Manual for XCLASS-Interface

T. Möller, P. Schilke

November 9, 2017

Version 1.2.5

Contents

1 Introduction

The Common Astronomy Software Applications package (CASA)[1] package provides a powerful tool for data post-processing, but contains only rudimentary functions for modeling the data. On the one hand the toolbox offers the possibility to model data using the myXCLASS program, which is a one-dimensional radiative transfer program using CDMS/JPL molecular data. Additionally, the toolbox provides an interface for the MAGIX[2] package. MAGIX provides a framework of an easy interface between existing codes and an iterating engine that attempts to minimize deviations of the model results from available observational data, constraining the values of the model parameters and providing corresponding error estimates. Many models can be plugged into MAGIX to explore their parameter space and find the set of parameter values that best fits observational/experimental data. Furthermore, the toolbox contains an interface for VAMDC (Virtual Atomic and Molecular Data Center) and the CDMS (Cologne Database for Molecular Spectroscopy) database.

1.1 CASA

You can download the latest CASA version from https://svn.cv.nrao.edu/casa/linux_distro/

Untar the file with

```
tar xzvf casa-release-##version##.tar.gz
```

and change to the created directory and type

```
casa
```

at the command prompt.

1.2 Install interface

Before you start this installation script, please make sure that path of the current CASA distribution is already added to your PATH environment variable. You can do this by simply changing to the directory of your current CASA distribution and type

```
pwd
```

at the command prompt. Now, add the following line to your `.bashrc` and `.bash_profile`, respectively:

```
export PATH=`pwd`:$PATH
```

where `` `pwd` `` represents the path of the CASA directory, provided by the `pwd` command mentioned before.

After updating your PATH variable, decompress the file change to the new created directory and execute the shell script at the command prompt:

```
python install-in-casa.py --smp
```

This file installs the SMP parallelized version (see § 10.6.1) of the XCLASS-to-CASA interface, so that you can use it in CASA without any additional commands. Please note, that the XCLASS-interface requires `OpenMP`, `gcc` and `gfortran`).

---

[1] CASA home page: http://http://casa.nrao.edu/
[2] MAGIX home page: http://www.astro.uni-koeln.de/projects/schilke/MAGIX

Note, you have to re-execute the install script `install-in-casa.py` once again, if you move the XCLASS interface directory or if you install a new CASA version.

In order to install the MPI parallelized version, please add the MPI flag to the call of the installation script, i.e.

```
python install-in-casa.py --mpi
```

Please note, the MPI version requires the installation of OpenMPI[3] on all computers in the cluster.

1.3   Settings for parallelization

In order to use the parallelization of the interface, the user might increase the stack size for OpenMP by adding the following lines to the .bashrc (or .bash_profile)

```
ulimit -s unlimited
export KMP_STACKSIZE='3999M'
export OMP_STACKSIZE='3999M'
export GOMP_STACKSIZE='3999M'
```

Please note, if more or less RAM is available, please increase/decrease the value "3999" to a value useful for your machine.

1.4   Environment variables for XCLASS

The XCLASS interface creates so-called job directories for each function of the interface where all files created by a function call are stored in. By default, all these job-directories are stored in a so-called run directory which is created within the XCLASS root directory with name "run", i.e. "path-of-XCLASS-Interface/run/". Sometimes it is useful to create the run directory not within the XCLASS root directory. By defining the environment variable `myXCLASSRunDirectory`

```
export myXCLASSRunDirectory="run_somewhere_else"
```

the user can define another location for the run directory. Please note, a relative path has to be defined relative to the XCLASS root directory!

In addition to this environment variable, the MAGIX function requires further environment variables which are described in (§ 10.2).

1.5   Using XCLASS without CASA

Please note, using the XCLASS interface without CASA requires the installation of the following python packages: numpy (version 1.11.1 or newer), scipy (version 0.17.0 or newer), pyfits (version 3.3 or newer) (astropy, version 0.4.2 or newer)), matplotlib (version 1.5.3 or newer) and sqlite3 (version 2.6.0 or newer).

In order to use the XCLASS interface without CASA, it is necessary to extend the python sys.path environment variable. It is recommended to describe the path of the XCLASS installation by defining the `XCLASSRootDir` environment variable in the .bashrc file

```
export XCLASSRootDir="path-to-XCLASS"
```

---

[3]http://www.open-mpi.org/

In the following python example, the sys.path variable is extended and the MAGIX function is executed.

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# import sys package
import sys

# get environment variable for XCLASS root directory
XCLASSRootDir = str(os.environ.get('XCLASSRootDir', ''))
XCLASSRootDir = XCLASSRootDir.strip()
if (XCLASSRootDir != ""):
    NewModulesPaths = [XCLASSRootDir + "/build_tasks/"]
else:
    # use the following line to define XCLASS root directory manually
    NewModulesPath = "path-of-XCLASS-Interface/build_tasks/"

# extend sys.path variable
already_included_flag = False
for entries in sys.path:
  if (entries == NewModulesPath):
      already_included_flag = True
      break
  if (not already_included_flag):
      sys.path.append(NewModulesPath)

# import MAGIX package
import task_MAGIX

# define parameters for MAGIX
MAGIXExpXML = "Reflectance_Data.xml"
MAGIXInstanceXML = "parameters.xml"
MAGIXFitXML = "algorithm-settings.xml"
MAGIXRegXML = "Generalized_Drude-Lorentz__sym__freq-damping+Rp.xml"
MAGIXOption = ""

# start MAGIX function
task_MAGIX.MAGIX(MAGIXExpXML, MAGIXInstanceXML, MAGIXFitXML, MAGIXRegXML, \
                 MAGIXOption)
```

Please note, without CASA the new XCLASS functions can be used as python subroutines. So, it is necessary to define the input parameters in the right order. Therefore, the descriptions of the new functions contain the descriptions of the subroutine calls in python as well.

1.6 Using XCLASS in a SCREEN session

All XCLASS functions (except the `GetTransition` function) can be used in combination with the SCREEN command but without CASA.

In order to avoid a matplotlib error, the starting python script requires two additional lines at the beginning of the script

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import matplotlib
matplotlib.use('Agg')
import sys
```

```
# get environment variable for XCLASS root directory
XCLASSRootDir = str(os.environ.get('XCLASSRootDir', ''))
...
```

2 New Functions

The XCLASS-interface for CASA is a toolbox which makes the following functions available in CASA:

- ▶ UPDATEDATABASE (§ 3): This function downloads the latest version of the sqlite database from the CDMS server which is need by XCLASS.

- ▶ DATABASEQUERY (§ 4): This function sends a given query string to the XCLASS database.

- ▶ LISTDATABASE (§ 5): This function reads in entries from the sqlite database.

- ▶ GETTRANSITIONS (§ 6): This function reads in entries from the sqlite database table `transitions` around a selected frequency.

- ▶ LOADASCIIFILE (§ 7): A very primitive routine to import data from an ASCII file using the numpy.loadtxt function.

- ▶ MYXCLASSPLOT (§ 8): A simple plot routine.

- ▶ MYXCLASS (§ 9): The function starts the myXCLASS program and reads in the calculated spectrum.

- ▶ MAGIX (§ 10): CASA interface for MAGIX.

- ▶ MYXCLASSFIT (§ 11): Simplified interface for MAGIX with myXCLASS program to fit single spectra.

- ▶ MYXCLASSMAPFIT (§ 12): Simplified CASA interface for MAGIX with myXCLASS program to fit a complete data cube instead of a single spectrum.

- ▶ MYXCLASSMAPREDOFIT (§ 13): Redo one or more pixel fits of a previous myXCLASSMapFit function run.

- ▶ LINEIDENTIFICATION (§ 14): Line identification routine

Please note, that

- ▶ the user is free to define a different name for the output variable(s) by defining the names of the output variables in the following way:

```
expdata = LoadASCIIFile("demo/LoadASCIIFile/ASCII.dat", 0)
```

Here, the contents of the ASCII file "ASCII.dat" is stored in the variable `expdata`.

- ▶ whenever a function call requires an input parameter defining the path of a file, the path can be relative or absolute. Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ the XCLASS interface for CASA uses a sqlite 3 database. You can manage this database by using external sqlite browser like `sqliteman` [4] as well.

In the following we describe each new function in detail, which are available in CASA after the installation.

---

[4] http://sqliteman.com

3 UpdateDatabase

This function updates the XCLASS database file located in the directory `Database` from spectroscopic databases connected to the Virtual Atomic and Molecular Datacentre, VAMDC (www.vamdc.eu). Currently, data access to the CDMS database and the JPL catalog, is supported.

The XCLASS interface uses a SQLite3 database which contains two tables:

The values for the partition function for many molecules are saved within the sqlite database table `PartitionFunctions`. For each molecule the following data are saved:

- ► column 1: name of molecule

- ► column 6 - 115: the partition function for the temperatures (all in K): 1.072, 1.148, 1.23, 1.318, 1.413, 1.514, 1.622, 1.738, 1.862, 1.995, 2.138, 2.291, 2.455, 2.63, 2.725, 2.818, 3.02, 3.236, 3.467, 3.715, 3.981, 4.266, 4.571, 4.898, 5, 5.248, 5.623, 6.026, 6.457, 6.918, 7.413, 7.943, 8.511, 9.12, 9.375, 9.772, 10.471, 11.22, 12.023, 12.882, 13.804, 14.791, 15.849, 16.982, 18.197, 18.75, 19.498, 20.893, 22.387, 23.988, 25.704, 27.542, 29.512, 31.623, 33.884, 36.308, 37.5, 38.905, 41.687, 44.668, 47.863, 51.286, 54.954, 58.884, 63.096, 67.608, 72.444, 75, 77.625, 83.176, 89.125, 95.499, 102.329, 109.648, 117.49, 125.893, 134.896, 144.544, 150, 154.882, 165.959, 177.828, 190.546, 204.174, 218.776, 225, 234.423, 251.189, 269.153, 288.403, 300, 309.03, 331.131, 354.813, 380.189, 407.38, 436.516, 467.735, 500, 501.187, 537.032, 575.44, 616.595, 660.693, 707.946, 758.578, 812.831, 870.964, 933.254, 1000

Additionally, the data for many radiative transitions are stored in a table called `Transitions`. Here, the following data are saved for each radiative transition:

- ► column 1*: Name of the molecule

- ► column 2*: Frequency (in MHz)

- ► column 3: Intensity (in nm^2 MHz)

- ► column 4*: Einstein A coefficient (in s^{-1})

- ► column 5*: Uncertainty (in MHz)

- ► column 6*: Energy_Lower (in cm^{-1})

- ► column 7*: upper state degeneracy

- ► column 8: nuclear spin isomer

- ► column 9: HFS

- ► column 10: not used at the moment

- ► column 11: upper state quantum numbers

- ► column 12: lower state quantum numbers

Please note, if you add private entries to the database, please make sure, that the partition function is given for all temperatures described above and that all entries/columns in table "transitions" which are marked with a "*" sign are defined as well!

During the update process, the old sqlite database file is renamed by the current date and time. For example, the database is stored within the file "`cdms_sqlite.db`". The XCLASS interface renames this file to `cdms_sqlite__16-09-2013__10-40-15.db` and then downloads/updates the latest version of the database from the CDMS server. So, the database which is used for a previous simulated spectra is not removed.

Input parameters:

- ▶ `DBUpdateNew`: indicates, if a complete sqlite database file is downloaded from the CDMS web server ("new") or if the existing database file is updated ("update") via the VAMDC infrastructure, i.e. new entries are added to the tables and existing entries are overwritten with new data. The complete sqlite database file is regularly updated via the VAMDC infrastructure and contains data from CDMS and JPL.

Output parameters:

- ▶ `None`

Example 1:
```
DBUpdateNew = "new"
UpdateDatabase()
```

Example 2:
```
DBUpdateNew = "update"
UpdateDatabase()
```

Usage without CASA:
```
# extend sys.path variable
...


# import UpdateDatabase package
import task_UpdateDatabase


# call UpdateDatabase function
DBUpdateNew = "update"
task_UpdateDatabase.UpdateDatabase(DBUpdateNew)
```

4 DatabaseQuery

This function sends a given SQL query string to the XCLASS database.

Input parameters:

▶ `QueryString`: the query string

Output parameters:

▶ `Contents`: contains the screen output of the query command.

Please note, the user is free to define a different name for the output parameter.

Example:
```
QueryString = "select PF_Name from Partitionfunctions"
Contents = DatabaseQuery()
```

Usage without CASA:
```
# extend sys.path variable
...


# import DatabaseQuery package
import task_DatabaseQuery


# call DatabaseQuery function
QueryString = "select PF_Name from Partitionfunctions"
Contents = task_DatabaseQuery.DatabaseQuery(QueryString)
```

Please note, in the example described above all names stored in table "Partitionfunctions" are saved in the output variable "Contents". Here, "Contents[0]" includes the first name and so on. Additionally, you can use an external sqlite browser like `sqliteman`[5] to manage entries in the database as well.

---

[5]http://sqliteman.com

5 ListDatabase

This function reads in entries from the table `transitions` located in the SQLite3 database file and prints out the contents to the screen or file (defined by the input parameter `OutputDevice`). The user can limit the output by defining a minimum and maximum for the frequency (or for the lower energy) for the transitions.

Input parameters:

- ▶ `FreqMin`: minimum frequency (in MHz) for transitions in the `transitions` table (default: 0)

- ▶ `FreqMax`: maximum frequency (in MHz) for transitions in the `transitions` table (default: 10^8)

- ▶ `ElowMin`: minimum for lower energy (in K) (default: 0)

- ▶ `ElowMax`: maximum for lower energy (in K) (default: 10^6)

- ▶ `SelectMolecule`: a (python) list containing the names of all molecules which should be considered or a string defining the path and the name of an ASCII file including the molecules which should be selected.

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `OutputDevice`: path and name of the file where the output is written to. If no file name is defined, the contents of the database is written to the screen. If this parameter is set to ``quiet'' no informations are printed to screen.

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

Output parameters:

- ▶ `Contents`: contents of the database (as an python array, e.g. `Contents[0]` contains the entries for the first molecule within the frequency/energy range).

  Note, the user is free to define a different name for the output parameter.

Example 1:
```
FreqMin = 20000.0
FreqMax = 20100.0
ElowMin = 100.0
ElowMax = 1000.0
SelectMolecule = "demo/ListDatabase/molecules.txt"
OutputDevice = ""
Contents = ListDatabase()
```

Example 2:
```
FreqMin = 20000.0
FreqMax = 20100.0
ElowMin = 100.0
ElowMax = 2000.0
SelectMolecule = []
OutputDevice = ""
Contents = ListDatabase()
```

Please note, that in the last example all molecules located in the defined range are printed out to screen.

Usage without CASA:

```
# extend sys.path variable
...


# import ListDatabase package
import task_ListDatabase


# call ListDatabase function
FreqMin = 20000.0
FreqMax = 20100.0
ElowMin = 100.0
ElowMax = 2000.0
SelectMolecule = []
OutputDevice = ""
Contents = task_ListDatabase.ListDatabase(FreqMin, FreqMax, ElowMin, \
                                          ElowMax, SelectMolecule, \
                                          OutputDevice)
```
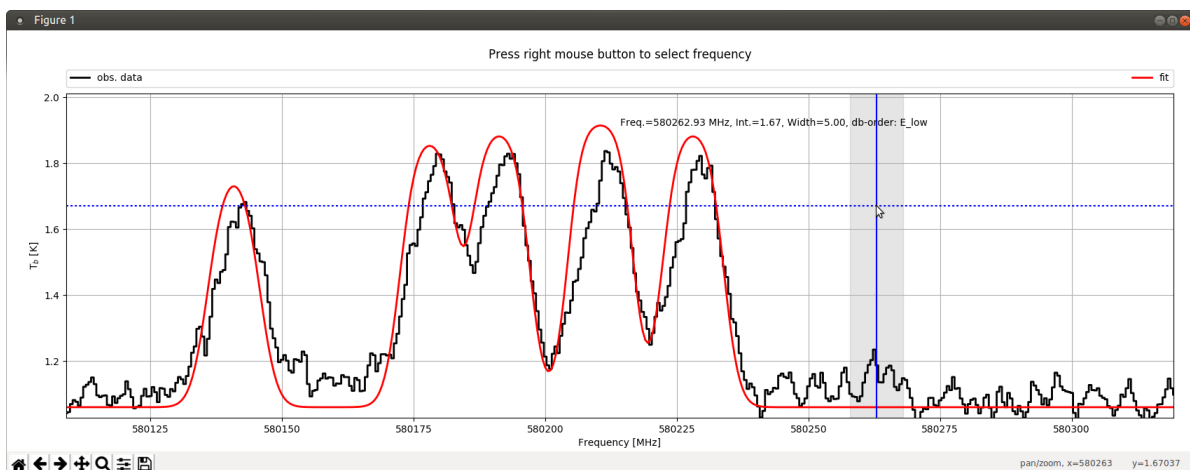
Figure 1: Graphical user interface (GUI) of the `GetTransitions` function. Here, some observational data (black line) are shown together with a synthetic spectrum (red line). By using the right mouse button the user defines the central frequency (blue vertical line) of a frequency range. The width of the range is given by an user defined parameter and can be enlarged (shrinked) by using the "+" ("-") key. In the following XCLASS prints out informations (name of molecule, transition frequency, uncertainty of transition frequency, Einstein A coefficient, quantum number for lower and upper state, and energy of lower state) of all transitions in the database located in the given frequency range.

6 GetTransitions

Similar to the `ListDatabase` function, the `GetTransitions` function displays informations from the embedded database about transitions within a user-defined frequency range. Here, the frequency range selection is done in an interactive way. Therefore, the `GetTransitions` function starts a graphical user interface (GUI) which describes observational data (defined by parameter `expdata`) within a frequency range defined by the parameters `FreqMin` and `FreqMax`. Additionally, the functions offers the possibility to plot a synthetic spectrum (defined by parameter `modeldata`) of a previous `myXCLASS` or `myXCLASSFit` function call as well.

By using the mouse in combination with the right mouse button, the user defines the central frequency (blue vertical line) of the frequency range (grey vertical bar) which is used for the database query. The initial half width of this range is defined by parameter `FrequencyWidth`. Note, the width of the range can be enlarged (shrinked) by pressing the "+" ("-") key. In the following XCLASS prints out informations of all transitions within the selected frequency range to the screen starting with the transition with the lowest transition frequency. In order to sort the transitions by lower energies, the user has to press the "1" key before selecting the central frequency. Using the "2" ("3") key, sorts the transitions by the Einstein A coefficients (by a products of upper state degeneracies and Einstein A coefficients (gA). Pressing the "0" key restores the initial order, i.e. by transition frequencies.

Additionally, the user can reduce the number of transitions by defining a lower and a upper limit for the lower energies using parameters `ElowMin` and `ElowMax`, respectively. Furthermore, the screen output can be limited to a few molecules by using parameter `SelectMolecule`.

Note, the GUI has to be closed to stop the function!

Input parameters:

- ▶ `expdata`: 2D numpy array containing the observational data.

  Note, the data has to be given as a function of frequency (in MHz).

- ▶ `modeldata`: 2D numpy array containing the synthetic spectrum.

  Note, the data has to be given as a function of frequency (in MHz).

- ▶ `FreqMin`: minimum frequency (in MHz) for the observational data (default: 0).

- ▶ `FreqMax`: maximum frequency (in MHz) for the observational data (default: 10^8).

- ▶ `SelectMolecule`: a (python) list containing the names of all molecules which should be considered or a file name including the molecules which should be selected.

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `FrequencyWidth`: defines the width of the frequency interval in MHz (selected central frequency ±`FrequencyWidth` where the line informations are printed out to the screen. (default: 5)

- ▶ `ElowMin`: minimum for lower energy (in K) (default: 0) in table `transitions`.

- ▶ `ElowMax`: maximum for lower energy (in K) (default: 10^6) in table `transitions`.

Output parameters:

- ▶ `None`

Example 1:

```
FreqMin = 580102.0
FreqMax = 580546.5
FreqStep = 0.5
TelescopeSize = 3.5
Inter_Flag = False
t_back_flag = True
tBack = 1.06
tslope = 0.0
nH_flag = True
N_H = 3.E+20
beta_dust = 2.0
kappa_1300 = 0.02
MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
iso_flag = T
IsoTableFileName = "demo/myXCLASS/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
modeldata, log, TransEnergies, IntOptical, jobDir = myXCLASS()

FileName = "demo/myXCLASS/band1b.dat"
NumHeaderLines = 1
expdata = LoadASCIIFile()

FreqMin = 580102.0
FreqMax = 580546.5
SelectMolecule = "demo/ListDatabase/molecules.txt"
```

```
    FrequencyWidth = 5.0
    ElowMin = 0.0
    ElowMax = 1000.0
    GetTransitions()
```

Example 2:

```
    FreqMin = 580102.0
    FreqMax = 580546.5
    FreqStep = 0.5
    TelescopeSize = 3.5
    Inter_Flag = False
    t_back_flag = True
    tBack = 1.06
    tslope = 0.0
    nH_flag = True
    N_H = 3.E+20
    beta_dust = 2.0
    kappa_1300 = 0.02
    MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
    iso_flag = T
    IsoTableFileName = "demo/myXCLASS/iso_names.txt"
    RestFreq = 0.0
    vLSR = 0.0
    modeldata, log, TransEnergies, IntOptical, jobDir = myXCLASS()

    FileName = "demo/myXCLASS/band1b.dat"
    NumHeaderLines = 1
    expdata = LoadASCIIFile()

    FreqMin = 580102.0
    FreqMax = 580546.5
    SelectMolecule = ["CH3OH;v=0;", "SO2;v=0;"]
    FrequencyWidth = 5.0
    ElowMin = 0.0
    ElowMax = 3000.0
    GetTransitions()
```

Usage without CASA:

```
    # extend sys.path variable
    ...

    # import XCLASS functions
    import task_LoadASCIIFile
    import task_GetTransitions
    import task_myXCLASS

    # use myXCLASS function to create synthetic spectrum
    FreqMin = 580102.0
    FreqMax = 580546.5
    FreqStep = 0.5
    TelescopeSize = 3.5
    Inter_Flag = False
    t_back_flag = True
    tBack = 1.06
    tslope = 0.0
    nH_flag = True
    N_H = 3.E+20
    beta_dust = 2.0
    kappa_1300 = 0.02
    MolfitsFileName = XCLASSBaseDir + "demo/myXCLASS/CH3OH__pure.molfit"
```

```
iso_flag = True
IsoTableFileName = XCLASSBaseDir + "demo/myXCLASS/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
modeldata, log, TransEnergies, IntOpt, jobDir = task_myXCLASS.myXCLASS(
                                     FreqMin, FreqMax, FreqStep, \
                                     TelescopeSize, Inter_Flag, \
                                     t_back_flag, tBack, \
                                     tslope, nH_flag, N_H, \
                                     beta_dust, kappa_1300, \
                                     MolfitsFileName, \
                                     iso_flag, IsoTableFileName, \
                                     RestFreq, vLSR)


# use LoadASCIIFile function to import obs. data
FileName = XCLASSBaseDir + "demo/myXCLASS/band1b.dat"
NumHeaderLines = 1
RestFreq = 0.0
vLSR = 0.0
expdata = task_LoadASCIIFile.LoadASCIIFile(FileName, NumHeaderLines, \
                                     RestFreq, vLSR)

# call GetTransitions function
SelectMolecule = []
FrequencyWidth = 5.0
ElowMin = 0.0
ElowMax = 3000.0
task_GetTransitions.GetTransitions(expdata, FreqMin, FreqMax, \
                                     SelectMolecule, FrequencyWidth, \
                                     ElowMin, ElowMax, modeldata)
```

17

7 LoadASCIIFile

A very primitive routine to import data from an ASCII as well as FITS file.

Input parameters:

- ▶ `FileName`: path and name of an ASCII or a FITS file.

  Note, a relative path has to be defined relative to the current working directory! Additionally, FITS files has to end with `.fits`.

- ▶ `NumHeaderLines` (has to be defined only for ASCII files): number of header lines, which are ignored (default: 0).

- ▶ `RestFreq` (has to be defined only for ASCII files): rest frequency in MHz (default: 0). (If this parameter is set to zero, the intensity is plotted against frequency (in MHz) otherwise against velocity (in km s^{-1}).

- ▶ `vLSR` (has to be defined only for ASCII files): velocity (local standard of rest) in km s^{-1} (default: 0), only used, if RestFreq $\neq$ 0. (velocity(Frequency = RestFreq) = vLSR)

Output parameters:

- ▶ `ASCIIdata`: contents of the ASCII (FITS) file (as an python array, e.g. `ASCIIdata[0]` contains the data of the first data point). For FITS files, the first n columns of the python array describe the coordinates, e.g. rectascension, declination, and frequency, whereas the last column describe the corresponding intensity (in e.g. Jy/beam or Kelvin).

  Note, the user is free to define a different name for the output parameter.

Example:

```
FileName = "demo/LoadASCIIFile/ASCII.dat"
NumHeaderLines = 0
RestFreq = 0.0
vLSR = 0.0
ASCIIdata = LoadASCIIFile()
```

Usage without CASA:

```
# extend sys.path variable
...


# import LoadASCIIFile package
import task_LoadASCIIFile


# call LoadASCIIFile function
FileName = "demo/LoadASCIIFile/ASCII.dat"
NumHeaderLines = 0
RestFreq = 0.0
vLSR = 0.0
expdata = task_LoadASCIIFile.LoadASCIIFile(FileName, NumHeaderLines, \
                                           RestFreq, vLSR)
```
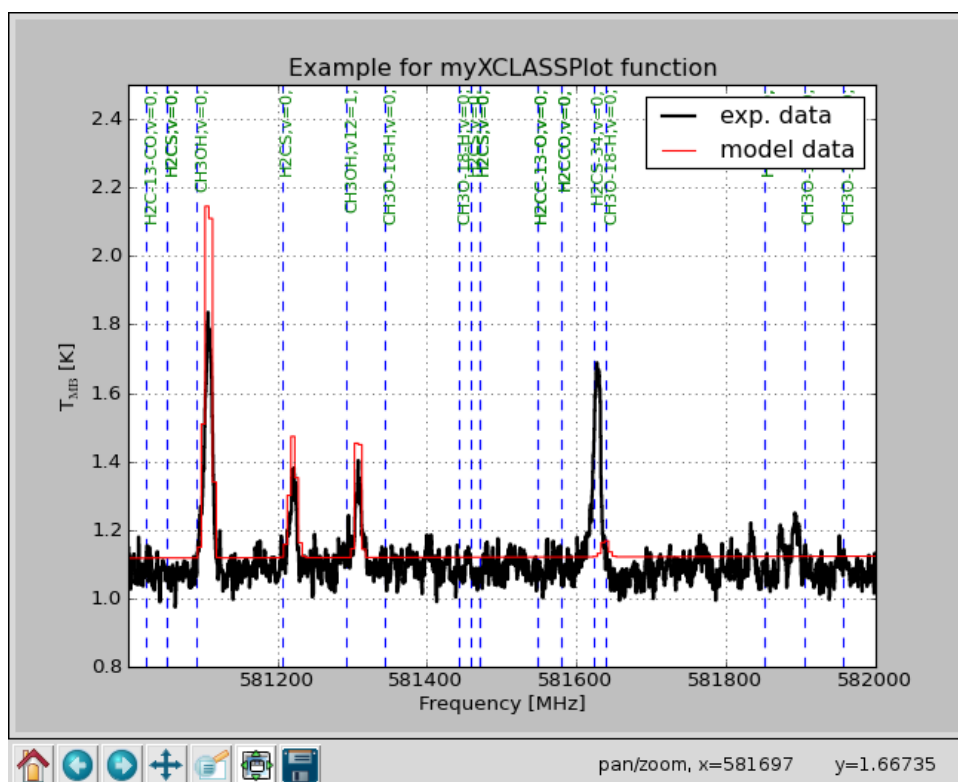
Figure 2: Example of the myXCLASSPlot GUI. Here the experimental data (black line) are shown together with the modeled spectrum (red line). Additionally, the transition frequencies (dotted blue lines) are shown with the corresponding molecule names (green).

8 myXCLASSPlot

This function creates a plot showing the (observational) data stored in the parameter `expdata` as black dots, the calculated model function stored in the parameter `modeldata` as green line and the transition energies from the table `transitions` stored in the parameter `TransEnergies` as blue vertical lines.

Additionally, the user is free to define a rest frequency (in MHz) and a velocity (local standard of rest) in km s^{-1} to draw the data as a function of velocity (in km s^{-1}).

Input parameters:

- ▶ `expdata`: 2D numpy array containing the observational data

  Note, the data has to be given as a function of frequency (in MHz).

- ▶ `modeldata`: 2D numpy array containing the values of a model function

  Note, the data has to be given as a function of frequency (in MHz).

- ▶ `TransEnergies`: python list containing the transition frequencies (in MHz) from the last myXCLASS run, the Doppler-shifted transition frequencies (in MHz), the corresponding intensities (in K), the energy of the lower level (in K), the upper state degeneracy, the Einstein A coefficient (in s^{-1}), and the molecule names.

- ▶ `RestFreq`: rest frequency in MHz (default: 0). (If this parameter is set to zero, the intensity is plotted against frequency (in MHz) otherwise against velocity (in km s^{-1}).

- ▶ vLSR: velocity (local standard of rest) in km s^{-1} (default: 0), only used, if RestFreq $\neq$ 0. (velocity(Frequency = RestFreq) = vLSR)

- ▶ MinIntensity: minimal intensity (in K) of a transition for plotting molecule names (default: 0, i.e. plot all names)

- ▶ xLowerLimit: lower limit (in MHz / km s^{-1}) of the frequency/velocity (default: 0). (Depending on the value of the rest frequency: If rest frequency is set to zero, the lower limit has to be given as frequency) If parameter is not given, all data will be plotted.

- ▶ xUpperLimit: upper limit (in MHz / km s^{-1}) of the frequency/velocity (default: 10^8). (Depending on the value of the rest frequency: If rest frequency is set to zero, the upper limit has to be given as frequency) If parameter is not given, all data will be plotted.

- ▶ yLowerLimit: lower limit of the intensity (y-axis) (default: 0). (If value is not given or if yUpperLimit is equal to yLowerLimit then the y-axis range is set automatically.)

- ▶ yUpperLimit: upper limit of the intensity (y-axis) (default: 0). (If value is not given or if yUpperLimit is equal to yLowerLimit then the y-axis range is set automatically.)

- ▶ PlotTitle: defines the title of the plot (default: "")

- ▶ LegendFlag: defines, if legend is plotted (true) or not (false) (default: "T")

- ▶ SaveFigureFile: defines the path and name of the file to which the current figure is stored. If no file is defined, i.e. "", the figure is shown in a GUI and not saved (default: "").

Output parameters:

- ▶ None

Example:

```
FreqMin = 580102.0
FreqMax = 580546.5
FreqStep = 5.0000000000E-01
TelescopeSize = 3.5
Inter_Flag = F
t_back_flag = T
tBack = 1.1
tslope = 0.0000000000E+00
nH_flag = T
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
iso_flag = T
IsoTableFileName = "demo/myXCLASS/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
modeldata, log, TransEnergies, IntOptical, jobDir = myXCLASS()

FileName = "demo/myXCLASS/band1b.dat"
NumHeaderLines = 1
expdata = LoadASCIIFile()

MinIntensity = 0.0
xLowerLimit = 580102.0
```

```
    xUpperLimit = 580546.5
    yLowerLimit = 0.8
    yUpperLimit = 2.5
    PlotTitle = "Example for myXCLASSPlot function"
    LegendFlag = T
    SaveFigureFile = "test.png"
    myXCLASSPlot()
```

Usage without CASA:

```
    # extend sys.path variable
    ...


    # import task_myXCLASS, task_LoadASCIIFile, and task_myXCLASSPlot packages
    import task_myXCLASS
    import task_LoadASCIIFile
    import task_myXCLASSPlot


    # call myXCLASS, LoadASCIIFile, and myXCLASSPlot functions
    FreqMin = 580102.0
    FreqMax = 580546.5
    FreqStep = 5.0000000000E-01
    TelescopeSize = 3.5
    Inter_Flag = False
    t_back_flag = True
    tBack = 1.1
    tslope = 0.0000000000E+00
    nH_flag = True
    N_H = 3.0000000000E+24
    beta_dust = 2.0
    kappa_1300 = 0.02
    MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
    iso_flag = True
    IsoTableFileName = "demo/myXCLASS/iso_names.txt"
    RestFreq = 0.0
    vLSR = 0.0
    modeldata, log, TransEnergies, IntOpt, jobDir = task_myXCLASS.myXCLASS(\
                                              FreqMin, FreqMax, FreqStep, \
                                              TelescopeSize, Inter_Flag, \
                                              t_back_flag, tBack, \
                                              tslope, nH_flag, N_H, \
                                              beta_dust, kappa_1300, \
                                              MolfitsFileName, \
                                              iso_flag, \
                                              IsoTableFileName, \
                                              RestFreq, vLSR)
    FileName = "demo/myXCLASS/band1b.dat"
    NumHeaderLines = 1
    expdata = task_LoadASCIIFile.LoadASCIIFile(FileName, NumHeaderLines, \
                                              RestFreq, vLSR)
    MinIntensity = 0.0
    xLowerLimit = 580102.0
    xUpperLimit = 580546.5
    yLowerLimit = 0.8
    yUpperLimit = 2.5
    PlotTitle = "Example for myXCLASSPlot function"
    LegendFlag = True
    SaveFigureFile = "test.png"
    task_myXCLASSPlot.myXCLASSPlot(expdata, modeldata, TransEnergies, RestFreq, \
```

```
                            vLSR , MinIntensity , xLowerLimit , \
                            xUpperLimit , yLowerLimit , yUpperLimit , \
                            PlotTitle , LegendFlag , SaveFigureFile )
```

Please note, logical variables in python has to be defined with `True` and `False` instead of `T` and `F` in CASA.
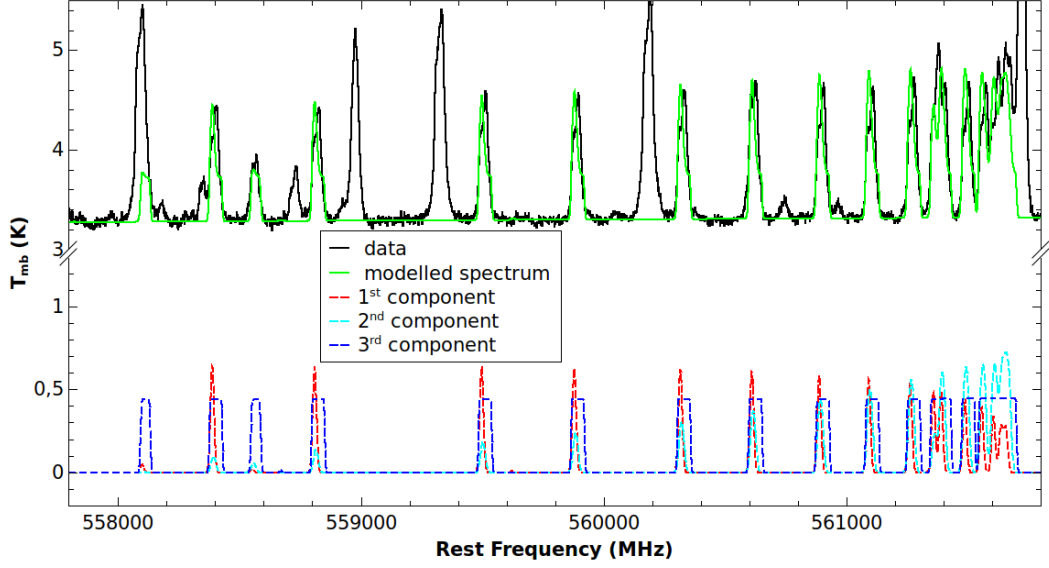
Figure 3: Here, the MYXCLASS function was used to model HIFI data of SgrB2m (black) using SO_2 with three different components. The intensities of each component is shown in the bottom half.

9 myXCLASS

The function models a spectrum by solving the radiative transfer equation for an isothermal object in one dimension, called detection equation. Furthermore, non-extended sources and dust attenuation is considered.

The files produced by a MYXCLASS function call are copied to a so-called "job-directory" located in the XCLASS interface working directory `path-of-XCLASS-Interface/run/myXCLASS/`! The name of a job directory for a myXCLASS run is made up of four components: The first part consists of the phrase "`job_`" whereas the second and third part describe the date and the time stamp (hours, minutes, seconds) of the function execution, respectively. The last part indicates a so-called job ID which is composed of the so-called PID followed by a four digit random integer number to create a really unambiguous job number, e.g.

`path-of-XCLASS-Interface/run/myXCLASS/job__25-07-2013__12-02-03__189644932/`

Input parameters:

- ▶ `FreqMin`: start frequency of simulated spectrum (in MHz), (default: 0).

- ▶ `FreqMax`: end frequency of simulated spectrum (in MHz), (default: 10^8).

- ▶ `FreqStep`: step frequency of simulated spectrum (in MHz), (default: 1).

- ▶ `vLSR`: velocity (local standard of rest) in km s^{-1} (default: 0) used in the calculation of the synthetic spectra, i.e. all velocity offsets described in the molfit file have to be defined relative to the v_{LSR} parameter. For example, the user defines $v_{LSR} = 20$ km/s, and the molfit file describes CH_3CN with one component and $v_{off}^{molfit} = 2$ km/s, XCLASS uses $v_{off} = v_{LSR} + v_{off}^{molfit} = 20$ km/s $+ 2$ km/s $= 22$ km/s for the calculation of the spectra.

  Additionally, XCLASS considers the v_{LSR} parameter for the determination of the transition frequencies contained in the database: For each frequency range, XCLASS shifts the

lowest and highest frequency by v_{LSR} (e.g. to describe lines in high-redshifted galaxies) and expands each range by taking the lowest and highest velocity offset defined in the molfit file into account. This offers the possibility to describe lines which are located at the edges of the frequency ranges.

▶ `TelescopeSize`: for single dish observations (`Inter_Flag = F`): `TelescopeSize` describes the size of telescope (in m), (default: 1); for interferometric observations (`Inter_Flag = T`): `TelescopeSize` describes the interferometric beam FWHM size (in arcsec), (default: 1).

▶ `Inter_Flag` (T/F): defines, if single dish (F) or interferometric observations (T) are described, (default: F).

▶ `t_back_flag` (T/F): defines, if the user defined background temperature T_{bg} and temperature slope T_{slope} given by the input parameters `tBack` and `tslope` describe the continuum contribution completely (`t_back_flag = T`) or not (`t_back_flag = F`) (default: T).

▶ `tBack` background temperature (in K), (default: 0).

▶ `tslope` temperature slope (dimensionless), (default: 0).

▶ `nH_flag` (T/F): defines, if column density, spectral index for dust and kappa are given by the molfit file (F) or if `nH_flag` is set to T, the following three parameters define the H density, spectral index for dust and kappa for all components (default: T).

▶ `N_H` (has to be given only if `nH_flag` is set to T): Hydrogen column density (in cm^{-2}), (default: 10^{24}).

▶ `beta_dust` (has to be given only if `nH_flag` is set to T): spectral index for dust (dimensionless), (default: 0.1).

▶ `kappa_1300` (has to be given only if `nH_flag` is set to T): kappa at 1.3 mm (cm^2 g^{-1}), (default: 0.01).

▶ `MolfitsFileName`: ABSOLUTE path and name of the molfit file, including the `source_size` (in arcsec), `T_rot` (rotation temperature in K), `N_tot` (total column density in cm^{-2}), `V_width` (velocity width in km s^{-1}), `V_off` (velocity offset in km s^{-1}), `AbsorptionFlag` (core or foreground). A detailed description of the molfit file is given in section (§ 9.2).

▶ `iso_flag`: use isotopologues (T/F). If `iso_flag` is set to T isotopologues defined in the iso ratio file are used (default: F).

▶ `IsoTableFileName` (has to be given only if `iso_flag` is set to T): ABSOLUTE path and file name of an ASCII file including the iso ratios between certain molecules. The so-called "iso ratio file" defines the iso ratios between molecules. The ASCII file consists of three columns, where the first two columns indicates the molecules, respectively. The third column defines the ratio for both molecules. The columns are separated by blanks or tabs. So, the names of the molecules must not contain blanks. Comments have to be marked with the "%" sign, i.e. all characters on the right side of a "%" sign are ignored.

The MYXCLASSFIT function offers the possibility to optimize the ratios between isotopologues as well. For that purpose, the user has to add two additional columns on the right indicating the lower and the upper limit of a certain ratio, respectively. A detailed description of the iso ratio file is given in section (§ 9.3).

► `RestFreq`: rest frequency in MHz (default: 0). (If this parameter is unequal zero, the intensity is also plotted against velocity (in km s^{-1}) using parameter `vLSR`, see above.

Output parameters:

► `myXCLASSspectrum`: contains the calculated myXCLASS spectrum

If `RestFreq` is unequal zero, the MYXCLASS function adds a column to the output parameter `myXCLASSspectrum` which contains the velocities. So, for a rest frequency unequal zero, the parameter `myXCLASSspectrum` represents a python array with three columns, where the first column describes the frequencies, the second column describes the velocities and the third column the corresponding intensities.

► `myXCLASSlog`: contains the corresponding log file

► `myXCLASSTrans`: (python) list containing the transition frequencies (in MHz) from the last myXCLASS run, the Doppler-shifted transition frequencies (in MHz), the corresponding absolute and integrated intensities (in K), the energy of the lower level (in K and K MHz), the upper state degeneracy, the Einstein A coefficient (in s^{-1}), and the molecule names within the defined range

► `myXCLASSIntOptical`: contains intensities and optical depths for each molecule and component

The intensities and optical depths are stored as follows:

   – `myXCLASSIntOptical[0]`, contains all informations about the intensities

     * `myXCLASSIntOptical[0][i][0]` contains the name of the ith molecule

     * `myXCLASSIntOptical[0][i][1]` contains the index of the component of the ith molecule

     * `myXCLASSIntOptical[0][i][2]` contains the intensities of the ith molecule as a 2D numpy array.

     * `myXCLASSIntOptical[0][i][3]` contains the integrated intensities of the ith molecule.

Please note, the intensities for each core component $[T_{\text{mb}}^{\text{core}}]^{m,c}(\nu)$ are given by

$$[T_{\text{mb}}^{\text{core}}]^{m,c}(\nu) = \left[\eta\left(\theta^{m,c}\right)\left[S^{m,c}(\nu)\left(1 - e^{-\tau_{\text{total}}^{m,c}(\nu)}\right) + I_{\text{bg}}^{\text{core}}(\nu)\left(e^{-\tau_{\text{total}}^{m,c}(\nu)} - 1\right)\right]\right]$$
$$+ \frac{1}{N_{\text{all}}^{\text{core}}}\left[I_{\text{bg}}^{\text{core}}(\nu) - J_{\text{CMB}}\right], \tag{1}$$

where $N_{\text{all}}^{\text{core}}$ indicates the total number of all core components of all molecules.

The intensities for each foreground component $\left[T_{\text{mb}}^{\text{fore}}\right]^{m,c=i}$ are given by

$$\left[T_{\text{mb}}^{\text{fore}}\right]^{m,c=i}(\nu) = \left[S^{m,c=i}(\nu)\left(1 - e^{-\tau_{\text{total}}^{m,c=i}(\nu)}\right) + \left[T_{\text{mb}}^{\text{fore}}\right]^{m,c=(i-1)}e^{-\tau_{\text{total}}^{m,c=i}(\nu)}\right], \tag{2}$$

with $\left[T_{\text{mb}}^{\text{fore}}\right]^{m,c=(0)}(\nu) \equiv T_{\text{mb}}^{\text{core}}(\nu)$.

- – `myXCLASSIntOptical[1]`, contains all informations about the optical depths.
  - ∗ `myXCLASSIntOptical[1][i][0]` contains the name of the ith molecule
  - ∗ `myXCLASSIntOptical[1][i][1]` contains the index of the component of the ith molecule
  - ∗ `myXCLASSIntOptical[1][i][2]` contains the optical depth of the ith molecule as a 2D numpy array

- ▶ `myXCLASSJobDir`: absolute path of the job directory created for the current run

Note, the user is free to define different names for the output parameters. In the example describe below, we use the name `"modeldata"` for output parameter `myXCLASSspectrum`, `"log"` for output parameter `myXCLASSlog`, `"TransEnergies"` for output parameter `myXCLASSTrans`, `"IntOptical"` for output parameter `myXCLASSIntOptical`, and `"jobDir"` for output parameter `myXCLASSJobDir`.

Example:

```
FreqMin = 580102.0
FreqMax = 580546.5
FreqStep = 5.0000000000E-01
TelescopeSize = 3.5
Inter_Flag = F
t_back_flag = T
tBack = 1.1
tslope = 0.0000000000E+00
nH_flag = T
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
iso_flag = T
IsoTableFileName = "demo/myXCLASS/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
modeldata, log, TransEnergies, IntOptical, jobDir = myXCLASS()

FileName = "demo/myXCLASS/band1b.dat"
NumHeaderLines = 1
expdata = LoadASCIIFile()

MinIntensity = 0.0
xLowerLimit = 580102.0
xUpperLimit = 580546.5
yLowerLimit = 0.8
yUpperLimit = 2.5
PlotTitle = "Example for myXCLASSPlot function"
LegendFlag = T
SaveFigureFile = ""
myXCLASSPlot()
```

Usage without CASA:

```
# extend sys.path variable
...


# import task_myXCLASS, task_LoadASCIIFile, and task_myXCLASSPlot packages
import task_myXCLASS
import task_LoadASCIIFile
import task_myXCLASSPlot
```

```
# call myXCLASS, LoadASCIIFile, and myXCLASSPlot functions
FreqMin = 580102.0
FreqMax = 580546.5
FreqStep = 5.0000000000E-01
TelescopeSize = 3.5
Inter_Flag = False
t_back_flag = True
tBack = 1.1
tslope = 0.0000000000E+00
nH_flag = True
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
MolfitsFileName = "demo/myXCLASS/CH3OH__pure.molfit"
iso_flag = True
IsoTableFileName = "demo/myXCLASS/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
modeldata, log, TransEnergies, IntOpt, jobDir = task_myXCLASS.myXCLASS(
                                    FreqMin, FreqMax, FreqStep, \
                                    TelescopeSize, Inter_Flag, \
                                    t_back_flag, tBack, \
                                    tslope, nH_flag, N_H, \
                                    beta_dust, kappa_1300, \
                                    MolfitsFileName, \
                                    iso_flag, IsoTableFileName, \
                                    RestFreq, vLSR)
```

Please note, logical variables in python has to be defined with `True` and `False` instead of `T` and `F` in CASA. Additionally, using the XCLASS functions outside of CASA requires the definition of all parameters, e.g. parameters such as `N_H`, `beta_dust`, and `kappa_1300` have to be defined independent of the value of `nH_flag`.

Note, the bottom half in the example load the experimental data from file and plots the calculated spectrum using the MYXCLASSPLOT function.

9.1   What is myXCLASS?

The MYXCLASS function models a spectrum by solving the radiative transfer equation for an isothermal object in one dimension, called *detection equation* {Stahler & Palla 2005}. Here, LTE is assumed, i.e. the source function is given by the Planck function of an excitation temperature, which does not need to be the physical temperature, but is constant for all transitions. The `myXCLASS` function is designed to describe line-rich sources which are often dense, where LTE is a reasonable approximation. Additionally, a non-LTE description requires collision rates which are available only for a few molecules.

The `myXCLASS` function is able to model a spectrum with an arbitrary number of molecules where the contribution of each molecule is described by an arbitrary number of components. The 1-d assumption imposes a very simplistic geometrical structure. We recognize two classes of components.

One, the core objects (in earlier implementations called emission component), consists of an ensemble of objects centered at the middle of the beam. These could be identified with clumps, hot dense cores etc. which overlaps but do not interact either because they do not overlap in physical or in velocity space. For computational convenience, they are assumed to be centered
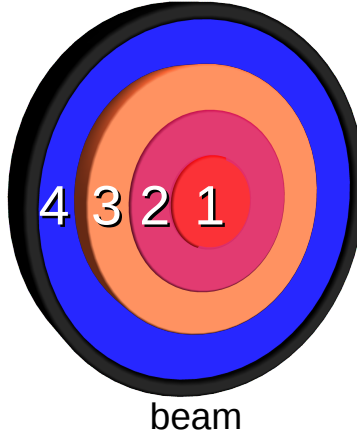
Figure 4: Sketch of a distribution of core layers within the Gaussian beam of the telescope (black ring). Here, we assume three different core components **1**, **2**, and **3**, centered at the middle of the beam with different source sizes, excitation temperatures, velocity offsets (relative to v_{LSR}) etc. indicated by different colors. Additionally, we assume that all core components have the same distance to the telescope, i.e. all core layers are located within a plane perpendicular to the line of sight. Furthermore, we assume that this plane is located in front of a background layer **4** with homogeneous intensity $I_{bg}^{core}(\nu)$ over the whole beam. Core components do not interact with each other radiatively.

in the beam, as shown in Fig. 4. It is also assumed that the dust emission emanates (partly) from these components. Their intensities are added, weighted with the beam filling factor, see Eq. (5).

The second class, foreground objects (in earlier implementations called absorption components), are assumed to be in layers in front of the core components. In the current 1-d implementation, they would have a beam-averaged intensity of the core sources as background, and would fill the whole beam. Examples for such structures would be source envelopes in front of dense cores, or foreground components along the line-of-sight.

As shown in Fig. 4, we assume that core components do not interact with each other radiatively, i.e. one core layer is not influenced by the others. But the core layers may overlap to offer the possibility to model sources consisting of several molecules and compounds. The solution of the radiative transfer equation for core layers is[6],

$$T_{mb}^{core}(\nu) = \sum_m \sum_c \left[\eta\left(\theta^{m,c}\right) \left[S^{m,c}(\nu) \left(1 - e^{-\tau_{total}^{m,c}(\nu)}\right) + I_{bg}^{core}(\nu) \left(e^{-\tau_{total}^{m,c}(\nu)} - 1\right) \right] \right] \qquad (3)$$

$$+ \left(I_{bg}^{core}(\nu) - J_{CMB} \right), \qquad (4)$$

where the sums go over the indices m for molecule, and c for (core) component, respectively. In the following we will briefly describe each term in Eq. (3).

The beam filling (dilution) factor $\eta(\theta^{m,c})$ of molecule m and component c in Eq. (3) for a source with a Gaussian brightness profile, see below, and a Gaussian beam is given by[7]

$$\eta(\theta^{m,c}) = \frac{(\theta^{m,c})^2}{\theta_t(\nu)^2 + (\theta^{m,c})^2}, \qquad (5)$$

---

[6] A derivation of the expression can be found in the appendix A.1.

[7] Derivations of the beam filling factor Eq. (5), are described in the appendix A.2.

where $\theta^{m,c}$ and θ_t represents the source and telescope beam full width half maximum (FWHM) sizes, respectively. The sources beam FWHM sizes $\theta^{m,c}$ for the different components are defined by the user in the molfit file, described in Sect. 9.2. Additionally, the MYXCLASS program assumes for single dish observations (indicated by `Inter_Flag = F`), that the telescope beam FWHM size is related to the diameter of the telescope by the diffraction limit

$$\theta_t(\nu) = \left(1.22 \cdot \frac{\lambda}{D}\right) \cdot \xi = \left(1.22 \cdot \frac{c_{\text{light}}}{\nu\,D}\right) \cdot \xi, \tag{6}$$

where D describes the diameter of the telescope, c_{light} the speed of light, and $\xi = 3600 \cdot 180/\pi$ a conversion factor to get the telescope beam FWHM size in arcsec. For interferometric observations (indicated by `Inter_Flag = T`, the user has to define the interferometric beam FWHM size directly using parameter `TelescopeSize`. In contrast to single dish observations the MYXCLASS program assume a constant interferometric beam FWHM size for the whole frequency range.

The term $\eta_{\text{max}}^{\text{core}}$ in Eq. (3) indicates the largest beam filling factor of all core components of all molecules, i.e. $\eta_{\text{max}}^{\text{core}} = \max\limits_{m,c}\{\eta(\theta^{m,c})\}$.

In general, the brightness temperature of radiation temperature $J(T, \nu)$ is defined as

$$J(T, \nu) = \frac{h\,\nu}{k_B}\frac{1}{e^{h\,\nu/k\,T} - 1}. \tag{7}$$

The expression J_{CMB} describes the radiation temperature Eq. (7) of the cosmic background $T_{\text{cbg}} = 2.7$ K, i.e. $J_{\text{CMB}} \equiv J(T_{\text{cbg}}, \nu)$.

In Eq. (3), the expression $S^{m,c}(\nu)$ represents the source function and is according to Kirchhoff's law of thermal radiation given by

$$\begin{aligned}
S^{m,c}(\nu) &= \frac{\epsilon_l^{m,c}(\nu) + \epsilon_d^{m,c}(\nu)}{\kappa_l^{m,c}(\nu) + \kappa_d^{m,c}(\nu)} \\
&= \frac{\kappa_l^{m,c}(\nu)\,J(T_{\text{ex}}^{m,c}, \nu) + \kappa_d^{m,c}(\nu)\,J(T_d^{m,c}, \nu)}{\kappa_l^{m,c}(\nu) + \kappa_d^{m,c}(\nu)} \\
&= (1 - \delta_{\gamma,0}) \cdot \left[\frac{\tau_l^{m,c}(\nu)\,J(T_{\text{ex}}^{m,c}, \nu) + \tau_d^{m,c}(\nu)\,J(T_d^{m,c}, \nu)}{\tau_l^{m,c}(\nu) + \tau_d^{m,c}(\nu)}\right] + \delta_{\gamma,0}\,J(T_{\text{ex}}^{m,c}, \nu),
\end{aligned} \tag{8}$$

where $\epsilon_{l,d}^{m,c}(\nu)$ and $\kappa_{l,d}^{m,c}(\nu)$ are the core and foreground coefficients for line and dust, respectively. Additionally, the optical depth is given by $\tau^{m,c}(\nu) = \int \kappa^{m,c}(\nu)\,ds = \kappa^{m,c}(\nu)\,s$. This assumes that molecules and dust are well mixed, i.e. it would not be correct if the molecule exists only in part of the cloud, but the dust everywhere. In older versions, the background temperature could only be defined as the measured continuum offset, which corresponds to the beam-averaged continuum brightness temperature. At the same time, the dust, as agent of line attenuation, was described by column density and opacity. This is practical, because the observable T_{bg} is used, but does not constitute a self-consistent and fully physical description. Therefore, we now use optionally either a physical ($\gamma \equiv 1$, defined by the input parameter setting `t_back_flag = F`) or phenomenological ($\gamma \equiv 0$, defined by the input parameter setting `t_back_flag = T`) description of the background[8] indicated by the Kronecker delta $\delta_{\gamma,0}$, i.e. $S^{m,c}(\nu) \equiv J(T_{\text{ex}}^{m,c}, \nu)$ for $\gamma \equiv 0$. Note, if $\gamma \equiv 0$, the definition of the dust temperature $T_d^{m,c}(\nu)$, Eq. (16), is superfluous.

---

[8]Here, the phrase "background" means the "layer" with intensity $I_{\text{bg}}^{\text{core}}(\nu)$ which is located behind the core components, i.e. the background of the core layers, see Fig. 4.

The total optical depth $\tau_{\text{total}}^{m,c}(\nu)$ of each molecule m and component c is defined as the sum of the optical depths $\tau_l^{m,c}(\nu)$ of all lines of each molecule m and component c plus the dust optical depth $\tau_d^{m,c}(\nu)$, i.e.

$$\tau_{\text{total}}^{m,c}(\nu) = \tau_l^{m,c}(\nu) + \tau_d^{m,c}(\nu), \tag{9}$$

where the dust optical depth $\tau_d^{m,c}(\nu)$ takes the dust attenuation into account and is given by

$$\tau_d^{m,c}(\nu) = \tau_{d,\text{ref}}^{m,c} \cdot \left(\frac{\nu}{\nu_{\text{ref}}}\right)^{\beta^{m,c}}$$
$$= \left(N_H^{m,c} \cdot \kappa_{\nu_{\text{ref}}}^{m,c} \cdot m_{H_2} \cdot \frac{1}{\zeta_{\text{gas}-\text{dust}}}\right) \cdot \left(\frac{\nu}{\nu_{\text{ref}}}\right)^{\beta^{m,c}}. \tag{10}$$

Here, $N_H^{m,c}$ describes the hydrogen column density, $\kappa_{\nu_{\text{ref}}}^{m,c}$ the dust mass opacity for a certain type of dust {Ossenkopf & Henning 1994} at the reference frequency ν_{ref}, and $\beta^{m,c}$ the spectral index of $\kappa_{\nu_{\text{ref}}}^{m,c}$. These parameters are defined by the user, see Sect. 9.2. In addition, $\nu_{\text{ref}} =$ 230 GHz indicates the reference frequency of the reference dust opacity $\tau_{d,\text{ref}}^{m,c}$, m_{H_2} describes the mass of a hydrogen molecule, and $1/\zeta_{\text{gas}-\text{dust}}$ describes the dust to gas ratio and is set here to $(1/100)$ {Hillebrand 1983}. The equation is valid for dust and gas well mixed.

The optical depth $\tau_l^{m,c}(\nu)$ of all lines for each molecule m and component c is described as[9]

$$\tau_l^{m,c}(\nu) = \sum_t \left[\frac{c_{\text{light}}^2}{8\pi\nu^2} A_{ul}^t N_{\text{tot}}^{m,c} \frac{g_u^t \, e^{-E_l^t/k_B \, T_{\text{ex}}^{m,c}}}{Q\left(m, T_{\text{ex}}^{m,c}\right)} \left(1 - e^{-h\,\nu^t/k_B \, T_{\text{ex}}^{m,c}}\right) \cdot \phi^{m,c,t}(\nu)\right], \tag{11}$$

where the sum with index t runs over all spectral line transitions of molecule m within the given frequency range. The Einstein A_{ul} coefficient[10], the energy of the lower state E_l, the upper state degeneracy g_u, and the partition function[11] $Q\left(m, T_{\text{ex}}^{m,c}\right)$ of molecule m are taken from the embedded SQLite3 database. In addition, the values of the excitation temperatures $T_{\text{ex}}^{m,c}$ and the column densities $N_{\text{tot}}^{m,c}$ for the different components and molecules are taken from the user defined molfit file.

In order to take broadening caused by the thermal motion of the gas particles and micro-turbulence into account we assume a normalized Gaussian line profile, i.e. $\int_0^\infty \phi(\nu)\,d\nu = 1$, for a spectral line t:

$$\phi^{m,c,t}(\nu) = \frac{1}{\sqrt{2\pi}\,\sigma^{m,c,t}} \cdot e^{-\frac{\left(\nu - \left(\nu^t + \delta\nu_{\text{LSR}}^{m,c,t}\right)\right)^2}{2(\sigma^{m,c,t})^2}}. \tag{12}$$

The source frequency $\delta\nu_{\text{LSR}}^{m,c,t}$ for each component c of a molecule m is related to the user defined velocity offset (relative to v_{LSR}) $\left(\delta\text{v}_{\text{offset}}^{m,c}\right)$ taken from the aforementioned molfit file, by the following expression

$$\delta\nu_{\text{LSR}}^{m,c,t} = -\frac{\left(\delta\text{v}_{\text{offset}}^{m,c} + \text{v}_{\text{LSR}}\right)}{c_{\text{light}}} \cdot \nu^t, \tag{13}$$

where ν^t indicates the frequency of transition t taken from the SQLite3 database mentioned above. Additionally, the standard deviation $\sigma^{m,c}$ of the profile is defined by the velocity width

---

[9]A derivation of the expression is given in the appendix A.3.

[10]The indices u and l represent upper and lower state of transition t, respectively.

[11]Because the database usually does not describe the partition functions at the given excitation temperature $T_{\text{ex}}^{m,c}$, the value of $Q\left(m, T_{\text{ex}}^{m,c}\right)$ is computed from a linear interpolation. (With the new catalog, extrapolation should not be necessary for most conditions encountered in molecular cores.)

$(\Delta v_{width}^{m,c})$ described in the molfit file for each component c of a molecule m:

$$\sigma^{m,c,t} = \frac{\frac{\Delta v_{width}^{m,c}}{c_{light}} \cdot \left(\nu^t + \delta\nu_{LSR}^{m,c,t}\right)}{2\sqrt{2\ln 2}}. \tag{14}$$

The beam-averaged continuum background temperature $I_{bg}^{core}(\nu)$ is parametrized as

$$I_{bg}^{core}(\nu) = T_{bg} \cdot \left(\frac{\nu}{\nu_{min}}\right)^{T_{slope}} + J_{CMB} \tag{15}$$

to allow the user to define the continuum contribution for each frequency range, individually. Here, ν_{min} indicates the lowest frequency of a given frequency range. T_{bg} and T_{slope}, defined by the user, describe the background continuum temperature and the temperature slope, respectively. Here, the treatment of the dust is not entirely self-consistent. To amend that, we would need to define a more precise source model, which we consider to be outside the scope of this effort.

In Eq. (3), the continuum contribution is described through the source function $S^{m,c}(\nu)$, Eq. (8), by an effective dust temperature $T_d^{m,c}$ (through $J(T_d^{m,c},\nu)$) for each component which is given by

$$\begin{aligned} T_d^{m,c}(\nu) &= T_{ex}^{m,c}(\nu) + \Delta T_d^{m,c}(\nu) \\ &= T_{ex}^{m,c}(\nu) + T_{d,off}^{m,c} \cdot \left(\frac{\nu}{\nu_{min}}\right)^{T_{d,slope}^{m,c}}, \end{aligned} \tag{16}$$

where $T_{d,off}^{m,c}$ and $T_{d,slope}^{m,c}$ can be defined by the user for each component in the molfit. If $T_{d,off}^{m,c}$ and $T_{d,slope}^{m,c}$ are not defined for a certain component, we assume $T_d^{m,c}(\nu) \equiv T_{ex}^{m,c}(\nu)$ for all components. For a physical ($\gamma \equiv 1$) description of the background intensity, see Eq. (8), the user can define the dust opacity, Eq. (10), and dust temperature, Eq. (16), for each component.

Finally, the last term J_{CMB} in Eq. (3) describes the OFF position for single dish observations (defined by Inter_Flag = F) where we have an intensity caused by the cosmic background J_{CMB}. For interferometric observations, the contribution of the cosmic background is filtered out and has to be subtracted as well.

In contrast to core layers, foreground components may interact with each other, as shown in Fig. 5, where absorption takes places only, if the excitation temperature for the absorbing layer is lower than the temperature of the background.

Hence, the solution of the radiative transfer equation for foreground layers can not be given in a form similar to Eq. (3). Foreground components have to be considered in an iterative manner. The solution of the radiative transfer equation for foreground layers can be expressed as

$$T_{mb}^{fore}(\nu)_{m,c=1} = \eta\left(\theta^{m,c=1}\right)\left(S^{m,c=1}(\nu) - T_{mb}^{core}(\nu)\right)\left(1 - e^{-\tau_{total}^{m,c=1}(\nu)}\right) + T_{mb}^{core}(\nu)$$

$$T_{mb}^{fore}(\nu)_{m,c=i} = \eta\left(\theta^{m,c=i}\right)\left(S^{m,c=i}(\nu) - T_{mb}^{fore}(\nu)_{m,c=(i-1)}\right)\left(1 - e^{-\tau_{total}^{m,c=i}(\nu)}\right) + T_{mb}^{fore}(\nu)_{m,c=(i-1)}, \tag{17}$$

where m indicates the index of the current molecule and i represents an index running over all foreground components c of all molecules. Additionally, we assume that each foreground
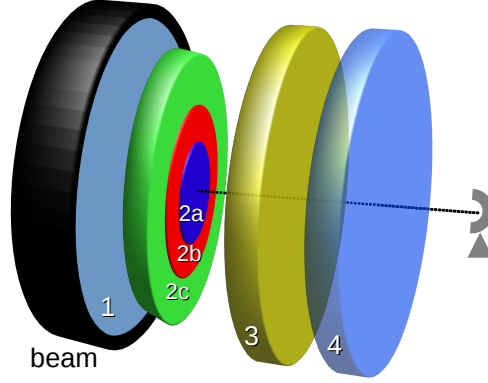
Figure 5: Sketch of a distribution of core and foreground layers within the Gaussian shaped beam of the telescope (black ring). Here, we assume three different core components **2a**, **2b**, and **2c** located in a plane perpendicular to the line of sight which lies in front of the background layer **1** with intensity $I_{\mathrm{bg}}^{\mathrm{core}}(\nu)$, see Eq. (15). The foreground layers **3** and **4** are located between the core layers and the telescope along the line of sight (black dashed line). Here, each component is described by different excitation temperatures, velocity offsets etc. indicated by different colors. The thickness of each layer is described indirectly by the total column density $N_{\mathrm{tot}}^{m,c}$, see appendix A.3. For each foreground layer we assume a beam filling factor of 1, i.e. each foreground layer covers the whole beam.

component covers the whole beam, i.e. $\eta\left(\theta^{m,c=1}\right) \equiv 1$ for all foreground layer. Thus, Eq. (17) simplifies to

$$T_{\mathrm{mb}}^{\mathrm{fore}}(\nu)_{m,c=1} = \left[S^{m,c=1}(\nu)\left(1 - e^{-\tau_{\mathrm{total}}^{m,c=1}(\nu)}\right) + T_{\mathrm{mb}}^{\mathrm{core}}(\nu)e^{-\tau_{\mathrm{total}}^{m,c=1}(\nu)}\right]$$

$$T_{\mathrm{mb}}^{\mathrm{fore}}(\nu)_{m,c=i} = \left[S^{m,c=i}(\nu)\left(1 - e^{-\tau_{\mathrm{total}}^{m,c=i}(\nu)}\right) + T_{\mathrm{mb}}^{\mathrm{fore}}(\nu)_{m,c=(i-1)}e^{-\tau_{\mathrm{total}}^{m,c=i}(\nu)}\right], \qquad (18)$$

where $T_{\mathrm{mb}}^{\mathrm{core}}(\nu)$ describes the core spectrum, see Eq. (3), including the beam-averaged continuum background temperature $I_{\mathrm{bg}}^{\mathrm{core}}(\nu)$. For foreground lines the contribution by other components is considered by first calculating the contribution of core objects and then use this as new continuum for foreground lines reflecting the fact that cold foreground layers are often found in front of hotter emission sources. The `myXCLASS` function assumes, that the cosmic background describes together with the core components one end of a stack of layers. Additionally, the foreground components are located between this plane and the telescope, see Fig. 5. The total column density $N_{\mathrm{tot}}^{m,c}$ depends on the abundance of a certain molecule and on the thickness of a layer containing the molecule. The order of components along the line of sight is defined by the occurrence of a certain foreground component in the molfit file.

By fitting all species and their components at once, line blending and optical depth effects are taken into account. The modeling can be done simultaneously with isotopologues (and higher vibrational states) of a molecule assuming an isotopic ratio stored in the so-called iso ratio file, see Sect. 9.3. Here, all parameters are expected to be the same except the column density which is scaled by one over the isotopic ratio for each isotopologue. Additionally, it is assumed that radiation emitted by all isotopologues of a molecule in a component interact with all other isotopologues, but the radiation emitted in one component does not interact with other molecules or with the same molecule in different components, i.e. their intensities are added.

In order to correctly take instrumental resolution effects into account in comparing the modeled spectrum with observations myXCLASS integrates the calculated spectrum over each

channel. Thereby, myXCLASS assumes that the given frequencies ν describe the center of each channel, respectively. The resulting value is than given as

$$T_{\mathrm{mb}}(\nu) = \frac{1}{\Delta_c} \int_{\nu - \frac{\Delta_c}{2}}^{\nu + \frac{\Delta_c}{2}} T_{\mathrm{mb}}(\tilde{\nu}) d\tilde{\nu}, \tag{19}$$

where Δ_c represents the width of a channel. Due to the complexity of Eqn. (3), (18) the integration in Eq. (19) can not be done analytically. Therefore, myXCLASS performs a piecewise integration of each component and channel using the trapezoidal rule and summaries the resulting values to get the final value used in Eq. (19). In order to reduce the computation effort myXCLASS determines the minimal variance σ_{min}, see Eq. (14), for each component. If the lines in the corresponding component are broad compared to the channel width, i.e. $\sigma_{\mathrm{min}} > 2 \cdot \Delta_c$, myXCLASS determines the intensities at $\nu \pm \frac{\Delta_c}{2}$ and evaluates Eq. (19) by applying the trapezoidal rule. But, if $\sigma_{\mathrm{min}} \leq 2 \cdot \Delta_c$, myXCLASS re-samples the corresponding channel and calculates all Doppler-shifted transition frequencies within each channel and calculates $T_{\mathrm{mb}}(\nu)$ at these frequencies if these frequencies are separated by at least $10^{-3} \cdot \Delta_c$. In order to achieve a good description of the modeled spectrum between these frequencies and the channel edge frequencies $\nu \pm \frac{\Delta_c}{2}$ as well, myXCLASS determines the intensities at further frequencies whereby the number of inserted frequencies depends on σ_{min}, i.e. if the distance d between two Doppler-shifted transition frequencies or between a Doppler-shifted transition frequency and one of the channel edge frequencies $\nu \pm \frac{\Delta_c}{2}$ is lower than $\sigma_{\mathrm{min}}/2$, myXCLASS determines the intensity at 20 equidistant frequencies in between. If d is lower than $\sigma_{\mathrm{min}}/4$ ($\sigma_{\mathrm{min}}/6$), myXCLASS inserts 40 (60) frequencies. Finally, myXCLASS performs a piecewise integration using the trapezoidal rule. Additionally, myXCLASS re-samples neighboring channel, which contain no Doppler-shifted transition frequency, as well to describe tails of lines located close to the channel edges properly, i.e. myXCLASS determines the intensities in a neighboring channel without transition frequencies at 20 equidistant frequencies and evaluates Eq. (19) using the trapezoidal rule, see Fig. 6.

9.2 The molfit file

Within the *molfit file* the user defines both which molecules are taken into account and the number of components for each molecule. Additionally, the user has to define for each component the source size $\theta^{m,c}$ in arcsec (`size`), the excitation temperature T_{ex} in K (`T_ex`), the column density N_{tot} in cm^{-2} (`N_tot`), the velocity width (FWHM) $\Delta\nu$ in km s^{-1} (`V_width`), the velocity offset (relative to v$_{\mathrm{LSR}}$) in km s^{-1} (`V_off`), and the flag (`CFFlag`) indicating if a component is considered for core `c` or foreground `f`. The definition of the source size $\theta^{m,c}$ for an absorbing component will be ignored, because we assume that all foreground layers cover the whole beam. So, the definition of this parameter is not necessary, but can be given.

Example of a molfit file:

```
% Number of molecules = 2
% size:    T_ex:    N_tot:   V_width:   V_off:   CFFlag:
CS;v=0;    3
 48.470  300.00  3.91E+17      2.86   -20.564         c
 21.804  320.00  6.96E+17      8.07    30.687         c
 81.700  208.00  1.46E+17      5.16   -10.124         c
HCS+;v=0;   2
% size:    T_ex:    N_tot:   V_width:   V_off:   CFFlag:
        150.00  1.10E+18      5.00    -0.154         f
        200.00  2.20E+17      3.10    -2.154         f
```
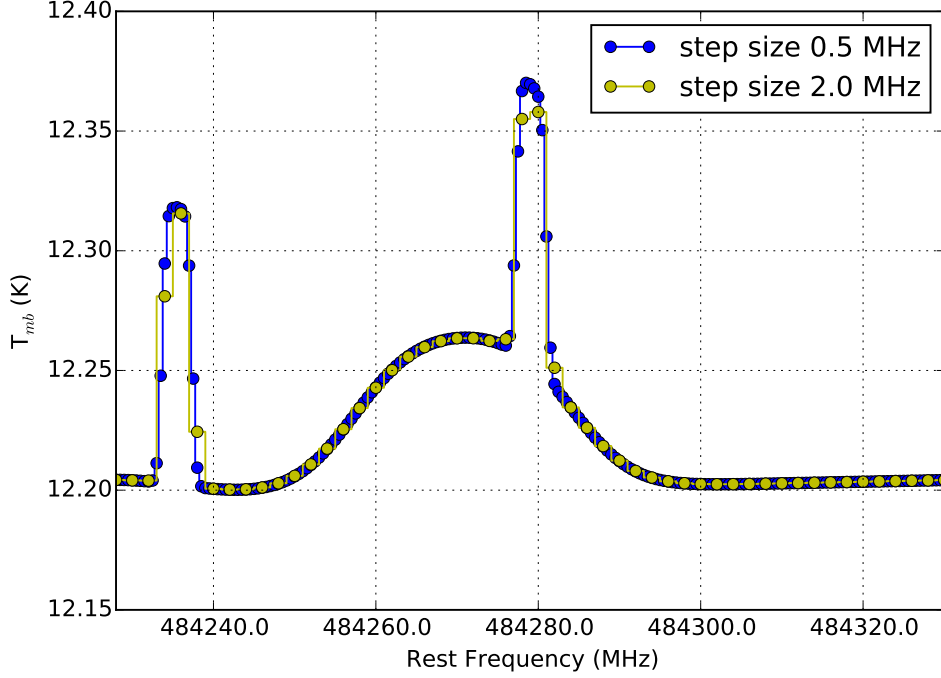
Figure 6: Here, the `myXCLASS` function was used to calculate a spectrum for different step sizes.

The definition of parameters for a molecule starts with a line describing the name of the molecule, which must be identical to the name of the molecule included in the database, followed by the number of components N for this molecule. The following N lines describe the parameters for each components, separately. Generally, all parameters have to be separated by blanks, comments are marked with the % character.

In order to define a dust temperature for each component which is not identical to the excitation temperature T_{ex} of the corresponding component, the molfit file has to contain two additional columns between columns `V_off` and `CFFlag`, describing the parameters $T_{\text{d,off}}^{m,c}$ (`T_doff`) and $T_{\text{d,slope}}^{m,c}$ (`T_dslope`), Eq. (16), respectively.

Example of an extended molfit file defining a dust temperature for each component:

```
% Number of molecules =      2
CS;v=0;     3
% s_size:  T_rot:    N_tot:   V_width:    V_off:   T_doff:   T_dslope:   CFFlag:
  48.47039  50.000  3.9E+16   2.862117   -2.5643      3.0        0.0          c
  40.10603  56.531  2.3E+18   4.210278   -7.3161      3.0        0.0          c
  29.09758  68.441  2.0E+17   4.024519    0.2130      2.5        1.0          c
            6.0857  2.2E+17   1.000000   -1.9000      3.0        0.0          f
HCS+;v=0;   1
% s_size:  T_rot:    N_tot:   V_width:    V_off:   T_doff:   T_dslope:   CFFlag:
  32.43532  43.234  2.1E+16   1.354217   -9.5211      4.0        0.1          c
```

Additionally, the `myXCLASS` program allows to define a hydrogen column density N_H (in cm^{-2}), dust mass opacity $\kappa_{\nu_{\text{ref}}}$ (in cm^2 g^{-1}), and the spectral index β for each component (`nH_flag = F`) or globally (`nH_flag = T`), i.e. $N_H^{m,c} \rightarrow N_H$, $\kappa_{\nu_{\text{ref}}}^{m,c} \rightarrow \kappa_{\nu_{\text{ref}}}$, and $\beta^{m,c} \rightarrow \beta$. In order to define these parameters individually for each component, the molfit file has to contain three additional columns on the left side of column `CFFlag`.

For globally defined dust parameters, the `myXCLASS` program assumes that all core components do not contain dust except the core component with the largest beam filling factor, see Eq. (5). This avoids an overestimation of the dust contribution caused by the overlap of the core components.

Example of an extended molfit file, if `nH_flag` is set to `F` (false):

```
% Number of molecules =      2
CS;v=0;     3
% s_size:  T_rot:    N_tot:   V_width:    V_off:   n_H:   kappa:   beta:   CFFlag:
  48.47039  50.000  3.9E+16  2.862117  -2.5643  3e24    0.42     2.0   c
            56.531  2.3E+18  4.210278  -7.3161  2e24    0.42     2.1   f
            68.441  2.0E+17  4.024519   0.2130  3e21    0.42     1.9   f
             6.857  2.2E+17  1.000000  -1.9000  3e24    0.42     1.8   f
HCS+;v=0;   1
% s_size:  T_rot:    N_tot:   V_width:    V_off:   n_H:   kappa:   beta:   CFFlag:
            43.234  2.1E+16  1.354217  -9.5211  1e24    0.42     2.0   f
```

In order to define the dust parameters $T_{d,\mathrm{off}}^{m,c}$ (in K), $T_{d,\mathrm{slope}}^{m,c}$, hydrogen column density N_H (in cm^{-2}), dust mass opacity $\kappa_{\nu_{\mathrm{ref}}}$ (in cm^2 g^{-1}), and the spectral index β for each component (`nH_flag = F`) the columns defining the dust temperature $T_{d,\mathrm{off}}^{m,c}$ (`T_doff`), $T_{d,\mathrm{slope}}^{m,c}$ (`T_dslope`) have to be given before the hydrogen column density, kappa and beta, i.e. in the order of `s_size`, `T_rot`, `N_tot`, `V_width`, `V_off`, `T_doff`, `T_dslope`, `n_H`, `kappa`, `beta`, and `CFFlag`.

9.3   The iso ratio file

The so-called `iso ratio file` defines the iso ratios between molecules/isotopologues. The ASCII file consists of three columns separated by tabs or at least two blank characters, where the first two columns indicates the molecules, respectively. The third column defines the ratio for both molecules. Comments are marked with a "%" character, i.e. all characters on the right side of a "%" are ignored.

Example for an iso ratio file:

```
% isotopologue:        molecule:        ratio:
CS;v=4;                CS;v=0;             2.3
CS;v=3;                CS;v=0;             2.1
CS;v=2;                CS;v=0;             2.1
CS;v=1;                CS;v=0;             2.0
CS-34;v=0;             CS;v=0;            22.5
CS-34;v=1;             CS;v=0;            22.5
CS-33;v=1;             CS;v=0;            75.0
CS-33;v=0;             CS;v=0;            75.0
HCS+;v=2;              HCS+;v=0;           1.0
HCS+;v=1;              HCS+;v=0;           1.0
HC-33-S+;v=0;          HCS+;v=0;          75.0
HC-34-S+;v=0;          HCS+;v=0;          22.5
```

Here, the first line sets the iso ratio between the `CS`, `v=4` and the `CS`, `v=0` molecule to `2.3`.

9.3.1   The iso ratio file for the myXCLASSFit function

The `myXCLASSFit` function offers the possibility to optimize the ratios between isotopologues as well. For that purpose, the user has to add two additional columns on the right indicating the lower and the upper limit of a certain ratio.

Example for an iso ratio file where the ratios are optimized by the `myXCLASSFit` function:

% isotopologue:	molecule:	ratio:	lower:	upper:
CS;v=4;	CS;v=0;	2.3	0.1	500.0
CS;v=3;	CS;v=0;	2.1	0.1	500.0
CS;v=2;	CS;v=0;	2.1	0.1	500.0
CS;v=1;	CS;v=0;	2.0	0.1	500.0
CS-34;v=0;	CS;v=0;	22.5	0.1	500.0
CS-34;v=1;	CS;v=0;	22.5	0.1	500.0
CS-33;v=1;	CS;v=0;	75.0	0.1	500.0
CS-33;v=0;	CS;v=0;	75.0	0.1	500.0
HCS+;v=2;	HCS+;v=0;	1.0	0.1	500.0
HCS+;v=1;	HCS+;v=0;	1.0	0.1	500.0
HC-33-S+;v=0;	HCS+;v=0;	75.0	0.1	500.0
HC-34-S+;v=0;	HCS+;v=0;	22.5	0.1	500.0

If the lower and upper limit are equal or if the lower limit is higher than the upper limit, the ratio is kept constant and is not optimized by the `myXCLASSFit` function. Note, if either the iso master molecule or a corresponding isotopologue has no transition within at least one given frequency range, the `myXCLASSFit` function does not optimize the corresponding iso-ratio. For example, if the isotopologue `HNC-13;v=0;` (used in the example described above) has no transition in at least one given frequency range, the given iso-ratio (here 60) is kept constant. Additionally, if a iso master molecule has no transition within at least one given frequency range, the iso-ratios to all of its isotopologues are kept constant.

10 MAGIX

The function starts the MAGIX program.

Input parameters:

- ▶ `MAGIXExpXML`: path and name of the experimental xml-file.

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `MAGIXInstanceXML`: path and name of the instance xml-file

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `MAGIXFitXML`: path and name of the xml file controlling the fitting process

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `MAGIXRegXML`: path and name of the so-called registration xml file containing the description of the input and output files of the external model program

  Note, if the parameter defines a relative path, this path has to be defined relative to the current working directory!

- ▶ `MAGIXOption`: option for the MAGIX run (default is ""):

  - `""` (default): all informations are printed out to the screen
  - `quiet`: no informations are printed to the screen except warning and error messages
  - `plotsaveonly`: MAGIX disables the interactive GUI of matplotlib but creates all plots and saves them into files
  - `debug`: Stop MAGIX after the first function call. This flag can be very helpful to analyze problems occurring with the call of the external model program.

Output parameters:

- ▶ `JobDir`: absolute path of the job directory created for the current run.

Example:
```
MAGIXExpXML = "demo/MAGIX/TwoOscillators_RefFit_R.xml"
MAGIXInstanceXML = "demo/MAGIX/parameters.xml"
MAGIXFitXML = "demo/MAGIX/Levenberg-Marquardt_Parameters.xml"
MAGIXRegXML = "Fit-Functions/Drude-Lorentz_conv/xml/"
MAGIXRegXML += "Conventional_Drude-Lorentz.xml"
MAGIXOption = ""
JobDir = MAGIX(MAGIXExpXML, MAGIXInstanceXML, MAGIXFitXML, \
                MAGIXRegXML, MAGIXOption)
```

Usage without CASA:
```
# extend sys.path variable
...


# import task_MAGIX package
```

```
    import task_MAGIX


    # call MAGIX function
    MAGIXExpXML = "demo/MAGIX/TwoOscillators_RefFit_R.xml"
    MAGIXInstanceXML = "demo/MAGIX/parameters.xml"
    MAGIXFitXML = "demo/MAGIX/Levenberg-Marquardt_Parameters.xml"
    MAGIXRegXML = "Fit-Functions/Drude-Lorentz_conv/xml/"
    MAGIXRegXML += Conventional_Drude-Lorentz.xml"
    MAGIXOption = ""
    JobDir = task_MAGIX.MAGIX(MAGIXExpXML, MAGIXInstanceXML, MAGIXFitXML, \
                             MAGIXRegXML, MAGIXOption)
```

The MAGIX function copies the different xml-files (except the registration xml-file) to a so-called "job-directory" located in the MAGIX working directory `path-of-XCLASS/run/MAGIX/`! The name of a job directory for a MAGIX run is made up of four components: The first part of the name consists of the phrase "`job_`" whereas the second part describes the date (day, month, year), the third part the time stamp (hours, minutes, seconds) of the function execution. The last part describes a so-called job ID which is composed of the so-called PID followed by a four digit random integer number to create a really unambiguous job number, e.g. `path-of-XCLASS-Interface/run/MAGIX/job__25-07-2013__12-02-03__189644932/`

All file(s), which are created by the current MAGIX run, are stored in such a job directory!

In addition to the xml files, the function copies the experimental data file(s), i.e., the files which contains the experimental/observational data, to the current job directory as well. The path(s) of the experimental data file(s) defined in the experimental xml-file are adjusted, so that these path(s) become absolute and point to the current job directory. Please note, that all modifications are applied to the copies of the xml-files. The original xml-files are not modified.

As mentioned above, the registration xml-file is neither copied to the job directory nor modified!

The io-control file (which is required for each MAGIX run) is created automatically!

10.1   What is MAGIX

Most physical or chemical models use a set of parameters. Finding the best description of observational/experimental data using a certain model implies determining the parameter set that most closely reproduces the data by some criteria, typically the minimum of a merit function. Often the χ^2 distribution[12] is used, and we will use this term throughout, although it should be understood that it could be replaced by other appropriate merit functions. Other important results are the goodness of fit, in absolute terms, and confidence levels for determined parameters. This is a generic problem independent of the actual model, and instead of implementing an optimizer in each and every program, parameter optimization can be separated. Therefore, a software package is needed that finds the best parameter set in an iterative procedure for arbitrary models by comparing the results of the physical model for a given parameter set with the experimental data set and modifying the parameter set to improve the quality of the description, i.e., by reducing the value of χ^2. In general, the physical and chemical models are multidimensional non-linear functions of the input parameters. Thus, finding the best description for a given experimental data set means finding the global minimum of the χ^2 function, which is a function of the model input parameters.

---

[12]The χ^2 distribution is a function of relative quadratic differences between experimental and model values.

Many optimization functions will find minima, but they could be local minima, which do not describe the results adequately, and can lead to misleading interpretations. Therefore, one has to find the global minimum of the χ^2 function to obtain a good description of the experimental data. However, finding the global minimum of an arbitrary function is challenging and has been practically impossible for many problems so far. To circumvent this, we need algorithms that allow us to explore the landscape of the χ^2 function and calculate probabilities for the occurrence of minima. Combining certain algorithms and making use of the different advantages of the applied algorithms allows a reliable but not absolutely unique interpretation of the experimental data. Most of the algorithms are very computationally expensive, and the computational effort tends to scale with the degree of reliability.

These requirements are very general. Hence it makes sense to generate a package that is able to read in experimental data, communicate with any registered external model program[13] and compare automatically the result of the physical model with the given data through the figure of merit. It should improve the quality of the fit within an iterative procedure by adjusting the input parameters using several algorithms that fulfill the wide range of requirements mentioned above.

To make MAGIX as flexible as possible, we developed it as a stand-alone program instead of a library for a certain programming language. (Please note, the MAGIX function in CASA provides an interface to the MAGIX program which is also included in the XCLASS interface.) A library is always coupled to a certain language and requires knowledge of their usage. A user without sufficient experience would not benefit from MAGIX while a stand-alone program requires only the knowledge of how to start the model program. No further experience in software programming is necessary. Additionally, many model programs such as Radmc-3 or Lime are controlled by input files or by partial modification of their source code. Therefore, writing one or more files on the disk, starting the model program, and finally reading in the result is inevitable. In addition, MAGIX offers the possibility to use a so-called RAM disk (or RAM drive), which is orders of magnitude faster than a normal hard disk. By using an RAM disk, the function evaluation / model computation becomes the dominant part in the whole process for nearly all external model programs. Therefore our approach will likely be more beneficial for the majority of users.

In the following we describe the structure and functionality of the MAGIX package that implements this system. We start with an overview of the different parts of MAGIX, followed by a detailed description of the so-called registration process that couples the extended model to the fitter code. In (§ 10.5) we explain the different algorithms included in the MAGIX package in more detail and end with an astrophysical application of MAGIX [14].

10.2 Environment variables

Before you use MAGIX, you might need to set some general environment variables (§ 10.2.1). Those environment variables have to be set by typing the corresponding commands at the command line (or adding the corresponding line in your `~/.bashrc` file, if you want a permanent setting for the corresponding environment variable).

Note, the following environment variables must not be changed during a fit process.

---

[13]Here, the phrase "external model program" means the external program that calculates the model function depending on several input parameters.

[14]An earlier version of this code was written by {Boone *et al.* 2006}.

10.2.1 General environment variables

► During the fit process, MAGIX creates several subdirectories located in a temporary directory (`temp`) which is by default located in the root directory of MAGIX `path-of-XCLASS-interface/programs/MAGIX/` (it is created if it doesn't exist). By setting the environment variable `MAGIXTempDirectory`

```
export MAGIXTempDirectory="temp_somewhere_else"
```

the user can define another location of this temporary directory. It is strongly recommended that the user should use a RAM drive, i.e. set the environment variable to

```
export MAGIXTempDirectory="/dev/shm/MAGIX/"
```

whenever possible. (The RAM drive is a common name for a temporary file storage facility on many Unix-like operating systems. The usage of a RAM drive improve the performance of MAGIX because the input and output file(s) of the external model programs are not written to the hard drive but to the RAM which is orders of magnitude faster.)

A detailed description of the structure of the MAGIX temp directory can be found in the MAGIX manual.

► In case of programs that deal with large data arrays, the error message `segmentation error` (or `Speicherzugriffsfehler`, in German) may occur. In that case, the user has to set the soft limit on the maximum amount of memory which is available, using the following command:

```
ulimit -s unlimited
```

Additionally, the user has to increase the size of the OMP stack writing the following command:

```
export OMP_STACKSIZE='999M'
```

Here the size of the stack depends on the memory that is available. If `OMP_STACKSIZE` is not set by the user, the default is `999M`.

► For the plot for each iteration daemon, the user can define a time interval by using the following command:

```
export MAGIXTimePlotIter="5000"
```

For further informations see (§ 10.6.3).

10.3 Registration

In contrast to other astrophysical optimization packages MAGIX does not include any intrinsic model program that calculates the model function depending on a given parameter set. To communicate with the external model program, MAGIX has to create the required input file(s) for every call of the external model program, including the modified parameter values and a directive how to read in the result of the model program. During every optimization step the values of the parameters to be optimized will have been modified. Consequently, MAGIX has to produce the actual input files that contain the new parameter values for the model to run at each subsequent function call. Therefore MAGIX has to be given directives how to create/write the input files that will be used in the function call. Hence the user has to define the structure of the input file(s), including loops with the information which parameter has to be written to

which location. After each optimization step, MAGIX compares the experimental data with the values of the model function calculated by the external program with the latest values of the parameters. The difference between data and model is quantified by the value of χ^2, i.e., low values of χ^2 correspond to small differences. Therefore, the path, the name(s), and the format of the output file(s) of the model program have to be defined as well. The descriptions of both the input and the output files of the model are given in the xml-file that we call registration file.

A detailed description of the registration xml file and the whole registration process can be found in the MAGIX manual, located in the subdirectory:
`path-of-XCLASS-Interface/programs/MAGIX/Documentation/`.

Example of a registration xml-file (defined in parameter `MAGIXRegXML`):

```
<!-- define commands to start script -->
<ModelProgramCall>
    <PathStartScript>Fit-Functions/ckRtm/RtmGreybody.py</PathStartScript>
    <ExeCommandStartScript>python RtmGreybody.py</ExeCommandStartScript>
    <ParallelizationPossible>Yes</ParallelizationPossible>
    <InputDataPath>data.dat</InputDataPath>
</ModelProgramCall>


<!-- define number of input files -->
<NumberInputFiles>1</NumberInputFiles>


<!-- describe first input file -->
<InputFile>


    <!-- describe first input file -->
    <InputFileName>in.txt</InputFileName>


    <!-- define number of lines in first input file name -->
    <NumberLines>5</NumberLines>


    <!-- describe first line of first input file -->
    <line group="false">
        <NumberParameterLine>1</NumberParameterLine>
        <Parameter group="false">
            <NumberReplicationParameter> </NumberReplicationParameter>
            <Name>sourceSize</Name>
            <Format>ES30.15</Format>
            <LeadingString></LeadingString>
            <TrailingString></TrailingString>
        </Parameter>
    </line>
    . . .
```

and the corresponding input file `in.txt`:

```
4.000000000000000E+01
8.700000000000000E+02
3.235219643279549E+01
1.714226271972070E+00
1.239041499402653E+04
```

Additionally, the registration file indicates whether the external model program can be used in a parallelized MAGIX run or not, i.e., if it is possible to execute two or more instances of the same external model program on the same machine at the same time. This depends on how and where output or intermediate files of the model are written, since different instances of models running in parallel must not overwrite each other's files. This is mainly a bookkeeping problem.

Ideally, a model has to be registered only once, i.e., it is not necessary to register an already registered model again as long as the structure of the input and output file(s) is unchanged. Whenever one wants to optimize some parameter(s) of the model with MAGIX, it should be sufficient to edit the instance xml-file.

Example of an instance xml-file (defined in parameter `MAGIXInstanceXML`):

```
<!-- define total number of parameter in the first input file -->
<NumberParameters>5</NumberParameters>


<!-- describe first parameter -->
<Parameter fit="true">
    <name>sourceSize</name>
    <value>40.0</value>
    <error> </error>
    <lowlimit>0</lowlimit>
    <uplimit>80</uplimit>
</Parameter>


<!-- describe second parameter -->
<Parameter fit="false">
    <name>WaveRef</name>
    <value>870.0</value>
    <error> </error>
    <lowlimit>0</lowlimit>
    <uplimit>100</uplimit>
</Parameter>


<!-- describe third parameter -->
<Parameter fit="false">
    <name>Temperature</name>
    <value>250.0</value>
    <error> </error>
    <lowlimit>0</lowlimit>
    <uplimit>100</uplimit>
</Parameter>
. . .
```

MAGIX comes with a suite of pre-registered models[15], and the authors are willing to assist with the registration of new models.

10.4   Model instance

The model instance is where the values of all parameters are set. This file also specifies whether each parameter is one to be optimized. If yes, then the starting values, as well as the lower and upper limits are also provided. If the parameter is not one to be optimized, then the lower and upper limits are ignored.

---

[15]At the moment the following software packages are registered: SimLine, Lime, Radmc-3D, myXCLASS, RADEX

10.4.1 Necessary tags in the instance

Listing 1: Example of a parameter xml-file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ModelParameters>


    <!-- define total number of parameters -->
    <NumberParameters>8</NumberParameters>


    <!-- define parameter "EpsilonInfinity" -->
    <Parameter fit="false">
        <name>EpsilonInfinity</name>
        <value>2.5</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>10</uplimit>
    </Parameter>


    <!-- define parameter "NumberOscillators" -->
    <Parameter fit="false">
        <name>NumberOscillators</name>
        <value>2</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>100</uplimit>
    </Parameter>


    <!-- define 1st parameter "EigenFrequency" -->
    <Parameter fit="false">
        <name>EigenFrequency</name>
        <value>150.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>


    <!-- define 1st parameter "PlasmaFrequency" -->
    <Parameter fit="true">
        <name>PlasmaFrequency</name>
        <value>200.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>


    <!-- define 1st parameter "Damping" -->
    <Parameter fit="true">
        <name>Damping</name>
        <value>10.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>
```

```xml
    <!-- define 2nd parameter "EigenFrequency" -->
    <Parameter fit="false">
        <name>EigenFrequency</name>
        <value>600.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>


    <!-- define 2nd parameter "PlasmaFrequency" -->
    <Parameter fit="true">
        <name>PlasmaFrequency</name>
        <value>400.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>


    <!-- define 2nd parameter "Damping" -->
    <Parameter fit="true">
        <name>Damping</name>
        <value>10.0</value>
        <error> </error>
        <lowlimit>0</lowlimit>
        <uplimit>1000</uplimit>
    </Parameter>
</ModelParameters>
```

▶ A parameter name should appear within the instance as many times as it appears in the registration file, i.e. preferably once. Exceptions for this are the following cases:

   – If a parameter belongs to a group, then its name should appear in the instance so many times as the number of replications in this group.

   – If a parameter belongs to a group, its name can also be given to another parameter that does not belong to this group.

   – If a parameter is declared more than once in the same input file (with double square brackets appended in their name) or in different input files (with no double square brackets appended), then it has to be declared only once in the instance, and that is the first time it appears in the registration file.

   – If a parameter is declared more than once in the same input file (with no double square brackets appended), then all occurrences of the name are considered to belong to different parameters and have to exist also in the instance.

▶ The number of the model parameters defined in this file (`<NumberParameters>`) must be equal to the number of all the parameters defined in the registration file (for all files and all of their lines – no exact tag exists for the total number in the registration file, but it can be derived summing up the contents of the `NumberParameterLine` of all lines and all files), *taking into account all existing replication of lines and parameters.* (The total number of parameters is altered by the `group` attribute and line replications).

▶ Each parameter is described inside the `<Parameter>` tag. There have to occur as many `<Parameter>` tags as defined in the tag `<NumberParameters>` in the same parameter xml-file.

44

▶ The names of the model parameters defined within the `<name>` tags must be identical with the corresponding names as defined in the registration file of the given model (§ 10.3). Otherwise the program stops. The only parameter names that may appear more than once in an instance are those who belong to a group, if the replication number of that group is > 1.

▶ In order to include a parameter in the fitting process, set the `fit` attribute of the `<Parameter>` tag to `true`.
For example: In case you want to optimize the value of the parameter named as `EpsilonInfinity` during the fit process:

```
<Parameter fit="true">
        <name>EpsilonInfinity</name>
        <value>3.0</value>
        <error></error>
        <lowlimit>0</lowlimit>
        <uplimit>9999</uplimit>
</Parameter>
```

If the value of this parameter should not be optimized, then you have to set the `fit` attribute to `false`:

```
<Parameter fit="false">
...
```

▶ The tags `<uplimit>` and `<lowlimit>` indicate the upper and the lower limits of the model parameters, respectively. If the value of the model parameter runs out of this defined range during the fit process, MAGIX will print out a warning message on the screen and corrects the value of the parameter to the closest value within the range.

Note, the value of the tag `<uplimit>` has to be greater than the value of the tag `<lowlimit>`. Please avoid using non-number specifications like "±inf", because the range definitions are essential for the swarm algorithms like Bees or PSO. Setting a parameter limit to "±inf" leads to an dramatic enhancement of the computational effort.

▶ The `<error>` tag must occur for every parameter, even if empty. The content of this tag is replaced by the error of the optimized parameter in the end.

10.4.2 Instance for myXCLASS

In contrast to users of other external model programs, a user of myXCLASS can use a so-called molfit file to define the start values as well as the upper and lower limits of each parameter. In addition to the format of the molfit file described in (§ 9) the extended molfit file required one (three) additional column(s) for each parameter of each component.

Listing 2: Example of an input file (`old` format) for myXCLASS

```
% Number of molecules =     2
%limit:  size:   limit:  T_rot:   limit:     N_tot:   limit: V_width:   limit:    V_off:  CFFlag:
CS;v=0;    3
0.00     48.470   50.00   3.000    0.00   1.39E+17    0.00    2.861     0.00  -20.564   c
         500.106    0.00   2.730   10.00   1.60E+16    2.00    2.276     0.00    6.547   f
0.00     40.106    0.00  56.531    0.00   1.23E+19    0.00    4.288     0.00   -7.316   c
HCS+;v=0;   1
         500.000    0.00   6.085    0.00   1.22E+18    0.00    1.000     0.00   -1.900   f
```

In the so-called `old` format (see File sample 2), the limits of each parameter are given by a column on the left side of each parameter. The limits of the parameters source size, column

45

density, and hydrogen column density are determined by

```
lower limit = abs(parameter value) / limit
upper limit = abs(parameter value) * limit
```

The limits of the other parameters are determined by simply adding/subtracting the limit to/from the value itself. All lower limits which are < 0 are set to 0 (except for velocity offset). Please note, that the flag indicating core or foreground does not require an additional column.

Listing 3: Example of an input file for myXCLASS

```
% Number of molecules =    2
CS;v=0;    3
n 0.0 0.0   48.40   n 0.0 0.0   3.00   n 0.0 0.0  0.39E+17   n 0.0 0.0  2.81   n 0.0 0.0 -20.54
c
n 0.0 0.0  500.16   n 0.0 0.0   2.70   n 0.0 0.0  0.60E+16   y 0.0 2.0  2.26   n 0.0 0.0   6.57
f
n 0.0 0.0   40.16   n 0.0 0.0  56.51   n 0.0 0.0  0.23E+19   n 0.0 0.0  4.28   n 0.0 0.0  -7.36
c
HCS+;v=0;    1
n 0.0 0.0    3.48   y 1.0 8.0   6.80   n 0.0 0.0  0.27E+17   n 0.0  0.0 5.00   n 0.0 0.0  -7.99
c
```

In the `new` format (see File sample 3), the limits of each parameter are defined by three additional columns on the left side of each parameter. The first column indicates, if the current parameter should be optimized `y` or not `n`. The second column defines the lower, the third column the upper limit of the current parameter. The lower limit must not be bigger than the upper limit! Please note, that the flag indicating core or foreground does not require additional columns as well.

A user of myXCLASS can fit the ratio(s) of isotopologues as well. For that purpose the iso file (§ 9.3) has to include two additional columns as described in section (§ 9.3.1).

10.5   Optimization algorithms

MAGIX provides optimization through one of the following algorithms or via a combination of several of them (algorithm chain, see § 10.5.12): the Levenberg-Marquardt (conjugate gradient) method (§ 10.5.1), which is fast, but can get stuck in local minima, simulated annealing (§ 10.5.2) and particle swarm optimization methods (§ 10.5.3), which are slower, but more robust against local minima. Other, more modern methods, such as bees (§ 10.5.4), genetic (§ 10.5.5), Markov chain Monte Carlo (MCMC) (§ 10.5.6), nested sampling (§ 10.5.7), or interval nested sampling algorithms (§ 10.5.8) are included as well for exploring the solution landscape, checking for the existence of multiple solutions, and giving confidence ranges. Additionally, MAGIX provides an interface to make several algorithms included in the `scipy`[16] package available.

In the following, we give a short description of the optimization algorithms that are implemented in MAGIX using the analytic Himmelblau-, Rosenbrock-, and Rastrigin functions (Fig. 7) as test functions for demonstration, with multiple minima (Himmelblau, Rastrigin) and a very shallow minimum (Rosenbrock). The optimization problem is solved directly through these algorithms via stochastic searching without derivatives or gradient information (except for the Levenberg-Marquardt algorithm)[17].

---

[16]http://www.scipy.org/

[17]While we often refer to *Numerical Recipes* (NR) to explain the algorithms, no actual NR algorithms are included in the package.
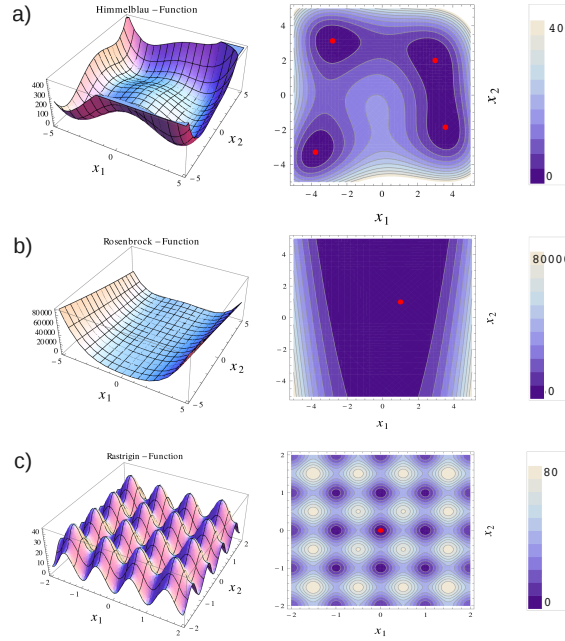
Figure 7: Analytical test functions: a) The Himmelblau function $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$ has four identical minima at $f(3.0, 2.0) = f(-2.805118, 3.131312) = f(-3.779310, -3.283186) = f(3.584428, -1.848126) = 0$. b) The Rosenbrock function $f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ has one global minimum at $f(1, 1) = 0$. c) The Rastrigin function $f(x_1, x_2) = 10 + (x_1^2 - 10\cos(2\pi \cdot x_1)) + 10 + (x_2^2 - 10\cos(2\pi \cdot x_2))$ has one global minimum at $f(0, 0) = 0$. The contour plots of the different test functions are plotted in the second column. The positions of the global minima for each test function are indicated by red dots.

Figure 8: Results for the Rosenbrock function using the LM algorithm with start values $x_1 = 4.5$, $x_2 = 1.0$: The distribution of the parameter values after five function calls ($\chi^2 = 1.38 \cdot 10^{-9}$) is indicated by green-yellow points. The sequence of iterations is color-coded (color bar on the lower right). Early points are denoted in dark green, points toward the end of the iteration are light yellow. The red dot denotes the global minimum of the Rosenbrock function.
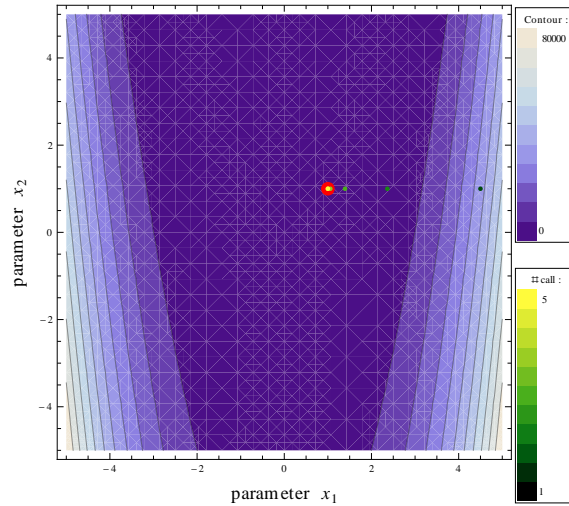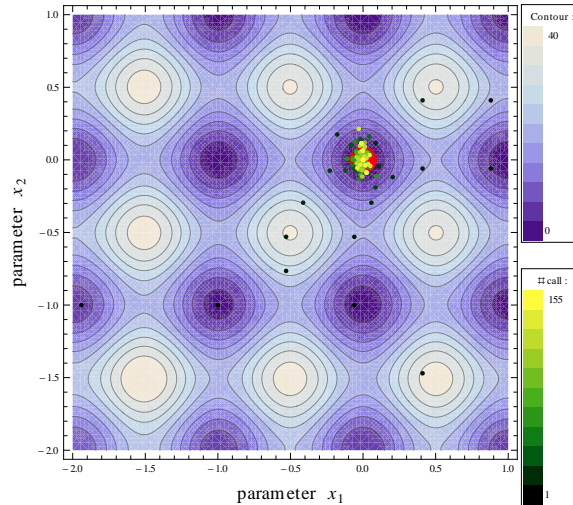
Figure 9: Results for the Rastrigin function using the SA algorithm with start values $x_1 = -1.0$, $x_2 = -1.0$: The distribution of the parameter values after 155 function calls ($\chi^2 = 4.37 \cdot 10^{-7}$) is indicated by green-yellow points. The sequence of iterations is color-coded (color bar on the lower right). Early points are denoted in dark green, points toward the end of the iteration are light yellow. The red dot denotes the global minimum of the Rastrigin function.

10.5.1 Levenberg–Marquardt algorithm

The Levenberg–Marquardt algorithm (LM), {Marquardt 1963}, {Nocedal & Wright 2006}, {Press *et al.* 2007} is a hybrid between the Gauss–Newton algorithm and the method of gradient descent. The Gauss–Newton algorithm is a method for solving non-linear least-squares problems. It is a modification of Newton's method to find the minimum of a function, but is constrained so that it can only minimize a sum of square function values. It requires knowledge of the gradients in χ^2 space, which can be obtained from differential steps for sufficiently smooth functions. The LM can find a minimum (possibly local) even if it starts very far from it, but the efficiency depends on the landscape of the parameters. On the other hand, for functions and starting values of parameters that are very close to the final minimum, the LM tends to be slower than Gauss–Newton. The LM is an algorithm that strongly depends on the starting values of the parameters that are to be optimized, and the user should choose the starting values very carefully. Otherwise the algorithm can easily become stuck in a side minimum of the global solution. MAGIX contains a modified version of the MINPACK package implementation {Garbow *et al.* 1980}. The gradient of the χ^2 function is calculated in a parallel environment using `OpenMP` and `OpenMPI`. Furthermore, the user can define the variation *var* used for the gradient determination. Because MAGIX cannot determine the gradient analytically, MAGIX has to use a numerical approximation: $(\partial/\partial x_i) f(\vec{x}) = (f(x_i + h) - f(x_i))/h$, where the variation h is defined by $h = x_i \cdot var$. Varying the size of the variation may be necessary if the χ^2 function is not a smooth function and the calculation of the gradient produces awkward results. As shown in Fig. 8, the fast convergence of the LM is obvious, where the minimum is found after five (!) iterations.

10.5.2 Simulated annealing

Simulated annealing (SA), {Press *et al.* 2007} is a generic probabilistic computational method that is used for the problem of global optimization, i.e., to find a good approximation to the global optimum of a given function in a large parameter space. For certain problems, SA is more effective than exhaustive enumeration – provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

In comparison to the LM, SA is more robust. Its result does not depend so much on the neighborhood of the starting point. The LM searches for the highest (negative) gradient and stops when it detects a local minimum. In contrast, SA can check if the gradient around a local minimum is flat (low perturbation), in which case it will continue to find a better minimum, i.e., a lower value than the one found before. In Fig. 9, the SA algorithm is able to find the global minimum of the Rastrigin function at $x_1 = 0$ and $x_2 = 0$ although it starts in a local minimum at $x_1 = -1$ and $x_2 = -1$.

The name and inspiration of this algorithm come from the process of annealing in metallurgy. This technique involves heating and controlled cooling of a material, with the aim to gradually increase the size of its crystals and reduce their defects. The heat provides energy and causes the atoms to move from their initial position (which was a local minimum of internal energy) and wander randomly through states of higher energy. Then, a slow cooling gives them more chances of finding configurations of lower internal energy than the initial one.

By analogy to the physical process, each step of SA replaces the current solution by a random nearby solution. This nearby solution is chosen with a probability that depends both on the difference between the corresponding function values and on a global temperature, T. The temperature T is gradually decreased (multiplied by the temperature reduction coefficient, $k < 1$) during the process.

The dependency of the temperature difference between two subsequent steps is such that the solution changes almost randomly when T is high, but it is modified increasingly toward lower values as T becomes zero. Allowing for increasing values prevents the method from becoming stuck at local minima – which can happen with gradient methods such as the LM. Subsequent points of the SA algorithm follow a perpendicular direction.

MAGIX uses a partially parallelized implementation of the `scipy` algorithm using `OpenMP` and `OpenMPI`.

10.5.3 Particle swarm optimization

The particle swarm optimization (PSO) algorithm implemented in MAGIX is a hybrid {Fan & Zahara 2007} between a particle swarm optimization algorithm {Kennedy & Eberhart 1995} and a Nelder–Mead simplex search method {Nelder & Mead 1965}. The PSO optimizes a problem by iteratively trying to improve a candidate solution according to some measure of quality; the particles are sent toward a better solution flying so much faster according to the technique's performance in previous step. The Nelder–Mead technique or downhill simplex search method is a traditional technique for the direct search of function minima; it is easy to use, does not need calculation of derivatives and therefore can converge even to non-stationary solutions.

Both techniques have been modified by Fan and Zahara {Fan & Zahara 2007} and their combination is the hybrid algorithm that is available in MAGIX. The PSO algorithm performs a kind of heuristics: It starts from an initial random population of particles and searches in the neighborhood for a global optimum. The particles with the best-fit function values are updated
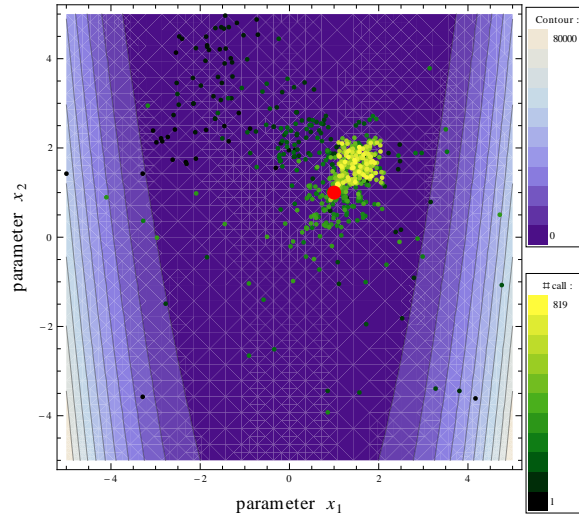
Figure 10: Results for the Rosenbrock function using the PSO algorithm: The distribution of the parameter values after 819 function calls ($\chi^2 = 2.31 \cdot 10^{-5}$) is indicated by green-yellow points. The sequence of iterations is color-coded (color bar on the lower right). Early points are denoted in dark green, points toward the end of the iteration are light yellow. The red dot denotes the global minimum of the Rosenbrock function. (The clustering of the function calls right above the global minimum is caused by the very flat gradient of the Rosenbrock function in this area.)

with the simplex method, while the particles with the poorest function values are updated with the PSO.

The whole procedure prevents the algorithm from being trapped locally and at the same time allows it to find the global minimum. It repeats itself until a termination criterion or the maximum number of iterations is reached. The iteration shown in Fig. 10 stops after 819 function calls because the value of χ^2 dropped below the limit of $\chi^2 = 9 \cdot 10^{-5}$. The corresponding parameter values are $x_1 = 0.9954$, and $x_2 = 0.9907$ instead of $x_1 = 1$ and $x_2 = 1$. Reducing the limit of the χ^2 value would lead to a better description of the global minimum but it would require more function calls. The algorithm implemented in MAGIX is parallelized using `OpenMP` and `OpenMPI`.

10.5.4 Bees algorithm

The bees algorithm {Pham *et al.* 2005} is a swarm algorithm that performs a kind of neighborhood search combined with a random search (see, Fig. 11). This name was given to the algorithm because it tries to mimic the collection of nutriments by honey bees, in that there is always a part of the population that performs the role of scouts, traveling far away in random directions to detect new nutrition sources.

The algorithm starts with an initial set of parameter vectors (a collection of particles or scout bees, i.e., the hive of bees), randomly selected and such that it spreads throughout the entire parameter space. After the fitness of each bee is evaluated in terms of the quality ranking it just visited, the bees with the highest fitness visit the neighborhoods of the sites they currently are at. Of the remaining bees some are sent away to random sites and some are sent to search in the neighborhood of the very best sites as well (so that there are more bees searching for food in places where it is more probable to find nutrition sources). At the end of the step, the fitness of each visited site is evaluated, and the bees who just visited them move away or search
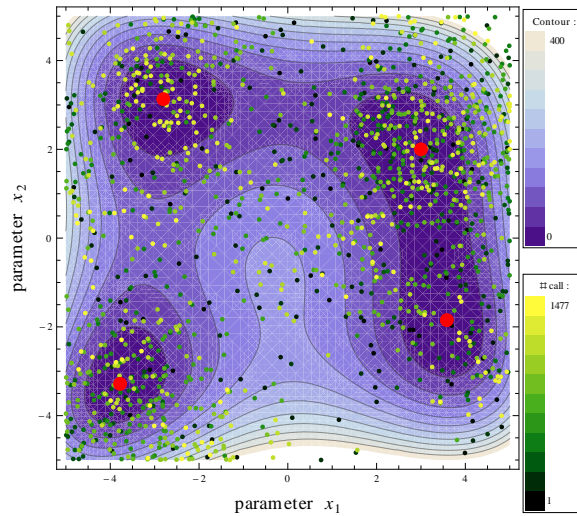
50

Figure 11: Results for the Himmelblau function using the bees algorithm: The distribution of the parameter values after 1477 function calls is indicated by green-yellow points. The sequence of iterations is color-coded (color bar on the lower right). Early points are denoted in dark green, points toward the end of the iteration are light yellow. The red dots denote the four global minima of the Himmelblau function.

in the closer vicinity of the best sites. The result is that the bees algorithm finds areas of local minima, see Fig. 11.

10.5.5 Genetic algorithm

The genetic algorithm (GA) is a probabilistic search algorithm that mimics the process of natural evolution. It iteratively transforms a set of parameter vectors (population), each with an associated fitness value, into a new population of objects.

The procedure takes place using the Darwinian principle of natural selection, with operations that are patterned after naturally occurring genetic operations such as recombination and mutation. Recombination is the joined process of reproduction and crossover, i.e., mixing the genetic matter (parameter values) of the parents (parameter vector) {Whitley 1994}. Mutation is the complete disappearance of specific genetic material and its transformation to a completely different material; this happens especially in combinations of genetic material (parameter sets) that does not fit.

The evolution starts from a randomly selected population (of parameter sets) and proceeds in evolutionary generations (stages/iteration steps). When a generation step is completed, the fitness of all members of the population is evaluated. Then a number of parameter sets is stochastically selected for modification; the fittest members are more likely to be kept unmodified. The rest of the population members are modified; the modifications they go through are more intense because they are fit. The modified and unmodified parameter sets make up the new population whose fitness will be evaluated at the end of the step {Herrera *et al.* 1998, Herrera *et al.* 2005} (see, Fig. 12).
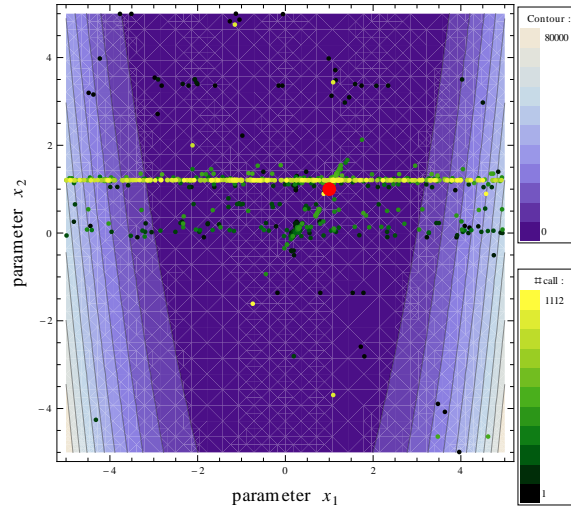
Figure 12: Results for the Rosenbrock function using the GA: The distribution of the parameter values after 1112 function calls ($\chi^2 = 9.33 \cdot 10^{-3}$) is indicated by green-yellow points, where the dark green points indicate function calls that are made at the beginning of the fit process and light yellow points represent function calls at the end of the fit process. The red dot denotes the global minimum of the Rosenbrock function.
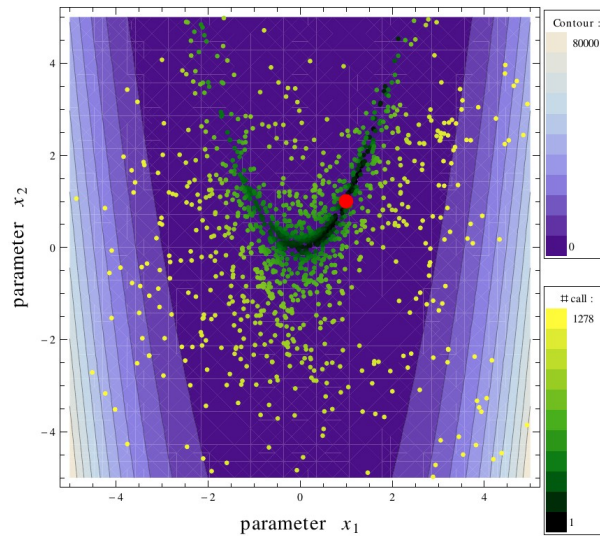
Figure 13: Results for the Rosenbrock function using MCMC: The distribution of the parameter values after 1278 function calls ($\chi^2 = 2.35 \cdot 10^{-2}$) is indicated by green-yellow points, where the dark green points indicate function calls that are made at the beginning of the fit process and light yellow points represent function calls at the end of the fit process. The red dot denotes the global minimum of the Rosenbrock function.

10.5.6   Markov chain Monte Carlo (MCMC)

We use the `emcee`[18] package {Foreman-Mackey *et al.* 2012}, which implements the affine-invariant ensemble sampler of {Goodman & Weare 2010}, to perform a full-parallelized MCMC algorithm. An additional installation of the `emcee` package is not necessary.

---

[18]http://dan.iel.fm/emcee

The following discussion is largely taken from {Foreman-Mackey *et al.* 2012}: The general goal of MCMC algorithms is to draw M samples $\{\Theta_i\}$ from the posterior probability density

$$p(\Theta, \alpha | D) = \frac{1}{Z} p(\Theta, \alpha) p(D | \Theta, \alpha), \tag{20}$$

where the prior distribution $p(\Theta, \alpha)$ and the likelihood function $p(D | \Theta, \alpha)$ can be relatively easily (but not necessarily quickly) computed for any particular value of (Θ_i, α_i). The normalization $Z = p(D)$ is independent of Θ and α once we have chosen the form of the generative model. This means that it is possible to sample from $p(\Theta, \alpha | D)$ without computing Z – unless one would like to compare the validity of two different generative models. This is important because Z is generally very expensive to compute.

Once the samples produced by MCMC are available, the marginalized constraints on Θ can be approximated by the histogram of the samples projected into the parameter subspace spanned by Θ. In particular, this implies that the expectation value of a function of the model parameters $f(\Theta)$ is

$$\langle f(\Theta) \rangle = \int p(D | \Theta) f(\Theta) \, d\Theta \approx \frac{1}{M} \sum_{i=1}^{M} f(\Theta_i). \tag{21}$$

Generating the samples Θ_i is a non-trivial process unless $p(\Theta, \alpha, D)$ is a very specific analytic distribution (for example, a Gaussian). MCMC is a procedure for generating a random walk in the parameter space that, over time, draws a representative set of samples from the distribution. Each point in a Markov chain $X(t_i) = [\Theta_i, \alpha_i]$ depends only on the position of the previous step $X(t_i - 1)$.

The simplest and most commonly used MCMC algorithm is the Metropolis-Hastings (M-H) method. The iterative procedure is as follows: (1) given a position $X(t)$ sample a proposal position Y from the transition distribution $Q(Y; X(t))$, (2) accept this proposal with probability

$$\min\left(1, \frac{p(Y|D)}{p(X(t)|D)} \frac{Q(X(t); Y)}{Q(Y; X(t))}\right). \tag{22}$$

The transition distribution $Q(Y; X(t))$ is an easy-to-sample probability distribution for the proposal Y given a position $X(t)$. A common parameterization of $Q(Y; X(t))$ is a multivariate Gaussian distribution centered on $X(t)$ with a general covariance tensor that has been tuned for performance. It is worth emphasizing that if this step is accepted $X(t + 1) = Y$; Otherwise, the new position is set to the previous one $X(t + 1) = X(t)$ (in other words, the position X(t) is repeated in the chain). The M-H algorithm converges (as $t \to \infty$) to a stationary set of samples from the distribution but there are many algorithms with faster convergence and varying levels of implementation difficulty. Faster convergence is preferred because of the reduction of computational cost due to the smaller number of likelihood computations necessary to obtain the equivalent level of accuracy. The inverse convergence rate can be measured by the autocorrelation function and more specifically, the integrated autocorrelation time. This quantity is an estimate of the number of steps needed in the chain in order to draw independent samples from the target density. A more efficient chain has a shorter autocorrelation time.

The stretch move {Goodman & Weare 2010} proposed an affine-invariant ensemble sampling algorithm informally called the "stretch move." This algorithm significantly outperforms standard M-H methods producing independent samples with a much shorter autocorrelation time. This method involves simultaneously evolving an ensemble of K walkers $S = \{X_k\}$ where the proposal distribution for one walker k is based on the current positions of the $K - 1$ walkers in the complementary ensemble $S_{[k]} = \{X_j, \forall_j \neq k\}$. Here, "position" refers to a vector in the

N-dimensional, real-valued parameter space. To update the position of a walker at position X_k, a walker X_j is drawn randomly from the remaining walkers $S_{[k]}$ and a new position is proposed:

$$X_k(t) \to Y = X_j + Z[X_k(t) - X_j] \tag{23}$$

where Z is a random variable drawn from a distribution $g(Z = z)$. It is clear that if g satisfies

$$g(z^{-1}) = zg(z), \tag{24}$$

the proposal of Eq. (23) is symmetric. In this case, the chain will satisfy detailed balance if the proposal is accepted with probability

$$q = \min\left(1, Z^{N-1}\frac{p(Y)}{p(X_k(t))}\right), \tag{25}$$

where N is the dimension of the parameter space. This procedure is then repeated for each walker in the ensemble in series. {Goodman & Weare 2010} advocate a particular form of $g(z)$, namely

$$g(z) \propto \begin{cases} \frac{1}{\sqrt{z}} & \text{if } z \in [\frac{1}{a}, a] \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

where a is an adjustable scale parameter that {Goodman & Weare 2010} set to 2. It is tempting to parallelize the stretch move algorithm by simultaneously advancing each walker based on the state of the ensemble instead of evolving the walkers in series, see Fig. 14. Unfortunately, this subtly violates detailed balance. Instead, we must split the full ensemble into two subsets ($S(0) = \{X_k, \forall_k = 1, \ldots, K/2\}$ and $S(1) = \{X_k, \forall_k = K/2 + 1, \ldots, K\}$) and simultaneously update all the walkers in $S(0)$ based only on the positions of the walkers in the other set ($S(1)$). Then, using the new positions $S(0)$, we can update $S(1)$. In this case, the outcome is a valid step for all of the walkers. The performance of this method – quantified by the autocorrelation time – is comparable to the serial stretch move algorithm but the fact that one can now take advantage of generic parallelization makes it extremely powerful.

The autocorrelation time is a direct measure of the number of evaluations of the posterior PDF required to produce independent samples of the target density. {Goodman & Weare 2010} show that the stretch-move algorithm has a significantly shorter autocorrelation time on several non-trivial densities. This means that fewer PDF computations are required – compared to a M-H sampler – to produce the same number of independent samples. The autocovariance function of a time series $X(t)$ is

$$C_f(T) = \lim_{t\to\infty} \text{cov}[f(X(t+T)), f(X(t))]. \tag{27}$$

This measures the covariances between samples at a time lag T. The value of T where $C_f(T) \to 0$ measures the number of samples that must be taken in order to ensure independence. In particular, the relevant measure of sampler efficiency is the integrated autocorrelation times

$$\tau_f = \sum_{T=-\infty}^{\infty} \frac{C_f(T)}{C_f(0)} = 1 + 2\sum_{T=1}^{\infty} \frac{C_f(T)}{C_f(0)}. \tag{28}$$

In practice, one can estimate $C_f(T)$ for a Markov chain of M samples as

$$C_f(T) \approx \frac{1}{M-T} \sum_{m=1}^{M-T} [f(X(T+m)) - \langle f\rangle] [f(X(m)) - \langle f\rangle]. \tag{29}$$
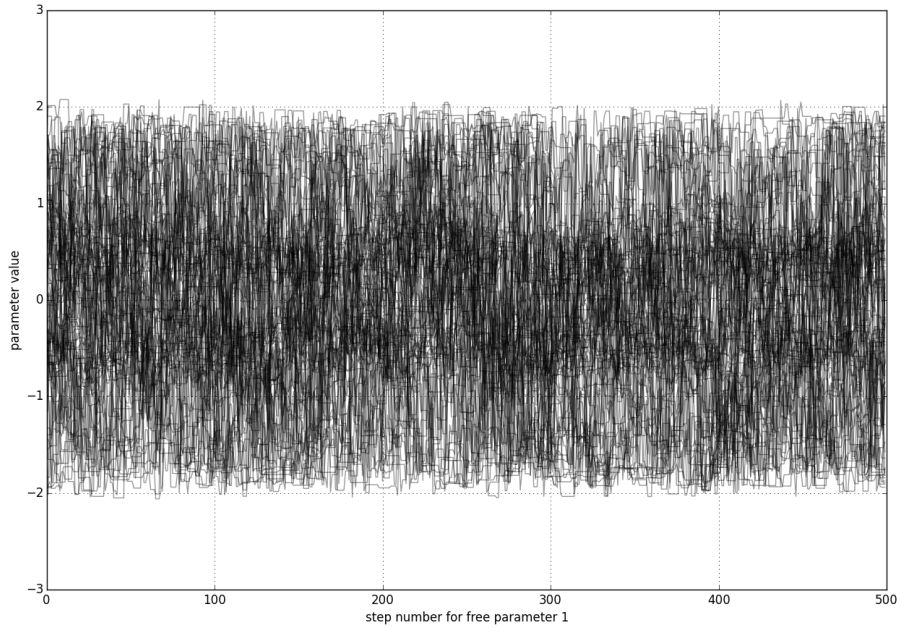
Figure 14: The figure shows the positions of each walker as a function of the number of steps in the chain.

We advocate for the autocorrelation time as a measure of sampler performance for two main reasons. First, it measures a quantity that we are actually interested in when sampling in practice. The longer the autocorrelation time, the more samples that we must generate to produce a representative sampling of the target density. Second, the autocorrelation time is affine invariant. Therefore, it is reasonable to measure the performance and diagnose the convergence of the sampler on densities with different levels of anisotropy.

10.5.7 Nested sampling

The nested sampling (NS) {Skilling 2006, Feroz & Hobson 2008} algorithm is a combination of a Monte Carlo method and Bayesian statistics {Sivia & Skilling 2006}. The Monte Carlo methods are a family of computational algorithms that perform repeated random sampling and compute the results for every sample. The Bayesian statistics is a statistical inference technique, i.e., it attempts to draw conclusions from data subject to random variation. In particular, the Bayesian inference calculates the probability that a hypothesis is true, i.e., how probable it is that the newly calculated value of a parameter is closer to the real one (the value that best fits experimental data); if it is more probable than the previously calculated, then the parameter value is updated.

The principle of the NS algorithm is illustrated in Fig. 15: The Bayesian approach is used to obtain a set of physical parameters $\Theta = (\theta_1; \theta_2, \ldots, \theta_n)$ that attempts to describe the experimental data. The Bayesian analysis is assumed to incorporate the prior knowledge with a given set of current observations to make statistical inferences. The prior information $\pi(\Theta)$ can come from observational data or from previous experiments. Bayes theorem states that the posterior probability distribution of the model parameters is given by
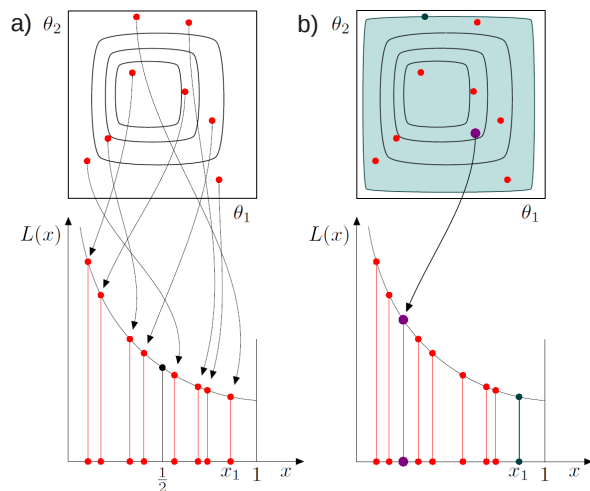
Figure 15: Principle of the NS algorithm (taken from {Skilling & MacKay 2012}): The upper part of both panels describes a contour plot of a likelihood function $\mathcal{L}(\theta)$. Left panel a): Each point θ within parameter space $\wp$ defined by the parameter ranges of θ_1 and θ_2 is associated with the volume that would be enclosed by the contour $L = \mathcal{L}(\theta)$. ($L(x)$ is the contour value such that the volume enclosed is x). If the points θ are uniformly distributed under the prior probability distribution (prior), all these volumes (x-values) are uniformly distributed between 0 and 1. Right panel b): Using a Markov chain method, the NS algorithm takes a point (purple dot) from $\wp$ satisfying $L \geq L(x_1)$. Inserting the new point into this distribution, we can find the highest x-value x_2 used for the next iteration.

$$\mathrm{Pr}(\Theta) = \frac{\mathcal{L}(\Theta)\,\pi(\Theta)}{Z}, \tag{30}$$

where $\mathrm{Pr}(\Theta)$ is the posterior probability distribution of the model parameters, $\mathcal{L}(\Theta)$ is the likelihood of the data for the given model and its parameters, $\pi(\Theta)$ is a prior information, and Z is Bayesian evidence. The Bayesian evidence is the average likelihood of the model in parameter space. It is given by the following integral over the n-dimensional space:

$$Z = \int \mathcal{L}(\Theta)\pi(\Theta)\,d\Theta. \tag{31}$$

The NS algorithm transforms the integral (31) to the single dimension by re-parametrization to a new linear variable[19] – a *prior volume X*. The volume of parameter space can be divided into elements $dX = \pi(\Theta)d\Theta$. The prior volume X can be accumulated from its elements dX in any order, so we construct it as a function of decreasing likelihood:

$$X(\lambda) = \int_{\mathcal{L}(\Theta)>\lambda} \pi(\Theta)\,d\Theta. \tag{32}$$

That means that the cumulative prior volume covers all likelihood values exceeding λ. As λ increases, the enclosed volume X decreases from $X(0) = 1$ to $X(1) = 0$. If the prior information $\pi(\Theta)$ is uniformly distributed in the parameter space, equation (31) for the evidence transforms into

$$Z = \int_0^1 \mathcal{L}(X)\,dX. \tag{33}$$

---

[19]Although we do not know the values of these volumes X, we know the order of them because $L(x_i) = \mathcal{L}(\theta)$
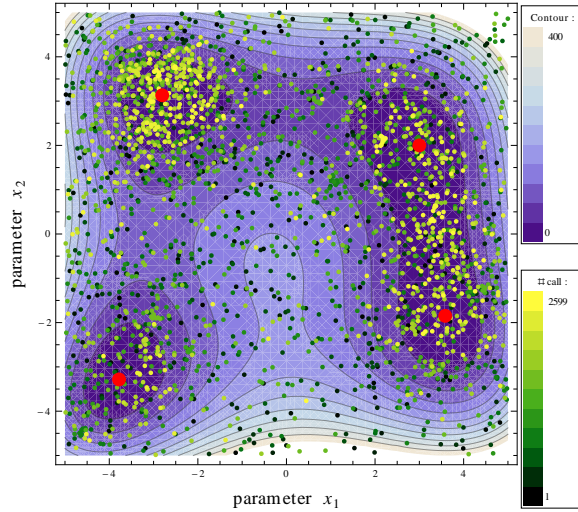
Figure 16: Results for the Himmelblau function using the NS algorithm: The distribution of the parameter values after 2599 function calls is indicated by green-yellow points, where the dark green points indicate function calls which are made at the beginning of the fit process and light yellow points represent function calls at the end of the fit process. The red dots denote the four minima of the Himmelblau function.

One can calculate the partial likelihood as $L_i = \mathcal{L}(X_i)$, where the X_i is a sequence of decreasing values, such that

$$0 < X_m < \ldots < X_2 < X_1 < 1. \tag{34}$$

MAGIX uses the trapezoid rule to approximate the evidence

$$Z = \sum_{i=1}^{m} Z_i, \quad \text{where} \quad Z_i = L_i \frac{X_{i-1} - X_{i+1}}{2}. \tag{35}$$

The NS algorithm is targeted at calculating Bayesian evidence, but it assists in obtaining a posterior sample of points from which one can estimate uncertainties of parameter values. The prior volume X_i, which corresponds to the likelihood contour L_i, is usually evaluated with a random number generator.

The actual algorithm is based on the Markov chain Monte Carlo (MCMC) methods {Press *et al.* 2007, Gilks *et al.* 1996, Diaconis 2009}. Those are a family of algorithms that sample from probability distributions, with the aim to construct a (Markov) chain with the desired distribution (a distribution with the desired properties) as the equilibrium distribution. The quality (how well the parameter sets fit the data) of the sample is a monotonically increasing function of the number of steps (see, Fig. 15).

An MCMC algorithm is used to reduce the dimensionality of the parameter space through integration. The Bayesian approach to this technique allows one not only to find multiple solutions, but also to define proportional weights of parameter values and to evaluate the Bayesian evidence[20]. The NS algorithm included in MAGIX requires fewer samples than standard MCMC

---

[20]Bayesian interpretation of probability: As the number of steps increases, we collect evidence with regard to the consistency or inconsistency of that evidence with a given hypothesis. More specifically, as the evidence accumulates, we tend to believe in the given hypothesis more or less, depending on the degree to which the increasing evidence agrees with that hypothesis.
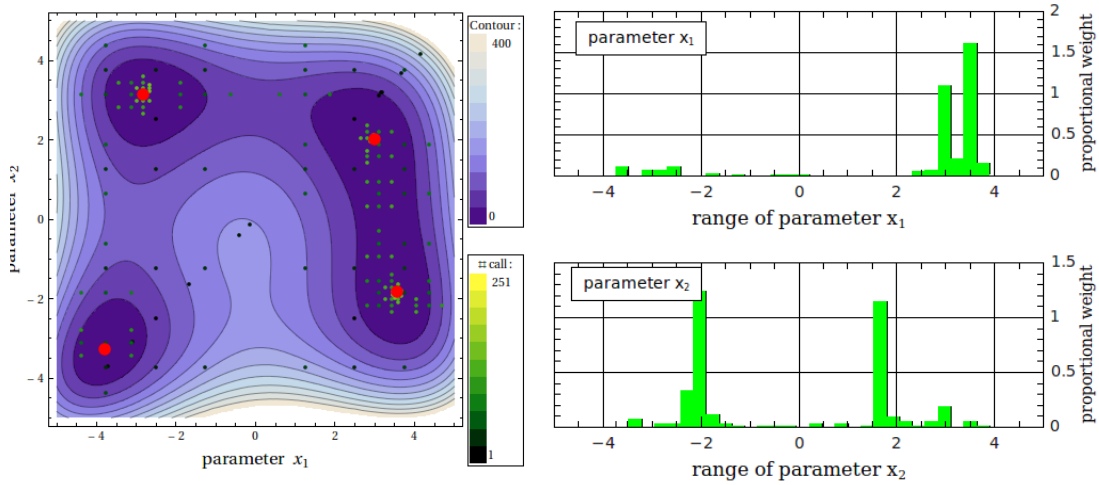
Figure 17: Results for the Himmelblau function using the INS algorithm: Left panel: The distribution of the parameter values after 251 function calls is indicated by green-yellow points, where the dark green points indicate function calls that are made at the beginning of the fit process and light yellow points represent function calls at the end of the fit process. The red dots denote the four minima of the Himmelblau function. Right panel: Posterior weights of points after the NS process.

methods {Feroz & Hobson 2008} while also providing posterior probabilities for each one of the best parameter vectors. The NS algorithm has the following advantages: It is a non-derivative method and investigates the landscape of optimization function. Additionally, it can find multiple minima. On the other hand, the algorithm does not converge to a global minimum and depends strongly on the random number generator. The results for the Himmelblau function, shown in Fig. 16, are similar to the results of the bees algorithm. Both algorithms can be used to explore the landscape of the χ^2 distribution and for finding areas of global minima. The implementation of the algorithm included in MAGIX is parallelized using `OpenMP` and `OpenMPI`.

10.5.8 Interval nested sampling algorithm

The Interval Nested Sampling (INS) algorithm (developed by I. Bernst[21]) included in MAGIX is an implementation of the branch-and-bound algorithm {Ichida & Fujii 1979} to find the next prior volume X_i. The algorithm is based on the NS algorithm (§ 10.5.7) and uses an interval method for the definition of a next part of the prior volume. The main principle of the interval method is a division of the parameter space into interval boxes and an estimation of the optimization function value over the boxes. The estimate is called inclusion function and can be calculated by various methods. The centered form of inclusion with slopes is used in the current version of the INS algorithm {Krawczyk 1985}. The interval method assists in finding the next prior volume X_i by determining the ratio of the volume of the working interval box Z_i to the whole volume Z of the parameter space. Posterior weights of points after the NS process are calculated using the Bayes theorem (see, Fig. 17):

$$w_i = \frac{Z_i}{Z}. \tag{36}$$

Using the sequence of posterior samples of parameter vectors we are able to determine the reliability of model parameters such as standard deviations or to construct posterior distribu-

---

[21]A paper describing the INS algorithm in detail is in preparation.

tions of parameter values. The mean value $\mu(\theta_j)$ of each model parameter θ_j, $j = (1, \ldots, n)$ and its standard deviation $\sigma(\theta_j)$ are given as

$$\mu(\theta_j) = \sum_{i=1}^{k} w_i \, \theta_j \tag{37}$$

and

$$\sigma(\theta_j) = \sqrt{\sum_{i=1}^{k} w_i \, (\theta_j - \mu(\theta_j))^2}, \tag{38}$$

where k indicates the number of points in the sample. The INS algorithm is capable of handling multi-modality of the optimization function, phase transitions, and strong correlations between model parameters. As shown in Fig. 17, the Interval Nested Sampling algorithm requires fewer function calls than other global optimizers. Additionally, it is used to determine the confidence intervals of parameters, which is described in the next section.

10.5.9 Additional Packages

This package makes the following algorithms included in the `scipy` package {scipy} available in MAGIX:

1. "fmin": Minimize a function using the downhill simplex algorithm. This algorithm only uses function values, not derivatives or second derivatives.

2. "fmin_powell": Minimize a function using modified PowellâĂŹs method. This method only uses function values, not derivatives.

3. "fmin_cg": Minimize a function using a nonlinear conjugate gradient algorithm.

4. "fmin_bfgs": Minimize a function using the BFGS algorithm.

5. "fmin_ncg": Unconstrained minimization of a function using the Newton-CG method.

6. "fmin_l_bfgs_b": Minimize a function func using the L-BFGS-B algorithm.

7. "fmin_tnc": Minimize a function with variables subject to bounds, using gradient information in a truncated Newton algorithm.

8. "anneal": Minimize a function using simulated annealing.

9. "brute": Minimize a function over a given range by brute force.

10.5.10 Error estimation

MAGIX provides an error estimation for each single parameter at the point of minimum using different methods:
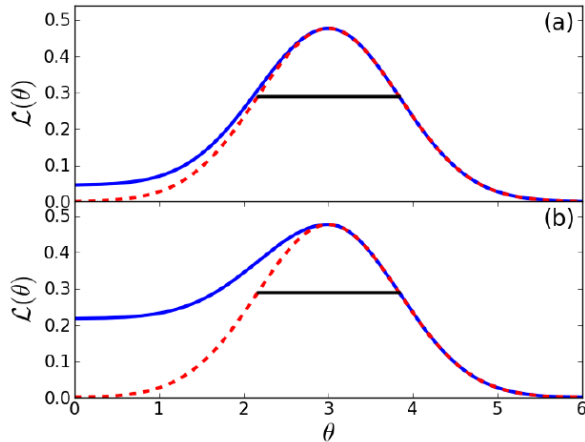
Figure 18: (Taken from {Andrae 2010}): Possible failures of the central-limit theorem. This figure shows an example likelihood function $\mathcal{L}(\theta)$ (solid blue curves), its Gaussian approximation at the maximum (dashed red curves), and the widths of these Gaussians (horizontal solid black lines). In panel (a) the Gaussian approximation and the corresponding error estimate is useful, whereas in panel (b) the error estimate is substantially underestimated.

10.5.10.1   Error estimation using Fisher matrix

This error estimation technique is based on the central-limit theorem which assumes that any well-behaved likelihood function is asymptotically Gaussian near its minimum. Assuming that $\log \mathcal{L} = \log \mathcal{L}(\vec{\theta})$ is a function of P parameters $\vec{\theta} = \{\theta_1, \ldots, \theta_P\}$ we expand this function in a Taylor series around the maximum $\vec{\theta}_{\max}$ to second order and get

$$\log \mathcal{L}(\vec{\theta}) \approx \log \mathcal{L}(\vec{\theta}_{\max}) + \frac{1}{2} \left.\frac{\partial^2 \log \mathcal{L}}{\partial \theta_i \partial \theta_j}\right|_{\vec{\theta}_{\max}} (\theta - \theta_{\max})_i \, (\theta - \theta_{\max})_j. \tag{39}$$

Close to the maximum $\vec{\theta}_{\max}$, the linear term vanishes, so that $\log \mathcal{L}(\vec{\theta})$ is approximately quadratic in $\vec{\theta}$, i.e. $\mathcal{L}(\vec{\theta}) = e^{\log \mathcal{L}(\vec{\theta})}$ is a Gaussian. The goodness of this approximation scales with the distance to the maximum. But for error estimation we cannot go arbitrarily close to the maximum. Fig. 18 describes two cases: In panel (a) the central-limit theorem gives a reasonable approximation of the likelihood and the error estimation is useful. In panel (b) the Gaussian is not a good approximation and the corresponding error estimates are useless.

In the following we will assume, that we can apply the central-limit theorem and express the likelihood $\mathcal{L}(\vec{\theta})$ as described by {Heavens 2009} as

$$\mathcal{L}(\vec{\theta}) = \frac{1}{(2\pi)^{P/2} \sqrt{\det \hat{\Sigma}^{-1}}} \exp\left[-\frac{1}{2} \left(\vec{\theta} - \vec{\theta}_0\right)^T \cdot \hat{\Sigma}^{-1} \cdot \left(\vec{\theta} - \vec{\theta}_0\right)\right], \tag{40}$$

where $\vec{\theta}$ describes a parameter vector for a given model function and $\vec{\theta}_0$ the vector of the (local) minimum of the χ^2 function. Eq. (40) describes a *P*-dimensional (*P-variate*) Gaussian with mean $\vec{\theta}_0$ and covariance matrix $\hat{\Sigma}$. This matrix describes the desired error estimates of $\vec{\theta}_0$. The variance estimates of each parameter θ_0^p is described by the diagonal entries of $\hat{\Sigma}$, whereas the off-diagonals are the estimates of the covariances. Comparing Eqs. (39) and (40) we see

$$\hat{\Sigma} = \left(-\frac{\partial^2 \log \mathcal{L}}{\partial \theta_i \partial \theta_j}\right)^{-1}. \tag{41}$$

The matrix of second derivatives of $\log\mathcal{L}$ is called *Fisher-matrix* or *Fisher information matrix*. If the second derivatives of $\log\mathcal{L}$ can be calculated, this method is order of magnitudes faster than all the other methods, described below, and for some high-dimensional problems the only applicable error estimation method. But this method can only describe elliptical error contours, i.e. it is not possible to obtain banana-shaped error contours. Additionally, this method assumes that the second-order Taylor expansion of Eq. (39) is a good approximation, see Fig. 18. A valid covariance-matrix $\hat{\Sigma}$ has to be positive definite, i.e. $\vec{x}^T \cdot \hat{\Sigma} \cdot \vec{x} > 0$, for any nonzero vector $\vec{x}$. In order to check this, one can use the following tests:

1. Compute the determinant $\det\hat{\Sigma}$. If $\det\hat{\Sigma} \le 0$, $\hat{\Sigma}$ is not valid.

2. Compute the eigenvalues of the matrix $\hat{\Sigma}$. If any eigenvalue is negative or zero, $\hat{\Sigma}$ is not valid.

Unfortunately, these tests are only rule-out criteria. If $\hat{\Sigma}$ fails any of these two tests, it is clearly ruled out, i.e. the central limit theorem can not be applied. But even if $\hat{\Sigma}$ passes both tests, the Gaussian might not be a good description of the likelihood around the maximum.

Finally, the error (or *marginal error*) $\Delta\theta_i$ of a parameter θ_i is than given as

$$\Delta\theta_i = \sqrt{\left(\hat{\Sigma}\right)^{-1}_{i,i}}, \tag{42}$$

where $\left(\hat{\Sigma}\right)^{-1}_{i,i}$ describes the ith diagonal element of the inverse of the covariance matrix $\hat{\Sigma}$.

10.5.10.2 Error estimation using Markov chain Monte Carlo (MCMC)

By choosing the MCMC method, the error estimation algorithm starts an MCMC algorithm at the estimated maximum $\vec{\theta}_0$ of the likelihood function and draws M samples of model parameters $\left\{\vec{\theta}_1, \ldots, \vec{\theta}_M\right\}$ from the likelihood function in a small ball around the a priori preferred position. After finishing the algorithm the probability distribution and the corresponding highest posterior density (HPD) interval of each free parameter is calculated. Additionally, the *corner* package[22] is used to plot each one- and two-dimensional projection of the sample to reveal covariances, see Fig. 19.

Before we describe the calculation of the HPD intervals in detail, we briefly summarize some important statistical expressions.

We start with the cumulative distribution function (CDF) of the standard normal distribution which is given as

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \, dt. \tag{43}$$

In statistics one often uses the related error function, or $\mathrm{erf}(x)$, defined as the probability of a random variable with normal distribution of mean 0 and variance $1/2$ falling in the range $[-x, x]$ that is

$$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} \, dt. \tag{44}$$

The two functions are closely related, namely

$$\Phi(x) = \frac{1}{2}\left[1 + \mathrm{erf}(\frac{x}{\sqrt{2}})\right]. \tag{45}$$

---

[22]https://pypi.python.org/pypi/corner
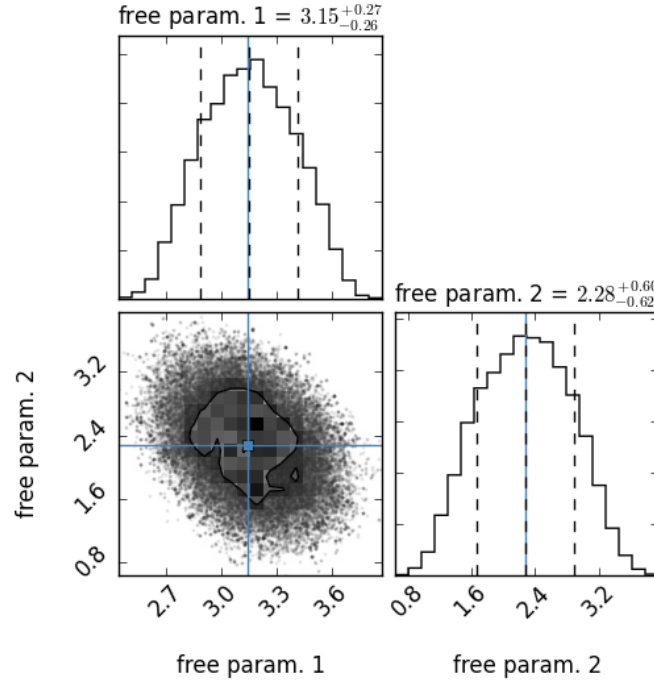
Figure 19: Example of a so-called *corner plot* created with the `corner` package. On top of each column the probability distribution for each free parameter is shown. The left and right dashed lines indicates the lower and upper limit of the corresponding HPD interval, respectively. The dashed line in the middle indicates the mode of the distribution. The blue lines indicate the parameter values of the best fit, calculated in previously applied optimization algorithm(s). The plot in the lower left corner describes the projected 2D histograms of two parameters. The contours indicate the HPD region.

For a generic normal distribution f with mean μ and deviation σ, the cumulative distribution function (CDF) for standard score $z = \left(\frac{x-\mu}{\sigma} \right)$ is

$$F(x) = \Phi \left(\frac{x - \mu}{\sigma} \right) = \frac{1}{2} \left[1 + \text{erf} \left(\frac{x - \mu}{\sigma \sqrt{2}} \right) \right]. \tag{46}$$

Following the so-called 68-95-99.7 (empirical) rule, or 3-sigma rule, about 68 % of values drawn from a normal distribution are within one standard deviation σ away from the mean; about 95.4 % of the values lie within two standard deviations; and about 99.7 % are within three standard deviations.

More precisely, the probability that a normal deviate lies in the range $\mu - n\sigma$ and $\mu + n\sigma$ is given by

$$P(n) = F(\mu + n\,\sigma) - F(\mu - n\,\sigma) = \Phi(n) - \Phi(-n) = \text{erf}\left(\frac{n}{\sqrt{2}} \right) \approx \begin{cases} 0.682, & n = 1 \\ 0.954, & n = 2 \\ 0.997, & n = 3 \end{cases}. \tag{47}$$

As mentioned above, the error estimation algorithm calculates the highest posterior density (HPD) interval of each parameter, respectively. A HPD interval is basically the shortest interval on a posterior density for some given confidence level, i.e. 68 % for 1σ, 95.4 % for 2σ etc. For example, if we're considering a 95.4 % (or 2σ) confidence interval, the HPD interval is the

---

[23]Taken from http://bebi103.caltech.edu/2015/tutorials/l06_credible_regions.html
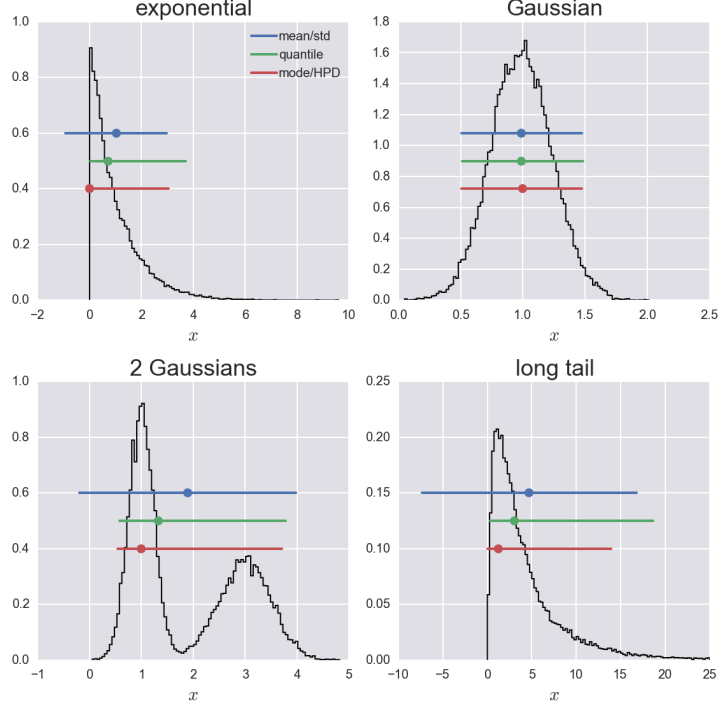
Figure 20: Three commonly used ways of plotting a value plus error bar for a 95 % credible region for four different probability distributions[23]: 1.) *Mean ± standard deviation (blue line)*: The most commonly used confidence interval is $\mu \pm k\sigma$, where k is chosen to give the appropriate confidence interval, assuming the posterior is Gaussian. Here, $k = 1.96$. 2.) *Median with quantile (green line)*: The posterior need not be Gaussian. If it is not, we would like a more robust way to summarize it. A simple method is to report the median, and then give lower and upper bounds to the error bar based on quantile. For a 95 % credible region we would report the 2.5th percentile and the 97.5th percentile. 3.) *Mode with HPD (red line)*: This method uses the highest posterior density (HPD) interval. If we're considering a 95 % confidence interval, the HPD interval is the shortest interval that contains 95 % of the probability of the posterior.

shortest interval that contains 95.4 % of the probability of the posterior. Mathematically a $100 \cdot (1 - P(n))$ % HPD interval (or region) for a subset $\mathcal{C} \in \Theta$ defined by

$$\mathcal{C} = \{\theta : \pi(\theta|x \geq k\}, \tag{48}$$

where k is the largest number such that

$$\int_{\{\theta : \pi(\theta|x \geq k\}} \pi(\theta|x)d\theta = 1 - P(n). \tag{49}$$

The value k can be thought of as a horizontal line placed over the posterior density whose intersection(s) with the posterior define regions with probability $1 - P(n)$, see Fig. 20. In order to compute a HPD interval we rank-order the MCMC trace. We know that the number of samples that are included in the HPD is 0.954 (or another confidence level) times the total number of MCMC sample. We then consider all intervals that contain that many samples and find the shortest one. In the case of a normal distribution an HPD interval coincides with the usual probability region symmetric about the mean, spanning the $\frac{n\sigma}{2}$ and $1 - \frac{n\sigma}{2}$ quantiles[24]. The same is true for any unimodal, symmetric distribution, see Fig. 20.

---

[24]In statistics and the theory of probability, quantiles are cut points dividing the range of a probability distribution into contiguous intervals with equal probabilities, or dividing the observations in a sample in the same way.

So, the error estimation algorithm reports the (first) mode[25] and then the bounds on the HPD intervals, see Fig. 19. Note, the mode must not coincidence with the best fit result of the previously applied algorithms!

Please note, the calculated HPD intervals are credible intervals not confidence intervals. In Bayesian statistics, a credible interval is an interval in the domain of a posterior probability distribution or predictive distribution used for interval estimation. The generalization to multivariate problems is the credible *region*. Credible intervals are analogous to confidence intervals in frequentist statistics, although they differ on a philosophical basis; Bayesian intervals treat their bounds as fixed and the estimated parameter as a random variable, whereas frequentist confidence intervals treat their bounds as random variables and the parameter as a fixed value. (Taken from https://en.wikipedia.org/wiki/Credible_interval).

10.5.10.3 Error estimation using Interval Nested Sampling (INS)

A schematic diagram of the error estimation module using INS is shown in Fig. 21: After some optimization procedure MAGIX determines a point of minimum. The input values for the error estimation are given by the point of minimum and the parameter space. To determine the error of a parameter θ_j at the minimum, MAGIX varies this parameter within the given parameter range, while the other parameters are kept constant. The INS algorithm is applied and returns a set of parameter values with proportional weights and the logarithm of the Bayesian evidence for parameter θ_j for a sequence of parameter values distributed over the whole parameter space. If the distribution of parameter values has only one minimum, Eqn. (37) - (38) can be applied to calculate the mean value and the standard deviation. Sometimes there are several minima in the sequence, hence these formulas cannot be used directly because the resulting estimation of the mean value and standard deviation produces meaningless results (see Figs. 17 and 22).

We are interested in the uncertainty around the considered minimum. Therefore, we need to estimate the mean value and the standard deviation in the minimum. This is done using a clustering method:

▶ Calculate the distances from the minimum to all points of the sample.

▶ Sort the points depending on their distances (ascending order).

▶ Select the points with a function value lower than the χ^2 boundaries for the 99 percent confidence region $\Delta\chi^2_{\alpha,n}$ ($\alpha = 0.99$).

Finally, the new sample of m points ($m < k$) is distributed around the minimum point (Fig. 22, gray box) and the mean value of the parameter θ_j is calculated as follows:

$$\mu\left(\theta_j\right) = \frac{\sum_{i=1}^{m} w_i \theta_j}{\sum_{i=1}^{m} w_i}. \tag{50}$$

There is one less quantile than the number of groups created. Thus quartiles are the three cut points that will divide a dataset into four equal-size groups. q-Quantiles are values that partition a finite set of values into q subsets of (nearly) equal sizes. There are q - 1 of the q-quantiles, one for each integer k satisfying 0 < k < q. In some cases the value of a quantile may not be uniquely determined, as can be the case for the median (2-quantile) of a uniform probability distribution on a set of even size. (Taken from https://en.wikipedia.org/wiki/Quantile)

[25]Generally, the *mode* is the value that appears most often in a set of data. For a normal distribution the numerical values of mode, mean, and median are identical, and may be very different in highly asymmetric distributions.

Using the mean value $\mu(\theta_j)$ and the standard deviation $\sigma(\theta_j)$, the 3σ confidence interval of the parameter θ_j is given by

$$Pr(\mu(\theta_j) - \sigma(\theta_j) < \theta_j < \mu(\theta_j) + \sigma(\theta_j)) \approx 0.99. \tag{51}$$

In Fig. 23 the example of a histogram of distribution of parameter values after error estimation is shown. Hence for the parameter value at the point of minimum (using the error left and the error right),

$$Pr(\theta_i(\text{min}) - \text{error}_{\text{left}} < \theta_j < \theta_i(\text{min}) + \text{error}_{\text{right}}) \approx 0.99. \tag{52}$$

The Bayesian evidence usually plays an important role in model selection but in parameter estimation the evidence factor is ignored because it is an integrated value over the whole parameter space. The INS algorithm calculates the logarithm of the evidence and can be used to estimate the quality of the fitting procedure. A high absolute value of the evidence logarithm indicates a big uncertainty of the parameters.

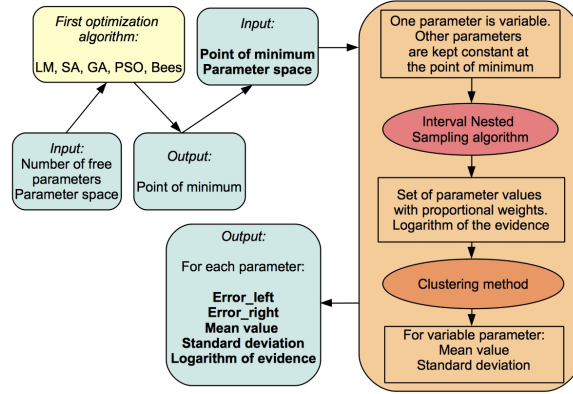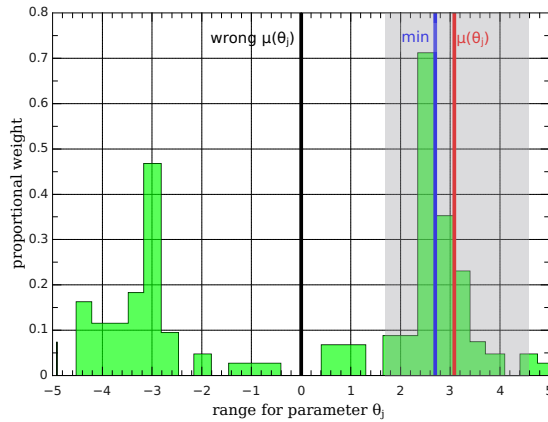**Figure 21:** Schematic diagram of error estimation module of MAGIX.

Figure 22: Distribution of parameter values with several minima where a direct application of Eqn. (37) - (38) is not possible. (Here, $\mu(\theta_j)$ indicates the mean value $\mu(\theta_j)$, $\sigma(\theta_j)$ is the standard deviation $\sigma(\theta_j)$, and "min" represents the value of the parameter θ_j of the best-fit result).
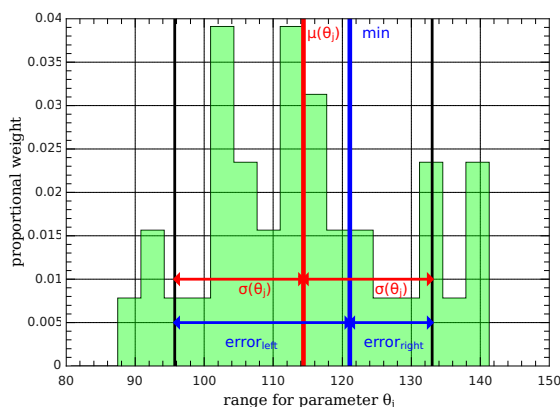
Figure 23: Schematic diagram of error estimation module of MAGIX. (Here, $\mu(\theta_j)$ indicates the mean value $\mu(\theta_j)$, $\sigma(\theta_j)$ represents the standard deviation $\sigma(\theta_j)$, and "min" the value of the parameter θ_j of the best-fit result).

10.5.11 Which algorithm should be used?

In principle, it is impossible to answer the question which algorithm is best. All algorithms included in MAGIX try to find the global minimum of the χ^2 function, which depends on the observational data, on the external model, on the free and fixed parameters, and on the ranges for each free parameter. Every algorithm has a different strategy for finding the minimum of the χ^2 function. Whether the strategy is successful depends on the χ^2 function and many other conditions. For example, the bees algorithm requires a huge computational effort, but this algorithm explores the whole landscape of the χ^2 function within the given ranges. It gives a good overview of the landscape of the problem and can in principle find even minima in narrow valleys. On the other hand the computational effort is heavy, and the algorithm is inefficient if computing the external model requires more than a few seconds. Here, the computation time of an algorithm depends on the number of function calls for each iteration, the computation time of the external model program, the possibility of executing the external model program more than once on the same machine at the same time and on the size of the parameter space. Other swarm algorithms such as the particle swarm or NS algorithms might be better if the χ^2 function has a smoother shape. For example, the particle swarm algorithm included in MAGIX makes use of a Nelder–Mead simplex search method, which accelerates the whole computation if the χ^2 function has no narrow valleys, i.e, the χ^2 function does not change drastically when one or more parameters are varied.

In Table 1 the total cost (in function evaluations) for each test function (Rastrigin, Rosenbrock, and Himmelblau function) for each global optimization algorithm is shown. As mentioned above, the total cost of an algorithm, i.e., the number of function evaluations depends strongly on many parameters. Therefore, Table 1 can give only a very rough overview of the efficiency for each algorithm. For example, the INS algorithm is quite efficient in finding the global minimum of the Rastrigin function but less efficient in finding the global minimum of the Rosenbrock function.

In general, the global optimization algorithms are more efficient in finding the areas of global minima. But they are less efficient in finding the exact properties of the global minima. For that purpose, local optimizers, such as the LM or SA are much more efficient. The combination of different algorithms is the best strategy to find the complete description of the experimental data.
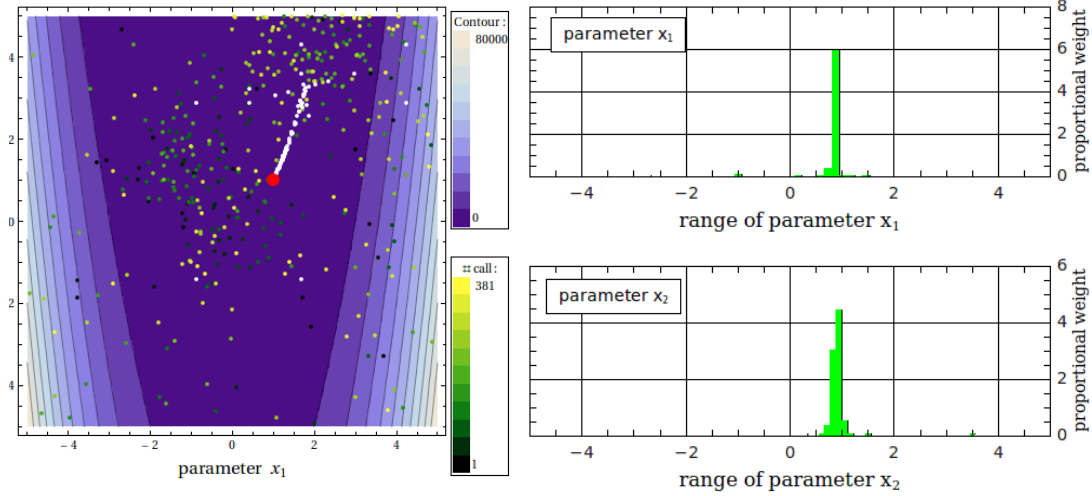
Figure 24: Results for the Rosenbrock function using an algorithm chain: Left panel: The distribution of the parameter values for the bees algorithm (green-yellow points, where the dark green points indicate function calls that are made at the beginning of the fit process and light yellow points represent function calls at the end of the fit process) and for the following simulated annealing algorithm (white points). The red dot indicates the global minimum of the Rosenbrock function. Right panel: Posterior weights of points after the NS process used by the error estimation module.

10.5.12   Algorithm chain

Therefore, MAGIX includes the possibility to send the results of the optimization process performed by one algorithm to another optimization loop through some different algorithm. As mentioned above, the SA as well as the LM algorithm require starting values of the parameters that are optimized, i.e., the user has to find a good fit by hand before applying these algorithms produces useful results. Often, the location of the minimum can be guessed with sufficient accuracy to give good starting values, but sometimes one is completely in the dark. Using an algorithm chain, the user can first apply one of the swarm algorithms, e.g., the bees or NS algorithm, to determine the starting values for the subsequent local optimization algorithm using SA or the LM algorithm. As shown in Fig. 24, we used an algorithm chain to determine the global minimum of the Rosenbrock function where we first applied the bees algorithm to explore the landscape of the problem. Using the best result of the bees algorithm (the parameter set that

algorithm name	Rastrigin function $\chi^2_{\text{limit}} = 1$	Rosenbrock function $\chi^2_{\text{limit}} = 4 \cdot 10^{-3}$	Himmelblau function $\chi^2_{\text{limit}} = 5 \cdot 10^{-4}$
Bees	1220	14491	101664
PSO	1317	535	770
Genetic	241	533	1626
NS	4230	5080	8720
INS	20	1144	168

Table 1: Total cost (in function evaluations) for each test function for each global optimization algorithm. The algorithms stop if the value of χ^2 dropped below the given limit of χ^2. We neglect here the so-called local optimizer such as LM and SA algorithm because the efficiency of these algorithms depends strongly on the starting values.

parameter name	value at minimum	error left	error right	log (evidence)
x_1	0.9954	$3.43 \cdot 10^{-2}$	$2.85 \cdot 10^{-2}$	-3.309
x_2	0.9905	$7.38 \cdot 10^{-2}$	$8.21 \cdot 10^{-2}$	-2.846

Table 2: Best parameter set with the corresponding confidence intervals ($\chi^2 = 2.5576 \cdot 10^{-5}$) after applying an algorithm chain. We used an algorithm chain consisting of the bees, the SA and the error estimation algorithm, to determine the global minimum of the Rosenbrock function. The bees algorithm requires only a specification of the range for each free parameter. Here, the parameters x_1 and x_2 vary between -5 and +5.

corresponds to the lowest χ^2 value, here $x_1 = \mathbf{1.7044}$ and $x_2 = \mathbf{2.8679}$) as starting point for the SA algorithm, we find the global minimum of the Rosenbrock function. MAGIX does not only allow one to use the best but also the second best etc. result of a swarm algorithm as starting values for other algorithms. Therefore, we are able to find multiple minima of models that behave as the Himmelblau function. To determine the confidence intervals for the parameters that are optimized, the user has to set the error estimation algorithm as the last algorithm of the algorithm chain. The confidence intervals for the parameters that are used in the example shown in Fig. 24 are given in Table 2. The upper right panel of Fig. 24 shows the probability for a minimum along the x_1 axis holding parameter x_2 fixed at $x_2 = \mathbf{0.9905}$. The lower right panel of Fig. 24 shows the probability for a minimum along the x_2 axis holding parameter x_1 fixed at $x_1 = \mathbf{0.9954}$. These two panels clearly show that there is only one minimum within the given parameter range.

10.5.13 Examples

We fit HIFI bands 4b and 5a toward SgrB2(M) {Schilke *et al.* 2010} with myXCLASS, simultaneously using an algorithm chain starting with the genetic algorithm and 78 free parameters (we used 26 velocity components where each component has three free parameters.). Here, we used the parameter values of the best fit-result of the genetic algorithm as starting values for the SA. At the end of the algorithm chain, we applied the error estimation algorithm to determine the left and right error for each optimized parameter. The final result is shown in Fig. 25, where the dashed red (blue) lines indicate a model where we reduced (increase) the free parameter values of the best fit by the left (right) error of each free parameter. Clearly, we achieve an excellent description of the absorption lines and quantify the uncertainty of the model.

10.6 Fit control xml file

Listing 4: Example of a fit control file (The expressions "`<!--`" and "`-->`" indicate a remark line in a xml-file.)

```
<?xml version="1.0" encoding="UTF-8"?>
<FitControl>
    <!-- settings for fit process -->


    <!-- set number of used algorithms -->
    <NumberOfFitAlgorithms>1</NumberOfFitAlgorithms>


    <algorithm>
        <!-- define algorithm -->
```
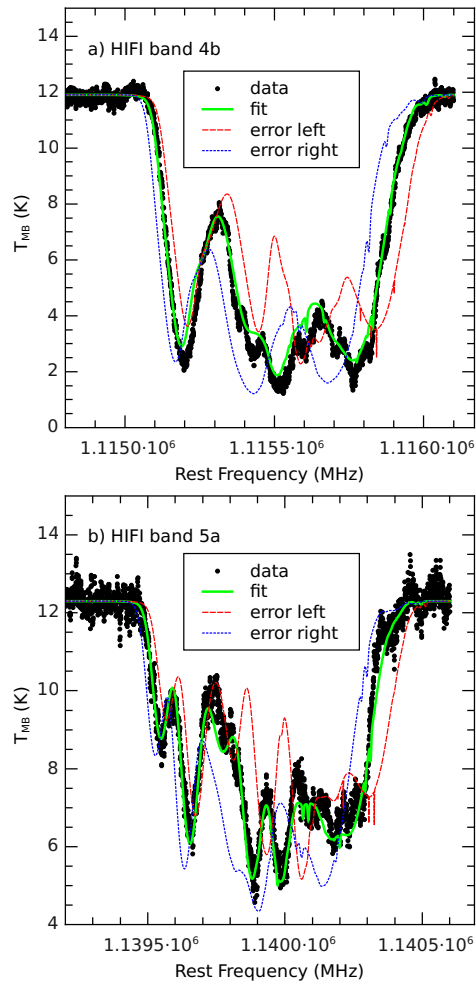
Figure 25: Fit of ortho H_2O^+ ground state lines in HIFI bands a) 4b and b) 5a toward SgrB2(M) with myXCLASS using an algorithm chain consisting of the genetic, the SA, and the error estimation algorithm. Owing to the high number of free parameters (78) it is not possible to show the distribution of the parameter values as in Figs. 8 - 24.

```
<FitAlgorithm>bees</FitAlgorithm>


<!-- special settings for bees algorithm -->
<!-- BestSiteCounter (number of best sites) > 0 -->
<BestSiteCounter>5</BestSiteCounter>


<!-- set max. number of iterations -->
<number_iterations>10</number_iterations>


<!-- set max. number of processors -->
<NumberProcessors>8</NumberProcessors>


<!-- set path and name of host file -->
<MPIHostFileName>hostfile.txt</MPIHostFileName>
```

```
        <!-- settings for chi^2 -->
        <limit_of_chi2>0.001</limit_of_chi2>
        <RenormalizedChi2>yes</RenormalizedChi2>
        <DeterminationChi2>default</DeterminationChi2>
        <SaveChi2>yes</SaveChi2>


        <!-- set plot options -->
        <PlotAxisX>Frequency [Hz]</PlotAxisX>
        <PlotAxisY>Intensity</PlotAxisY>
        <PlotIteration>no</PlotIteration>
    </algorithm>
</FitControl>
```

▶ The tag `<NumberOfFitAlgorithms>` defines the number of algorithms which should be used within the fit process. A number greater than 1 defines a so-called *algorithm chain* (see example 5).

▶ The settings for each algorithm are enclosed inside the `<algorithm>` tag. The tag has to occur as many times as specified by the tag `<NumberOfFitAlgorithms>`.

▶ Each algorithm is described by the tags
`<FitAlgorithm>`, `<number_iterations>`, `<NumberProcessors>`, (`<MPIHostFileName>`),
`<limit_of_chi2>`, `<DeterminationChi2>`, `<SaveChi2>`, `<RenormalizedChi2>`,
`<PlotAxisX>`, `<PlotAxisY>`, (`<PlotAxisZ>`) and `<PlotIteration>`.
Depending on the chosen algorithm, a couple of additional tags have to be added (§ 10.6.4).

▶ The tag `<FitAlgorithm>` defines the algorithm that is used in the fit process. The content of the `<FitAlgorithm>` tag has to be identical with one of the algorithm names (information on the available algorithms in section § 10.5; it does not matter if these words are written in lower or upper case letters):

  – `levenberg-marquardt` (§ 10.5.1),

  – `simulated-annealing` (§ 10.5.2),

  – `pso` (§ 10.5.3),

  – `bees` (§ 10.5.4),

  – `genetic` (§ 10.5.5),

  – `mcmc` (§ 10.5.6)

  – `nested-sampling` (§ 10.5.7),

  – `additionalpackages` (§ 10.5.9),

  – `interval-ns` (§ 10.5.8),

  – `errorestim_ins` (§ 10.5.10).

▶ The tag `<number_iterations>` sets the number of iterations for each algorithm and has to be an integer greater zero (N_I, integer > 0).

▶ The tag `<NumberProcessors>` defines the number of processors to be used by MAGIX.

NOTE that a value > 1 can be used only for external model programs that allow parallelized work (see MAGIX manual for more detail).

▶ The tag `<MPIHostFileName>` defines the path and name of a so-called host file required for the MPI parallelized version of MAGIX, see § 10.6.1. The host file contains names of all computers on which the MPI job will execute. For ease of execution, the user should be sure that all of these computers have SSH access, and that an authorized keys file is defined to avoid a password prompt for SSH. Additionally, the number of cores which should be used for an MPI run on each machine can be limited by using the "slots" command. For that purpose, the user has to extend the name of each machine by the definition of cores.

```
meslam slots =16
lugal slots =8
anu slots =2
```

In the example described above, we use 16 cores on "meslam", 8 cores on "lugal" and two cores on "anu".

NOTE, if the total number of processors defined in the host file is smaller than the number of processors defined by the tag `<NumberProcessors>` MAGIX will reduce this value.

For some supercomputers the user does not need to specify a host file, because there is already a host file defined. In order to use a globally defined MPI host file, please insert the phrase `MPI_HOSTS` into the tag `<MPIHostFileName>`.

If you've installed the MPI version of the XCLASS interface but do not define a host file, i.e. leave the tag `<MPIHostFileName>` empty, the XCLASS interface will use only the current machine (localhost) with the number of cores defined by the tag `<NumberProcessors>`.

10.6.1 Different parallelization techniques used by MAGIX

MAGIX supports two different parallelization techniques. One the one hand MAGIX provides the algorithms in a SMP (Symmetric multiprocessing) parallelized version using OpenMP. A symmetric multiprocessor system where two or more identical processors are connected to a single, shared main memory, have full access to all I/O devices, and are controlled by a single operating system instance that treats all processors equally. Modern multiprocessors offers up to 32 different processor cores which can be used for a MAGIX run. This parallelization technique is very fast, because all processors use the same memory, but the number of threads is limited by the number of available cores on the current machine.

In contrast to the SMP parallelization, MPI (Message Passing Interface) is a standardized and portable message-passing system, where one or more computers are connected in a so-called cluster. MPI parallelization is somewhat slower than SMP, caused by the network, but it can be used with an (in principle) unlimited number of cores. In order to use the MPI parallelization, the user has to install the OpenMPI package on all computers in the cluster. Additionally, the user has to provide a temp directory which is visible by all computers. Please note, by using the environment variable `MAGIXTempDirectory` (see § 10.2.1), the user can define different paths for the temp directory on each machine in the cluster. Depending on the external model program, this makes the definition of a directory which is visible by all computers in the cluster dispensable.

10.6.2 Tags concerning χ^2

MAGIX is able to fit the values of parameters, providing confidence intervals presented by the value of χ^2.

▶ `<DeterminationChi2>`: Specifies the method that is used for the determination of χ^2. At the moment the following options are included in MAGIX:

– default:

$$\chi^2 = \sum_{i=1}^{N} \left(y_i^{\text{obs}} - y_i^{\text{fit}} \right)^2$$

where y_i^{obs} represents the value of the experimental data at point i, and y_i^{fit} the corresponding value of the fit function.

If the tag `<DeterminationChi2>` is set to `default` or `difference`, then the content of the tag `<SaveChi2>` is read (yes/no, default value `yes`), which specifies whether the difference $y_i^{\text{obs}} - y_i^{\text{fit}}$ is saved for all experimental points to a file (in the `.chi2` log file).

Note, if the experimental data file(s) includes error values then the χ^2 value is defined as follows:

$$\chi^2 = \sum_{i=1}^{N} \left[\left(y_i^{\text{obs}} - y_i^{\text{fit}} \right)^2 \cdot \frac{1}{(\sigma_i^{\text{error}})^2} \right],$$

where σ_i^{error} represents the error of the ith data point.

– ...

▶ The tag `<limit_of_chi2>` specifies the value of χ^2 where the fitting process stops, i.e. if the value of χ^2 drops below this value the algorithm stops. The limit of χ^2 should be a real number > 0.

▶ `<RenormalizedChi2>` (yes/no, default value `yes`): specifies if MAGIX uses a re-normalized value for the limit of χ^2. If you set the flag to `yes` or `y`, then MAGIX determines the limit of χ^2 through the relation

$$\left(\chi_{\text{limit}}^2 \right)_{\text{renom}} = \left(\sum_{i=1}^{N_{\text{exp}}} N_Y(i) \cdot N_{\text{points}}(i) - N_{\text{par}} \right) \cdot \left(\chi_{\text{limit}}^2 \right)_{\text{orig}}$$

where N_{exp} is the number of observation files `Number_ExpFiles`; $N_Y(i)$ represents the number of Y columns of observation file i; $N_{\text{points}}(i)$ indicates the number of observation data points in the observation file i; N_{par} `NumberParameters` is the total number of all parameters; $\left(\chi_{\text{limit}}^2 \right)_{\text{orig}}$ is the original unmodified value of χ^2.

10.6.3 Tags available only for 2D and 3D plots of 1D functions $y = f(x)$ and $y = f(x, y)$

▶ The tags `<PlotAxisX>` and `<PlotAxisY>` define the labels for the X and Y axis, respectively.

▶ The tag `<PlotAxisZ>` is used only for 3D plots.

▶ The observed data and the fit function are plotted for each iteration step, if the tag `<PlotIteration>` is set to `yes` (default value `no`). Please note, that this option starts a daemon, which refreshes the plot window every 3 sec. By setting the environment variable `MAGIXTimePlotIter`

```
export MAGIXTimePlotIter="5000"
```

the user can define another time interval (in milliseconds). Note, if the time interval is to short no spectrum is plotted.

Please take into account, that the creation of the plot window requires time as well.

10.6.4 Tags required only for certain algorithms

▶ The following tag is only relevant when the algorithm chosen is **NS**, **PSO**, **Bees**, **Genetic**, **MCMC** or **INS**:

  – `<BestSiteCounter>`: Defines the number of best sites. MAGIX writes the results (parameter set and value of the model function) of these sites to files. A number of best sites greater than one, is especially useful when the χ^2 function has multiple minima, i.e. there is more than one best fit (description) of the experimental data. Additionally, a number greater than one is useful when you want to use a so-called *algorithm chain*, and the current algorithm (§ 10.5.12) is not the last one in the chain.

  Imagine you use an algorithm chain (§ 10.5.12) of two algorithms: The first algorithm is the Bees, with the best site counter set to 2, so MAGIX will search and find the two best parameter sets. Then, if the next algorithm in the chain is the Levenberg-Marquardt, it will find the best fitting parameter sets, starting from the two sites found previously by the Bees. File sample 5 shows the fit control file of a similar scheme with the best site counter for Bees set to 3.

▶ The following tags are only relevant when the algorithm chosen is **Levenberg-Marquardt** (§ 10.5.1):

  – `<VariationValue>` (var, real positive number > 0, typical value 10^{-6}): For each iteration step the Levenberg-Marquardt algorithm has to determine the gradient of the χ^2 function. Due to the fact that MAGIX can not determine the components of the gradient analytically, MAGIX has to use a numerical approximation:

  $$\frac{\partial}{\partial x_i} f(\vec{x}) = \frac{f(x_i + h) - f(x_i)}{h},$$

  where the variation h is defined by

  $$h = x_i \cdot var.$$

  Varying the value of `<VariationValue>` could be very useful if the χ^2 function is not a smooth function and the calculation of the gradient produces awkward results.

▶ The following tags are only relevant when the algorithm chosen is **Simulated Annealing** (§ 10.5.2):

  – `<Temperature>` (T_0, real number > 0, typical value 1000): This tag defines the starting value for the global temperature, which is updated (decreased) at every step of the fit process.

  – `<TemperatureReductionKoeff>` (k, real number > 0 and < 1, typical value 0.8): Defines coefficient for the temperature reduction.

  – `<NumberOfReduction>` (N_R, integer number > 0, typical value 10): The value defines the number of temperature reductions. The number of reductions N_R does not need to be the same as the number of iterations N_I specified by the `number_iterations` tag. If $N_R < N_I$ and the first N_R reductions have been completed, then the global temperature is reset to T_0, but with the configuration being the one that resulted from the last reduction. The procedure continues until the total number of reductions completed is N_I.

– `<NumberOfReheatingPhases>` (N_{rH}, integer number > 0, typical value 3): The value defines the number of the reheating phases. The Simulated Annealing algorithm "heats" with the temperature T_0 which leads to a modification of the starting values. Then, T_0 is reduced by k for N_R iterations. If the algorithm is not able to find a better χ^2 value after N_R iterations, MAGIX "heats" again with temperature T_0. MAGIX will repeat this process N_{rH} times before it stops the algorithm.

– `<ScheduleSA>` This tag (only used for `scipy` version) defines the annealing schedule. Available ones are 'fast', 'cauchy', 'boltzmann'.

▶ The following tag is only relevant for the **Bees** algorithm (§ 10.5.4):

– `<NumberBees>`: This tag defines the number of the so-called bees, which should be used within the Bees algorithm. (The default setting is `automatic`.) The user can define a value which has to be larger than

$$N_{\mathrm{Bess}} = N_{\mathrm{site}} \cdot (5 + 11 \cdot N_{\mathrm{site}}) \cdot N_{\mathrm{free}},$$

where N_{site} indicates the number of best sites, see above, and N_{free} the number of free parameters. Otherwise, MAGIX determines the number of bees automatically. Note that a bigger number would lead to an increased computational effort, whereas a smaller number can produce a worse result. But this depends immensely on the model function used in the fitting process.

▶ The following tags are only relevant for the **Genetic** algorithm (§ 10.5.5):

– `<NumberChromosomes>`: This tag defines the number of the so-called chromosomes, which should be used within the Genetic algorithm. (The default setting is `automatic`.) The user defined value has to be larger than zero. Note that a bigger number would lead to an increased computational effort, whereas a smaller number can produce a worse result. But this depends immensely on the model function used in the fitting process.

– `<UseNewRange>`: This tag defines if the algorithm should determine new (shrinked) ranges for each free parameter (`yes`), or not (`no`). (The default setting is `yes`.)

▶ The following tags are only relevant when the algorithm chosen is **Nested Sampling** (§ 10.5.7):

– `<NumberObjects>`: This tag defines the number of the so-called objects, which should be used within the NS algorithm. A typical value is 100. Note that a bigger number would lead to an increased computational effort, whereas a smaller number can produce a worse result. But this depends immensely on the model function used in the fitting process.

▶ The following tags are only relevant when the algorithm chosen is **MCMC** (§ 10.5.6):

– `<NumberMCMCSampler>`: This tag defines the number of the so-called walkers, which should be used within the MCMC algorithm. A typical value is 100. Note that a bigger number would lead to an increased computational effort, whereas a smaller number can produce a worse result. But this depends immensely on the model function used in the fitting process.

Following {Foreman-Mackey *et al.* 2012} there is no reason not to go large when it comes to walker number, until you hit performance issues. Although each step takes twice as much compute time if you double the number of walkers, it also

returns to you twice as many independent samples per autocorrelation time. So go large. In particular, we have found that-in almost all cases of low acceptance fraction-increasing the number of walkers improves the acceptance fraction. The one disadvantage of having large numbers of walkers is that the burn-in phase (from initial conditions to reasonable sampling) can be slow; burn-in time is a few autocorrelation times; the total run time for burn-in scales with the number of walkers. These considerations, all taken together, suggest using the smallest number of walkers for which the acceptance fraction during burn-in is good, or the number of samples you want out at the end (see below), whichever is greater. A more ambitious project would be to increase the number of walkers after burn-in; this requires thought beyond the scope of this document; it can be accomplished by burning in a set of small ensembles and then merging them into a big ensemble for the final run. One mistake many users of MCMC methods make is to take too many samples! If all you want your MCMC to do is produce one- or two-dimensional error bars on two or three parameters, then you only need dozens of independent samples. With ensemble sampling, you get this from a single snapshot or single time step, provided that you are using dozens of walkers (and we would recommend that you use hundreds in most applications). The key point is that you should run the sampler for a few (say 10) autocorrelation times. Once you have run that long, no matter how you initialized the walkers, the set of walkers you obtain at the end should be an independent set of samples from the distribution, of which you rarely need many. Another common mistake, of course, is to run the sampler for too few steps. You can identify that you haven't run for enough steps in a couple of ways: If you plot the parameter values in the ensemble as a function of step number, you will see large-scale variations over the full run length if you have gone less than an autocorrelation time. You will also see that if you try to measure the autocorrelation time (with, say, acor), it will give you a time that is always a significant fraction of your run time; it is only when the correlation time is much shorter (say by a factor of 10) than your run time that you are sure to have run long enough. The danger of both of these methods-an unavoidable danger at present-is that you can have a huge dynamic range in contributions to the autocorrelation time; you might think it is 30 when in fact it is 30 000, but you don't "see" the 30 000 in a run that is only 300 steps long. There is not much you can do about this; it is generic when the posterior is multi-modal: The autocorrelation time within each mode can be short but the mode-mode migration time can be long. See above on low acceptance ratio; in general when your acceptance ratio gets low your autocorrelation time is very, very long.

- `<NumberBurnInIter>`: This tag defines the number of iterations (default 50) used for the so-called *burn-in* phase, see (§ 10.5.10.2).

▶ The following tags are only relevant when the algorithm chosen is **Additional packages** (§ 10.5.9):

  – `<minAlgorithm>`: This tag defines the name of the `scipy` algorithm which should be used. The following algorithms are available:

    1. "fmin",
    2. "fmin_powell",
    3. "fmin_cg",
    4. "fmin_bfgs",
    5. "fmin_ncg",

6. "fmin_l_bfgs_b",

7. "fmin_tnc",

8. "anneal",

9. "brute".

For further documentation see documentation of `scipy` package.

Note that the tag `<minAlgorithm>` has to include one of the above listed names of algorithms.

▶ The following tags are only relevant when the algorithm chosen is **Interval-Nested-Sampling** (§ 10.5.8):

- `<vol_bound>`: This tag indicates the critical element of the volume. If the tag is empty, then MAGIX determines the value using the following expression:

$$\text{vol\_bound} = 0.1 \cdot \left(1.0 - \sqrt{\frac{(N_{\text{free}} - 0.75)}{N_{\text{free}}}} \right),$$

where N_{free} indicates the number of free parameters.

- `<delta_incl>`: This tag defines the difference between maximal and minimal value of inclusion function. (The default setting is `0.001`.)

▶ The following tags are only relevant when the algorithm chosen is the **Error estimation** (§ 10.5.9):

- `<ErrorMethod>`: This tag defines the method ("MCMC" (default), "INS", "Fisher') which is used for error estimation (§ 10.5.10).

- `<NumberMCMCSampler>`: This tag (relevant only for the MCMC method) describes the number of samples / walkers (default $2N$, where N indicates the number of free parameters) which are used by the MCMC algorithm, see description of tags used by MCMC algorithm above.

- `<NumberBurnInIter>`: This tag (relevant only for the MCMC method) defines the number of iterations (default 50) used for the so-called *burn-in* phase, see (§ 10.5.10.2).

- `<UsePrevResults>`: This tag (relevant only for the MCMC method) indicates, if parameter vectors calculated by other algorithms in the algorithm-chain, are used for the burn-in phase (`True`) or not (`False`, default). Using previous calculated parameter vectors reduces the computational effort, but the parameters are not calculated at the position which where generated by the MCMC algorithm. So, the underlying probability distribution might be not well sampled.

- `<MultiplicitySigma>`: This tag (relevant only for the MCMC method) defines the multiplicity (default 2) of the standard deviation σ, which defines the error bounds of the free parameters. For example, by setting the multiplicity to 2 the error estimation algorithm computes the 2σ errors for the free parameters.

- `<VariationValue>`: This tag (relevant only for the Fisher method) specifies the variation value (default 10^{-3}, see description of the `<VariationValue>` tag for the Levenberg-Marquardt algorithm described above) used for the computation of the covariance matrix, see (§ 10.5.10.1).

10.6.5 Optimization through an algorithm chain

It is possible to send the results of the optimization process performed by a certain algorithm, to another optimization procedure through some other algorithm. Some directives about how to select the order of the algorithms to use are given in § 10.5.11.

In file 5, the fitting process starts with the Bees algorithm. Thereafter, the Levenberg-Marquardt algorithm is applied to the best three sites found previously by the Bees algorithm.

Listing 5: Example of a fit control file with an algorithm chain

```xml
<?xml version="1.0" encoding="UTF-8"?>
<FitControl>
    <!-- settings for fit process -->


    <!-- set number of used algorithms -->
    <NumberOfFitAlgorithms>2</NumberOfFitAlgorithms>


    <algorithm>
        <!-- define algorithm -->
        <FitAlgorithm>bees</FitAlgorithm>


        <!-- special settings for bees algorithm -->
        <!-- BestSiteCounter (number of best sites) > 0 -->
        <BestSiteCounter>3</BestSiteCounter>


        <!-- set max. number of iterations -->
        <number_iterations>30</number_iterations>


        <!-- set max. number of processors -->
        <NumberProcessors>8</NumberProcessors>


        <!-- set path and name of host file -->
        <MPIHostFileName>hostfile.txt</MPIHostFileName>


        <!-- settings for chi^2 -->
        <limit_of_chi2>0.001</limit_of_chi2>
        <RenormalizedChi2>yes</RenormalizedChi2>
        <DeterminationChi2>default</DeterminationChi2>
        <SaveChi2>yes</SaveChi2>


        <!-- set plot options -->
        <PlotAxisX>Frequency [Hz]</PlotAxisX>
        <PlotAxisY>Intensity</PlotAxisY>
        <PlotIteration>yes</PlotIteration>
    </algorithm>

    <algorithm>
        <!-- define algorithm -->
        <FitAlgorithm>Levenberg-Marquardt</FitAlgorithm>


        <!-- set max. number of iterations -->
```

```
        <number_iterations>20</number_iterations>


        <!-- set max. number of processors -->
        <NumberProcessors>8</NumberProcessors>


        <!-- set path and name of host file -->
        <MPIHostFileName>hostfile.txt</MPIHostFileName>


        <!-- settings for chi^2 -->
        <limit_of_chi2>0.0008</limit_of_chi2>
        <RenormalizedChi2>yes</RenormalizedChi2>
        <DeterminationChi2>default</DeterminationChi2>
        <SaveChi2>yes</SaveChi2>


        <!-- set plot options -->
        <PlotAxisX>Frequency [Hz]</PlotAxisX>
        <PlotAxisY>Intensity</PlotAxisY>
        <PlotIteration>yes</PlotIteration>
    </algorithm>
</FitControl>
```

10.7 Experimental data

Before the experimental files can be imported, the xml-file containing all settings required for the import of the experimental data has to be loaded. More specifically, the *experimental xml-file* defines the number of experimental files, it describes and gives the path and file name for each one of them.

10.7.1 General tags

All tags are necessary (except for `<MinExpRange>` and `<MaxExpRange>` when `<NumberExpRanges>` is set to 0, see § 10.7.2).

- ▶ MAGIX is able to read files in a variety of experimental data formats. At the moment it can load experimental data stored in ASCII, FITS format. The user can select the format of the experimental data file with the `<ImportFilter>` tag:

  – If the user sets the content of the tag `<ImportFilter>` to `automatic` (this option does not exist in XMLgen – if it existed, then it would not be possible to make the necessary tags for each filter appear), then MAGIX chooses the correct import filter depending the ending of the file. For that setting to function properly, the files have to end with either one of `.dat`, `.fits` and `.cso` for ASCII, FITS and CLASS files respectively.

  – The user can also specify the correct import filter, by setting the tag `<ImportFilter>` to ASCII, FITS, or CLASS.

- ▶ The names of the tags `<ExpFiles>`, `<NumberExpFiles>` etc. have to be written in the same way as presented in the example above.

- ▶ The tag `<file>` must occur as many times as the number of experimental files defined in the tag `<NumberExpFiles>`.

► Note, the order of the different experimental data files in the xml-file becomes important if the external model program creates for each function call more than one output file. MAGIX assumes that the first experimental data file is described by the first output file of the model program etc. The order of the output files is defined in the registration file (§ 10.3). Therefore, the user has to declare the different experimental data files in the correct order:

For example, the external model program creates two output files: The first file contains the transmission as a function of frequency and the second file describes the velocity as a function of frequency. If the first experimental data file (1) includes the measured transmission as a function of frequency, and the second file (2) the corresponding data for the velocity, the user has to declare file (1) first.

10.7.2 Experimental data ranges

► The number of ranges `<NumberExpRanges>` must always be given! If no range is desired (i.e. all data contained in the file are to be included in the fitting process), then the number of ranges must be set to 0. For XCLASS files it is a bit more complicated than that, see § 10.7.6).

► If the number of ranges is set to 0, then the tags `<MinExpRange>` and `<MaxExpRange>` need not be given.

► If the user does not want to use all data (number of ranges > 0), then the tags `<MinExpRange>` and `<MaxExpRange>` have to occur as many times as defined by `<NumberExpRanges>`.

► Note that the XML description of experimental data files for ASCII, FITS and CLASS files differ by some tags (see below for each case, § 10.7.5-10.7.6).

10.7.3 X and Y columns

► The expression *X column* refers to an independent variable of the model. The X columns are the columns of an array defining the X position of the experimental data point.

For a function $f(x_1, x_2, x_3)$, the number of X columns is 3; for a function $f(x_1, x_2)$, the number of X columns is 2.

► The tag `<NumberColumnsX>` defines the number of columns (starting from the left column) that belong to the X points of each experimental data file.

If the user wants to import an ASCII file containing 3D data, then `<NumberColumnsX>` has to be set to 3. The first 3 columns will then define the X, Y and Z position.

► If the number of X columns is > 1, then the min and max of X columns of the ranges have to be separated by the comma (,) character.

That's for one X column:

```
<NumberColumnsX>1</NumberColumnsX>
<MinExpRange>0</MinExpRange>
<MaxExpRange>2000</MaxExpRange>
```

For three X columns:

```
<NumberColumnsX>3</NumberColumnsX>
<MinExpRange>0, 0, 0</MinExpRange>
<MaxExpRange>2000, 100, 20</MaxExpRange>
```

▶ The expression *Y column* refers to a dependent variable of the model. The `<NumberColumnsY>` is only relevant for ASCII files (§ 10.7.4). For a given ASCII file, there can be defined exactly one number of X columns and exactly one number of Y columns. This means that the independent variables have to be of equal number for all the dependent variables– functions.

Imagine an experimental file that includes the values of three independent variables, i.e. some 3D grid coordinates, x_1, x_2, x_3, of two functions, say the temperature $T(x_1, x_2, x_3)$ and the density $n(x_1, x_2, x_3)$. Then the number of X columns is 3 and the number of dependent variables – Y columns – is 2.

10.7.4 Experimental data from ASCII files

Listing 6: XML structure to import experimental data from ASCII files

```
<?xml version="1.0" encoding="UTF-8"?>
<ExpFiles>


    <!-- define number of experimental data files -->
    <NumberExpFiles>1</NumberExpFiles>


    <!-- define import settings for 1st exp. data file -->
    <file>


        <!-- define path and name of experimental data file -->
        <FileNamesExpFiles>examples/TwoOscillators_RefFit_R.dat</FileNamesExpFiles>


        <!-- define import filter -->
        <ImportFilter>ascii</ImportFilter>


        <!-- define number of header lines -->
        <NumberHeaderLines>0</NumberHeaderLines>


        <!-- define character, which separate columns -->
        <SeparatorColumns> </SeparatorColumns>


        <!-- define number of X- and Y-columns -->
        <NumberColumnsX>1</NumberColumnsX>
        <NumberColumnsY>1</NumberColumnsY>


        <!-- are errors included? -->
        <ErrorY>no</ErrorY>


        <!-- define number and limits of ranges -->
        <NumberExpRanges>1</NumberExpRanges>
        <MinExpRange>50</MinExpRange>
        <MaxExpRange>1000</MaxExpRange>
    </file>
</ExpFiles>
```

▶ The tag `<NumberHeaderLines>` defines the number of header lines that must be ignored at import of an ASCII file.

▶ The user can specify a separator character (the tag `<SeparatorColumns>` defines the character that separates the columns from each other) for each file.

▶ For ASCII files, it may be necessary to specify the number of Y columns. This means that for a given X position in the experimental data, you can specify several Y values.

You measured several spectra under different polarization angles at the same frequency. A line in the corresponding ASCII file may look like:

```
100.12,  0.34134,  0.12341,  0.78901,  0.13361
```

Here, the first column describes the frequency and the other columns describe the transmission at different polarization angles. The number of X columns is 1 and the number of Y columns is 4.

▶ The tag `<NumberColumnsY>` defines the number of columns that belong to the Y points of the experimental data. The Y columns have to be next to the X values!

If the user wants to import an ASCII file that contains values of four Y points at every given X point, then the tag `<NumberColumnsY>` has to be set to 4.

▶ If the error tag `<ErrorY>` is set to `yes`, then the columns containing the errors have to be next to the Y columns. The number of these error columns have to be equal to the number of Y columns given in the tag `<NumberColumnsY>`.

Note, the error values are used for the calculation of the χ^2 value, see (§ 10.6.2).

Listing 7: Example of an ASCII file with 3 Y columns and the corresponding Y-errors (`ErrorY="YES"`)

```
       NumberColumnsX=2      NumberColumnsY=3      3 <ErrorY> columns
       100.2313  20.6578   0.5846  40.1  1.4218   0.020  0.451  0.017
       102.2463  21.7548   0.5947  60.3  1.5432   0.039  0.230  0.092
       140.5671  21.9998   0.3450  93.0  1.6725   0.091  0.561  0.005
```

10.7.5 Experimental data from FITS files

Listing 8: XML structure to import experimental data from FITS files

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ExpFiles>


    <!-- define number of experimental data files -->
    <NumberExpFiles>2</NumberExpFiles>


    <!-- define import settings for 1st exp. data file -->
    <file>


        <!-- define path and name of experimental data file -->
        <FileNamesExpFiles>one_parameter_free/File3.fits</FileNamesExpFiles>


        <!-- define import filter -->
```

```
        <ImportFilter>automatic</ImportFilter>


        <!-- define number of HDU -->
        <NumberHDU>0</NumberHDU>


        <!-- define number and limits of ranges -->
        <NumberExpRanges>1</NumberExpRanges>
        <MinExpRange>0</MinExpRange>
        <MaxExpRange>1000</MaxExpRange>
    </file>


    <!-- define import settings for 2nd exp. data file -->
    <file>


        <!-- define path and name of experimental data file -->
        <FileNamesExpFiles>one_parameter_free/File4.fits</FileNamesExpFiles>


        <!-- define import filter -->
        <ImportFilter>automatic</ImportFilter>


        <!-- define number of HDU -->
        <NumberHDU>0</NumberHDU>


        <!-- define number and limits of ranges -->
        <NumberExpRanges>2</NumberExpRanges>
        <MinExpRange>0</MinExpRange>
        <MaxExpRange>2000</MaxExpRange>
        <MinExpRange>3130</MinExpRange>
        <MaxExpRange>3200</MaxExpRange>
    </file>
</ExpFiles>
```

▶ For FITS files, the number of Y columns is always 1!

▶ Although the `<NumberColumnsX>` tag is ignored (if it exists) when importing a FITS file, the content of this tag is defined by the dimension of the FITS file. Thus, the ranges settings, namely the way that the beginning and ending of each range are specified, have to be given as in example § 10.7.3, if the dimension of the FITS file is > 1.

▶ The user has to specify the Header Data Unit (HDU) that should be loaded for each FITS file. This tag is needed only for FITS files.

▶ MAGIX distinguishes between image and table HDUs.

10.7.6 Experimental data and myXCLASS

Listing 9: XML structure to import experimental data from a CLASS file

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<ExpFiles>


    <!-- define number of experimental data file(s) -->
    <NumberExpFiles>1</NumberExpFiles>



    <!-- ****************************************************************** -->
    <!-- define parameters for first file -->
    <file>
        <FileNamesExpFiles>demo/myXCLASSFit/band1b.dat</FileNamesExpFiles>
        <ImportFilter>xclassASCII</ImportFilter>


        <!-- define number of frequency ranges for the current data file -->
        <NumberExpRanges>1</NumberExpRanges>


        <!-- define parameters for each frequency range -->
        <FrequencyRange>
            <MinExpRange>580102.0</MinExpRange>
            <MaxExpRange>580546.5</MaxExpRange>
            <StepFrequency>0.5</StepFrequency>


            <!-- define background temperature and temperature slope -->
            <t_back_flag>True</t_back_flag>
            <BackgroundTemperature>0.88</BackgroundTemperature>
            <TemperatureSlope>3.0</TemperatureSlope>


            <!-- define hydrogen column density, beta for dust, and kappa -->
            <HydrogenColumnDensity>3.e+24</HydrogenColumnDensity>
            <DustBeta>0.0</DustBeta>
            <Kappa>0.02</Kappa>
        </FrequencyRange>


        <!-- define local standard of rest (vLSR) -->
        <GlobalvLSR>0.0</GlobalvLSR>


        <!-- define size of telescope -->
        <TelescopeSize>3.5</TelescopeSize>


        <!-- define if interferrometric observation is modeled -->
        <Inter_Flag>False</Inter_Flag>


        <!-- define parameters for ASCII file import -->
        <ErrorY>no</ErrorY>
        <NumberHeaderLines>1</NumberHeaderLines>
        <SeparatorColumns> </SeparatorColumns>
    </file>


    <!-- parameters for isotopologues -->
    <iso_flag>True</iso_flag>
    <IsoTableFileName>demo/myXCLASS/iso_names.txt</IsoTableFileName>
```

```
    <!-- define path and name of database file -->
    <dbFilename>Database/cdms_sqlite__2016-06-15.db</dbFilename>
</ExpFiles>
```

In addition to the so-called general tags, the user has to define additional tags for the myXCLASS program, which are defined in the experimental xml-file:

▶ For each observational file the tag `<ImportFilter>` has to be always given and has to be set to `xclassASCII`, if the observational data file is an ASCII file or to `xclassFITS`, if the observational data file is a FITS file.

Please note, this tag has to be given for each observational data file!

▶ The number of ranges `<NumberExpRanges>` must be set ≥ 1, because myXCLASS requires the first `<MinExpRange>` and the last frequency `<MaxExpRange>` for re-sampling purposes (i.e. if you want to use all experimental data, then you still have to set a single range constrained by the first and last frequency).

▶ The step frequency of the simulated spectrum has to be given by the tag `<StepFrequency>`.

▶ The user can specify values for the hydrogen column density, the dust spectral index, and for kappa in two different ways: On the one hand, the user can define values which are used for all components of all molecules in the molfit file by using the tags `<HydrogenColumnDensity>`, `<DustBeta>` and `<Kappa>`. On the other hand, the user can define different values for each component in the molfit file. In addition, the tags `<BackgroundTemperature>` and `<TemperatureSlope>` describe for each frequency range the background temperature and for the temperature slope, respectively. The tag `<t_back_flag>` indicates if the user defined background temperature (described by `<BackgroundTemperature>`) and temperature slope (described by `<TemperatureSlope>`) describe the continuum contribution completely or not, see (§ 9).

▶ For each observation file the user has to specify the size of the telescope (> 0) by defining the tag `<TelescopeSize>` and the tag `<Inter_Flag>`, indicating if single dish or interferometric observations are described.

▶ Using the tag `<GlobalvLSR>` the user can defined different local standard of rest velocities (v_{LSR}) for each obs. data file. Thereby, the value defined by the input parameter `vLSR` is ignored.

▶ The tags `<ErrorY>`, `<NumberHeaderLines>`, and `<SeparatorColumns>` control the import of the ASCII file containing the observational data. Please note, these tags are read only if the observational data file is an ASCII file, i.e. that the tag `<ImportFilter>` is set to `xclassASCII`.

For further details please take a look at the previous section about the import of an ASCII file.

▶ In order to use isotopologues, the user has to set the tag `<iso_flag>` to `yes`, otherwise the tag has to be set to `no`.

▶ The tag `<IsoTableFileName>` has to be given only if the tag `<iso_flag>` is set to `yes`. The contents of the tag `<IsoTableFileName>` has to define the path and name of an ASCII file including the iso ratios between certain molecules, see (§ 9.3).

▶ The step size `<StepFrequency>`, the background temperature `<BackgroundTemperature>` and the temperature slope `<TemperatureSlope>` have to be given for each range.

▶ The value of the tag `<MinExpRange>` has to be smaller than the value of the tag `<MaxExpRange>`.

▶ By default XCLASS uses the SQLite3 database file `cdms_sqlite.db` located in the directory `path_to_XCLASS_interface/Database/`. In order to use a different database file, the user has to define the path and name of the other database file using the tag `<dbFilename>`.

10.8 MAGIX Output files

A series of files are created during a run of MAGIX:

10.8.1 Log files

MAGIX creates three different log files for each application of an algorithm referred to in the fit control file (see § 10.6):

▶ a "normal" log-file with ending ".`log`", which corresponds to the screen output. The log file contains for every iteration step the best χ^2 value and the corresponding values of the parameters that are being optimized.

▶ a file with ending ".`log.param`" including for every iteration step the best χ^2 value and the corresponding values of the parameters that are being optimized as well as all input files for the external model program. This allows the user to verify that MAGIX writes the parameters at the right positions.

▶ a file with ending ".`log.chi2`" including all χ^2 values and the corresponding free parameter values for all calls of the external model program starting with the smallest χ^2 value.

Note, if the user specifies only the path for the log-files, MAGIX creates the files `fit.log`, `fit.log.param`, and `fit.log.chi2`. Otherwise MAGIX creates a log file with filename specified in the I/O control file and extends the filename within ".`log.param`" and ".`log.chi2`". For example the user specifies `PathToYourFiles/mylogfile` in the I/O control file. Then MAGIX creates three different log-files:

▶ `PathToYourFiles/mylogfile.log`,

▶ `PathToYourFiles/mylogfile.log.param`, and

▶ `PathToYourFiles/mylogfile.log.chi2`.

The names of the log-files become more complicate, if the user applies a algorithm chain:

MAGIX extends the name of the log-files by an abbreviation for the algorithm (e.g. "LM" for Levenberg-Marquardt) and by "`__call_`" followed by the number of the call of the algorithm.

For example, the Levenberg-Marquardt algorithm is applied to the three best sites of a previous used Bees algorithm, then the names of the log-files (`fit.log` for simplification) are
`fit_LM__call_1.log, fit_LM__call_2.log, fit_LM__call_3.log,`
`fit_LM__call_1.log.param, fit_LM__call_2.log.param, fit_LM__call_3.log.param,`
`fit_LM__call_1.log.chi2, fit_LM__call_2.log.chi2, fit_LM__call_3.log.chi2.`

10.8.2 Files for fit function comparison and χ^2

For each experimental data file MAGIX creates two (three) additional output files in the directory where the experimental data files are located:

▶ *File of optimized parameter values:* After finishing a fit algorithm MAGIX writes all parameter values and the corresponding error values (if calculated) for the best fit to a file which has the same names as the instance file. Additionally, MAGIX extends the ending of the filename with an abbreviation for the algorithm (e.g. "`LM`" for Levenberg-Marquardt) followed by the phrase "`.out.xml`".

For example, the instance xml-file is named `parameter.xml`. MAGIX writes the optimized parameter values and the corresponding error values (if calculated) to the file `parameter.LM.out.xml`.

▶ *File of fit function values:* Additionally, MAGIX writes the values of the model function for each data point of the best fit to files which have the same names as the experimental data files. But, MAGIX extends the ending of the filenames with an abbreviation for the algorithm (e.g. "`LM`" for Levenberg-Marquardt) followed by the phrase "`.out.dat`" for ASCII and CLASS files and "`.out.fits`" for fits files.

For example, the name of the experimental data file is `datafile.dat`. MAGIX writes the values of the model function to the file `datafile.LM.out.dat`.

▶ *File with χ^2 values:* If the user sets the value of the tag `<SaveChi2>` to "`yes`", MAGIX writes the values of χ^2 for each data point of the best fit to further files. These files have the same names as the experimental data files except that MAGIX extends the ending of the filenames with an abbreviation for the algorithm (e.g. "`LM`" for Levenberg-Marquardt) followed by the phrase "`.out.chi2.dat`" for ASCII and CLASS files and "`.out.chi2.fits`" for fits files.

For example, the name of the experimental data file is `datafile.dat`. MAGIX writes the values of χ^2 for each data point to the file `datafile.LM.out.chi2.dat`.

▶ *Error Estimation:* If the user selects the error estimation algorithm MAGIX produces additional files for each experimental data file. Two of these files contain the model function for each data point where each free parameter is reduced (enhanced) by the corresponding lower (upper) error value. If the INS method is used, XCLASS creates two other files containing the corresponding χ^2 functions. The filenames corresponding to the reduced (lower) parameter values contain the phrase "`LowerErrorValues`" the enhanced contain "`UpperErrorValues`".

For example, the name of the experimental data file is `datafile.dat`. MAGIX writes the values for the reduced parameters to `datafile.ErrorEstim_INS__LowerErrorValues__call_1.out.dat`" and to `datafile.ErrorEstim_INS__LowerErrorValues__call_1.out.chi2.dat` whereas the enhanced parameters are written to `datafile.ErrorEstim_INS__UpperErrorValues__call_1.out.dat` and to `datafile.ErrorEstim_INS__UpperErrorValues__call_1.out.chi2.dat`.

Additionally, using the **INS** method XCLASS determines the χ^2 distribution for each free parameter. Here, XCLASS varies each parameter within the given range, whereas the other parameters are kept constant. Furthermore, XCLASS plots the χ^2 values as a function of the free parameter j and and saves the plot to a file named "`ErrorEstim_INS__chi2-distribution_of_free-parameter_parm_`" followed by an integer number indicating the free
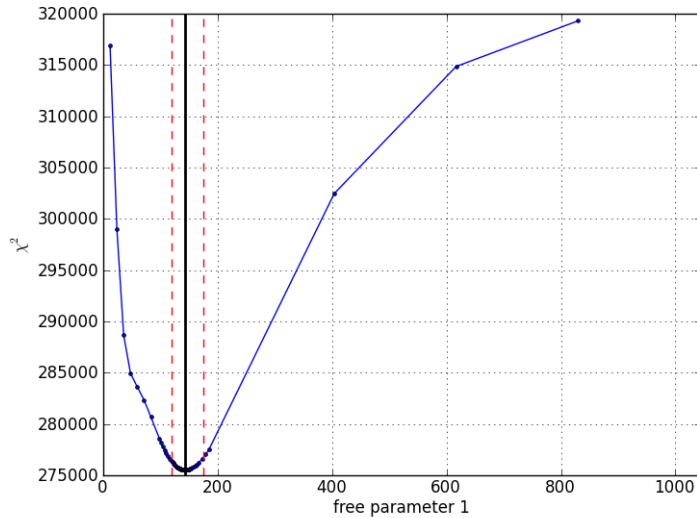
Figure 26: Example of a χ^2 distribution plot for the seventh free parameter. The solid vertical thick black line indicates the optimized parameter value and the two dashed vertical red lines describe the optimized parameter value reduced by the left and increased by the corresponding right errors, respectively.

parameter. Additionally, the plotted χ^2 values together with the corresponding parameter values are stored to an ASCII file having the same name as the corresponding png-file. For example, the file `ErrorEstim_INS__chi2-distribution_of_free-parameter_parm_1.png` contains the plot of the χ^2 distribution of the first free parameter and the ASCII file `ErrorEstim_INS__chi2-distribution_of_free-parameter_parm_1.dat` contains the data points plotted in the png file.

In addition each plot indicates the value of the parameter which corresponds to the best fit result by a solid vertical thick black line. The error range determined by the Error Estimation algorithm using INS method is marked with two dashed vertical red lines, see Fig. 26.

If the **MCMC** method is selected, XCLASS produces for each optimized parameter png-files describing the evolution of the parameter for each iteration step, similar to Fig. 14, the probability distribution for the whole parameter range and the so-called corner-plots, see Fig. 19.

If the **Fisher-matrix** method is used, XCLASS produces no further output files.

The names of the output files produced by MAGIX become more complicate, if an algorithm chain is used or if the number of best sites is set to a value greater one:

In addition to the extensions of the file names described above, MAGIX adds in case of an algorithm chain the phrase "`__call_`" followed by the number of the call of the algorithm as well.

For example, the Levenberg-Marquardt algorithm is applied to the three best sites of a previous used Bees algorithm, then the names of the instance xml-files are named `parameters.LM__call_1.out.xml`, `parameters.LM__call_2.out.xml`, and `parameters.LM__call_3.out.xml`.

In order to distinguish between different "best" sites, MAGIX adds in addition to the extensions described above the phrase "`__site_`" followed by the number of the best site as well.

For example, the user applies the Bees algorithm and sets the number of best sites to 3. The names of the instance xml-files are named `parameters.Bees__call_1__site_1.out.xml`, `parameters.Bees__call_1__site_2.out.xml`, and `parameters.Bees__call_1__site_3.out.xml`.

In cases of special FITS files that are images, the X column is declared in the header through the declarations of the first reference pixel (`CRPIX1`), its value (`CRVAL1`), and the distance between two pixels (`CDELT1`). If the user specifies more than one frequency range for a observational FITS file (e.g. `datafile.fits`) XCLASS creates different output FITS files, one for each frequency range. For example, the user defines two frequency ranges described by `LowFreq1`, `UpFreq1` and `LowFreq2`, `UpFreq2`, respectively and applies the Levenberg-Marquardt algorithm. After finishing the fitting procedure, XCLASS writes the calculated model function values and the corresponding χ^2-values to `datafile_LowFreq1_UpFreq1.LM.out.fits`, `datafile_LowFreq1_UpFreq1.LM.out.chi2.fits`, `datafile_LowFreq2_UpFreq2.LM.out.fits`, and `datafile _LowFreq2_UpFreq2.LM.out.chi2.fits`.

10.8.3   Plots

If the user do not select the `--noplot` option, MAGIX creates a plot containing the experimental data, the model function values and the χ^2 values for each data point. The plot is divided into two parts: The left side contains plots for each experimental data file where the observation data are plotted together with the model function for the best fit result. The right side contains plots for the corresponding χ^2 values for each data point. Finally, the plot is saved to a file where the name contains the phrase "`final_plot`", but become more complicate, if the user applies an algorithm chain or sets the number of best sites to a value greater 1, see the general description of the output file names above. For example, the user applies the Bees algorithm and sets the number of best sites to 3. The names of the plot files are named `final_plot.Bees__site_1.out.xml`, `final_plot.Bees__site_2.out.xml`, and `final_plot.Bees__site_3.out.xml`.

11 myXCLASSFit

This function provides a simplified interface for MAGIX using the myXCLASS program. The function starts MAGIX using the Levenberg-Marquardt algorithm (with four processors) to fit experimental data with the myXCLASS program. The user has to specify the max. number of iterations, the experimental data (or path and name of the experimental xml-file), and the path and name of the extended molfit file. The fit procedure stops, if the max. number of iterations is reached, or if χ^2 drops below 10^{-7}. (This χ^2-limit as well as the variation value of 10^{-3} are defined by the MYXCLASSFIT function. In order to use different settings, please use a MAGIX algorithm xml-file, see below.)

For each run the MYXCLASSFIT function creates a so-called job directory located in the run directory (`path-of-XCLASS-Interface/run/myXCLASSFit/`) where all files created by the MYXCLASS-FIT function are stored in. The name of this job directory is made up of four components: The first part consists of the phrase "`job_`" whereas the second and third part describe the date (day, month, year) and the time stamp (hours, minutes, seconds) of the function execution, respectively. The last part indicates a so-called job ID which is composed of the so-called PID followed by a four digit random integer number to create a really unambiguous job number, e.g. `path-of-XCLASS-Interface/run/myXCLASSFit/job__25-07-2013__12-02-03__189644/`.

Before the fit procedure starts, the function copies the molfit, the experimental data, and xml-files to the myXCLASSFit job directory. The path(s) defined in the experimental xml-file are adjusted so that these paths point to the current myXCLASSFit job directory.

Please note, that the original experimental xml- and data files are not modified. Only the copy of the experimental xml file located in the myXCLASSFit job directory is modified!

If no experimental xml-file is defined, the MYXCLASSFIT function creates an experimental xml-file containing the settings for the different parameters in the myXCLASSFit job directory. Additionally, the function creates an ASCII file located in the myXCLASSFit job directory containing the experimental data given by the parameter "experimentalData". If the parameter "experimentalData" defines the path and the name of an experimental data file, then the data file is copied to the myXCLASSFit job directory as well. The path and the name of this ASCII file is pasted in the created experimental xml file. Furthermore, the function creates automatically the algorithm and i/o control xml files, need by MAGIX, so that the user does not need to edit any xml-file.

Additionally, the MYXCLASSFIT function converts the column density N_{tot} (and if the hydrogen column density N_H is given for each component) the hydrogen column density N_H as well to a log scale, i.e. these two densities are converted automatically to their log10 values to get a better fit. At the end of the fitting process, the log10 values are converted back to the original linear values. So, the MAGIX log files contain the log10 values of these parameters, whereas the input and output molfit file contain the linear values, respectively.

Input parameters:

- ▶ `NumberIteration`: max. number of iterations (default: 50).

- ▶ `AlgorithmXMLFile`: only necessary, if the user wants to use another fit algorithm (than Levenberg-Marquardt) for fitting. Therefore, the path and name of a MAGIX xml-file defining settings for an algorithm or algorithm chain has to be given. (A relative path has to be defined relative to the current working directory!)

  NOTE, if the user specify a xml file, the number of iterations given by the parameter

`NumberIteration` is ignored. The number of iteration is then given by the xml file. In order to use the implemented fit algorithm (Levenberg-Marquardt) clear the AlgorithmXMLFile parameter, i.e. `AlgorithmXMLFile = ""`, and define the max. number of iterations by using parameter `NumberIteration`.

For a detailed description of the required xml-file, see (§ 10.6).

▶ `MolfitsFileName`: path and name of the extended molfit file, including the source size, the rotation temperature the column density, the velocity width, the velocity offset and the flag indicating if the current component is considered for core `c` or foreground `f`.

In contrast to the format of the molfit file described in (§ 9) the extended molfit file required by the MYXCLASSFIT function contains one (three) additional column(s) for each parameter of each component.

For a detailed description of the extended molfit file required by the MYXCLASSFIT function see (§ 10.4.2).

In order to fit the ratio(s) of isotopologues as well, the iso file (§ 9.3) has to include two additional columns as described in section (§ 9.3.1).

NOTE, a relative path has to be defined relative to the current working directory!

▶ `experimentalData`: This parameter offers two different possibility to send the experimental data to the MYXCLASSFIT function:

  – the parameter `experimentalData` defines the path and name of and experimental xml-file suitable for MAGIX. For a detailed description of the MAGIX xml-file, see (§ 10.7) and (§ 10.7.6).

  – the parameter `experimentalData` defines the path and name of an ASCII file called *observational data file*, where the first column describe the frequency (in MHz) and the second column the beam temperature (intensity).

NOTE, if the parameter `experimentalData` defines a relative path, the path has to be defined relative to the current working directory!

The following parameters are needed, if the parameter `experimentalData` does NOT describe the path and name of a MAGIX xml-file:

▶ `vLSR`: velocity (local standard of rest) in km s^{-1} (default: 0) used in the calculation of the synthetic spectra, see description for myXCLASS function (§ 9). Please note, for the myXCLASSFit function XCLASS uses the user-defined lower and upper limits of a velocity offset parameter in the molfit file, if this parameter is fitted within a MAGIX run.) Please note, using a xml-file, the `vLSR` parameter can be defined for each obs. data file, see (§ 10.7.6)!

▶ `TelescopeSize`: for single dish observations (`Inter_Flag = F`): `TelescopeSize` describes the size of telescope (in m), (default: 1); for interferometric observations (`Inter_Flag = T`): `TelescopeSize` describes the interferometric beam FWHM size (in arcsec), (default: 1).

▶ `Inter_Flag` (`T`/`F`): defines, if single dish (`F`) or interferometric observations (`T`) are described, (default: `F`).

▶ `t_back_flag` (T/F): defines, if the user defined background temperature T_{bg} and temperature slope T_{slope} given by the input parameters `tBack` and `tslope` describe the continuum contribution completely (`t_back_flag = T`) or not (`t_back_flag = F`) (default: T).

▶ `tBack` background temperature (in K), (default: 0).

▶ `tslope` temperature slope (dimensionless), (default: 0).

▶ `nH_flag`: (T/F), defines, if column density, spectral index for dust and kappa are given by the molfit file (F) or, if `nH_flag` is set to T, the following three parameters define the H column density, spectral index for dust and kappa for all components (default: F):

▶ `N_H`: (has to be given only if `nH_flag` is set to T) Hydrogen column density (in cm^{-2}), (default: 0).

▶ `beta_dust`: (has to be given only if `nH_flag` is set to T) spectral index for dust (dimensionless), (default: 0).

▶ `kappa_1300`: (has to be given only if `nH_flag` is set to T) kappa ($cm^2 \, g^{-1}$), (default: 0.01).

▶ `iso_flag`: use isotopologues (T/F). If `iso_flag` is set to T isotopologues defined in the iso ratio file are used (default: F).

▶ `IsoTableFileName` (has to be given only if `iso_flag` is set to T): path and name of an ASCII file including the iso ratios between certain molecules. A detailed description of the iso ratio file is given in (§ 9.3). If no file name is given (default), the so-called iso-flag is set to F.

NOTE, if the parameter `experimentalData` defines a relative path, the path has to be defined relative to the current working directory!

The following parameter is needed to add a velocity axis to the output array `model_values`, see below:

▶ `RestFreq`: rest frequency in MHz (default: 0). (If this parameter is set to zero, the intensity is plotted against frequency (in MHz) otherwise against velocity (in km s^{-1}), see description for myXCLASS function (§ 9)

The settings for mini. and max. frequency as well as for the step size is taken from the experimental data.

Output parameters:

▶ `input_file`: the contents of the molfit file containing the parameters for the best fit.

▶ `model_values`: the model function values for each data point, which correspond to the best fit.

If `RestFreq` is unequal zero, the MYXCLASSFIT function adds a column to the output parameter `model_values` which contains the velocities. So, for a rest frequency unequal zero, the parameter `model_values` represents a python array with three columns, where the first column describes the frequencies, the second column describes the velocities and the third column the corresponding intensities.

▶ `JobDir`: absolute path of the job directory created for the current run.

Note, the user is free to define different names for the output parameters.

Please note, if MAGIX produces more than one result, the output parameters `input_file` and `model_values` contain all input files and all model function values of all results. For example, `input_file[0]` contains the molfit file and `model_values[0]` the model function values for the first result.

The order of the results, i.e. the name of the molfit file, which correspond to `input_file[0]`, is printed to the screen.

Example with MAGIX xml files (using a molfit file in old format):

```
NumberIteration = 10
MolfitsFileName = "demo/myXCLASSFit/CH3OH__old.molfit"
experimentalData = "demo/myXCLASSFit/observation.xml"
RestFreq = 0.0
vLSR = 0.0
newmolfit, modeldata, JobDir = myXCLASSFit()
```

Example with MAGIX xml files (using a molfit file in new format:

```
NumberIteration = 10
MolfitsFileName = "demo/myXCLASSFit/CH3OH__new.molfit"
experimentalData = "demo/myXCLASSFit/observation.xml"
RestFreq = 0.0
vLSR = 0.0
newmolfit, modeldata, JobDir = myXCLASSFit()
```

Example using a molfit file in the old format and an ASCII file containing the experimental data:

```
NumberIteration = 10
MolfitsFileName = "demo/myXCLASSFit/CH3OH__old.molfit"
experimentalData = "demo/myXCLASSFit/band1b.dat"
TelescopeSize = 3.5
Inter_Flag = F
t_back_flag = T
tBack = 1.1
tslope = 0.0000000000E+00
nH_flag = T
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
iso_flag = F
IsoTableFileName = ""
RestFreq = 0.0
vLSR = 0.0
newmolfit, modeldata, JobDir = myXCLASSFit()
```

Example with MAGIX xml files (using another fit algorithm):

```
MolfitsFileName = "demo/myXCLASSFit/CH3OH__new.molfit"
experimentalData = "demo/myXCLASSFit/observation.xml"
AlgorithmXMLFile = "demo/myXCLASSFit/algorithm-settings.xml"
RestFreq = 0.0
vLSR = 0.0
newmolfit, modeldata, JobDir = myXCLASSFit()
```

Usage without CASA:

```
# extend sys.path variable
...



# import task_myXCLASSFit package
import task_myXCLASSFit



# call myXCLASSFit function
MolfitsFileName = "demo/myXCLASSFit/CH3OH__new.molfit"
experimentalData = "demo/myXCLASSFit/observation.xml"
AlgorithmXMLFile = "demo/myXCLASSFit/algorithm-settings.xml"
RestFreq = 0.0
vLSR = 0.0
newmolfit, modeldata, JobDir = task_myXCLASSFit.myXCLASSFit(\
                                 NumberIteration, AlgorithmXMLFile, \
                                 MolfitsFileName, experimentalData, \
                                 TelescopeSize, Inter_Flag, t_back_flag, \
                                 tBack, tslope, nH_flag, N_H, beta_dust, \
                                 kappa_1300, iso_flag, IsoTableFileName, \
                                 RestFreq, vLSR)
```

Please note, if the Error Estimation algorithm is applied the myXCLASSFit function creates
further molfit file(s), with ending `__error.molfit` corresponding to the other molfit file(s). These
molfit file(s) contain(s) the fixed and optimized parameters for each component and molecule.
Additionally, the molfit files include five further columns for each parameter describing the left
and right error, and (if INS method was used) the mean value, the standard deviation and the
logarithm of evidence. For fixed (non-optimized) parameters, the new columns are identical to
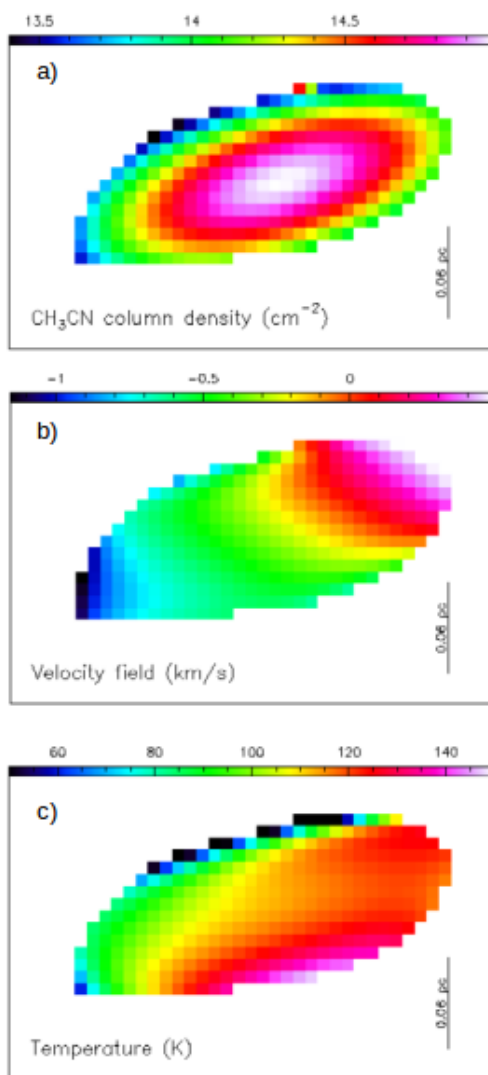zero.

Figure 27: Example of parameter maps created by the myXCLASSMapFit function. In a) each pixel corresponds to the column density of the best fit for the corresponding pixel. Here, only pixels are fitted with intensities above a given threshold. b) and c) shows the fitted velocity offsets and temperature respectively.

12 myXCLASSMapFit

The function starts MAGIX using the Levenberg-Marquardt algorithm to fit experimental data with the myXCLASS program. Instead of the myXCLASSFit function the MYXCLASSMAP-FIT function fits a complete data cube. So, the MYXCLASSMAPFIT function reads in the data cube(s), extracts the spectra for each pixel and, fits these spectra, separately using the Levenberg-Marquardt (or another) algorithm. The fit procedure for each pixel stops, if the max. number of iterations is reached, or if χ^2 drops below 10^{-7}. (This χ^2-limit as well as the variation value of 10^{-3} are defined by the MYXCLASSMAPFIT function. In order to use different settings or another algorithm (chain), please use a MAGIX algorithm xml-file, see below.)

Please note, the FITS data cube has to be given in the World Coordinate System (WCS) described in {Greisen *et al.* 2002, Calabretta *et al.* 2002, Greisen *et al.* 2006}.

For each run the MYXCLASSMAPFIT function creates a so-called job directory "job-directory" located in the myXCLASSMapFit working directory `path-of-XCLASS-Interface/run/myXCLASSMapFit/`! The name of a job directory is made up of four components: The first part of the name consists of the phrase "`job_`" whereas the second part describes the date (day, month, year), the third part the time stamp (hours, minutes, seconds) of the function execution. The last part describes a so-called job ID which is composed of the so-called PID followed by a four digit random integer number to create a really unambiguous job number, e.g.`ath-of-XCLASS-Interface/run/myXCLASSMapFit/job__25-07-2013__12-02-03__189644932/`.

Additionally, the MYXCLASSMAPFIT function creates a subdirectory within this job directory called "Pixel-Fits/" which contains further subdirectories for each pixel, where the names of these subdirectories are made up of the values (in degrees) for the right ascension and for the declination, respectively, e.g. "`83.65498__-5.24258`". Additionally, the indices of the corresponding pixel are added as well separated by "`_-_`", e.g. "`83.65498__-5.24258__1_-_2`". All output files, created by MAGIX for this pixel are stored in this subdirectory. If the user applies the Levenberg-Marquardt algorithm, the MYXCLASSMAPFIT function uses the parameter given in the molfit file as initial guess for all pixels in the first row of the selected region. In order to speed up the fit procedure, the myXCLASSMapFit function uses for the following rows, the result of the fit for the pixel of the previous row (and the same column) as initial guess for the current pixel.

At the end of the whole fit procedure, the MYXCLASSMAPFIT function creates FITS images for each optimized parameter, where each pixel corresponds to the value of the optimized parameter taken from the best fit for this pixel. The name of each parameter FITS image consists of the phrase "BestResult" followed by the name of the parameter, e.g. `T_rot` for the rotation temperature etc, see (§ 9.2), the name of the corresponding molecule, and finally of the index of the related component. For example, the FITS file
`BestResult___parameter__T_rot___molecule__CH3OH_v=0____component__1.fits`, contains the optimized excitation (rotation) temperature of the first component of molecule "$CH_3OH,v=0$" for each pixel of the selected map.

Furthermore, the MYXCLASSMAPFIT function creates FITS cubes for each used algorithm and fitted data cube, where each pixel contains the modeled spectrum. Additionally, the MYXCLASSMAPFIT function creates one FITS image, where each pixel corresponds to the χ^2 value of the best fit for each pixel.

In addition, the MYXCLASSMAPFIT function converts the column density N_{tot} (and if the hydrogen column density N_H is given for each component) the hydrogen column density N_H as well to a log scale, i.e. these two densities are converted automatically to their log10 values to get a better fit. At the end of the fitting process, the log10 values are converted back to the original linear values. So, the MAGIX log files contain the log10 values of these parameters, whereas the input and output molfit files contain the linear values, respectively.

Note, the MYXCLASSMAPFIT function offers the possibility to fit more than one pixel of a line of a selected map at once using a cluster consisting of at least two computers. The nodes (computers) of the cluster are defined by the input parameter `clusterdef`, see below. For example, the user wants to fit a map of 10 times 10 pixel using a cluster with two nodes (computers) where each node offers 8 cores, respectively. The corresponding molfit file contains two free parameters (e.g. `T_rot` and `V_off`). Applying the Levenberg-Marquardt algorithm we need three cores for each pixel fit. Therefore, we can fit 5 pixel ($5 \times 3 = 15$ cores) in parallel, because the cluster offers ($2 \times 8 =$) 16 cores in total.

Input parameters:

- ▶ `NumberIteration`: max. number of iterations (default: 50).

- ▶ `AlgorithmXMLFile`: only necessary, if the user wants to use another fit algorithm (than Levenberg-Marquardt) for fitting. Therefore, the path and name of a MAGIX xml-file describing settings for an algorithm or algorithm chain has to be given. (A relative path has to be defined relative to the current working directory!)

  NOTE, if the user specify a xml file, the number of iterations given by the parameter "NumberIteration" is ignored. The number of iteration is then given by the xml file. In order to use the implemented fit algorithm clear the AlgorithmXMLFile parameter, i.e. AlgorithmXMLFile = "", and define the max. number of iterations using parameter "NumberIteration".

  For a detailed description of the required xml-file, see (§ 10.6).

- ▶ `MolfitsFileName`: path and name of the extended molfit file, including the source size, the rotation temperature the column density, the velocity width, the velocity offset and the flag indicating if the current component is considered for core `c` or foreground `f`.

  In contrast to the format of the molfit file described in (§ 9) the extended molfit file required by the MYXCLASSMAPFIT function contains one (three) additional column(s) for each parameter of each component.

  For a detailed description of the extended molfit file required by the MYXCLASSMAPFIT function see (§ 10.4.2).

  NOTE, if the given path is a relative path, i.e. the path does not start with `/`, this path has to be defined relative to the current working directory!

- ▶ `experimentalData`: This parameter offers two different possibility to send the experimental data to the myXCLASSMapFit function:

  – the parameter `experimentalData` defines the path and name of and experimental xml-file suitable for MAGIX. Here, the xml-file contains the path and the name of each FITS or CASA image (IMAGE) file/directory given by the tag `<FileNamesExpFiles>`.

    NOTE, the names of a FITS file has to end with `.fits` whereas the name of a CASA image file/directory has to end with `.image`!! If XCLASS is used without CASA, CASA image files/directories can not be imported!!

    Please note, if more than one data cube is specified in the xml file, the data cubes must describe the same map, i.e. the right ascension and the declination has to be identical! The data cubes has to differ only in the frequency/velocity axis. So, it is possible to specify different units for the frequency axis for different data cubes.

  – the parameter `experimentalData` defines the path and name of a FITS or CASA image (IMAGE) containing the data cube. (Please note, without using a MAGIX xml file, it is not possible to fit more than one data cube, simultaneously.)

    NOTE, if the parameter experimentalData defines a relative path, the path has to be defined relative to the current working directory!

Please note, the MYXCLASSMAPFIT function assumes, that the first three axes of the data cube(s) describe the right ascension, the declination, and the frequency, respectively. If the frequencies are not given in MHz, the FITS header has to contain the CUNIT3 command word which defines the unit of the frequency axis. For velocities, the header has to contain the command word RESTFRQ as well.

NOTE, if the parameter `experimentalData` defines a relative path, the path has to be defined relative to the current working directory!

▶ `regionFileName`: the so-called region file (defined by the ds9 program or the CASA viewer). At the moment the shape of the region has to be rectangular. (Other formats and other shapes will be available soon).

▶ `UsePreviousResults`: (T/F) defines if the molfit file (described by the parameter `MolfitsFileName`) is used as initial guess for all pixels (F) or if the result for a previous pixel is used.

Please note, that this parameter is valid only for so-called local optimization algorithm like Levenberg-Marquardt or Simulated annealing.

▶ `Threshold`: defines a threshold for a pixel. If the spectrum of a pixel has an max. intensity lower than the value defined by this parameter the pixel is not fitted (ignored).

Please note, the value for the `Threshold` parameter has to be given in the same unit than the spectrum in the FITS file.

NOTE, if parameter `experimentalData` defines the path and name of an experimental xml-file suitable for MAGIX the user can define different threshold values for each frequency range by using the tag `<Threshold>`. Please note, that the tag `<Threshold>` has to be given for each frequency range. If tag `<Threshold>` is defined in the xml file, the value of the input parameter `Threshold` is ignored.

The following parameters are needed, if the parameter `experimentalData` does NOT describe the path and name of a MAGIX xml-file:

▶ `FreqMin`: start frequency of simulated spectrum (default: 0).

Please note, if no start frequency is given, or if start frequency is lower or equal to the end frequency, the MYXCLASSMAPFIT function will use the max. frequency range defined in the FITS header. Additionally, the step size, i.e. the difference between two frequencies is taken from the FITS header.

▶ `FreqMax`: end frequency of simulated spectrum (default: 0).

▶ `vLSR`: velocity (local standard of rest) in km s^{-1} (default: 0) used in the calculation of the synthetic spectra, see description for myXCLASS function (§ 9). Please note, for the myXCLASSMapFit function, XCLASS uses the user-defined lower and upper limits of a velocity offset parameter in the molfit file, if this parameter is fitted within a MAGIX run.) Please note, using a xml-file, the `vLSR` parameter can be defined for each obs. data file, see (§ 10.7.6)!

▶ `TelescopeSize`: for single dish observations (`Inter_Flag = F`): `TelescopeSize` describes the size of telescope (in m), (default: 1); for interferometric observations (`Inter_Flag = T`): `TelescopeSize` describes the interferometric beam FWHM size (in arcsec), (default: 1).

▶ `Inter_Flag` (T/F): defines, if single dish (F) or interferometric observations (T) are described, (default: F).

- ▶ `t_back_flag` (`T`/`F`): defines, if the user defined background temperature T_{bg} and temperature slope T_{slope} given by the input parameters `tBack` and `tslope` describe the continuum contribution completely (`t_back_flag = T`) or not (`t_back_flag = F`) (default: `T`).

- ▶ `tBack` background temperature (in K), (default: 0).

- ▶ `tslope` temperature slope (dimensionless), (default: 0).

- ▶ `nH_flag`: (`T`/`F`), defines, if column density, spectral index for dust and kappa are given by the molfit file (`F`) or if `nH_flag` is set to `T`, the following three parameters define the hydrogen column density, spectral index for dust and kappa for all components (default: `F`):

- ▶ `N_H`: (has to be given only if `nH_flag` is set to `T`) Hydrogen column density (default: 0).

- ▶ `beta_dust`: (has to be given only if `nH_flag` is set to `T`) spectral index for dust (default: 0).

- ▶ `kappa_1300`: (has to be given only if `nH_flag` is set to `T`) kappa (default: 0.01).

- ▶ `iso_flag`: use isotopologues (`T`/`F`). If `iso_flag` is set to `T` isotopologues defined in the iso ratio file are used (default: `F`).

- ▶ `IsoTableFileName` (has to be given only if `iso_flag` is set to `T`): path and name of an ASCII file including the iso ratios between certain molecules. A detailed description of the iso ratio file is given in (§ 9.3). If no file name is given (default), the so-called iso-flag is set to `F`.

  NOTE, if the given path is a relative path, i.e. the path does not start with `/`, this path has to be defined relative to the current working directory!

The following parameter is needed, to add a velocity axis:

- ▶ `RestFreq`: rest frequency in MHz (default: 0). (If this parameter is set to zero, the intensity is plotted against frequency (in MHz) otherwise against velocity (in km s^{-1}).

Finally, the last parameter is needed to start the MYXCLASSMAPFIT function on a cluster

- ▶ `clusterdef`: path and name of a file containing information for the cluster as described in the CASA cookbook[26] (subsection "10.3 Parallelization control"). In contrast to CASA's "cluster configuration file", the MYXCLASSMAPFIT function requires only two entries for each node: the HOSTNAME of the target node where the cluster is deployed and the number of CORES (engines) separated by comma. (Comments are marked with the "#" character.) Please note, the third column can be used to define the positions of the XCLASS directory on each computer in the cluster. This is essential, if the cluster contains computers with different operating systems, e.g. Linux and MAC OS. If no path is defined, the MYXCLASSMAPFIT function assumes that the XCLASS directory is visible by all nodes in the cluster and mounted in the same path of the file-system.

  Note, a "cluster configuration file" described in the CASA cookbook can be used as well whereat the definitions for the work directories are interpreted as definition for the XCLASS directory on the different nodes of the cluster. Additionally, the definitions of RAM usage and RAM per engine are ignored.

The following requirements are necessary for all nodes which are included in the cluster:

---

[26]http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf

1. Password-less ssh access from the controller (user) machine to all other nodes included in the cluster.
   NOTE: This is not necessary when using only "localhost", i.e. if the cluster is deployed only on the machine where CASAPY is running.

2. XCLASS interface, CASA and gfortran (with OpenMP/OpenMPI) are installed on all nodes of the cluster.

3. The XCLASS job directory must be located in a shared file-system, accessible from all nodes comprising the cluster, and mounted in the same path of the file-system.
   NOTE: If the cluster contains computers with different operating systems, e.g. Linux and MAC OS, the user has to define the different paths of the XCLASS interface directories.

Example of a "cluster configuration file" used for the MYXCLASSMAPFIT function:

```
# cluster configuration file
# node       number of cores       path of XCLASS interface
anu,         2
lugal,       3
```

In the example described above, the nodes "anu" and "lugal" will be used with two (three) cores, respectively.

NOTE: The total number of cores used on a node of the cluster is determined by the number of processors defined in the algorithm xml file times the number of cores defined in the "cluster configuration file". For example, the user wants to apply the Levenberg-Marquardt algorithm with eight processors on a cluster defined in the example above. On node "anu" 16 (2×8) cores and on node "lugal" 18 (3×8) cores are used.

Output parameters:

▶ `JobDir`: absolute path of the job directory created for the current run.

Example using a molfit file in the old format:

```
NumberIteration = 10
MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
AlgorithmXMLFile = ""
experimentalData = "demo/myXCLASSMapFit/Orion.methanol.cbc.contsub.ms.fits"
regionFileName = "demo/myXCLASSMapFit/region__box__ds9.reg"
TelescopeSize = 3.5
Inter_Flag = F
t_back_flag = T
tBack = 0.5000000000E-01
tslope = 0.0000000000E+00
nH_flag = T
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
iso_flag = T
IsoTableFileName = "demo/myXCLASSMapFit/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
clusterdef = ""
JobDir = myXCLASSMapFit()
```

Example using a algorithm chain:

```
NumberIteration = 20
MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
AlgorithmXMLFile = "demo/myXCLASSMapFit/algorithm-settings.xml"
experimentalData = "demo/myXCLASSMapFit/Orion.methanol.cbc.contsub.ms.fits"
regionFileName = "demo/myXCLASSMapFit/region__box__ds9.reg"
TelescopeSize = 3.5
Inter_Flag = F
t_back_flag = T
tBack = 0.5000000000E-01
tslope = 0.0000000000E+00
nH_flag = T
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
iso_flag = T
IsoTableFileName = "demo/myXCLASSMapFit/iso_names.txt"
RestFreq = 0.0
vLSR = 0.0
clusterdef = ""
JobDir = myXCLASSMapFit()
```

Example with a MAGIX xml files (using a old molfit format):

```
MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
experimentalData = "demo/myXCLASSMapFit/Orion_observation__myXCLASSMapFit.xml"
regionFileName = "demo/myXCLASSMapFit/region__box__ds9.reg"
AlgorithmXMLFile = ""
RestFreq = 0.0
vLSR = 0.0
clusterdef = ""
JobDir = myXCLASSMapFit()
```

Usage without CASA:

```
# extend sys.path variable
...


# import task_myXCLASSMapFit package
import task_myXCLASSMapFit


# call myXCLASSMapFit function
MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
experimentalData = "demo/myXCLASSMapFit/Orion_observation__myXCLASSMapFit.xml"
regionFileName = "demo/myXCLASSMapFit/region__box__ds9.reg"
AlgorithmXMLFile = ""
RestFreq = 0.0
vLSR = 0.0
clusterdef = ""
JobDir = task_myXCLASSMapFit.myXCLASSMapFit(NumberIteration, \
                                           AlgorithmXMLFile, MolfitsFileName, \
                                           experimentalData, regionFileName, \
                                           UsePreviousResults, \
                                           Threshold, FreqMin, FreqMax, \
                                           TelescopeSize, Inter_Flag, \
                                           t_back_flag, tBack, tslope, \
                                           nH_flag, N_H, beta_dust, \
                                           kappa_1300, iso_flag, \
                                           IsoTableFileName, RestFreq, \
                                           vLSR, clusterdef)
```

13 myXCLASSMapRedoFit

This function offers the possibility to redo one or more so called pixel fits of a previous MYXCLASSMAPFIT run. The function performs fits for the selected pixels and recreates the different parameter maps using the new optimized parameter values. The names of these maps are created in the same way as described in (§ 12). In addition the MYXCLASSMAPREDOFIT function adds the phrase "`__RedoFit`" at the end of the file names of the new parameter maps. The user has to define the pixel coordinates, which should be re-fitted, the job number of the previous MYXCLASSMAPFIT run, and the new molfit file. All files created by the fit of the previous MYXCLASSMAPFIT run are moved to a new subdirectory within the selected pixel directory. The name of this new subdirectory is made up by the phrase "`backup_for_RedoFit__`" followed by the current date and time stamp.

Input parameters:

- ▶ `JobNumber`: job number of a previous MYXCLASSMAPFIT run which should be improved.

- ▶ `PixelList`: list of all pixel coordinates which should be re-fitted. (These pixel coordinates are used to identify the pixel directories created by the selected MYXCLASSMAPFIT call, see (§ 12).) Each coordinate consists of two numbers separated by a comma, where the first number corresponds to the right ascension and second to the declination. Different pixel coordinates are enclosed by squared brackets and separated by commas. For example, we want to refit the pixels 83.2013422325, -15.1345324232 and 83.1111111111, -15.2211111111. The `PixelList` parameter has to be defined as follow

    ```
    PixelList = [[83.2013422325,  -15.1345324232],
                 [83.1111111111,  -15.2211111111]]
    ```

    Please note, that these coordinates are used to identify the pixel directories in the selected MYXCLASSMAPFIT job directory, i.e. the values defined in the `PixelList` parameter have to be included in the names of the corresponding pixel directories. For example, we want to refit the pixel with pixel directory `83.2013422325__-15.1345324232__1_-_3`. In order to identify this pixel the `PixelList` parameter can be defined as

    ```
    PixelList = [[83.2013422325,  -15.1345324232]]
    ```

    Additionally, the MYXCLASSMAPREDOFIT function offers the possibility to define the indices of the pixels which should be refitted. In the example described above, the `PixelList` parameter can be defined as

    ```
    PixelList = [[1, 3]]
    ```

    to identify the pixel 83.2013422325, -15.1345324232.

- ▶ `NumberIteration`: max. number of iterations (default: 50).

- ▶ `AlgorithmXMLFile`: only necessary, if the user wants to use another fit algorithm (than Levenberg-Marquardt) for fitting. Therefore, the path and name of a MAGIX xml-file describing settings for an algorithm or algorithm chain has to be given. (A relative path has to be defined relative to the current working directory!)

    NOTE, if the user specify a xml file, the number of iterations given by the parameter `NumberIteration` is ignored. The number of iteration is then given by the xml file. In order to use the implemented fit algorithm clear the AlgorithmXMLFile parameter,

i.e. `AlgorithmXMLFile = ""`, and define the max. number of iterations using parameter `NumberIteration`.

For a detailed description of the required xml-file, see (§ 10.6).

▶ `MolfitsFileName`: path and name of the extended molfit file, including the source size, the rotation temperature the column density, the velocity width, the velocity offset and the flag indicating if the current component is considered for core `c` or foreground `f`.

In contrast to the format of the molfit file described in (§ 9) the extended molfit file required by the MYXCLASSMAPFIT and MYXCLASSMAPREDOFIT function contains one (three) additional column(s) for each parameter of each component. For a detailed description of the extended molfit file required by the MYXCLASSMAPFIT function see (§ 10.4.2).

Please note, if the molfit file contains a free parameter which was not optimized in the previous MYXCLASSMAPFIT run, the MYXCLASSMAPREDOFIT function will create a new parameter map for this new free parameter at the end of the fitting process containing the optimized parameter values for the selected pixels only.

NOTE, if the given path is a relative path, i.e. the path does not start with `/`, this path has to be defined relative to the current working directory!

▶ `experimentalData`: This parameter offers the possibility to use a new experimental xml-file suitable for MAGIX for the selected pixel(s), where new settings for each frequency range (e.g. background temperature and slope) can be defined. For that purpose the parameter `experimentalData` has to define the path and name of the new xml file.

Please note, the MYXCLASSMAPREDOFIT function does not consider the tags `<FileNamesExpFiles>` and `<ImportFilter>` because these tags has been defined by the previous MYXCLASSMAPFIT run.

▶ `Threshold`: defines a threshold for a pixel (default: 0). If the spectrum of a pixel has a max. intensity lower than the value defined by this parameter the pixel is not fitted (ignored).

Please note, the value for the `Threshold` parameter has to be given in Kelvin.

NOTE, if the parameter `experimentalData` defines the path and name of an experimental xml-file suitable for MAGIX the user can define different threshold values for each frequency range by using the tag `<Threshold>`. Please note, that the tag `<Threshold>` has to be given for each frequency range. If tag `<Threshold>` is defined in the xml file, the value of the input parameter `Threshold` is ignored.

The following parameters can be used to update the settings of the already existing experimental xml-files which were used in the previous MYXCLASSMAPFIT run: (Please note, in order to use the following parameters the parameter `experimentalData` must not be defined, i.e. `experimentalData = ""`.)

▶ `FreqMin`: new start frequency of simulated spectrum (default: 0).

▶ `FreqMax`: new end frequency of simulated spectrum (default: 0).

▶ `t_back_flag` (`T`/`F`): defines, if the user defined background temperature T_{bg} and temperature slope T_{slope} given by the input parameters `tBack` and `tslope` describe the continuum contribution completely (`t_back_flag = T`) or not (`t_back_flag = F`) (default: `T`).

▶ `tBack` background temperature (in K), (default: 0).

- ▶ `tslope` temperature slope (dimensionless), (default: 0).

- ▶ `N_H`: (has to be given only if `nH_flag` is set to `T`) Hydrogen column density (default: 0).

- ▶ `beta_dust`: (has to be given only if `nH_flag` is set to `T`) spectral index for dust (default: 0).

- ▶ `kappa_1300`: (has to be given only if `nH_flag` is set to `T`) kappa (default: 0.01).

- ▶ `iso_flag`: use isotopologues (`T/F`). If `iso_flag` is set to `T` isotopologues defined in the iso ratio file are used (default: `F`).

- ▶ `IsoTableFileName` (has to be given only if `iso_flag` is set to `T`): path and name of an ASCII file including the iso ratios between certain molecules. A detailed description of the iso ratio file is given in (§ 9.3). If no file name is given (default), the so-called iso-flag is set to `F`.

  NOTE, if the given path is a relative path, i.e. the path does not start with `/`, this path has to be defined relative to the current working directory!

Output parameters:

- ▶ `None`

Example:

In this example, the MYXCLASSMAPREDOFIT function will re-fit the pixels (83.201, −15.1345) and (83.111, −15.221) of the previous MYXCLASSMAPFIT run with job number 1234 using the molfit file stored at "demo/myXCLASSMapFit/CH3OH.molfit".

```
JobNumber = 1234
PixelList = [[83.201, -15.1345], [83.111, -15.221]]
NumberIteration = 10
AlgorithmXMLFile = ""
MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
experimentalData = ""
Threshold = 0.0
FreqMin = 0.0
FreqMax = 0.0
t_back_flag = T
tBack = 9.5000000000E-01
tslope = 0.0000000000E+00
N_H = 3.0000000000E+24
beta_dust = 2.0
kappa_1300 = 0.02
iso_flag = T
IsoTableFileName = "demo/myXCLASSMapFit/iso_names.txt"
myXCLASSMapRedoFit()
```

Usage without CASA:

```
# extend sys.path variable
...


# import task_myXCLASSMapRedoFit package
import task_myXCLASSMapRedoFit


# call myXCLASSMapRedoFit function
JobNumber = 1234
```

```
    PixelList = [[83.201, -15.1345], [83.111, -15.221]]
    NumberIteration = 10
    AlgorithmXMLFile = ""
    MolfitsFileName = "demo/myXCLASSMapFit/CH3OH.molfit"
    experimentalData = ""
    Threshold = 0.0
    FreqMin = 0.0
    FreqMax = 0.0
    t_back_flag = True
    tBack = 9.5000000000E-01
    tslope = 0.0000000000E+00
    N_H = 3.0000000000E+24
    beta_dust = 2.0
    kappa_1300 = 0.02
    iso_flag = True
    IsoTableFileName = "demo/myXCLASSMapFit/iso_names.txt"
    task_myXCLASSMapRedoFit.myXCLASSMapRedoFit(JobNumber, PixelList, \
                                        NumberIteration, \
                                        AlgorithmXMLFile, \
                                        MolfitsFileName, \
                                        experimentalData, Threshold, \
                                        FreqMin, FreqMax, t_back_flag, \
                                        tBack, tslope, N_H, beta_dust, \
                                        kappa_1300, iso_flag, \
                                        IsoTableFileName)
```

Please note, logical variables in python has to be defined with "True" and "False" instead of "T" and "F" in CASA.

14 LineIdentification

This function identifies molecules in a given spectrum/spectra.

In general, the LineIdentification function reads in all molecule names from the database which have transitions within a defined frequency range and stores the names of these molecules in a so-called *molecule list*. In order to determine the contribution of each molecule, the LineIdentification function performs so-called *single molecule fits* for each molecule. If a molecule covers a defined fraction of the spectrum the molecule is "for now" identified and the optimized molfit file is append to a so-called *overall molfit file* which describes the contribution of all identified molecules. After all single molecule fits are done, the LineIdentification function performs a *final fit*, using the overall molfit file created before.

If an iso ratio file is defined, the LineIdentification function determines the isotopologues of all molecules in the molecule list using the given iso ratio file. The identified isotopologues are removed from the list of molecules for whom a single molecule fit has to be done. All single molecule fits are done without using isotopologues. The user defined iso ratio file is used again for the final overall fit, where all iso-molecules and their definitions, which were not identified, are removed. Please note, the XCLASS package offers the possibility to optimize iso ratios as well, see (§ 9.3).

The LineIdentification function offers different possibilities to control the single molecule fits, e.g. by using a so-called *default molfit file* or a so-called *source molfit file*, see below. The user is free to use the Levenberg-Marquardt algorithm for all fits, or to define a MAGIX xml file, described in (§ 10.6), which describes settings for another algorithm or for an algorithm chain. So, the user is able to control the accurateness of the whole line identification process.

Additionally, the LineIdentification function is able to identify molecules in more than one spectrum (or frequency range). For that purpose, the user has to apply a MAGIX xml file, described in (§ 10.7). Hereby, the LineIdentification function performs single molecule fits where all frequency ranges of all spectra are fitted simultaneously[27]. If a molecule contributes significantly to at least one frequency range and does not lead to artificial features in the modeled spectrum, the optimized molfit file of the current molecule is append to an overall molfit file. At the end of the line identification process the LineIdentification function fits all frequency ranges simultaneously using the overall molfit file containing all identified molecules. Depending on the number of identified molecules and used velocity components, the LineIdentification function has to optimize more than 1000 free parameters in the final overall fit.

For each run the LineIdentification function creates a so-called job directory located in the run directory (`path-of-XCLASS-Interface/run/LineIdentification/`) where all files created by the LineIdentification function are stored in. The name of this job directory is made up of four components: The first part of the name consists of the phrase `job_` whereas the second part describes the date (day, month, year), the third part the time stamp (hours, minutes, seconds) of the function execution. The last part describes a so-called job ID which is composed of the so-called PID followed by a four digit random integer number to create a really unambiguous job number, e.g.

`path-of-XCLASS-Interface/run/LineIdentification/ job__25-07-2014__12-02-03__189644/`.

Additionally, the LineIdentification function creates a subdirectory within the job directory called `single-molecule_fits`. Within this single molecule fit directory the LineIdentification function creates subdirectories for each molecule from the database (or template file etc.). All files produced by the single molecule fits are stored in these subdirectories.

---

[27]In order to reduce the computational effort, the LineIdentification divides for the single-molecule fits the given frequency ranges in small ranges around the non-doppler shifted transitions frequencies

Please note, that the original xml- and data files are not modified. Only the copies of these files located in the single molecule fit directories are modified!

In addition, the LINEIDENTIFICATION function converts the column density N_{tot} and (if the hydrogen column density N_H is given for each component) the hydrogen column density N_H as well to a log scale, i.e. these two densities are converted automatically to their log10 values to get a better fit. At the end of the fitting process, the log10 values are converted back to the linear values. So, the MAGIX log files contain the log10 values of these parameters, whereas the input and output molfit files contain the linear values, respectively.

Note, the LINEIDENTIFICATION function offers the possibility to perform more than one single molecule fit at the same time using a cluster consisting of at least two computers. The nodes (computers) of the cluster are defined in the "cluster configuration file", see input parameter `clusterdef` described below. For example, a cluster consisting of three nodes with four cores respectively offers the possibility to perform 12 (3×4) single molecule fits simultaneously. Please note, that the total number of cores used by the LINEIDENTIFICATION function is mostly even higher, because each single molecule fit requires further cores defined in the algorithm xml file as well. For example, using the Levenberg-Marquardt algorithm with eight processors (cores) for a single molecule fit on a cluster described above requires 96 (8×12) cores altogether.

14.1 Single molecule fits

14.1.1 Strong molecule fits

In order to take the contribution of one or more so-called strong (i.e. highly abundant) molecules for each single molecule fit into account, the user can define a list of strong molecules. The LINEIDENTIFICATION function performs the single molecule fits for these strong molecules first, where it starts with the first strong molecule. If this molecule contributes to the given spectra, the optimized molfit file is append to all molfits files of all other molecules. For example, if the first strong molecule contributes to the given spectra, the optimized molfit file is append to the molfit file related to the second strong molecule and so on. Please note, the (optimized) parameters describing the contribution of a strong molecules are kept constant for all other single molecule fits. These parameters will be optimized only in the final overall fit.

14.1.2 Does a molecule contribute?

After finishing a single molecule fit the LINEIDENTIFICATION function reads in the modeled spectra for all frequency ranges and checks, if the modeled spectra contains at least one peak with intensity above the user defined noise level(s). (All peaks in the modeled spectra below the noise level(s) are completely ignored.) Thereafter, the LINEIDENTIFICATION function searches for artificial peaks, i.e. peaks in the modeled spectrum above the noise level, which have no corresponding peak in the observational data. For that purpose, the user has to define the global overestimation factor (in %, input parameter `MaxOverestimationHeight`) valid for all single molecule fits. The LINEIDENTIFICATION function compares the intensities of the modeled and the observed spectra at the doppler-shifted transition frequencies $\nu_{\text{Doppler}}^{i} = \nu_t^i \cdot \left(1 - \frac{v_{\text{offset}}^{m,c}}{c_{\text{light}}}\right)$, where ν_t^i represents the ith non-doppler shifted transition frequency taken from the database and v_{offset}^c the velocity offset of component c taken from the molfit file for the current molecule m. By calculating the fraction η_{Peak} of intensities of modeled and observed spectra at these frequencies, i.e.

$$\eta_{\text{Peak}} = \frac{I_{\text{model}}\left(\nu^i_{\text{Doppler}}\right)}{I_{\text{observed}}\left(\nu^i_{\text{Doppler}}\right)} \times 100$$

the LINEIDENTIFICATION function decides if a molecule is included or not. A peak is over-estimated if η_{Peak} is larger than the overestimation limit defined by the input parameter `MaxOverestimationHeight` $+ 100$ %. If the number of overestimated lines compared to the total number of doppler-shifted transition frequencies ν^i_{Doppler} of the current molecule is higher than the user defined tolerance (given by the input parameter `Tolerance`) the current molecule is NOT identified.

If the modeled spectrum does not overestimate the observed spectrum the corresponding molecule is "for now" identified and the fitted molfit file is added to the overall molfit file which is used at the end of the line identification process to determine the final contribution of each molecule.

Furthermore, the LINEIDENTIFICATION function writes a short summary about the result of each single molecule fit to a file called `results.dat` located in the current job subdirectory. The file contains the input parameters, the min. and max. frequency of each frequency range, a list with all molecules considered in the defined frequency range(s), the noise level for the defined frequency range(s) (i.e. the minimal intensity of a line), and informations about the molecule identification process. Additionally, the LINEIDENTIFICATION function creates a file called `Identified_Molecules.dat` located in the same directory containing all identified molecules, i.e. all molecules which contributes significantly to the spectra (controlled by the input parameter `MaxOverestimationHeight`, see below). Finally, the LINEIDENTIFICATION function creates a further subdirectory within the job directory called `Intermediate_identified_molecules` containing plots of the fitted (continuum subtracted) spectrum together with the optimized molfit file for each identified molecule. Hereby the names of the spectrum plot files contain the name of the corresponding molecule plus the name of the exp. data file plus the lowest and highest frequencies of the corresponding frequency range. For example, the file `CH3OH_v=0___sgrb2m.dat__342282.0_-_345282.0_MHz.png` describes the modeled spectrum of the molecule $\text{CH}_3\text{OH}_{v=0}$ together with the observational data from file `sgrb2m.dat.png` for the frequency range between 342282.0 MHz and 345282.0 MHz. In addition, each plot contains one (or two) horizontal blue dotted line(s) indicating the noise levels[28], and one or more vertical green dashed lines describing the non-doppler shifted transition frequencies, see Fig. 28. In order to control the identification process, the current job directory contains another subdirectory, called `Not_identified_molecules`, including the plots of the non-identified molecules. The plots contain the same informations as the plots of the identified molecules. Please note, the directory `Not_identified_molecules` contains no molfit files.

14.2 Overall fit

After finishing all single molecule fits for all frequency ranges, the LINEIDENTIFICATION function performs an overall fit with all identified molecules, where all frequency ranges are fitted simultaneously. For this overall fit, the LINEIDENTIFICATION function creates a further subdirectory within the current job directory called `all`, where all required MAGIX files are stored in. The molfit file for this overall molfit file is made up by merging all optimized molfit files of the identified molecules from the single molecule fits. After the overall fit is done, the LINEIDENTIFICATION function determines the contribution of each molecule described in the optimized overall

---

[28]The second blue line is important for absorption: Only absorption lines below the second (lower) blue line are considered.
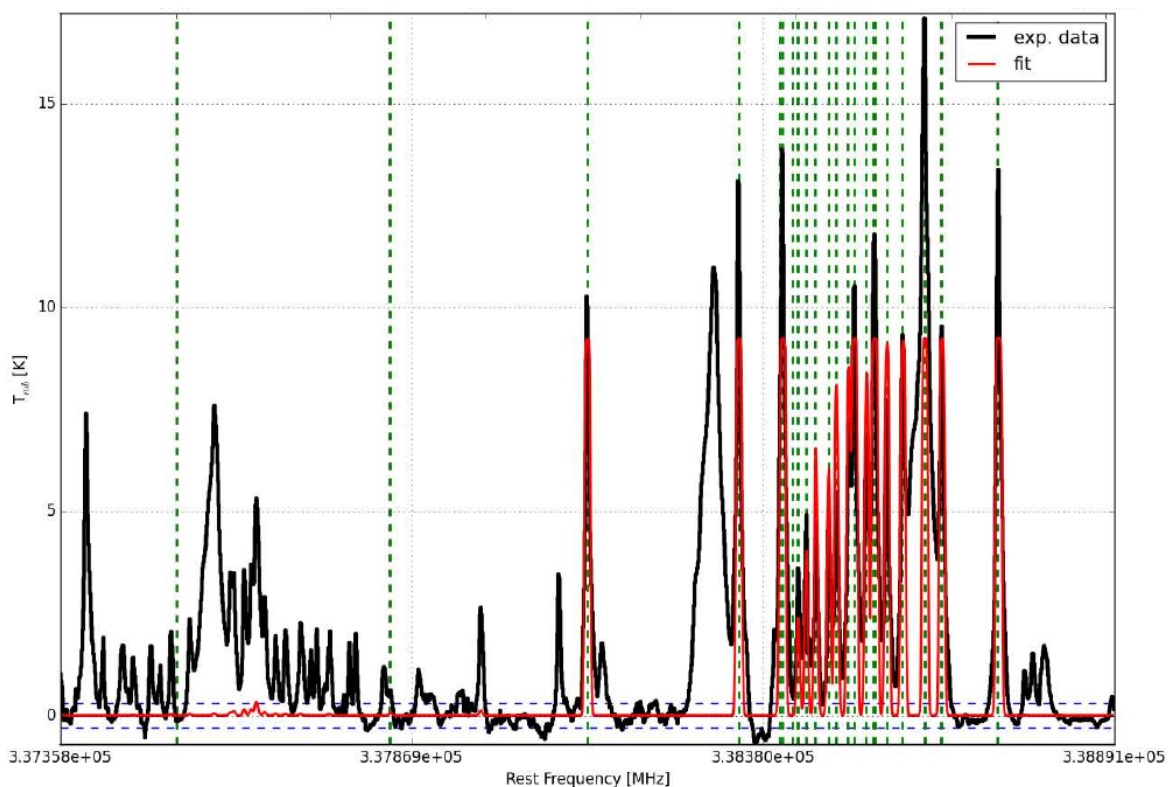
Figure 28: Result of a single molecule fit for CH_3OH (continuum subtracted) together with the observed data (continuum subtracted). The horizontal green dotted lines indicate the transition frequencies of CH_3OH, respectively.

molfit file using the MYXCLASS function. For that purpose the LINEIDENTIFICATION function creates a subdirectory within the `all` subdirectory called `final_fit`, determines the spectrum of each molecule for each frequency range using the MYXCLASS function, creates a plot showing the (continuum subtracted) observational data together with the (continuum subtracted) modeled spectrum of a certain molecule, and stores this plot file in a further subdirectory named with the name of the current molecule within the `final_fit` directory.

The name of the plot file is made up of the name of the molecule, the name of the observational data file and the lowest frequency (in MHz) of the corresponding frequency range. The plots for each molecule contribution contain one or more vertical green dotted lines indicating the transition frequencies stored in the database for the corresponding molecule in the given frequency range, see Fig. 29. In addition the horizontal blue dotted line(s) indicate(s) the band of noise, as described above.

14.2.1 Input parameters:

▶ `Noise`: noise level (in K), (default: 0.1 K). All parts of the spectrum with intensities lower than the noise level are ignored.

Using a MAGIX xml file for the import of the observational data (see description of input parameter `experimentalData` below) allows the user to specify a noise level for each frequency range. For that purpose, a tag called `NoiseLevel`, see below, has to be defined for each frequency range for all spectra defined in the xml file. If the definition is not
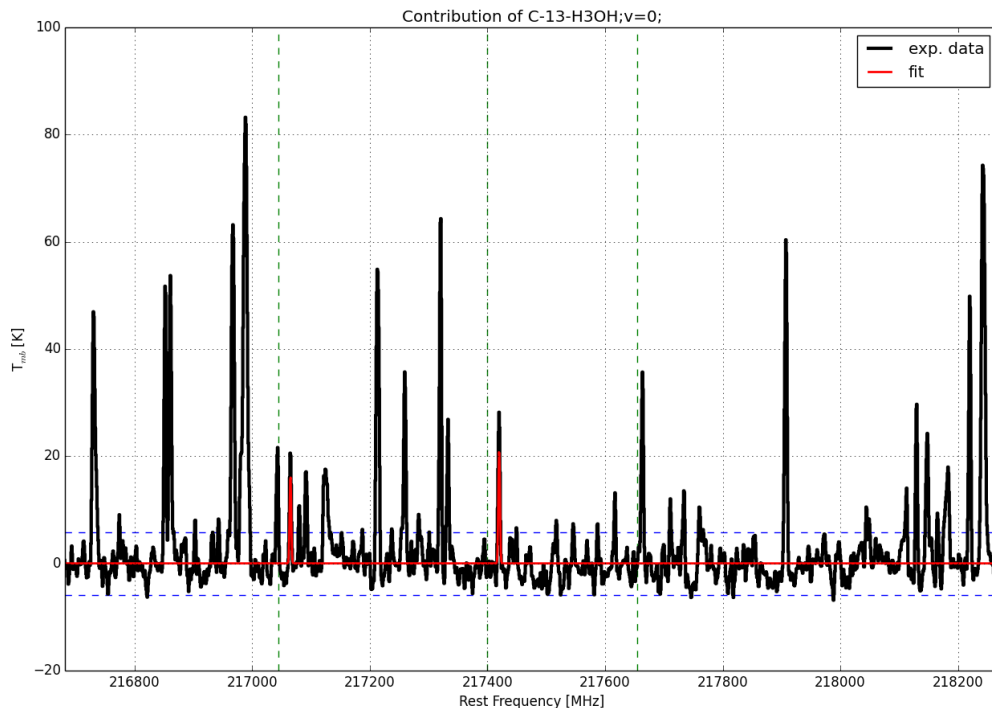
108

Figure 29: Example of a plot showing the contribution of a single molecule (here $C^{13}H_3OH_{v=0}$) after finishing the overall fit. The vertical green dotted lines indicate the transition frequencies in the database in the given frequency range.

given for all frequency ranges, the value defined by this parameter `Noise` is used instead for all frequency ranges of all spectra.

Listing 10: MAGIX xml file for the import of observational data containing new tag `NoiseLevel`

```
...

<!-- define parameters for each data ranges -->
<FrequencyRange>
    <MinExpRange>342285</MinExpRange>
    <MaxExpRange>344247</MaxExpRange>
    <StepFrequency>1</StepFrequency>


    <!-- define background temperature and temperature slope -->
    <t_back_flag>True</t_back_flag>
    <BackgroundTemperature>24.0</BackgroundTemperature>
    <TemperatureSlope>0.0</TemperatureSlope>


    <!-- define hydrogen column density, beta for dust, and kappa -->
    <HydrogenColumnDensity>1.4e+25</HydrogenColumnDensity>
    <DustBeta>1.4</DustBeta>
    <Kappa>0.02</Kappa>


    <!-- define noise level for current frequency range -->
    <NoiseLevel>18.37</NoiseLevel>
```

```
        </FrequencyRange>
        ...
```

▶ `MaxOverestimationHeight`: defines the overestimation factor of the modeled single molecule spectrum (in %), (default: 10 %).

Please note, the input parameter `MaxOverestimationHeight` has to be given in % and does not define the final overestimation limit directly which is given by `MaxOverestimationHeight` + 100 %.

If the modeled spectrum does not overestimate the observed spectrum, the corresponding molecule is "for now" identified and the fitted molfit file is added to the overall molfit file which is used at the end of the line identification process to determine the final contribution of each molecule.

▶ `SourceName`: This parameter defines the path and the name of a source (template) molfit file which is used for the single molecule fits. Whenever the LINEIDENTIFICATION function fits the contribution of one of the molecules included in the source molfit file the definitions herein are used instead of the definitions in the default molfit file, see description below. This offers the possibility to fit the contribution(s) of one or more molecules with a different number of components, ranges, initial values etc. as defined in the default molfit file. In order to perform the single molecule fits for molecules which are not described by the source molfit file, the LINEIDENTIFICATION function uses the definitions in the default molfit file, see below.

Please note, the source molfit file has to be given in the extended molfit format which is described in detail in (§ 10.4.2).

▶ `DefaultMolfitFile`: This parameter defines the path and name of a so-called default molfit file which is used to fit the contribution of each molecule.

The default molfit file defines for one molecule the number of components, the ranges and initial values for each parameter and has to be defined in the extended molfit format which is described in detail in (§ 10.4.2).

During the line identification process the name of the (first) molecule defined in the default molfit file is replaced by the name of the current molecule.

NOTE, if the parameter `DefaultMolfitFile` defines a relative path, the path has to be defined relative to the current working directory!

▶ `Tolerance`: defines the max. fraction (in %) of overestimated lines. If the fraction of overestimated lines in the result of a single molecule fit is lower than the given threshold, the corresponding molecule is "for now" identified and the optimized molfit file is considered in the final overall fit.

▶ `SelectedMolecules`: A (python) list defining molecules which should be excluded from or considered only by the line identification process.

In general, the LINEIDENTIFICATION function considers all molecules which are included in the database within the defined frequency range(s).

In order to exclude a molecule from the line identification process, the name of the molecule has to start with "--", e.g. to exclude the molecule `CH3SH;v=0;` the user has to define `SelectedMolecules = ["--CH3SH;v=0;"]`.

If the molecule names do not start with "--", the LINEIDENTIFICATION function consider only these molecules.

▶ `StrongMoleculeList`: A (python) list including the strong (highly abundant) molecules which should be fitted at the beginning (default: "[]").

▶ `MinColumnDensityEmis`: min. column density (for core lines) of an optimized component of a single molecule fit to be included in the *overall* molfit file (default: 0).

▶ `MinColumnDensityAbs`: min. column density (for foreground lines) of an optimized component of a single molecule fit to be included in the *overall* molfit file (default: 0).

▶ `NumberIteration`: max. number of iterations (default: 50). This parameter is used only if no MAGIX xml files are defined by the parameters `AlgorithmXMLFileSMF` and `AlgorithmXMLFileOverAll`.

▶ `AlgorithmXMLFileSMF` (only necessary, if the user wants to use another fit algorithm (than Levenberg-Marquardt) for each single molecule fit): path and name of a MAGIX xml-file defining settings for an algorithm or algorithm chain has to be given. (A relative path has to be defined relative to the current working directory!)

NOTE, if the user specify a xml file, the number of iterations given by the parameter `NumberIteration` is ignored. The number of iteration is then given by the xml file itself. In order to use the implemented fit algorithm (Levenberg-Marquardt) clear the `AlgorithmXMLFileSMF` parameter, i.e. `AlgorithmXMLFileSMF = ""`, and define the max. number of iterations by using parameter `NumberIteration`.

For a detailed description of the required xml-file, see (§ 10.6).

▶ `AlgorithmXMLFileOverAll`: only necessary, if the user wants to use another fit algorithm (than Levenberg-Marquardt) for the overall fit. Therefore, the path and name of a MAGIX xml-file defining settings for an algorithm or algorithm chain has to be given. (A relative path has to be defined relative to the current working directory!)

For more information see description of parameter `AlgorithmXMLFileSMF`.

▶ `experimentalData`: This parameter offers two different possibility to send the experimental data to the LINEIDENTIFICATION function:

– the parameter `experimentalData` defines the path and name of and experimental xml-file suitable for MAGIX. For a detailed description of the MAGIX xml-file, see (§ 10.7).

– the parameter `experimentalData` defines the path and name of and ASCII file called `experimental data file`, where the first column describe the frequency (in MHz) and the second column the beam temperature (intensity) of a spectrum.

NOTE, if the parameter `experimentalData` defines a relative path, the path has to be defined relative to the current working directory!

The following parameters are needed, if the parameter `experimentalData` does NOT describe the path and name of a MAGIX xml-file:

▶ `vLSR`: velocity (local standard of rest) in km s^{-1} (default: 0) used in the calculation of the synthetic spectra, see description for myXCLASS function (§ 9). Please note, using a xml-file, the `vLSR` parameter can be defined for each obs. data file, see (§ 10.7.6)!

▶ `TelescopeSize`: size of telescope (default: 1).

▶ `Inter_Flag` (`T`/`F`): defines, if single dish (`F`) or interferometric observations (`T`) are described, (default: `F`).

▶ `tBack`: background temperature (in K), (default: 0).

Please note, the background temperature and slope have to describe the continuum completely, i.e. `t_back_flag` is always set to `T`.

▶ `tslope`: temperature slope (dimensionless), (default: 0).

▶ `N_H`: Hydrogen column density (in cm^{-2}), (default: 0).

▶ `beta_dust`: spectral index for dust (dimensionless), (default: 0).

▶ `kappa_1300`: dust mass opacity kappa (cm^2 g^{-1}), (default: 0.01).

Finally, the last parameter is needed to start the LINEIDENTIFICATION function on a cluster

▶ `clusterdef`: path and name of a file containing information for the cluster as described in the CASA cookbook[29] (subsection "10.3 Parallelization control"). In contrast to CASA's "cluster configuration file", the LINEIDENTIFICATION function requires only two entries for each node: the HOSTNAME of the target node where the cluster is deployed and the number of CORES (engines) separated by comma. (Comments are marked with the "#" character.) Please note, the third column can be used to define the positions of the XCLASS directory on each computer in the cluster. This is essential, if the cluster contains computers with different operating systems, e.g. Linux and MAC OS. If no path is defined, the LINEIDENTIFICATION function assumes that the XCLASS interface is located in the same directory as the current XCLASS interface.

Note, a "cluster configuration file" described in the CASA cookbook can be used as well whereat the definitions for the work directories are interpreted as definition for the XCLASS directory on the different nodes of the cluster. Additionally, the definitions of RAM usage and RAM per engine are ignored.

The following requirements are necessary for all nodes which are included in the cluster:

1. Password-less ssh access from the controller (user) machine into all the hosts to be included in the cluster.
   NOTE: This is not necessary when using only "localhost", i.e. if the cluster is deployed only on the machine where CASAPY is running.

2. XCLASS interface, CASA and gfortran (with OpenMP) are installed on all nodes of the cluster.

3. The XCLASS job directory must be located in a shared file-system, accessible from all nodes comprising the cluster, and mounted in the same path of the file-system.
   NOTE: If the cluster contains computers with different operating systems, e.g. Linux and MAC OS, the user has to define the different paths of the XCLASS interface directories.

Example of a "cluster configuration file" used for the LINEIDENTIFICATION function:

---

[29] http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf

```
    # cluster configuration file
    # node        number of cores        path of XCLASS interface
    anu,          2
    lugal,        3
```

In the example described above, the nodes "anu" and "lugal" will be used with two (three) cores, respectively.

NOTE: The total number of cores used on a node of the cluster is determined by the number of processors defined in the algorithm xml file times the number of cores defined in the "cluster configuration file". For example, the user wants to apply the Levenberg-Marquardt algorithm with eight processors on a cluster defined in the example above. On node "anu" 16 (2×8) cores and on node "lugal" 18 (3×8) cores are used.

14.2.2 Output parameters:

▶ `IdentifiedLines`: contains the optimized molfit file including the fitted parameters of all identified molecules.

▶ `JobDir`: absolute path of the job directory created for the current run.

Example without experimental xml-file:

```
Noise = 0.5
MaxOverestimationHeight = 500.0
Tolerance = 65.0
MinColumnDensityEmis = 0.0
MinColumnDensityAbs = 0.0
SourceName = ""
DefaultMolfitFile ="demo/LineIdentification/demo__default.molfit"
SelectedMolecules = ["HCCCN;v=0;", "CH3OH;v=0;", "C2H5OH;v=0;", \
                     "CH3CN;v=0;", "SO;v=0;", "SO2;v=0;"]
StrongMoleculeList = []
NumberIteration = 10
AlgorithmXMLFileSMF = ""
AlgorithmXMLFileOverAll = ""
experimentalData ="demo/LineIdentification/SyntheticData.dat"
vLSR = 0.0
TelescopeSize = 350.0
Inter_Flag = F
tBack = 1.0
tslope = 0.0
N_H = 1.4E+23
beta_dust = 1.4
kappa_1300 = 0.0
IdentifiedLines, JobDir = LineIdentification()
```

Example using an experimental xml-file:

```
Noise = 0.5
MaxOverestimationHeight = 500.0
Tolerance = 65.0
MinColumnDensityEmis = 0.0
MinColumnDensityAbs = 0.0
SourceName = ""
DefaultMolfitFile ="demo/LineIdentification/demo__default.molfit"
```

```
    SelectedMolecules = ["HCCCN;v=0;", "CH3OH;v=0;", "C2H5OH;v=0;",  \
                         "CH3CN;v=0;", "SO;v=0;", "SO2;v=0;"]
    StrongMoleculeList = []
    NumberIteration = 10
    AlgorithmXMLFileSMF = ""
    AlgorithmXMLFileOverAll = ""
    experimentalData = "demo/LineIdentification/demo.xml"
    IdentifiedLines, JobDir = LineIdentification()
```

Example with experimental xml-file and algorithm xml-file:

```
    Noise = 0.5
    MaxOverestimationHeight = 500.0
    Tolerance = 65.0
    MinColumnDensityEmis = 0.0
    MinColumnDensityAbs = 0.0
    SourceName = ""
    DefaultMolfitFile ="demo/LineIdentification/demo__default.molfit"
    SelectedMolecules = ["HCCCN;v=0;", "CH3OH;v=0;", "C2H5OH;v=0;", \
                         "CH3CN;v=0;", "SO;v=0;", "SO2;v=0;"]
    StrongMoleculeList = []
    AlgorithmXMLFileSMF = "demo/LineIdentification/algorithm-settings___LM.xml"
    AlgorithmXMLFileOverAll = "demo/LineIdentification/"
    AlgorithmXMLFileOverAll += "algorithm-settings___LM__Final.xml"
    experimentalData = "demo/LineIdentification/demo.xml"
    IdentifiedLines, JobDir = LineIdentification()
```

Usage without CASA:

```
    # extend sys.path variable
    ...


    # import task_LineIdentification package
    import task_LineIdentification


    # call LineIdentification function
    Noise = 0.5
    MaxOverestimationHeight = 500.0
    Tolerance = 65.0
    MinColumnDensityEmis = 0.0
    MinColumnDensityAbs = 0.0
    DefaultMolfitFile ="demo/LineIdentification/demo__default.molfit"
    SelectedMolecules = ["HCCCN;v=0;", "CH3OH;v=0;", "C2H5OH;v=0;", \
                         "CH3CN;v=0;", "SO;v=0;", "SO2;v=0;"]
    StrongMoleculeList = []
    AlgorithmXMLFileSMF = "demo/LineIdentification/algorithm-settings___LM.xml"
    AlgorithmXMLFileOverAll = "demo/LineIdentification/"
    AlgorithmXMLFileOverAll += "algorithm-settings___LM__Final.xml"
    experimentalData = "demo/LineIdentification/demo.xml"
    vLSR = 0.0
    TelescopeSize = 350.0
    Inter_Flag = F
    tBack = 1.0
    tslope = 0.0
    N_H = 1.4E+23
    beta_dust = 1.4
```

```
kappa_1300 = 0.0
IdentifiedLines, JobDir = task_LineIdentification.LineIdentification( \
                         Noise, MaxOverestimationHeight, SourceName, \
                         DefaultMolfitFile, Tolerance, \
                         SelectedMolecules, StrongMoleculeList, \
                         MinColumnDensityEmis, MinColumnDensityAbs, \
                         NumberIteration, AlgorithmXMLFileSMF, \
                         AlgorithmXMLFileOverAll, experimentalData, \
                         vLSR, TelescopeSize, Inter_Flag, tBack, \
                         N_H, beta_dust, kappa_1300, clusterdef)
```

A Derivations

A.1 Detection Equation

In absence of scattering, the radiative transfer equation

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}s} = -\kappa_\nu(s)I_\nu + \epsilon_\nu(s) \tag{53}$$

describes the propagation of radiation which passes through a medium. During the propagation photons are absorbed and emitted indicated by the emission and absorption coefficients ϵ_ν and κ_ν, respectively.

The optical depth $\tau_\nu(s)$ which measures the distance in units of the mean free path, is given by

$$\tau_\nu(s) = \int_{s'=0}^{s'=s} \kappa_\nu \, ds'. \tag{54}$$

By using the source function $S_\nu = \frac{\epsilon_\nu}{\kappa_\nu}$, the radiative transfer equation can be expressed as

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}\tau} = I_\nu(\tau) + S_\nu(\tau). \tag{55}$$

Within the local thermodynamic equilibrium (LTE) the source function is described by Planck's law for black body radiation $B_\nu(T)$.

Integrating Eq. (55) over τ leads to

$$I_\nu(s) = I_\nu(0) \, e^{-\tau_\nu} + \int_{\tau'=0}^{\tau'=\tau} S_\nu(\tau'_\nu) e^{\tau'_\nu - \tau_\nu} d\tau'_\nu. \tag{56}$$

Assuming a constant source function, i.e. constant emission and absorption coefficients through the medium, the transfer equation can be written as

$$I_\nu(s) = I_\nu(0) \, e^{-\tau_\nu} + S_\nu \left(1 - e^{-\tau_\nu}\right). \tag{57}$$

Additionally, we have to take into account, that the different components may not cover the whole beam, i.e. that the background behind a certain component might contribute as well. So, we have to extend Eq. (57) by introducing the beam filling factor η, Eq. (5), which describes the fraction of the beam covert by a component

$$I_\nu(s) = \eta \left[I_\nu(0) \, e^{-\tau_\nu} + S_\nu \left(1 - e^{-\tau_\nu}\right)\right] + (1 - \eta) \, I_\nu(0). \tag{58}$$

Here, the term $\eta \, I_\nu(0) \, e^{-\tau_\nu}$ indicates the attenuated radiation from the background $I_\nu(0)$. The second term $\eta \, S_\nu \left(1 - e^{-\tau_\nu}\right)$ describes the self attenuated radiation emitted by a certain component. Finally, the last term $(1 - \eta) \, I_\nu(0)$ represents the contribution of the background which is not covert by a component.

In real observations, we do not measure absolute intensities but only differences of intensities, i.e. we have to subtract the OFF-position from Eq. (58) as well, where we have an intensity caused by the cosmic background J_{CMB}. So, we achieve

$$I_\nu(s) = \eta \left[I_\nu(0) \, e^{-\tau_\nu} + S_\nu \left(1 - e^{-\tau_\nu}\right)\right] + (1 - \eta) \, I_\nu(0) - J_{\mathrm{CMB}}. \tag{59}$$
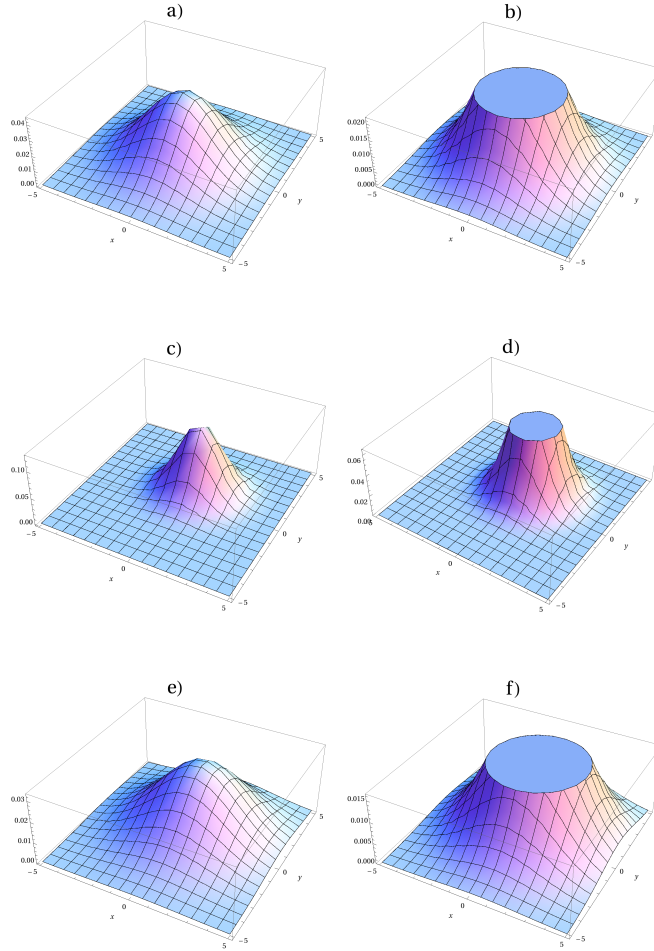
Figure 30: a) A single two-dimensional Gaussian function, Eq. (60), with $\sigma_x = \sigma_y = 1.9$ and $\mu_x = \mu_y = 0$ representing a Gaussian beam of a telescope. b) Cut through the Gaussian function described in a) at half height. c) A single two-dimensional Gaussian function, with $\sigma_x = \sigma_y = 1.1$ and $\mu_x = 0.5$ and $\mu_y = 0.9$ representing a Gaussian beam of a point source. d) Cut through the Gaussian function described in a) at half height. e) Convolved Gaussian of telescope and point source as given by Eq. (63). f) Cut through the convolved Gaussian function described in e) at half height.

A.2 Beam Filling Factor

The derivation of the beam filling factor (5) starts with the normalized two-dimensional Gaussian function

$$g(x, y, \sigma_x, \sigma_y, \mu_x, \mu_y) = \frac{1}{\sqrt{2\,\pi\,\left(\sigma_x^2 + \sigma_y^2\right)}}\,e^{-\left(\frac{(x-\mu_x)^2}{2\,\sigma_x^2} + \frac{(y-\mu_y)^2}{2\,\sigma_y^2}\right)}, \tag{60}$$

where σ_x^2 and σ_y^2 describe the variances and μ_x and μ_y the center along the x and y axis, respectively.

Observing a Gaussian shaped extended source with a telescope is described by a convolution

of two two-dimensional Gaussian functions:

$$(g_1 * g_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g_1(x - u, y - v, \sigma_{x,1}, \sigma_{y,1}, \mu_{x,1}, \mu_{y,1}) \cdot g_2(u, v, \sigma_{x,2}, \sigma_{y,2}, \mu_{x,2}, \mu_{y,2}) \, du \, dv$$

(61)

$$= \frac{1}{2\pi \sqrt{\left(\sigma_{x,1}^2 + \sigma_{x,2}^2\right)\left(\sigma_{y,1}^2 + \sigma_{y,2}^2\right)}} e^{-\left(\frac{(\mu_{x,1}+\mu_{x,2}-x)^2}{2\left(\sigma_{x,1}^2+\sigma_{x,2}^2\right)} + \frac{(\mu_{y,1}+\mu_{y,2}-y)^2}{2\left(\sigma_{y,1}^2+\sigma_{y,2}^2\right)}\right)}.$$

(62)

Assuming that g_1 describes the telescope with $\mu_{x,1} = \mu_{y,1} \equiv 0$ and that telescope and extended source are described by non-elliptical Gaussians, i.e. $\sigma_{x,1} = \sigma_{y,1} \equiv \sigma_1$ and $\sigma_{x,2} = \sigma_{y,2} \equiv \sigma_2$, Eq. (61) can be simplified to[30]

$$(g_1 * g_2) = \frac{1}{2\pi \left(\sigma_1^2 + \sigma_2^2\right)} e^{-\frac{(x-\mu_{x,2})^2 + (y-\mu_{y,2})^2}{2\left(\sigma_1^2+\sigma_2^2\right)}}.$$

(63)

The FWHM of the resulting Gaussian is given by

$$\text{FWHM} = 2\sqrt{2\log 2}\sqrt{\sigma_1^2 + \sigma_2^2}$$

(64)

which describe an area of

$$A_{\text{conv}}^{\text{FWHM}} = \pi \cdot 2\log 2 \cdot \left(\sigma_1^2 + \sigma_2^2\right)$$
$$= \frac{\pi}{4} \cdot \left(\theta_1^2 + \theta_2^2\right).$$

(65)

In the last line we used the relation between the variances $\sigma_{1,2}$ and the user defined FWHM of telescope ($\theta_1 \equiv \theta_t$) and source size ($\theta_2 \equiv \theta^{m,c}$) which is given by

$$\theta_i = 2\sqrt{2\log 2} \cdot \sigma_i.$$

(66)

The beam filling factor Eq. (5) is defined as ratios of areas

$$\eta_{g_1, g_2} = \frac{A_{\text{source}}^{\text{FWHM}}}{A_{\text{conv}}^{\text{FWHM}}} = \frac{\frac{\pi \theta_2^2}{4}}{\frac{\pi}{4} \cdot \left(\theta_1^2 + \theta_2^2\right)} = \frac{\theta_2^2}{\left(\theta_1^2 + \theta_2^2\right)},$$

(67)

which is completely independent of the position μ_x and μ_y of the extended source within the telescope beam.

If more extended sources are observed with the telescope, we have to convolve the already convolved Gaussian Eq. (63) with a further two-dimensional Gaussian function with variance $\sigma_{x,3} = \sigma_{y,3} \equiv \sigma_3$ and center $\mu_{x,3}$ and $\mu_{y,3}$. We get

$$((g_1 * g_2) * g_3) = (g_1 * g_2 * g_3) = \frac{1}{2\pi \left(\sigma_1^2 + \sigma_2^2 + \sigma_3^2\right)} e^{\left(-\frac{(\mu_{x,2}+\mu_{x,3}-x)^2 + (\mu_{y,2}+\mu_{y,3}-y)^2}{2\left(\sigma_1^2+\sigma_2^2+\sigma_3^2\right)}\right)}.$$

(68)

The FWHM of the resulting Gaussian is given by

$$\text{FWHM} = 2\sqrt{2\log 2}\sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2}$$

(69)

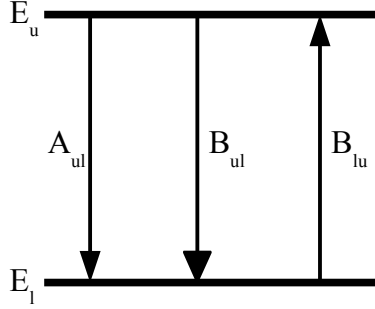[30]Here "*" indicates the convolution of two functions g_1 and g_2.

Figure 31: Transitions between the lower l and the upper u level with the corresponding Einstein coefficients.

which describe an area of

$$
\begin{aligned}
A_{\text{conv}}^{\text{next}} &= \pi \cdot 2 \, \log 2 \cdot \left(\sigma_1^2 + \sigma_2^2 + \sigma_3^2 \right) \\
&= \frac{\pi}{4} \cdot \left(\theta_1^2 + \theta_2^2 + \theta_3^2 \right).
\end{aligned}
\tag{70}
$$

In the last line we used again the relation between the variances and FWHM, Eq. (66). Now, we can again define a beam filling factor similar to Eq. (5) and get

$$
\eta_{g_1, g_2, g_3} = \frac{\frac{\pi \, \theta_2^2}{4} + \frac{\pi \, \theta_3^2}{4}}{\frac{\pi}{4} \cdot \left(\theta_1^2 + \theta_2^2 + \theta_3^2 \right)} = \frac{\theta_2^2 + \theta_3^2}{\left(\theta_1^2 + \theta_2^2 + \theta_3^2 \right)}.
\tag{71}
$$

When we observe more than three extended sources with a telescope we have to iteratively convolve the resulting Gaussian with a further two-dimensional Gaussian function and we can generalize Eq. (71) to

$$
\eta_{\text{general}} = \frac{\sum_{i>1} \theta_i^2}{\left(\sum_i \theta_i^2 \right)} = \frac{\sum_{i>1} \theta_i^2}{\theta_t^2 + \sum_{i>1} \theta_i^2},
\tag{72}
$$

where $\theta_1 \equiv \theta_t$ describes the FWHM of the Gaussian shaped beam of the telescope, defined by the diffraction limit, Eq. (6).

A.3 Optical depth

In order to derive Eq. (11) we consider a system which involves radiative transitions between a lower l and an upper u level only. As shown in Fig. 31, the lower level has an energy E_l and the upper level an energy $E_u > E_l$. With

$$
h \, \nu_{u,l} = E_u - E_l
\tag{73}
$$

describing the energy difference between these two levels we can express the emissivity due to spontaneous radiative decay as

$$
\epsilon_{l,u,\nu} = \frac{h \, \nu}{4 \, \pi} \, n_u \, A_{u,l} \, \phi_{l,u}(\nu),
\tag{74}
$$

where $A_{u,l}$ describes the *Einstein A-coefficient*, or radiative decay rate for the transition from the lower l to the upper u level. The expression $1/A_{u,l}$ gives the averaged time, that a quantum

mechanical system can stay in level u before radiatively decaying to level l, where we assume no collisional (de-)excitation. The expression $\phi_{l,u}(\nu)$ describes the line profile of the transition of photons of frequency ν and is normalized to 1, i.e. $\int_0^\infty \phi(\nu)\,d\nu = 1$.

Similar to Eq. (74) we can write the extinction coefficient, which describes the radiative excitation from the lower to the upper level

$$\kappa_{l,u,\nu}^{\text{ext}} = \frac{h\,\nu}{4\,\pi}\,n_l\,B_{l,u}\,\phi_{l,u}(\nu), \tag{75}$$

where $B_{l,u}$ describes the *Einstein B-coefficient for extinction*.

In addition to spontaneous emission and extinction we have to take the stimulated emission into account, which can be described by adding a negative opacity contribution to Eq. (75):

$$\kappa_{l,u,\nu} = \frac{h\,\nu}{4\,\pi}\,(n_l\,B_{l,u} - n_u\,B_{u,l})\,\phi_{l,u}(\nu), \tag{76}$$

where $B_{u,l}$ represents the *Einstein B-coefficient for stimulated emission*. So, for $n_l B_{l,u} < n_u\,B_{u,l}$ we get laser (maser) emission.

The different Einstein coefficients are related to each other by the Einstein relations:

$$A_{u,l} = \frac{2\,h\,\nu^3}{c_{\text{light}}^2}\,B_{u,l},$$
$$g_l\,B_{l,u} = g_u\,B_{u,l}. \tag{77}$$

Following Eq. (54), the differential optical depth τ_ν is defined as

$$\begin{aligned} d\tau_\nu = \kappa_\nu\,ds &= \left(\frac{h\,\nu}{4\,\pi}\,(n_l\,B_{l,u} - n_u\,B_{u,l})\,\phi_{l,u}(\nu) \right)\,ds \\ &= \left(\frac{c_{\text{light}}^2}{8\,\pi\,\nu^2}\,A_{u,l}\,\left(n_l\,\frac{g_u}{g_l} - n_u\right)\,\phi_{l,u}(\nu) \right)\,ds, \end{aligned} \tag{78}$$

where we used the Einstein relations Eqn. (77) in the last line. By assuming LTE conditions and therefore Boltzmann population distribution

$$\frac{n_u}{n_l} = \frac{g_u}{g_l}\,e^{(-E_u - E_l)/k_B T_{\text{ex}}} = \frac{g_u}{g_l}\,e^{-h\,\nu_{u,l}/k_B T_{\text{ex}}} \tag{79}$$

we can rewrite Eq. (78) by using Eq. (73)

$$d\tau_\nu = \left(\frac{c_{\text{light}}^2}{8\,\pi\,\nu^2}\,A_{u,l}\,n_u\,\left(e^{h\,\nu_{u,l}/k_B\,T_{\text{ex}}} - 1\right)\,\phi_{l,u}(\nu) \right)\,ds. \tag{80}$$

Finally, we have to integrate along the line of sight and obtain the optical depth τ_ν

$$\tau_\nu = \frac{c_{\text{light}}^2}{8\,\pi\,\nu^2}\,A_{u,l}\,N_u\,\left(e^{h\,\nu_{u,l}/k_B\,T} - 1\right)\,\phi_{l,u}(\nu), \tag{81}$$

where $N_u = \int n_u\,ds$ describes the column density of a certain molecule in the upper state.

In order to express Eq. (81) in terms of the total column density $N_{\text{tot}} = \sum_{i=0}^{\infty} n_i$ we start again with the Boltzmann population distribution

$$N_{\text{tot}} = \sum_{i=0}^{\infty} n_i = n_j \sum_{i=0}^{\infty} \frac{n_i}{n_j} = n_j \sum_{i=0}^{\infty} \frac{g_i}{g_j} e^{(-E_i + E_j)/k_B T_{\text{ex}}} = \frac{n_j}{g_j} e^{E_j/k_B T_{\text{ex}}} \cdot Q\left(T_{\text{ex}}\right), \qquad (82)$$

where we used the partition function $Q(T_{\text{ex}})$, which is defined as sum over all states

$$Q(T_{\text{ex}}) = \sum_{i=0}^{\infty} g_i \, e^{-E_i/k_B T_{\text{ex}}}. \qquad (83)$$

For our purpose we rewrite Eq. (82) in terms of N_u and N_l:

$$N_{\text{tot}} = \frac{N_u}{g_u} e^{E_u/k_B T_{\text{ex}}} \cdot Q\left(T_{\text{ex}}\right). \qquad (84)$$

Inserting this expression into Eq. (81) gives

$$\tau_\nu = \frac{c_{\text{light}}^2}{8\,\pi\,\nu^2} \, A_{u,l} \left[\frac{N_{\text{tot}}\, g_u}{Q\left(T_{\text{ex}}\right)} e^{-E_u/k_B T_{\text{ex}}} \right] \left(e^{h\,\nu_{u,l}/k_B T} - 1 \right) \phi_{l,u}(\nu)$$

$$= \frac{c_{\text{light}}^2}{8\,\pi\,\nu^2} \, A_{u,l} \, N_{\text{tot}} \frac{g_u \, e^{-E_l/k_B T_{\text{ex}}}}{Q\left(T_{\text{ex}}\right)} \left(1 - e^{-h\,\nu_{u,l}/k_B T} \right) \phi_{l,u}(\nu), \qquad (85)$$

where we used Eq. (73) to achieve Eq. (11) for a single line of a certain component and molecule.

References

[Andrae 2010] Andrae, R. 2010 (arXiv: 1009.2755)

[Boone *et al.* 2006] Boone, F. *et al.* 2006, Astronomical Data Analysis Software and Systems XV ASP Conference Series, Vol. 351, Proceedings of the Conference Held 2-5 October 2005 in San Lorenzo de El Escorial, Spain. Edited by Carlos Gabriel, Christophe Arviset, Daniel Ponz, and Enrique Solano. San Francisco: Astronomical Society of the Pacific, 2006., p.577

[Diaconis 2009] Diaconis, P. 2009, Bull. Amer. Math. Soc. 46, 179-205.

[Fan & Zahara 2007] Fan, S.-K. S., & Zahara, E. 2007, Journal of Operational Research 181, 527.

[Feroz & Hobson 2008] Feroz, F., & Hobson, M.P. 2008, MNRAS 384, 449.

[Foreman-Mackey *et al.* 2012] Foreman-Mackey, D., Hogg, D.W., Lang, D. & Goodman, J. 2012 (arXiv: 1292.3665)

[Garbow *et al.* 1980] Garbow, B., *et al.* 1980, MINPACK. in. www.netlib.org/minpack, Argonne National Lab.

[Gilks *et al.* 1996] Gilks, W.R., Richardson, S., Spiegelhalter, D. (Eds.) 1996 *Markov Chain Monte Carlo in Practice*, Chapman and Hall, London

[Greisen *et al.* 2002] Greisen, E.W., and Calabretta, M.R., 2002, A & A, 395, 1061-1075.

[Calabretta *et al.* 2002] Calabretta, M.R., and Greisen, E.W., 2002, A & A, 395, 1077-1122.

[Goodman & Weare 2010] Goodman, J. & Weare, J., 2010, Comm. App. Math. Comp. Sci., 5, 65

[Greisen *et al.* 2006] Greisen, E.W., *et al.*, 2006, A & A, 446, 747-771.

[Heavens 2009] Heavens, A. 2009 (arXiv: 0906.0664v3)

[Herrera *et al.* 1998] Herrera, F. *et al.* 1998, Artificial Intelligence Review 12, 265

[Herrera *et al.* 2005] Herrera, F. *et al.* 2005, Soft Comput. 9, 280.

[Hillebrand 1983] Hillebrand, R. H. 1983, QJRAS, 24, 267

[Ichida & Fujii 1979] Ichida, K., & Fujii, Y. 1979, Computing 23, 85.

[Jaynes 1976] Jaynes, E.T. 1976, Confidence Intervals vs Bayesian Intervals, in Foundations of Probability Theory, Statistical Inference, and Statistical Theories of Science, (W. L. Harper and C.A. Hooker, eds.), Dordrecht: D. Reidel, p. 175.

[Kennedy & Eberhart 1995] Kennedy, J. & Eberhart, R. 1995, Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942-1948.

[Krawczyk 1985] Krawczyk, R. 1985, Computing 34, 243.

[Marquardt 1963] Marquardt, D. W. 1963, J. Soc. Indust. Appl. Math. Vol. II, No.2.

[Müller *et al.* 2001] Müller, H.S.P., Thorwirth, S., Roth, D. A., & Winnewisser, G. 2001, A & A, 370, L49

[2005] Müller, H.S.P., Schlöder, F., Stutzki, J., & Winnewisser, G. 2005, J. Molec. Struct., 742, 215

[Nelder & Mead 1965] Nelder, J.A., & Mead, R. 1965, Computer Journal 7, 308.

[Nocedal & Wright 2006] Nocedal, J., & Wright, S. 2006, *Numerical Optimization*, Springer, 2nd edition.

[Ossenkopf & Henning 1994] Ossenkopf, V., & Henning, T. 1994, A & A, 291, 943

[Pham *et al.* 2005] Pham, D.T. *et al.* 2005, Manufacturing Engineering Centre, Cardiff University, UK.

[Pickett *et al.* 1998] Pickett, H. M., Poynter, R. L., Cohen, E. A., et al. 1998, J. Quant. Spectrosc. Radiat. Transf., 60, 883

[Press *et al.* 2007] Press, W.H. *et al.* 2007, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 3rd edition.

[Schilke *et al.* 2010] Schilke, P. *et al.* 2010, A & A, 521, L11.

[scipy] http://docs.scipy.org/doc/scipy/reference/optimize.html, call January 23rd 2012.

[Sivia & Skilling 2006] Sivia, D. & Skilling, J. 2006, *Data Analysis: A Bayesian Tutorial*, Oxford University Press.

[Skilling 2006] Skilling, J. 2006, Bayesian Analysis 1, 833.

[Skilling & MacKay 2012] Skilling, J, & MacKay, D. 2012, http://www.inference.phy.cam.ac.uk/bayesys/

[Stahler & Palla 2005] Stahler, S.W., & Palla, F. 2005, *The Formation of Stars* Wiley-VCH

[Whitley 1994] Whitley, D. 1994, Statistics and Computing 4, 65.