 wrange,wave,nu,Nw)
```

Reads the optical properties corresponding to the grain species identified by the label `lab` (vector). The returned quantities are the wavelength, radius and temperature grids. These grids are different for each species, thus `waveall`, `radiusall` and `tempall` are 2D arrays of size $N_{\mathrm{species}} \times N_{\mathrm{max}}$ where N_{max} is either `Nwmax`, `Nrmax` or `NTmax`, and the relevant part of the array are: `waveall(i,1:Nwall(i))`, `radiusall(i,1:Nrall(i))` and `tempall(i,1:NTall(i))`. The precomputed optical properties are returned in `Qabsall`, `Qscaall` and `gscaall` of size $N_{\mathrm{species}} \times$ `Nrmax` $\times$ `Nwmax` and `Qavall` of size $N_{\mathrm{species}} \times$ `Nrmax` $\times$ `NTmax`.

If `wave` is present, then the optical properties are reinterpolated on a common wavelength grid, without loosing spectral resolution. The spectral range can be truncated with `wrange`.

7.4 ENTHALPIES: MODULE Dust/grain_enthalpies.f90

7.4.1 Public Variables

lendustH: length of the string containing the enthalpy label.

rho_PAH: mass volume density of PAHs in $\mathrm{kg/m^3}$.

rho_gra: mass volume density of graphite in $\mathrm{kg/m^3}$.

rho_sil: mass volume density of silicates in $\mathrm{kg/m^3}$.

taumin minimum grain lifetime in s.

taumax age of the universe in s.

7.4.2 Public Functions

Function grain_enthalpy

```
H[N,M] = grain_enthalpy(radius[N],temp[M],species,accuracy)
```

Returns the enthalpy of a grain, `H` in J, as a function of temperature, `temp`, and radius, `radius`, for the various species in the database, indentified by label `species`.

The possible labels are listed in Table 7.1. More details are available in `Documentation/grain_properties.pdf`.

Procedure enthalpy_reference

```
CALL ENTHALPY_REFERENCE(species,name,reference)
```

For a given species label, `species`, returns the full name, `name`, and the bilbiographical reference, `reference`, of the enthalpy.

Code label	Species	Bibliographic reference
PAH_D97	PAH	Dwek et al. (1997)
Gra_D97	Graphite	Dwek et al. (1997)
Sil_DA85	Astronomical silicates	Draine & Anderson (1985)
PAH_DL01	PAH	Draine & Li (2001)
PAH_DL01_prox	PAH (approximation)	Draine & Li (2001, Eq. 33)
Gra_DL01	Graphite	Draine & Li (2001)
Sil_DL01	Astronomical silicates	Draine & Li (2001)
a-C_man20nm	a-C(:H) with mantle	Jones et al. (2013)
a-Forst_Fe_man5nm	a-Forsterite with mantle and inclusions	Jones et al. (2013)
a-Forst_Fe_man5nm	a-Enstatite with mantle and inclusions	Jones et al. (2013)

Table 7.1: Available grain enthalpies.

Function grain_lifetime

```
tau = grain_lifetime(radius,T[N],dPdT[N],H[N],species)
```

Returns the temperature averaged grain lifetime, `tau`, in s, following the formalism of (Guhathakurta & Draine, 1989, Sect. IIIb), with an adaptation to PAHs. One must enter the radius of the grain in microns, `radius`, as well as its label, `species`. One must also provide, the temperature distribution, `dPdT` and the enthalpy of the grain, `H`, mapped on the temperature grid, `T`.

7.5 SIZE DISTRIBUTIONS: MODULE Dust/grain_sizedist.f90

7.5.1 Public Variables

lendustcomp: length of the string containing the dust component label.

lendustmixt: length of the string containing the dust mixture label.

Ndustmixt: number of currently implemented dust mixtures.

dustmixt: list of labels of currently implemented dust mixtures (Table 7.2).

Ncomp_dust: number of dust components.

labcomp_dust: list of component labels (Table 7.3).

q_type: type containing the mass fractions of each components: `PAHi`, `PAHn`, `Scar`, `Ssil`, `Bcar`, `Bsil`, `PAH`, `car`, `sil`, `SG`, `BG`, `Zdust`.

q_X: structure containing the mass fraction for model `X` (Table 7.2).

Z04	Zubko et al. (2004, BARE-GR-S)
G11_AC	Galliano et al. (2011, AC)
C11	Compiègne et al. (2011)
J13	Jones et al. (2013)
J17	Jones et al. (2017)

Table 7.2: Labels of currently available dust mixtures.

7.5.2 Public Routines

Function sizedist

```
f(r) = SIZEDIST(r,model,comp,amin,amax)
```

Returns the size distribution of a dust component in $\mu\mathrm{m}^{-1}$, `comp`, for a model, `model`, as a function of grain radius, `r`. This distribution is normalized to the dust mass of the component, *i.e.*:

$$\int_{a_-}^{a_+} \frac{4\pi}{3} a^3 \rho \times f(a)\, \mathrm{d}a = 1. \tag{7.1}$$

Optional minimum and maximum radii, `amin` and `amax`, in μm, can be returned.

Label	THEMIS	Other Models
PAHi	Extremelly small a-C(:H)	Ionized PAHs
PAHn	Very small a-C(:H)	Neutral PAHs
Scar	Small a-C(:H)	Small carbonaceous
Ssil	...	Small siliactes
Bcar	Large a-C(:H)	Large carbonaceous
Bsil	Large silicates	Large silicates

Table 7.3: List of dust component labels.

Procedure dust_mixture_properties

```
CALL DUST_MIXTURE_PROPERTIES(dust_mixture="",labQabs=["",6]OUT,
                             labH=["",6]OUT,ref="",q=[6]OUT)
```

Returns several properties of the dust mixture, `dust_mixture`. The list of optical properties is returned in the vector `labQabs`. The list of enthalpies is returned in the vector `labH`. The bibliographical reference of the mixture is returned in `ref`. The mass fractions of each component is returned in the vector `q`.

Function dust_mass_fractions

```
q(6) = DUST_MASS_FRACTIONS(dust_mixture="",q_PAH,fionPAH,f_SG,f_sil)
```

Function returning the mass fractions of the homogenized dust components, as a function of input SED model parameters.

DUST_MIXTURE is the dust mixture label.

Q_PAH is the mass fraction of PAHs: $q_{\rm PAH} = M_{\rm PAH}/M_{\rm total\ dust}$.

fionPAH is the charge fraction of PAHs: $f^{+}_{\rm PAH} = M_{\rm PAHi}/M_{\rm PAH}$.

f_SG is the fraction of non-PAH small grains relative to non-PAH dust: $f_{\rm SG} = (M_{\rm Scar} + M_{\rm Ssil})/(M_{\rm Scar} + M_{\rm Bcar} + M_{\rm Ssil} + M_{\rm Bsil})$.

f_sil is the fraction of silicates relative to non-PAH dust: $f_{\rm sil} = (M_{\rm Ssil} + M_{\rm Bsil})/(M_{\rm Scar} + M_{\rm Bcar} + M_{\rm Ssil} + M_{\rm Bsil})$

The extra condition we impose when parameterizing the size distribution is to have a constant small-silicate-to-small-grain ratio: $f^{\rm sil}_{\rm SG} = M_{\rm Ssil}/(M_{\rm Scar} + M_{\rm Ssil}) = {\rm const}$. If a variable is negative, then it is assumed to be at the default value, and the variable is accordingly modified on output.

7.6 EMISSION: MODULE Dust/grain_spectrum.f90

7.6.1 Public Procedure grain_stochastheat

```
CALL GRAIN_STOCHASTHEAT(wave,temp,Unu,Cabs,Cav,H,Lnu,Tout,dPdT, &
                        equilibrium,E_el,nsigvf_el,accuracy,gridtype)
```

Returns the temperature distribution and infrared spectrum of a dust grain, in the approximation of the continuous cooling, using the algorithm by Guhathakurta & Draine (1989). The energy density of the ISRF is passed through `Unu`, in $\mathrm{J/m^3/sr/Hz}$. The wavelength and temperature grids, `WAVE` (μm) and `TEMP` (in K), must be sorted. The grain cross section, `Cabs` in m^2 must be mapped on the `WAVE` grid. The grain enthalpy, `H` (in J), and the Planck mean of the cross-section, `Cav`, must be mapped on the `TEMP` grid. If `equilibrium` is true, then equilibrium temperature is forced for all sizes. The temperature distribution is returned in `dPdT` and the adpatative grid on which it is mapped in `Tout`. IF `E_EL` and `NSGIVF_EL` are set, then collisional heating is switched on.

More details can be found in `Documentation/grain_properties.pdf`.

7.7 SED FITTING: MODULE Dust/fitSED_utilities.f90

7.7.1 Public Variables

SED Template Types

The different SED templates are present, in details, in Sect. 2 of Galliano (2018). If a prameter's starts with `ln`, it means it is the natural logarithm of the quantity.

MBB_type: structure containing the *MBB* templates, with: the grid sizes, `Nw`, `NlnT`, `Nbeta`; the MBB fixed parameters, `w0` and `kappa0`; the filter list, `filt`; the parameter grids, `w`, `lnT` and `beta`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

BBQ_type: structure containing the *BBQ* templates, with: the grid sizes, `Nw` and `NlnT`; the label of the grain species, `label`; the filter list, `filt`; the parameter grids, `w` and `lnT`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

BEMBB_type: structure containing the *BEMBB* templates, with: the grid sizes, `Nw`, `NlnT`, `Nbeta1`, `Nbeta2` and `Nlnwb`; the BEMBB fixed parameters, `w0` and `kappa0`; the filter list, `filt`; the parameter grids, `w`, `lnT` and `beta1`, `beta2` and `lnwb`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

deltaU_type: structure containing the *deltaU* templates, with: the grid sizes, `Nw` and `NlnU`; the labels of the dust mixture, `dust` and `comp`; the filter list, `filt`; the parameter grids, `w` and `lnU`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

powerU_type: structure containing the *powerU* templates, with: the grid sizes, `Nw`, `NlnUm`, `lnDU`, `alpha`; the labels of the dust mixture, `dust` and `comp`; the filter list, `filt`; the parameter grids, `w` and `lnUm`, `lnDU`, `alpha`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

starBB_type: structure containing the *starBB* templates, with: the grid size, `Nw`; the filter list, `filt`; the parameter grid, `w`; and the SED grid, `lnLnu`.

radio_type: structure containing the *radio* templates, with: the grid sizes, `Nalphas` and `Nw`; the filter list, `filt`; the wavelength where the continuum is normalized, `wnorm` and `norm`; the parameter grids, `alphas` and `w`; the synchrotron SED, `lnLnu_sync` and the free-free SED, `lnLnu_FF`.

AGN_type: the AGN templates of Siebenmorgen et al. (2015), with: the parameter grid sizes, `w`, `Nth`, `NlnR`, `NlnVc`, `NlnAc` and `NlnAd`; the filter labels, `filt`; the parameter grids, `w`, `th`, `lnR`, `lnVc`, `lnAc` and `lnAd`; the SED grid, `lnLnu`; and the luminosities, `lnL` and `lnFIR`.

SED Fit Variables

parinfo_type: containing the general settings of parameters to be fed to the various fitters, with: the name of the parameter, `name`; the name of the template component, `comp`; the default value of the parameter, `value`; the parameter limits, `limits` and `limited`; if the parameter is fixed, `fixed`, an hyperparameter, `hyper`, if ASIS is applied, `asis`; a boolean telling if the parameter is a model parameter or an ancillary data, `model`; the parameter with which it is tied, `tied`; the mean and standard deviation of the parameter's distribution, `mean` and `sigma`; the index of the parameter in the parameter list, `ind`.

indpar_type: contains the parameter indices in the parameter list; the field of this structure are the parameter labels and their values, the parameter index. If the parameter is absent, the index is -1.

dirtemp_type: contains the absolute path of the directory where the template of each component is. The name of the field is the label of the component.

lentemp: length of the character string containing the template name.

lencorr: length of the character string containing a parameter correlation name.

Ncompall: number of model components.

NparMBB: number of parameters of the *MBB* component.

NparBBQ: number of parameters of the *BBQ* component.

NparBEMBB: number of parameters of the *BEMBB* component.

NpardeltaU: number of parameters of the *deltaU* component.

NparpowerU: number of parameters of the *powerU* component.

NparstarBB: number of parameters of the *starBB* component.

Nparradio: number of parameters of the *radio* component.

NparAGN: number of parameters of the *AGN* component.

compall: list of all component labels.

parMBBname: list of parameter names for the *MBB*.

parBBQname: list of parameter names for the *BBQ*.

parBEMBBname: list of parameter names for the *BEMBB*.

pardeltaUname: list of parameter names for the *deltaU*.

parpowerUname: list of parameter names for the *powerU*.

parstarBBname: list of parameter names for the *starBB*.

parradioname: list of parameter names for the *radio*.

parAGNname: list of parameter names for the *AGN*.

iX_Y: indice of parameter with label `X` and model component `Y` in the `parY` structure.

7.7.2 Public Routines for Manipulating SED Templates

Procedure read_template_MBB

```
CALL READ_TEMPLATE_MBB(templ,dirtemp,limitslnT,limitsbeta, &
                       limitedlnT,limitedbeta,model,initialize, &
                       filtobs,limitswmod,unitMKS)
```

Reads the precomputed *MBB* templates into the `MBB_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The parameters can be limited with `limitslnT`, `limitedlnT`, `limitsbeta` and `limitedbeta`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`. If `unitMKS` is false (default), the masses are expressed in $M_\odot$, the SEDs, in $L_\odot/\mathrm{Hz}$ and the powers, in $L_\odot$. If `unitMKS` is true, the masses are expressed in $\mathrm{kg/m^2}$, the SEDs, in $\mathrm{W/m^2/Hz}$ and the powers, in $\mathrm{W/m^2}$.

Procedure read_template_BBQ

```
CALL READ_TEMPLATE_BBQ(templ,dirtemp,labQ,limitslnT, &
                       limitedlnT,model,initialize, &
                       filtobs,limitswmod,unitMKS)
```

Reads the precomputed *BBQ* templates into the `BBQ_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The Q_{abs} is label is entered with `labQ`. The parameters can be limited with `limitslnT` and `limitedlnT`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`. If `unitMKS` is false (default), the masses are expressed in $M_\odot$, the SEDs, in $L_\odot/\mathrm{Hz}$ and the powers, in $L_\odot$. If `unitMKS` is true, the masses are expressed in $\mathrm{kg/m^2}$, the SEDs, in $\mathrm{W/m^2/Hz}$ and the powers, in $\mathrm{W/m^2}$.

Procedure read_template_BEMBB

```
CALL READ_TEMPLATE_BEMBB(templ,dirtemp,limitslnT,limitsbeta1, &
                         limitsbeta2,limitslnwb, &
                         limitedlnT,limitedbeta1,limitedbeta2, &
                         limitslnwb,model,initialize, &
                         filtobs,limitswmod,unitMKS)
```

Reads the precomputed *BEMBB* templates into the `BEMBB_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The parameters can be limited with `limitslnT`, `limitedlnT`, `limitsbeta1`, `limitedbeta1`, `limitsbeta2`, `limitedbeta2`, `limitslnwb` and `limitedlnwb`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`. If `unitMKS` is false (default), the masses are expressed in $M_\odot$, the SEDs, in $L_\odot/\mathrm{Hz}$ and the powers, in $L_\odot$. If `unitMKS` is true, the masses are expressed in $\mathrm{kg/m^2}$, the SEDs, in $\mathrm{W/m^2/Hz}$ and the powers, in $\mathrm{W/m^2}$.

Procedure read_template_deltaU

```
CALL read_template_deltaU(templ,dirtemp,dustmixt,limitslnU,limitedlnU,&
                          lnq_PAH,fionPAH,f_SG,f_sil,free_lnq_PAH, &
                          free_fionPAH,free_f_SG,free_f_sil, &
                          model,filtobs,limitswmod,Ncomp,unitmks)
```

Reads the precomputed *deltaU* templates into the `deltaU_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The dust mixture label is specified with `dustmixt`. The number of subcomponents depends on which parameters are going to be varied. The dust mixture parameters are specified with `lnq_PAH`, the fraction of PAHs, `fionPAH`, the fraction of ionized PAHs, `f_SG`, the fraction of small grains, `f_sil`, the fraction of silicates, and the booleans specifying if they are free, `free_lnq_PAH`, `free_fionPAH`, `free_f_SG` and `free_f_sil`. The number of model components is returned in `Ncomp`. The parameters can be limited with `limitslnU` and `limitedlnU`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`. If `unitMKS` is false (default), the masses are expressed in $M_\odot$, the SEDs, in $L_\odot/\mathrm{Hz}$ and the powers, in $L_\odot$. If `unitMKS` is true, the masses are expressed in $\mathrm{kg/m^2}$, the SEDs, in $\mathrm{W/m^2/Hz}$ and the powers, in $\mathrm{W/m^2}$.

Procedure read_template_powerU

```
CALL read_template_powerU(templ,dirtemp,dustmixt,limitslnUm,&
                          limitedlnUm,limitsDU,limitedDU, &
                          limitsalpha,limitedalpha,&
                          lnq_PAH,fionPAH,f_SG,f_sil,free_lnq_PAH, &
                          free_fionPAH,free_f_SG,free_f_sil, &
                          model,filtobs,limitswmod,Ncomp,unitmks)
```

Reads the precomputed *powerU* templates into the `powerU_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The dust mixture label is specified with `dustmixt`. The number of subcomponents depends on which parameters are going to be varied. The dust mixture parameters are specified with `lnq_PAH`, the fraction of PAHs, `fionPAH`, the fraction of ionized PAHs, `f_SG`, the fraction of small grains, `f_sil`, the fraction of silicates, and the booleans specifying if they are free, `free_lnq_PAH`, `free_fionPAH`, `free_f_SG` and `free_f_sil`. The number of model components is returned in `Ncomp`. The parameters can be limited with `limitslnUm`, `limitedlnU`, `limitslnDU`, `limitedlnDU`, `limitsalpha` and `limitedalpha`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`. If `unitMKS` is false (default), the masses are expressed in $M_\odot$, the SEDs, in $L_\odot/\mathrm{Hz}$ and the powers, in $L_\odot$. If `unitMKS` is true, the masses are expressed in $\mathrm{kg/m^2}$, the SEDs, in $\mathrm{W/m^2/Hz}$ and the powers, in $\mathrm{W/m^2}$.

Procedure read_template_starBB

```
CALL read_template_starBB(templ,dirtemp,model,initialize,filtobs, &
                          limitswmod)
```

Reads the precomputed *starBB* templates into the `starBB_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`.

Procedure read_template_radio

```
CALL read_template_radio(templ,dirtemp,limitsalphas,limitedalphas, &
                         model,initialize,filtobs,limitswmod)
```

Reads the precomputed *radio* templates into the `radio_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The parameters can be limited with `limitsalphas`, and `limitedalphas`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`.

Procedure read_template_AGN

```
CALL read_template_AGN(templ,dirtemp,limitsth,limitslnR,limitslnVc, &
                       limitslnAc,limitslnAd,limitedth,limitedlnR, &
                       limitedlnVc,limitedlnAc,limitedlnAd,model, &
                       initialize,filtobs,limitswmod)
```

Reads the precomputed *AGN* templates into the `AGN_type` variable `templ`. The directory where the templates are is specified *via* `dirtemp`. The parameters can be limited with `limitsth`, `limitedth`, `limitslnR`, `limitedlnR`, `limitslnVc`, `limitedlnVc`, `limitslnAc`, `limitedlnAc`, `limitslnAd` and `limitedlnAd`. If `model` is true, then the fine wavelemgth-grid model is also returned; the spectral range of the model is optionally limited by `limitwmod`. If `initialize` is true, then the routine simply initializes the `templ` structure without reading the spectra. The list of photometric filters in which the model is integrated is `filtobs`.

Function simplified_q

```
q[] = simplified_q(comp,q_PAH,fionPAH,f_SG,f_sil)
```

Function returning the simplified mass fractions (mass fractions when the number of components is reduced due to some parameters being fixed). It reads the parameter `temp(:)%comp`, which contains the unique sequence of components. Each input parameter can either be a scalar or a vector. The output fraction is an array of size $N_{\rm comp} \times N_{\rm qPAH} \times N_{\rm fionPAH} \times N_{\rm fSG} \times N_{\rm fsil}$. If some parameters are scalars, the corresponding output dimensions are collapsed.

Function interp_SED_template

```
Lnu = interp_SED_template(jw,parvec,parname,compname,parinfo,indpar, &
                          parval,tempMBB1,tempMBB2,tempBBQ,tempBEMBB, &
                          tempdeltaU,temppowerU,tempradio,tempstarBB, &
                          tempAGN,LnuMBB1,LnuMBB2,LnuBBQ,LnuBEMBB, &
                          LnudeltaU,LnupowerU,Lnuradio,LnustarBB, &
                          LnuAGN,iBBQ)
```

Returns the total SED, at wavelength indices listed in `jw`. It interpolates the SED templates to provide a fast model evaluation. If `parvec`, `parname` and `compname` are present, then the model is calculated for a whole vector of values (`parvec`) of the parameter `parname` of component `compname`, the other parameters being set to the values in `par{comp}`. If these `parname` and `compname` are not set and `SIZE(parvec)=1`, then the model is calculated for a list of `jw`, and a single combination of parameter values. If any `Lnu{comp}` is set, then this component is not recomputed, we use the input value. The units of the output are $L_\odot/Hz$.

Function interp_lum_template

```
L = lum_tot_vec(Npar,parval,parinfo,indpar, tempMBB1,tempMBB2, &
                tempBBQ, tempBEMBB,tempdeltaU, temppowerU,tempAGN, &
                L_MBB1,L_MBB2,L_BBQ,L_BEMBB, L_deltaU,L_powerU,L_AGN, &
                FIR)
```

Returna the integrated power of the dust components, for a given set of parameters, by interpolating the SED templates. The units of the output are $L_\odot$. If `FIR` is true, then the power is integrated only between 60 and 200 microns.

Function inverse_SED_template

```
par = inverse_SED_template(Lnu,jw,parname,parinfo,indpar,parval, &
                           tempMBB1,tempMBB2,tempBBQ,tempBEMBB, &
                           tempdeltaU,temppowerU,tempradio, &
                           tempstarBB,tempAGN,proper)
```

Given a specific monochromatic luminosity (L_ν in $\mathrm{L}_\odot/\mathrm{M}_\odot/\mathrm{Hz}$), and all the parameters except one (of index `indpar`), the function returns, by interpolation, the missing parameter. These functions are necessary for ASIS, in the Bayesian code.

Function avU

```
<U> = avU(lnUm,lnDU,alpha)
ln(<U>) = lnavU(lnUm,lnDU,alpha)
std(U) = sigU(lnUm,lnDU,alpha)
ln(std(U)) = lnsigU(lnUm,lnDU,alpha)
```

Returns various moments of the starlight intensity distribution of the *powerU* component.

7.7.3 Public Routines for Managing Input Files

Procedure read_partuning

```
CALL read_partuning(comp,parinfo,dirtemp,dustmixt,labQ)
```

Reads the individual parameter tuning files. For a given component `comp`, it fills in the `parinfo` structure and optional arguments `dirtemp`, `dustmixt` and `labQ`.

Procedure set_indpar

```
CALL set_indpar(indpar,parinfo)
```

Fills the `indpar` structure with the info in the `parinfo` structure.

Procedure read_master

```
CALL read_master(Nmcmc,verbose,robust_RMS,robust_cal,skew_RMS, &
                 newinit,NiniMC,calib,resume,indresume,newseed, &
                 MBB1,MBB2,BBQ,BEMBB,deltaU,powerU,starBB,radio,AGN, &
                 Nextra,dostop,dirtemp,dustmixt_deltaU, &
                 dustmixt_powerU,labQ,parinfo,parhypinfo,parextinfo, &
                 parmodinfo,indpar,Npar,Nparmod,Nparhyp,Ncorrhyp, &
                 Ncorr,corrhypname,corrname,SED_unit, NHDF5file, &
                 double_length)
```

Reads the master input file of `fitSED_chi2` and `fitSED_HB`, and sets all the variables according to their value found in the input file.

Procedure read_analysis

```
CALL read_analysis(simu,overchi2,histogram,ACF,SED,verbose,FIR, &
                   ranwmod,t_burnin,t_end,Nindiv,indiv,surfaceunit, &
                   t_int_all,indivcorr)
```

Reads the analysis input file of `fitSED_chi2` and `fitSED_HB`, and sets all the variables according to their value found in the input file.

Procedure initparam

```
CALL initparam(NiniMC,ind,par,parinfo,itied,mask, &
               iwmaxFIR,wmaxFIR,LnumaxFIR,Lnunormrad,Lnuvis,ivis, &
               tempMBB1,tempMBB2,tempBBQ,tempBEMBB,tempdeltaU, &
               temppowerU,tempradio,tempstarBB,tempAGN,newinit, &
               filobs,extraOBS,dextraOBS)
```

Initialize the parameters before the SED fits with `fitSED_chi2` and `fitSED_HB`.

Procedure fracname

```
CALL fracname(NHDF5file,filename,dirOUT)
```

For a given number, `NHDF5file`, of fractionated HDF5 files, it returns the filenames, `filename`, following our convention. The directory where theses files will be written can optionally be set with `dirOUT`.

7.8 INSTRUMENTS: MODULE Dust/instrument_filters.f90

More details about this module are in the file Documentation/instrument_filters.pdf.

7.8.1 Public Types

filter_type: structure containing all the relevant quantities for a list of photometric filters. The filter transmissions are in the Nfilt×Nwmax matrix, trans, and are mapped on the wavelength (in μm) and frequency (in Hz), wave and nu. The actual number of wavelength samples for each filter is in the vector of size Nfilt, Nw. The central wavelengths and frequencies and the label of the filters are in wcen, nucen and namefilt.

calib_type: structure containing all the relevant quantities relative to the calibration uncertainties of various instrument wavebands. Nfilt contains the number of non fully correlated filters. Nfiltall contains the number of input filters (Nfiltall≥Nfilt). The correlation, standard-deviation and covariance matrices are matR(all), matS(all) and matcov(all). The fully correlated filters are identified with the vector bool. The filter lists are namefilt(all) and the corresponding instruments are instrument(all). The array labspecall is 0 if the index corresponds to a broad band, and it is the index of the SPECALL instrument, if it corresponds to a spectrometer. The vector ispecmod contains the index of the wavelength for which the calibration error is assumed non fully correlated.

7.8.2 Other Public Variables

lenfilter: length of the character string containing the photometric filter labels.

Nphotall: number of photometric filters currently implemented.

wcenall: central wavelength in microns of each photometric filter.

filtall: label of each photometric filter.

7.8.3 Public Routines

Function wcen_filter

```
wcen[N] = WCEN_FILTER(filter[N],spec,Nspec,maskspec,instrument)
```

Returns the nominal wavelengths of a list of photometric filters. If an element of filter belongs to the filtall filter list, then the nominal wavelength in μm is returned in wcen. If the filter has the form (spectrograph label)//(wavelength in microns) then the actual wavelenegth is returned.

Procedure read_filters

```
CALL READ_FILTERS(filters,list0[Nfilt])
```

Returns the filter structure, filters, corresponding to a vector of photometric filter labels, list0.

Precedure read_caliberr

```
CALL READ_CALIBERR (calib,list[Nfilt])
```

Read the structure, calib, of the instrumental calibration uncertainties, corresponding to a list of filters list.

Function synthethic_photometry

```
Fnu0[N] = SYNTHETIC_PHOTOMETRY(wave[Nw],Fnu[Nw],namefilt[N],filters)
```

Computes the synthetic photometry, Fnu0, corresponding to an finely sampled SED/spectrum, Fnu, mapped on the wavelength grid wave in μm, and seen throught the list of photometric filters namefilt.

Bibliography

Bohren, C. F. & Huffman, D. R. 1983, Absorption and scattering of light by small particles (Wiley)

Compiègne, M., Verstraete, L., Jones, A., et al. 2011, A&A, 525, A103+

Draine, B. T. & Anderson, N. 1985, ApJ, 292, 494

Draine, B. T. & Li, A. 2001, ApJ, 551, 807

Dwek, E. 1986, ApJ, 302, 363

Dwek, E., Arendt, R. G., Fixsen, D. J., et al. 1997, ApJ, 475, 565

Galliano, F. 2018, MNRAS, 476, 1445

Galliano, F., Hony, S., Bernard, J.-P., et al. 2011, A&A, 536, A88

Guhathakurta, P. & Draine, B. T. 1989, ApJ, 345, 230

Henyey, L. G. & Greenstein, J. L. 1941, ApJ, 93, 70

Jones, A. P., Fanciullo, L., Köhler, M., et al. 2013, A&A, 558, A62

Jones, A. P., Köhler, M., Ysard, N., Bocchio, M., & Verstraete, L. 2017, A&A, 602, A46

Laor, A. & Draine, B. T. 1993, ApJ, 402, 441

Mathis, J. S., Mezger, P. G., & Panagia, N. 1983, A&A, 128, 212

Mezger, P. G., Mathis, J. S., & Panagia, N. 1982, A&A, 105, 372

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing (Cambridge University Press)

Rybicky, G., Lightman, A. P., & Domke, H. 1982, Astronomische Nachrichten, 303, 142

Siebenmorgen, R., Heymann, F., & Efstathiou, A. 2015, A&A, 583, A120

Yu, Y. & Meng, X.-L. 2011, JCGS, 20, 531

Zubko, V., Dwek, E., & Arendt, R. G. 2004, ApJS, 152, 211