This manual documents how to install and use the ARoME Library, version 1.0.0.
Copyright © 2012, Centro de Astrofísica da Universidade do Porto
Contributed by
G. Boué, M. Montalto, I. Boisse, M. Oshagh, N. C. Santos
EXOEarths, Centro de Astrofísica, Universidade do Porto
arome@astro.up.pt
Table of Contents

1 ARoME Library Copying conditions ......... 1

2 Introduction to ARoME Library ............. 2

3 Installing ARoME Library .................. 3
   3.1 Installation on a Unix-like system (Linux, Mac OS X, ...) ...... 3
      3.1.1 Other ‘make’ Targets ................................ 4

4 Reporting bugs ............................. 5

5 ARoME Library Interface ................... 6
   5.1 C Usage ........................................ 6
      5.1.1 Headers and Libraries .......................... 6
         5.1.1.1 Compilation on a Unix-like system ........ 6
      5.1.2 Constants ..................................... 6
      5.1.3 Types ......................................... 6
   5.2 Fortran 2003 Usage ....................... 7
      5.2.1 Modules and Libraries ....................... 7
      5.2.2 Compilation on a Unix-like system ............ 7
   5.3 Fortran 77/90/95 Usage ................... 7
      5.3.1 Headers and Libraries ........................ 7
      5.3.2 Compilation on a Unix-like system ............ 7
   5.4 Single position functions ................ 8
      5.4.1 Usage ....................................... 8
      5.4.2 Functions .................................... 10
         5.4.2.1 arome_alloc_quad ......................... 10
         5.4.2.2 arome_alloc_nl .......................... 11
         5.4.2.3 arome_reset_quad ....................... 12
         5.4.2.4 arome_reset_nl ......................... 13
         5.4.2.5 arome_set_lineprofile .................. 13
         5.4.2.6 arome_set_planet ....................... 14
         5.4.2.7 arome_init_CCF .......................... 15
         5.4.2.8 arome_init_i iodine ..................... 15
         5.4.2.9 arome_calc_fvpbetap ..................... 16
         5.4.2.10 arome_set_flux ......................... 16
         5.4.2.11 arome_set_vp ........................... 17
         5.4.2.12 arome_set_betapR ....................... 17
         5.4.2.13 arome_set_betapT ....................... 18
         5.4.2.14 arome_get_flux ......................... 18
         5.4.2.15 arome_get_vp ........................... 19
         5.4.2.16 arome_get_betapR ....................... 19
         5.4.2.17 arome_get_betapT ....................... 20
5.4.2.18  arome_get_RM_CCF  .............................................. 20
5.4.2.19  arome_get_RM_CCF_e  .......................................... 20
5.4.2.20  arome_get_RM_iodine  .......................................... 21
5.4.2.21  arome_get_RM_iodine_e  ....................................... 21
5.4.2.22  arome_get_RM_mean  ............................................ 22
5.4.2.23  arome_get_RM_mean_e  ......................................... 22
5.4.2.24  arome_free  ....................................................... 23

5.5  Multiple position functions ........................................ 23
5.5.1  Usage ................................................................. 24
5.5.2  Functions ............................................................ 27
  5.5.2.1  arome_malloc .............................................. 27
  5.5.2.2  arome_mcalc_fvpbetap .................................. 27
  5.5.2.3  arome_mset_flux .......................................... 28
  5.5.2.4  arome_mset_vp ............................................. 29
  5.5.2.5  arome_mset_betapR ........................................ 29
  5.5.2.6  arome_mset_betapT ........................................ 30
  5.5.2.7  arome_mget_flux ........................................... 30
  5.5.2.8  arome_mget_vp ............................................. 31
  5.5.2.9  arome_mget_betapR ....................................... 32
  5.5.2.10  arome_mget_betapT ...................................... 32
  5.5.2.11  arome_mget_RM_CCF ...................................... 33
  5.5.2.12  arome_mget_RM_iodine ................................... 33
  5.5.2.13  arome_mget_RM_mean .................................... 34
  5.5.2.14  arome_mfree .............................................. 35

5.6  Error functions ...................................................... 35
5.6.1  Usage ................................................................. 35
5.6.2  Functions ............................................................ 37
  5.6.2.1  arome_set_continue_on_error .............................. 37
  5.6.2.2  arome_set_exit_on_error .................................. 37
  5.6.2.3  arome_set_func_on_error .................................. 38

5.7  Orbit computation functions ...................................... 38
5.7.1  Usage ................................................................. 38
5.7.2  Functions ............................................................ 39
  5.7.2.1  arome_new_orbit .......................................... 39
  5.7.2.2  arome_set_orbit_eo ....................................... 40
  5.7.2.3  arome_set_orbit_kh ...................................... 40
  5.7.2.4  arome_get_orbit_transit_time ............................ 41
  5.7.2.5  arome_get_orbit_xyz ...................................... 42
  5.7.2.6  arome_mget_orbit_xyz .................................... 42
  5.7.2.7  arome_free_orbit .......................................... 43

Appendix A  Release notes .............................................. 44
1 ARoME Library Copying conditions

Copyright © 2012, Centro de Astrofísica da Universidade do Porto
Contributed by
G. Boué, M. Montalto, I. Boisse, M. Oshagh, N. C. Santos
EXOEarths, Centro de Astrofísica, Universidade do Porto
arome@astro.up.pt

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.
2 Introduction to ARoME Library

This library is designed to generate analytical Rossiter-McLaughlin (RM) effects as described in the paper: G. Boué, M. Montalto, I. Boisse, M. Ohasgh, N. C. Santos: 2012, New analytical expressions of the Rossiter-McLaughlin effect adapted to different observation techniques, http://www.astro.up.pt/resources/arome/files/boue_et_al_2012.pdf. It models the Doppler-shift of a star during a transit measured by the following techniques:

- fit of a cross-correlation function by a Gaussian function.
- fit of an observed spectrum by a modeled one.
- weighted mean.

This library provides a C interface and, optionally, a Fortran 77 or 2003 interface to be called by the application.

Two groups of functions enable the calculation of RM effects:

- Single position functions
  These functions compute and return the RM effects at a single position of the planet with respect to the star.
- Multiple position functions
  These functions return the RM effects computed at different positions of the planet at the same time.

This library is based on a work supported by the European Research Council/European Community under the FP7 through Starting Grant agreement number 239953, as well as from Fundação para a Ciência e a Tecnologia (FCT) through program Ciência 2007 funded by FCT/MCTES (Portugal) and POPH/FSE (EC), and in the form of grants reference PTDC/CTE-AST/098528/2008, SFRH/BPD/71230/2010, and SFRH/BPD/81084/2011.
3 Installing ARoME Library

3.1 Installation on a Unix-like system (Linux, Mac OS X, ...)

You need a C compiler, such as gcc. A fortran compiler, compliant with the ANSI Fortran 77 specifications, is required to compile the fortran-77/90/95 interface of the library. A fortran compiler, compliant with the Fortran 2003 specifications, is required to compile the fortran-2003 interface of the library. And you need a standard Unix `make` program, plus some other standard Unix utility programs.

Here are the steps needed to install the library on Unix systems:

1. `tar xzf arome-1.0.0.tar.gz`
2. `cd arome-1.0.0`
3. `./configure`

   Running `configure` might take a while. While running, it prints some messages telling which features it is checking for.

   `configure` recognizes the following options to control how it operates.

   `--enable-fortran={yes|no}`
   Enable or disable the fortran-77 and fortran-2003 interface. The default is `yes`.

   `--disable-shared`
   Disable shared library.

   `--disable-static`
   Disable static library.

   `--help`
   `-h`
   Print a summary of all of the options to `configure`, and exit.

   `--prefix=dir`
   Use `dir` as the installation prefix. See the command `make install` for the installation names.

   The default compilers could be changed using the variable `CC` for C compiler and `FC` for the Fortran compiler. The default compiler flags could be changed using the variable `CFLAGS` for C compiler and `FCFLAGS` for the Fortran compiler.

4. `make`
   This compiles the ARoME Library in the working directory.

5. `make check`
   This will make sure that the ARoME Library was built correctly.

   If you get error messages, please report them to `arome@astro.up.pt` (See Chapter 4 [Reporting bugs], page 5, for information on what to include in useful bug reports).

6. `make install`
   This will copy the file `arome.h`, `module_arome.mod` and `f90arome.h` to the directory `/usr/local/include`, the file `libarome.a`, `libarome.so` to
the directory ‘/usr/local/lib’, and the file ‘arome.info’ to the directory ‘/usr/local/share/info’ (or if you passed the ‘--prefix’ option to ‘configure’, using the prefix directory given as argument to ‘--prefix’ instead of ‘/usr/local’). Note: you need write permissions on these directories.

3.1.1 Other ‘make’ Targets

There are some other useful make targets:

- ‘arome.info’ or ‘info’
  Create an info version of the manual, in ‘arome.info’.
- ‘arome.pdf’ or ‘pdf’
  Create a PDF version of the manual, in ‘arome.pdf’.
- ‘arome.dvi’ or ‘dvi’
  Create a DVI version of the manual, in ‘arome.dvi’.
- ‘arome.ps’ or ‘ps’
  Create a Postscript version of the manual, in ‘arome.ps’.
- ‘arome.html’ or ‘html’
  Create an HTML version of the manual, in ‘arome.html’.
- ‘clean’
  Delete all object files and archive files, but not the configuration files.
- ‘distclean’
  Delete all files not included in the distribution.
- ‘uninstall’
  Delete all files copied by ‘make install’.
4 Reporting bugs

If you think you have found a bug in the ARoME Library, first have a look on the ARoME Library web page http://www.astro.up.pt/resources/arome, in which case you may find there a workaround for it. Otherwise, please investigate and report it. We have made this library available to you, and it seems very important for us, to ask you to report the bugs that you find.

There are a few things you should think about when you put your bug report together. You have to send us a test case that makes it possible for us to reproduce the bug. Include instructions on the way to run the test case.

You also have to explain what is wrong; if you get a crash, or if the results printed are incorrect and in that case, in what way.

Please include compiler version information in your bug report. This can be extracted using ‘cc -V’ on some machines, or, if you’re using gcc, ‘gcc -v’. Also, include the output from ‘uname -a’ and the ARoME version.

Send your bug report to: arome@astro.up.pt. If you think something in this manual is unclear, or downright incorrect, or if the language needs to be improved, please send a note to the same address.
Chapter 5: ARoME Library Interface

5 ARoME Library Interface

5.1 C Usage

5.1.1 Headers and Libraries
All declarations needed to use ARoME Library are collected in the include file ‘arome.h’. It is designed to work with both C and C++ compilers.
You should include that file in any program using the ARoME library:

```c
#include <arome.h>
```

5.1.1.1 Compilation on a Unix-like system
All programs using ARoME must link against the ‘libarome’ library. On Unix-like system this can be done with ‘-larome’, for example

```c
gcc myprogram.c -o myprogram -larome
```

If ARoME Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.

5.1.2 Constants
AROME_VERSION_MAJOR
This integer constant defines the major revision of this library. It can be used to distinguish different releases of this library.

AROME_VERSION_MINOR
This integer constant defines the minor revision of this library. It can be used to distinguish different releases of this library.

AROME_VERSION_PATCH
This integer constant defines the patch level revision of this library. It can be used to distinguish different releases of this library.

```c
#if (AROME_VERSION_MAJOR>=2)  
  || (AROME_VERSION_MAJOR>=1 && AROME_VERSION_MINOR>=1) 
...  
#endif
```

5.1.3 Types

**t_arome**  
[Data type]  
This type contains all information to compute RM effects. There should be one variable of this type per star.

**t_orbit**  
[Data type]  
This type contains elliptical elements used to compute cartesian coordinates. The orbit structure is not mandatory to compute RM signals, but it may help.
Chapter 5: ARoME Library Interface

5.2 Fortran 2003 Usage

5.2.1 Modules and Libraries
All declarations needed to use ARoME Library are collected in the module files ‘module_arome.mod’. The library is designed to work with Fortran compilers compliant with the Fortran 2003 standard. All declarations use the standard ‘ISO_C_BINDING’ module.

You should include that module in any program using the ARoME library:

    use module_arome

When a fortran string is given as a parameter to a function of this library, you should append this string with ‘//C_NULL_CHAR’ because the C library works only with C string.

5.2.2 Compilation on a Unix-like system
All programs using ARoME must link against the ‘libarome’ library. On Unix-like system this can be done with ‘-larome’, for example

    gfortran -I/usr/local/include myprogram.f -o myprogram -larome

If ARoME Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.

5.3 Fortran 77/90/95 Usage

5.3.1 Headers and Libraries
It is designed to work with Fortran compilers compliant with the Fortran 77, 90 or 95 standard with wrappers. All declarations are implicit, so you should take care about the types of the arguments. All functions are prefixed by ‘f90’. This interface is only provided as compatibility layer and have a small overhead due to the wrappers. So if you have a fortran compiler compliant with 2003 standard, you should use the fortran 2003 interface of this library.

All declarations needed to use ARoME Library are collected in the header file ‘f90arome.h’. It is designed to work with Fortran compilers compliant with the Fortran 77, 90 or 95 standard.

You should include that file in every subroutine or function in any program using the ARoME library:

    include 'f90arome.h'

5.3.2 Compilation on a Unix-like system
All programs using ARoME must link against the ‘libarome’ library. On Unix-like system this can be done with ‘-larome’, for example

    gfortran -I/usr/local/include myprogram.f -o myprogram -larome

If ARoME Library has been installed to a non-standard location then it may be necessary to use ‘-I’ and ‘-L’ compiler options to point to the right directories, and some sort of run-time path for a shared library.
5.4 Single position functions

This section explains how to compute Rossiter McLaughlin (RM) effects at individual positions of the planet with respect to the star. To compute at once an array of RM effects for different positions of the planet, see Section 5.5 [Multiple position functions], page 23. When an error occurs, these functions execute error handlers according to the behavior defined by one of the following functions (see Section 5.6 [Error functions], page 35)

- arome_set_exit_on_error,
- arome_set_continue_on_error,
- arome_set_func_on_error.

5.4.1 Usage

The following examples, that can be founded in the directory ‘examples’ of the library sources, show the typical usage of this group of functions. The example in C language is ‘csingle.c’. The example in Fortran 2003 language is ‘f2003single.f’. The example in Fortran 77/90/95 language is ‘f77single.f’.
#include <stdio.h>
#include "arome.h"

/*-----------------------*/
/* main programme     */
/*-----------------------*/
int main(int argc, char **argv)
{
    double v_CCF, v_iiodine, v_mean;
    double own_f, own_betapR;
    t_arome *parome;
    int status = 0;

    /* planet coordinates in stellar radius */
    double x0 = 0.2;
    double y0 = 0.0;
    double z0 = 5.0;  /* z0 should be positive to have transit */

    /* limb-darkening */
    double u1 = 0.69;
    double u2 = 0.00;

    /* line profile */
    double beta0 = 2.0;  /* width of the non-rotating star */
    double Vsini = 15.0;  /* Vsini */
    double sigma0 = 8.0;  /* width of the best Gaussian fit */
    double zeta = 2.0;  /* macro-turbulence parameter */
    double Rp = 0.1;  /* radius of the planet */
    int Kmax = 4;  /* order of expansion for the Iodine */
    /* cell technique */

    /* allocation of the ARoME structure */
    parome = arome_alloc_quad(u1, u2);

    /* set the lineprofile */
    status +=
        arome_set_lineprofile(beta0, Vsini, sigma0, zeta, Kmax, parome);

    /* set the planet parameters */
    status += arome_set_planet(Rp, parome);

    /* initialize the techniques used in the following */
    status += arome_init_CCF(parome);
    status += arome_init_iiodine(parome);
if (!status)
{
    /* for all planet positions, do the following  --------------------*/
    /* compute the flux, vp and betap at a given position (x0, y0, z0) */
    status += arome_calc_fvpbetap(x0,y0,z0, parome);

    /* you can have access to the computed f, vp, betapR and betapT */
    printf("f = %.15E\n", arome_get_flux(parome) );
    printf("vp = %.15E\n", arome_get_vp(parome) );
    printf("betapR = %.15E\n", arome_get_betapR(parome));
    printf("betapT = %.15E\n", arome_get_betapT(parome));

    /* you can change any of the flux, vp, betapR, or betapT, e.g., */
    own_f = 0.01;
    own_betapR = 15.0; /* km/s */
    arome_set_flux(own_f, parome);
    arome_set_betapR(own_betapR, parome);

    /* then you can compute the RM signal for the technique(s) already */
    /* initialized using error-handling functions with suffix "_e" */
    status += arome_get_RM_CCF_e(parome, &v_CCF);

    /* or simply (without the suffix "_e") */
    v_iodine = arome_get_RM_iodine(parome);
    v_mean = arome_get_RM_mean(parome);

    printf("v_CCF = %.15E\n", v_CCF );
    printf("v_iodine = %.15E\n", v_iodine);
    printf("v_mean = %.15E\n", v_mean );

    /* end of the loop over the positions of the planet -----------------*/
}

/* release memory */
arome_free(parome);

return 0;
}

5.4.2 Functions

5.4.2.1 arome_alloc_quad

t_arome *arome_alloc_quad ( double u1, double u2 )

function arome_alloc_quad ( u1, u2 ) BIND(C)
    REAL(C,DOUBLE), VALUE, intent(in) :: u1, u2
    TYPE(C,PTR) :: arome_alloc_quad
function f90arome_alloc_quad (arome, u1, u2)  
    INTEGER*8, intent(out) :: arome
    DOUBLE PRECISION, intent(in) :: u1, u2
    INTEGER :: f90arome_alloc_quad

This function allocates and initializes an ARoME structure for a star with a quadratic limb-darkening whose coefficients are $u1$ and $u2$. The quadratic limb-darkening law is defined by $I(x,y) = I_0[1 - u_1(1 - \mu) - u_2(1 - \mu)^2]$, where $I_0 = 1/([\pi(1 - u_1/3 - u_2/6)])$ and $\mu = \cos \theta$ is the cosine of the angle between the line of sight and the normal of the stellar surface.

The function arome_free must be called to free the allocated memory by this function once the ARoME structure is not used anymore.

On exit, it returns NULL (0 for the fortran 77/90/95 interface) if an error occurs, otherwise the return value is a non-NULL value.

The arguments are:

$u1$ 1st quadratic limb-darkening coefficient.

$u2$ 2nd quadratic limb-darkening coefficient.

The following example allocates an ARoME structure with a quadratic limb-darkening ($u1=0.5$, $u2=0.0$)

```c
    t_arome *arome;
    arome = arome_alloc_quad(0.5,0.0);
    if (arome)
    {
        /*
        ... computation ...
        */
        /* free memory */
        arome_free(arome);
    }
```

5.4.2.2 arome_alloc_nl

t_arome *arome_alloc_nl (double c1, double c2, int c3, int c4 )  
function arome_alloc_nl (c1, c2, c3, c4 ) BIND(C)  
    REAL(C,DOUBLE), VALUE, intent(in) :: c1, c2, c3, c4
    TYPE(C_PTR) :: arome_alloc_nl

function f90arome_alloc_nl (arome, c1, c2, c3, c4 )  
    INTEGER*8, intent(out) :: arome
    DOUBLE PRECISION, intent(in) :: c1, c2, c3, c4
    INTEGER :: f90arome_alloc_nl
This function allocates and initializes an ARoME structure for a star with a nonlinear limb-darkening whose coefficients are $c_1$, $c_2$, $c_3$, and $c_4$. The nonlinear limb-darkening law is defined by

$$I(x, y) = I_0[1 - \sum_n c_n(1 - \mu^{n/2})],$$

where $I_0 = 1/[\pi(1 - c_1/5 - c_2/3 - 3c_3/7 - c_4/2)]$ and $\mu = \cos \theta$ is the cosine of the angle between the line of sight and the normal of the stellar surface.

The function `arome_free` must be called to free the allocated memory by this function once the ARoME structure is not used anymore.

On exit, it returns NULL (0 for the fortran 77/90/95 interface) if an error occurs, otherwise the return value is a non-NULL value.

The arguments are:

- $c_1$ 1st nonlinear limb-darkening coefficient.
- $c_2$ 2nd nonlinear limb-darkening coefficient.
- $c_3$ 3rd nonlinear limb-darkening coefficient.
- $c_4$ 4th nonlinear limb-darkening coefficient.

The following example allocates an ARoME structure with a nonlinear limb-darkening ($c_1$=0.5, $c_2$=0.0, $c_3$=0.1, $c_4$=0.0)

```c
  t_arome *arome;
  arome = arome_alloc_nl(0.5,0.0,0.1,0.0);
  if (arome)
  { /*
        ... computation ...
    */
    /* free memory */
    arome_free(arome);
  }
```

## 5.4.2.3 `arome_reset_quad`

```c
int arome_reset_quad ( double u1, double u2, t_arome *arome )

function arome_reset_quad ( u1, u2, arome ) BIND(C) [Fortran 2003]

REAL(C_DOUBLE), VALUE, intent(in) :: u1, u2
TYPE(C_PTR), VALUE, intent(in) :: arome
INTEGER(C_INT) :: arome_reset_quad

function f90arome_reset_quad ( u1, u2, arome ) [Fortran 77/90/95]

DOUBLE PRECISION, intent(in) :: u1, u2
INTEGER*8, intent(in) :: arome
INTEGER :: f90arome_reset_quad
```
This function resets the quadratic coefficients of an ARoME structure already allocated with \texttt{arome\_alloc\_quad}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{u1} new 1st quadratic limb-darkening coefficient.
- \texttt{u2} new 2nd quadratic limb-darkening coefficient.
- \texttt{arome} ARoME structure.

\textbf{5.4.2.4 \texttt{arome\_reset\_nl}}

\begin{verbatim}
int arome_reset_nl ( double c1, double c2, double c3, double c4, t_arome *arome )

function arome_reset_nl ( c1, c2, c3, c4, arome ) BIND(C)
    REAL(C\_DOUBLE), VALUE, intent(in) :: c1, c2, c3, c4
    TYPE(C\_PTR), VALUE, intent(in) :: arome
    INTEGER(C\_INT) :: arome\_reset\_nl

function f90arome_reset_nl ( c1, c2, c3, c4, arome ) [Fortran 77/90/95]
    DOUBLE PRECISION, intent(in) :: c1, c2, c3, c4
    INTEGER*8, intent(in) :: arome
    INTEGER :: f90arome\_reset\_nl
\end{verbatim}

This function resets the nonlinear coefficients of an ARoME structure already allocated with \texttt{arome\_alloc\_nl}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{c1} new 1st nonlinear limb-darkening coefficient.
- \texttt{c2} new 2nd nonlinear limb-darkening coefficient.
- \texttt{c3} new 3rd nonlinear limb-darkening coefficient.
- \texttt{c4} new 4th nonlinear limb-darkening coefficient.
- \texttt{arome} ARoME structure.

\textbf{5.4.2.5 \texttt{arome\_set\_lineprofile}}

\begin{verbatim}
int arome_set_lineprofile ( double beta0, double Vsini, double sigma0, double zeta, int Kmax, t_arome *arome )

function arome_set_lineprofile ( beta0, Vsini, sigma0, zeta, Kmax, arome ) BIND(C)
    REAL(C\_DOUBLE), VALUE, intent(in) :: beta0, Vsini, sigma0, zeta
\end{verbatim}
INTEGER(C_INT), VALUE, intent(in) :: Kmax
TYPE(C_PTR), VALUE, intent(in) :: arome
INTEGER(C_INT) :: arome

function f90arome_set_lineprofile ( beta0, Vsini, 
  sigma0, zeta, Kmax, arome )
  DOUBLE PRECISION, intent(in) :: beta0, Vsini, sigma0, zeta
  INTEGER, intent(in) :: Kmax
  INTEGER*8, intent(in) :: arome
  INTEGER :: arome

This function sets the parameters of the observed line profiles. It should be called at least once after the allocation of an ARoME structure and before computing Rossiter-McLaughlin quantities with arome_calc_fvpbetap. If this function is called several times, only the last call is taken into account.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

* `beta0` width of the spectral lines of the non-rotating star.
* `Vsini` projected rotation velocity of the star.
* `sigma0` width of the best Gaussian fit to the cross-correlation function. Only used by arome_get_RM_CCF or arome_get_RM_CCF_e.
* `zeta` macro-turbulence parameter.
* `Kmax` order of expansion in the computation of the RM effect measured by the iodine cell technique. Kmax=6 is usually enough. This parameter is only used by arome_get_RM_iodine or arome_get_RM_iodine_e.
* `arome` ARoME structure.

5.4.2.6 arome_set_planet

int arome_set_planet ( double Rp, t_arome *arome )

function arome_set_planet ( Rp, arome ) BIND(C)
  REAL(C_DOUBLE), VALUE, intent(in) :: Rp
  TYPE(C_PTR), VALUE, intent(in) :: arome
  INTEGER(C_INT) :: arome

function f90arome_set_planet ( Rp, arome )
  DOUBLE PRECISION, intent(in) :: Rp
  INTEGER*8, intent(in) :: arome
  INTEGER :: f90arome

This function sets the parameters of the planet. It should be called at least once after the allocation of an ARoME structure and before computing Rossiter-McLaughlin quantities.
with `arome_calc_fvpbetap`. If this function is called several times, only the last call is taken into account.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $Rp$ radius of the planet in units of the stellar radius.
- `arome` ARoME structure.

### 5.4.2.7 `arome_init_CCF`

```c
int arome_init_CCF ( t_arome *arome ) [C]
```

```fortran
function arome_init_CCF ( arome ) BIND (C) [Fortran 2003]
  TYPE(C_PTR), VALUE, intent(in) :: arome
  INTEGER(C_INT) :: arome_init_CCF
```

```fortran
function f90arome_init_CCF ( arome ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  INTEGER :: f90arome_init_CCF
```

This function initializes some parameters used to compute the RM effect as measured by the Gaussian fit of a cross-correlation. It needs to be called only if `arome_get_RM_CCF` or `arome_get_RM_CCF_e` is used. In that case, it must be called once after setting the line profiles with `arome_set_lineprofile` and the planet with `arome_set_planet`, and before computing the RM effect.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- `arome` ARoME structure.

### 5.4.2.8 `arome_init_iodine`

```c
int arome_init_iodine ( t_arome *arome ) [C]
```

```fortran
function arome_init_iodine ( arome ) BIND (C) [Fortran 2003]
  TYPE(C_PTR), VALUE, intent(in) :: arome
  INTEGER(C_INT) :: arome_init_iodine
```

```fortran
function f90arome_init_iodine ( arome ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  INTEGER :: f90arome_init_iodine
```

This function initializes some parameters used to compute the RM effect as measured by the iodine cell technique. It needs to be called only if `arome_get_RM_iodine` or `arome_get_RM_iodine_e` is used. In that case, it must be called once after setting the line profiles with...
arome_set_lineprofile and the planet with arome_set_planet, and before computing the RM effect.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

arome  ARoME structure.

5.4.2.9 arome_calc_fvpbetap

int arome_calc_fvpbetap ( double x, double y, double z, t_arome *arome )  [C]

function arome_calc_fvpbetap ( x, y, z, arome ) BIND(C)  [Fortran 2003]
REAL(C_DOUBLE), VALUE, intent(in) :: x, y, z
TYPE(C_PTR), VALUE, intent(in) :: arome
INTEGER :: arome_calc_fvpbetap

function f90arome_calc_fvpbetap ( x, y, z, arome )  [Fortran 77/90/95]
DOUBLE PRECISION, intent(in) :: x, y, z
INTEGER*8, intent(in) :: arome
INTEGER :: f90arome_calc_fvpbetap

This function computes the flux fraction \( f \) occulted by the planet, the subplanet velocity \( v_p \), and the widths of the subplanet line profile \( \beta_R \) and \( \beta_T \), associated to the radial and the tangential macro-turbulence, respectively. \((x, y, z)\) are the coordinates of the planet with respect to the star. This function can be called any number of times once the line profile and the planet are set with arome_set_lineprofile and arome_set_planet. The values of \( f, v_p, \beta_R, \) and \( \beta_T \) computed with arome_calc_fvpbetap can be recovered using arome_get_flux, arome_get_vp, arome_get_betapR, and arome_get_betapT, respectively. These quantities can also be set or modified by the user with arome_set_flux, arome_set_vp, arome_set_betapR, and arome_set_betapT.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

\( x \)  \( x \) coordinate of the planet in units of stellar radius.

\( y \)  \( y \) coordinate of the planet in units of stellar radius.

\( z \)  \( z \) coordinate of the planet in units of stellar radius.

arome  ARoME structure.

5.4.2.10 arome_set_flux

int arome_set_flux ( double f, t_arome *arome )  [C]

function arome_set_flux ( f, arome ) BIND(C)  [Fortran 2003]
REAL(C_DOUBLE), VALUE, intent(in) :: f
TYPE(C_PTR), VALUE, intent(in) :: arome
INTEGER(C_INT) :: arome_set_flux
function f90arome_set_flux (f, arome)  
    DOUBLE PRECISION, intent(in) :: f  
    INTEGER*8, intent(in) :: arome  
    INTEGER :: f90arome_set_flux

This function sets the flux fraction $f$ occulted by the planet, or modifies the value previously computed with arome_calc_fvpbetap.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $f$ flux fraction occulted by the planet.
- arome ARoME structure.

### 5.4.2.11 arome_set_vp

int arome_set_vp (double vp, t_arome *arome)  

function arome_set_vp (vp, arome) BIND(C)  
    REAL(C_DOUBLE), VALUE, intent(in) :: vp  
    TYPE(C_PTR), VALUE, intent(in) :: arome  
    INTEGER(C_INT) :: arome_set_vp

This function sets the subplanet velocity $v_p$, or modifies the value previously computed with arome_calc_fvpbetap.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- vp subplanet velocity in km/s.
- arome ARoME structure.

### 5.4.2.12 arome_set_betapR

int arome_set_betapR (double betaR, t_arome *arome)  

function arome_set_betapR (betaR, arome) BIND(C)  
    REAL(C_DOUBLE), VALUE, intent(in) :: betaR  
    TYPE(C_PTR), VALUE, intent(in) :: arome  
    INTEGER(C_INT) :: arome_set_betapR
function f90arome_set_betapR ( betaR, arome )  
    DOUBLE PRECISION, intent(in) :: betaR 
    INTEGER*8, intent(in) :: arome 
    INTEGER :: f90arome 

This function sets the width $\beta_R$ of the subplanet line profile associated to the radial macro-turbulence, or modifies the value previously computed with \texttt{arome_calc_fvpbetap}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $\beta_R$ width of the subplanet line profile in km/s associated to the radial macro-turbulence.
- \texttt{arome} ARoME structure.

5.4.2.13 \texttt{arome_set_betapT}

\begin{verbatim}
int arome_set_betapT ( double betaT, t_arome *arome )
function arome_set_betapT ( betaT, arome ) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: arome 
    INTEGER(C_INT) :: arome_set_betapT 

function f90arome_set_betapT ( betaT, arome )
    DOUBLE PRECISION, intent(in) :: betaT 
    INTEGER*8, intent(in) :: arome 
    INTEGER :: f90arome_set_betapT 

\end{verbatim}

This function sets the width $\beta_T$ of the subplanet line profile associated to the tangential macro-turbulence, or modifies the value previously computed with \texttt{arome_calc_fvpbetap}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $\beta_T$ width of the subplanet line profile in km/s associated to the tangential macro-turbulence.
- \texttt{arome} ARoME structure.

5.4.2.14 \texttt{arome_get_flux}

\begin{verbatim}
double arome_get_flux ( t_arome *arome )
function arome_get_flux ( arome ) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: arome 
    REAL(C_DOUBLE) :: arome_get_flux 

\end{verbatim}
function f90arome_get_flux (arome)
    INTEGER*8, intent(in) :: arome
    DOUBLE PRECISION :: f90arome_get_flux

This function returns the flux fraction $f$ occulted by the planet previously computed with arome_calc_fvpbetap or arome_set_flux. If an error occurs, the return value is NaN.

The arguments are:

arome ARoME structure.

5.4.2.15 arome_get_vp

double arome_get_vp (t_arome *arome)

function arome_get_vp (arome) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: arome
    REAL(C_DOUBLE) :: arome_get_vp

function f90arome_get_vp (arome)
    INTEGER*8, intent(in) :: arome
    DOUBLE PRECISION :: f90arome_get_vp

This function returns the subplanet velocity $v_p$ in km/s previously computed with arome_calc_fvpbetap or arome_set_vp. If an error occurs, the return value is NaN.

The arguments are:

arome ARoME structure.

5.4.2.16 arome_get_betapR

double arome_get_betapR (t_arome *arome)

function arome_get_betapR (arome) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: arome
    REAL(C_DOUBLE) :: arome_get_betapR

function f90arome_get_betapR (arome)
    INTEGER*8, intent(in) :: arome
    DOUBLE PRECISION :: f90arome_get_betapR

This function returns the width $\beta_R$ in km/s of the subplanet line profile associated to the radial macro-turbulence previously computed with arome_calc_fvpbetap or arome_set_betapR. If an error occurs, the return value is NaN.

The arguments are:
5.4.2.17 arome_get_betapT

double arome_get_betapT ( t_arome *arome ) [C]

function arome_get_betapT ( arome ) BIND(C) [Fortran 2003]
  TYPE(C_PTR), VALUE, intent(in) :: arome
  REAL(C_DOUBLE) :: arome_get_betapT

function f90arome_get_betapT ( arome ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  DOUBLE PRECISION :: f90arome_get_betapT

This function returns the width $\beta_T$ in km/s of the subplanet line profile associated to the radial macro-turbulence previously computed with arome_calc_fvpbetap or arome_set_betapT. If an error occurs, the return value is NaN.

The arguments are:

arome ARoME structure.

5.4.2.18 arome_get_RM_CCF

double arome_get_RM_CCF ( t_arome *arome ) [C]

function arome_get_RM_CCF ( arome ) BIND(C) [Fortran 2003]
  TYPE(C_PTR), VALUE, intent(in) :: arome
  REAL(C_DOUBLE) :: arome_get_RM_CCF

function f90arome_get_RM_CCF ( arome ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  DOUBLE PRECISION :: f90arome_get_RM_CCF

This function returns the value of the RM effect in km/s as measured by the cross-correlation technique. If an error occurs, the return value is NaN. This function must be called only once the flux fraction $f$, the subplanet velocity $v_p$, and the widths $\beta_R$ and $\beta_T$ of the subplanet line profile have been computed or set by the user, and after arome_init_CCF.

The arguments are:

arome ARoME structure.

5.4.2.19 arome_get_RM_CCF_e

int arome_get_RM_CCF_e ( t_arome *arome, double *v_CCF ) [C]

function arome_get_RM_CCF_e ( arome, v_CCF ) BIND(C) [Fortran 2003]
This function is similar to `arome_get_RM_CCF`, except that it enables error handling. It provides the value of the RM effect \( v_{CCF} \) in km/s as measured by the cross-correlation technique. This function must be called only once the flux fraction \( f \), the subplanet velocity \( v_p \), and the widths \( \beta_R \) and \( \beta_T \) of the subplanet line profile have been computed or set by the user, and after `arome_init_CCF`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \( \text{arome} \)  
  ARoME structure.

- \( v_{CCF} \)  
  value of the RM effect in km/s.

### 5.4.2.20 arome_get_RM_iodine

This function returns the value of the RM effect in km/s as measured by the iodine cell technique. If an error occurs, the return value is NaN. This function must be called only once the flux fraction \( f \), the subplanet velocity \( v_p \), and the widths \( \beta_R \) and \( \beta_T \) of the subplanet line profile have been computed or set by the user, and after `arome_init_iodine`.

The arguments are:

- \( \text{arome} \)  
  ARoME structure.

### 5.4.2.21 arome_get_RM_iodine_e

This function returns the value of the RM effect in km/s as measured by the iodine cell technique. If an error occurs, the return value is NaN. This function must be called only once the flux fraction \( f \), the subplanet velocity \( v_p \), and the widths \( \beta_R \) and \( \beta_T \) of the subplanet line profile have been computed or set by the user, and after `arome_init_iodine`.

The arguments are:

- \( \text{arome} \)  
  ARoME structure.
Chapter 5: ARoME Library Interface

TYPE(C_PTR), VALUE, intent(in) :: arome
REAL(C_DOUBLE), intent(out) :: v_iodine
INTEGER(C_INT) :: f90arome_get_RM_iodine_e

function f90arome_get_RM_iodine_e ( arome, v_iodine ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  DOUBLE PRECISION, intent(out) :: v_iodine
  INTEGER :: f90arome_get_RM_iodine_e

This function is similar to arome_get_RM_iodine, except that it enables error handling. It provides the value of the RM effect \( v_{iodine} \) in km/s as measured by the cross-correlation technique. This function must be called only once the flux fraction \( f \), the subplanet velocity \( v_p \), and the widths \( \beta_R \) and \( \beta_T \) of the subplanet line profile have been computed or set by the user, and after arome_init_iodine.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:
- \( arome \) ARoME structure.
- \( v_{iodine} \) value of the RM effect in km/s.

5.4.2.22 arome_get_RM_mean

double arome_get_RM_mean ( t_arome *arome ) [C]

function arome_get_RM_mean ( arome ) BIND(C) [Fortran 2003]
  TYPE(C_PTR), VALUE, intent(in) :: arome
  REAL(C_DOUBLE) :: arome_get_RM_mean

function f90arome_get_RM_mean ( arome ) [Fortran 77/90/95]
  INTEGER*8, intent(in) :: arome
  DOUBLE PRECISION :: f90arome_get_RM_mean

This function returns the value of the RM effect in km/s as measured by the weighted mean. If an error occurs, the return value is NaN. This function must be called only once the flux fraction \( f \) and the subplanet velocity \( v_p \) have been computed or set by the user. Unlike arome_get_RM_CCF and arome_get_RM_iodine, this function does not need to be initialized.

The arguments are:
- \( arome \) ARoME structure.

5.4.2.23 arome_get_RM_mean_e

int arome_get_RM_mean_e ( t_arome *arome, double *v_mean ) [C]

function arome_get_RM_iodine_e ( arome, v_mean ) BIND(C) [Fortran 2003]
Chapter 5: ARoME Library Interface

```fortran
function f90arome_get_RM_mean_e ( arome, v_mean )
    INTEGER*8, intent(in) :: arome
    DOUBLE PRECISION, intent(out) :: v_mean
    INTEGER :: f90arome_get_RM_mean_e

This function is similar to arome_get_RM_mean, except that it enables error handling. It provides the value of the RM effect \( v_{\text{mean}} \) in km/s as measured by the weighted mean. This function must be called only once the flux fraction \( f \) and the subplanet velocity \( v_p \) have been computed or set by the user. Unlike arome_get_RM_CCF_e and arome_get_RM_iodine_e, this function does not need to be initialized.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:
- `arome` ARoME structure.
- `v_mean` value of the RM effect in km/s.
```

5.4.2.24 arome_free

```c
void arome_free ( t_arome *arome )
```

```fortran
subroutine arome_free ( arome )
    TYPE(C_PTR), VALUE, intent(in) :: arome

subroutine f90arome_free ( arome )
    INTEGER*8, intent(in) :: arome
```

This function frees allocated memory by any of the functions arome_alloc_quad or arome_alloc_nl.

5.5 Multiple position functions

The following group of functions should be the preferred method to compute arrays of Rossiter-McLaughlin effects. The allocation, initialization, and free functions are the same as in the single position group (see Section 5.4 [Single position functions], page 8). The others have a prefix "arome_m" and are detailed in the following subsubsections.

When an error occurs, these functions execute error handlers according to the behavior defined by one of the following functions (see Section 5.6 [Error functions], page 35).

- `arome_set_exit_on_error`
- `arome_set_continue_on_error`
- `arome_set_func_on_error`
5.5.1 Usage

The following examples, that can be founded in the directory ‘examples’ of the library sources, show the typical usage of this group of functions. The example in C language is ‘cmultiple.c’. The example in Fortran 2003 language is ‘f2003multiple.f’. The example in Fortran 77/90/95 language is ‘f77multiple.f’.
program f2003multiple
use module_arome
implicit none
real(8), parameter :: pi = acos(-1.0d0)
integer, parameter :: ncoord = 20
real(8) :: c1, c2, c3, c4, beta0, Vsini, sigma0, zeta, Rp
real(8) :: sma, inc, lambda, anom
integer :: Kmax
real(8) :: M_init, M_end
real(8) :: tab_anom(ncoord)
real(8) :: tab_x(ncoord), tab_y(ncoord), tab_z(ncoord)
real(8) :: tab_f(ncoord), tab_vp(ncoord)
real(8) :: tab_betapR(ncoord), tab_betapT(ncoord)
real(8) :: tab_v_CCF(ncoord)
real(8) :: tab_v_iodine(ncoord)
real(8) :: tab_v_mean(ncoord)
type(C_PTR) :: arome
integer :: nerror, k

nerror = 0

!/* planet orbital parameters */
sma = 4.0d0 !/* stellar radii */
inc = 86.0d0*pi/180.0d0 !/* radian */

!/* spin-orbit angle */
lambda = 30.0d0*pi/180.0d0 !/* radian */

!/* Mean anomaly */
M_init = 70.0d0*pi/180.0d0
M_end = 110.0d0*pi/180.0d0

!/* planet’s coordinates */
do k=1,ncoord
   anom = (M_end-M_init)*(k-1.0)/(ncoord-1.0d0)+M_init
   tab_anom(k) = anom
   tab_x(k) = sma*(-cos(lambda)*cos(anom)+sin(lambda)*sin(anom)*cos(inc))
   tab_y(k) = sma*(-sin(lambda)*cos(anom)-cos(lambda)*sin(anom)*cos(inc))
   tab_z(k) = sma*sin(anom)*sin(inc)
end do

!/* limb-darkening */
c1 = 0.701

c2 = 0.149d0

c3 = 0.277

c4 = -0.297d0
Chapter 5: ARoME Library Interface

/* line profile */
beta0 = 2.0d0 !/* width of the non-rotating star */
Vsini = 15.0d0 !/* Vsini */
sigma0 = 8.0d0 !/* width of the best Gaussian fit */
zeta = 2.0d0 !/* macro-turbulence parameter */
Rp = 0.1d0 !/* radius of the planet */
Kmax = 4 !/* order of expansion for the Iodine technique */

/* allocation of the ARoME structure */
arome = arome_alloc_nl(c1, c2, c3, c4)

/* set the lineprofile */
nerror = arome_set_lineprofile(beta0, Vsini, sigma0, &
& zeta, Kmax, arome)

/* set the planet parameters */
nerror = arome_set_planet(Rp, arome)

/* initialize the techniques used in the following */
nerror = arome_init_CCF(arome)
nerror = arome_init_iodine(arome)

/* allocate memory for vectorial routines */
nerror = arome_malloc(ncoord, arome)

/* compute the flux, vp and betap */
nerror = arome_mcalc_fvpbetap(tab_x, tab_y, tab_z, &
& ncoord, arome)

/* recover the computed flux, subplanet vp, betapR, and betapT */
nerror = arome_mget_flux(arome, ncoord, tab_f)
nerror = arome_mget_vp(arome, ncoord, tab_vp)
nerror = arome_mget_betapR(arome, ncoord, tab_betapR)
nerror = arome_mget_betapT(arome, ncoord, tab_betapT)

/* get the RM signals */
nerror = arome_mget_RM_CCF(arome, ncoord, tab_v_CCF)
nerror = arome_mget_RM_iodine(arome, ncoord, tab_v_iodine)
nerror = arome_mget_RM_mean(arome, ncoord, tab_v_mean)

/* print the results */
do k=1,ncoord
    print '(SS,F9.6,SP,F9.4,F9.4,F9.4,SS,F9.6,SP,
&
& tab_anom(k)/(2.0d0*pi), tab_x(k), tab_y(k), tab_z(k), &
& tab_f(k), tab_vp(k), tab_betapR(k), tab_betapT(k), &
& tab_v_CCF(k), tab_v_iodine(k), tab_v_mean(k)
end do
Chapter 5: ARoME Library Interface

5.5.2 Functions

5.5.2.1 arome_malloc

```c
int arome_malloc ( int n, t_arome *arome )
```

This function allocates memory for computing multiple values of RM effects. It should be called after, and in addition to, `arome_alloc_quad` or `arome_alloc_nl`. The allocated memory by this function should be released with `arome_mfree`. On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0. The arguments are:

- `n` number of positions at which the RM effect will be computed.
- `arome` ARoME structure.

5.5.2.2 arome_mcalc_fvpbetap

```c
int arome_mcalc_fvpbetap ( double x[], double y[], double z[], int n, t_arome *arome )
```

This function computes multiple values of FVP Betap effects. It should be called after, and in addition to, `arome_alloc_quad` or `arome_alloc_nl`. The allocated memory by this function should be released with `arome_mfree`. On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0. The arguments are:

- `n` number of positions at which the RM effect will be computed.
- `arome` ARoME structure.
This function computes the flux fraction $f$ occulted by the planet, the subplanet velocity $v_p$, and the widths of the subplanet line profile $\beta_R$ and $\beta_T$ associated to the radial and the tangential macro-turbulence at each of the $n$ positions of the planet, respectively. $(x, y, z)$ are the coordinates of the planet with respect to the star. This function can be called once the line profile and the planet are set with `arome_set_lineprofile` and `arome_set_planet`, and after allocating memory with `arome_malloc`. The values of $f$, $v_p$, $\beta_R$, and $\beta_T$ computed with `arome_mcalc_fvpbetap` can be recovered using `arome_mget_flux`, `arome_mget_vp`, `arome_mget_betapR`, and `arome_mget_betapT`, respectively. These quantities can also be set or modified by the user with `arome_mset_flux`, `arome_mset_vp`, `arome_mset_betapR`, and `arome_mset_betapT`.

The number of positions $n$ in `arome_mcalc_fvpbetap` should be the same as in the call of `arome_malloc`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $x$ array of x coordinates of the planet in units of stellar radius.
- $y$ array of y coordinates of the planet in units of stellar radius.
- $z$ array of z coordinates of the planet in units of stellar radius.
- $n$ number of planet positions.
- `arome` ARoME structure.

### 5.5.2.3 `arome_mset_flux`

```c
int arome_mset_flux ( double f[], int n, t_arome *arome )
```

```fortran
function arome_mset_flux ( f, n, arome ) BIND(C)
    REAL(C_DOUBLE), dimension(n) :: f
    INTEGER(C_INT), VALUE, intent(in) :: n
    TYPE(C_PTR), VALUE, intent(in) :: arome
    INTEGER(C_INT) :: arome
```

This function sets the flux fraction $f$ occulted by the planet, or modifies the value previously computed with `arome_mcalc_fvpbetap`.

The number of positions $n$ in `arome_mset_flux` should be the same as in the call of `arome_malloc`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $f$ array of double precision values.
- $n$ number of positions.
- `arome` ARoME structure.
Chapter 5: ARoME Library Interface

5.5.2.4 arome_mset_vp

This function sets the subplanet velocity \( v_p \), or modifies the value previously computed with arome_mcalc_fvpbetap.

The number of positions \( n \) in arome_mset_vp should be the same as in the call of arome_malloc.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \( \text{vp} \) array of subplanet velocities in km/s.
- \( n \) number of planet positions.
- \( \text{arome} \) ARoME structure.

5.5.2.5 arome_mset_betapR
This function sets the width $\beta_R$ of the subplanet line profile associated to the radial macro-turbulence, or modifies the value previously computed with `arome_mcalc_fvpbetap`.

The number of positions $n$ in `arome_mset_betapR` should be the same as in the call of `arome_malloc`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $\beta_R$ array of widths of the subplanet line profile in km/s associated to the radial macro-turbulence.
- $n$ number of planet positions.
- `arome` ARoME structure.

### 5.5.2.6 `arome_mset_betapT` 

```c
int arome_mset_betapT ( double betaT[], n, t_arome *arome )
```

```fortran
function arome_mset_betapT ( betaT, n, arome ) BIND(C)
```

This function sets the width $\beta_T$ of the subplanet line profile associated to the tangential macro-turbulence, or modifies the value previously computed with `arome_mcalc_fvpbetap`.

The number of positions $n$ in `arome_mset_betapT` should be the same as in the call of `arome_malloc`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- $\beta_T$ array of widths of the subplanet line profile in km/s associated to the tangential macro-turbulence.
- $n$ number of planet positions.
- `arome` ARoME structure.

### 5.5.2.7 `arome_mget_flux` 

```c
int arome_mget_flux ( t_arome *arome, int n, double f[] )
```

```fortran
function arome_mget_flux ( arome, n, f ) BIND(C)
```

This function retrieves the flux array $f$ at $n$ positions, given an `ARoME` structure `arome`.
This function returns the flux fraction \( f \) occulted by the planet previously computed with \texttt{arome_mcalc_fvpbetap} or \texttt{arome_mset_flux}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{arome} ARoME structure.
- \texttt{n} number of planet positions.
- \texttt{f} output array which will contain the flux at each position.

### 5.5.2.8 arome_mget_vp

This function returns the subplanet velocity \( v_p \) in km/s previously computed with \texttt{arome_mcalc_fvpbetap} or \texttt{arome_mset_vp}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{arome} ARoME structure.
- \texttt{n} number of planet positions.
- \texttt{vp} output array which will contain the subplanet velocity at each position.
5.5.2.9 arome_mget_betapR

```c
int arome_mget_betapR ( t_arome *arome, int n, double betaR[] )
```

```fortran
function arome_mget_betapR ( arome, n, betaR ) BIND(C)
  TYPE(C_PTR), VALUE, intent(in) :: arome
  INTEGER(C_INT), VALUE, intent(in) :: n
  REAL(C_DOUBLE), dimension(n) :: betaR
  INTEGER(C_INT) :: arome_mget_betapR
endfunction
```

This function returns the width \( \beta_R \) in km/s of the subplanet line profile associated to the radial macro-turbulence previously computed with `arome_mcalc_fvpbetap` or `arome_mset_betapR`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- `arome` ARoME structure.
- `n` number of planet positions.
- `betaR` output array which will contain the width \( \beta_R \) at each position.

5.5.2.10 arome_mget_betapT

```c
int arome_mget_betapT ( t_arome *arome, int n, double betaT[] )
```

```fortran
function arome_mget_betapT ( arome, n, betaT ) BIND(C)
  TYPE(C_PTR), VALUE, intent(in) :: arome
  INTEGER(C_INT), VALUE, intent(in) :: n
  REAL(C_DOUBLE), dimension(n) :: betaT
  INTEGER(C_INT) :: arome_mget_betapT
endfunction
```

This function returns the width \( \beta_T \) in km/s of the subplanet line profile associated to the tangential macro-turbulence previously computed with `arome_mcalc_fvpbetap` or `arome_mset_betapT`.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- `arome` ARoME structure.
- `n` number of planet positions.
- `betaT` output array which will contain the width \( \beta_T \) at each position.
ARoME structure.

number of planet positions.

output array which will contain the width $\beta_T$ at each position.

5.5.2.11 arome_mget_RM_CCF

This function returns the value of the RM effect in km/s as measured by the cross-correlation technique. This function must be called only once the flux fraction $f$, the subplanet velocity $v_p$, and the widths $\beta_R$ and $\beta_T$ of the subplanet line profile have been computed or set by the user, and after arome_init_CCF.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

ARoME structure.

number of planet positions.

output array which will contain the value of the RM effect at each position.

5.5.2.12 arome_mget_RM_iodine

This function returns the value of the RM effect in km/s as measured by the cross-correlation technique. This function must be called only once the flux fraction $f$, the subplanet velocity $v_p$, and the widths $\beta_R$ and $\beta_T$ of the subplanet line profile have been computed or set by the user, and after arome_init_CCF.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

ARoME structure.

number of planet positions.

output array which will contain the value of the RM effect at each position.
This function returns the value of the RM effect in km/s as measured by the iodine cell technique. This function must be called only once the flux fraction $f$, the subplanet velocity $v_p$, and the widths $\beta_R$ and $\beta_T$ of the subplanet line profile have been computed or set by the user, and after \texttt{arome\_init\_iodine}.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{arome} \quad ARoME structure.
- \texttt{n} \quad number of planet positions.
- \texttt{v\_iodine} \quad output array which will contain the value of the RM effect at each position.

5.5.2.13 \texttt{arome\_mget\_RM\_mean}

\begin{verbatim}
int arome_mget_RM_mean ( t_arome *arome, int n, double v_mean[] )
function arome_mget_RM_mean ( arome, n, v_mean ) BIND(C)
    TYPE(C_PTR), VALUE, intent(in) :: arome
    INTEGER(C_INT), VALUE, intent(in) :: n
    REAL(C_DOUBLE), dimension(n) :: v_mean
    INTEGER(C_INT) :: arome_mget_RM_mean

function f90arome_mget_RM_mean ( arome, n, v_mean )
    INTEGER*8, intent(in) :: arome
    INTEGER, intent(in) :: n
    DOUBLE PRECISION :: v_mean(n)
    INTEGER :: f90arome_mget_RM_mean
\end{verbatim}

This function returns the value of the RM effect in km/s as measured by the weighted mean. This function must be called only once the flux fraction $f$ and the subplanet velocity $v_p$ have been computed or set by the user.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \texttt{arome} \quad ARoME structure.
- \texttt{n} \quad number of planet positions.
- \texttt{v\_mean} \quad output array which will contain the value of the RM effect at each position.
5.5.2.14 arome_mfree

void arome_mfree ( t_arome *arome )

subroutine arome_mfree ( arome )  BIND(C)  [Fortran 2003]
   TYPE(C_PTR), VALUE, intent(in) :: arome

subroutine f90arome_mfree ( arome )  [Fortran 77/90/95]
   INTEGER*8, intent(in) :: arome

This function releases the memory allocated by arome_malloc. It should be called before,
and in addition to, arome_free.
The argument is :

arome     ARoME structure.

5.6 Error functions

The following group of functions defines the behavior of the library when errors occur during
the execution.

5.6.1 Usage

The following examples, that can be founded in the directory ‘examples’ of the library
sources, show the typical usage of this group of functions. The example in C language is
‘cerror.c’. The example in Fortran 2003 language is ‘f2003error.f’. The example in
Fortran 77/90/95 language is ‘f77error.f’.
The following example shows how to stop the execution on the error with the Fortran 2003 interface.

```fortran
program f2003error
    USE, INTRINSIC :: ISO_C_BINDING
    use module_arome
    implicit none
    type(C_PTR) arome

    ! set the error handler to stop on error
    call arome_set_exit_on_error()

    ! allocate a new ARoME structure with quadratic limb-darkening
    arome = arome_alloc_quad(0.1,0.0)
    ...

    stop
end

! *------------------------------------------------------------------!
! /* custom error handler */
! *------------------------------------------------------------------!

subroutine myhandler(msg, msglen) BIND(C)
    USE, INTRINSIC :: ISO_C_BINDING
    implicit none
    character(kind=C_CHAR), dimension(msglen), intent(in) :: msg
    integer(C_INT), VALUE, intent(in) :: msglen
    write (*,*) "The ARoME calls the function myhandler"
    write (*,*) "The message contains ",msglen," characters"
    write(*,*), "The error message is :">
    write(*,*), "-------------"
    write(*,*), msg
    write(*,*), "-------------"
    write(*,*), "The error handler returns"
end
```
The following example shows how to define a custom error handler function with the Fortran 2003 interface.

```fortran
program f2003error
  USE, INTRINSIC :: ISO_C_BINDING
  use module_arome
  implicit none
  type(C_PTR) arome

  interface
    subroutine myhandler(msg, msglen) BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      implicit none
      character(kind=C_CHAR), dimension(msglen), intent(in) &
      & :: msg
      integer(C_INT), VALUE, intent(in) :: msglen
    end subroutine
  end interface

  ! set the error handler to use my own callback
  call arome_set_func_on_error(c_funloc(myhandler))

  ! allocate a new ARoME structure with quadratic limb-darkening
  arome = arome_alloc_quad(0.1,0.0)
  ....
  stop
end
```

### 5.6.2 Functions

#### 5.6.2.1 `arome_set_continue_on_error`

```c
void arome_set_continue_on_error (void) [C]
```

```fortran
subroutine arome_set_continue_on_error () BIND(C) [Fortran 2003]
```

```fortran
subroutine f90arome_set_continue_on_error () [Fortran 77/90/95]
```

This function sets the behavior of the library to continue when an error occurs during the execution of the library's functions. This is the default behavior, thus this function needs not to be called.

#### 5.6.2.2 `arome_set_exit_on_error`

```c
void arome_set_exit_on_error (void) [C]
```
This function sets the behavior of the library to exit when an error occurs during the execution of the library’s functions. This function should be called (not mandatory) before any other functions of the library.

5.6.2.3 arome_set_func_on_error

This function sets the behavior of the library to call the user function userfunc when an error occurs during the execution of the library’s functions. This function should be (not mandatory) called before any other functions of the library.

The function userfunc must be defined as

With Fortran 2003 interface, this function must have an explicit interface. With fortran 77/90/95 interface, this function must be declared as EXTERNAL.

5.7 Orbit computation functions

The following group of functions is not dedicated to the Rossiter-McLaughlin effect alone. It is designed to compute planet cartesian coordinates for given sets of orbital elliptical elements. These routines are thus not mandatory to compute the Rossiter-McLaughlin effect, but they may help since they provide the coordinates $x$, $y$, and $z$ required by the functions arome_calc_fvpbetap and arome_mcalc_fvpbetap.

5.7.1 Usage

The following examples, that can be founded in the directory ‘examples’ of the library sources, show the typical usage of this group of functions. The example in C language is ‘corbit.c’. The example in Fortran 2003 language is ‘f2003orbit.f’. The example in Fortran 77/90/95 language is ‘f77orbit.f’.
The following example shows how to stop the execution on the error with the Fortran 2003 interface.

```fortran
program f2003orbit
  USE, INTRINSIC :: ISO_C_BINDING
  use module_arome
  implicit none
  type(C_PTR) orbit1
  integer nerror
  real(8) x, y, z

  nerror = 0

  ! create a new orbit
  orbit1 = arome_new_orbit()

  ! set the orbital parameters (per, sma, ecc, inc, om, lambda)
  nerror = arome_set_orbit_eo(2.d0,4.d0,0.02d0,32.d0, &
      & 82.d0,20.d0,orbit1)

  ! compute planet’s coordinate at time t=0.2
  nerror = arome_get_orbit_xyz(orbit1,0.2d0,x,y,z)

  ! release allocated memory
  arome_free_orbit(orbit1)

  stop
end
```

### 5.7.2 Functions

#### 5.7.2.1 arome_new_orbit

```fortran

**t_orbit *arome_new_orbit (void)**  
[C]

**function arome_new_orbit () BIND(C)**  
[Fortran 2003]
  TYPE(C_PTR) :: arome_new_orbit

**function f90arome_new_orbit (orbit)**  
[Fortran 77/90/95]
  INTEGER*8, intent(out) :: orbit
  INTEGER :: f90arome_new_orbit
```

This function creates a new orbit structure.

The function `arome_free_orbit` must be called to free the allocated memory by this function once the orbit structure is not used anymore.

On exit, it returns `NULL` (0 for the fortran 77/90/95 interface) if an error occurs, otherwise the return value is a non-NULL value.
5.7.2.2 arome_set_orbit_eo

```c
int arome_set_orbit_eo ( double per, double sma, double ecc, double om,
                         double inc, double lambda, t_orbit *orbit )
```

function arome_set_orbit_eo ( per, sma, ecc, om, inc, lambda, orbit ) [Fortran 2003]

```fortran
function arome_set_orbit_eo ( per, sma, ecc, om, inc, lambda, orbit )
    REAL(C_DOUBLE), VALUE, intent(in) :: per, sma, ecc, om, inc, lambda
    TYPE(C_PTR), VALUE, intent(in) :: orbit
    INTEGER(C_INT) :: arome_set_orbit_eo
end function arome_set_orbit_eo
```

function f90arome_set_orbit_eo ( per, sma, ecc, om, inc, lambda, orbit ) [Fortran 77/90/95]

```fortran
function f90arome_set_orbit_eo ( per, sma, ecc, om, inc, lambda, orbit )
    DOUBLE PRECISION, intent(in) :: per, sma, ecc, om, inc, lambda
    INTEGER*8, intent(in) :: orbit
    INTEGER :: f90arome_set_orbit_eo
end function f90arome_set_orbit_eo
```

This function sets the elliptical elements of the orbit. It can be called an arbitrary number of times. Only the last call is taken into account. With this set of elliptical parameters \(e, \omega\), the origin of time is assumed to be at the passage of the pericenter \(t_{\text{peri}}\). Since the parameters \(e, \omega\), and \(t_{\text{peri}}\) are strongly correlated, we encourage people to use the variable \(k, h\), and passage at the ascending node \(t_{\text{node}}\) as required by the function `arome_set_orbit_kh`, instead.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \(per\) orbital period in days.
- \(sma\) semi-major axis in stellar radius.
- \(ecc\) orbital eccentricity.
- \(om\) argument of periastron in degrees.
- \(inc\) orbital inclination w.r.t the plane of the sky in degrees.
- \(lambda\) projected spin-orbit angle in degrees.
- \(orbit\) orbit structure.

5.7.2.3 arome_set_orbit_kh

```c
int arome_set_orbit_kh ( double per, double sma, double k, double h, double inc,
                         double lambda, t_orbit *orbit )
```

function arome_set_orbit_kh ( per, sma, k, h, inc, lambda, orbit ) [Fortran 2003]

```fortran
function arome_set_orbit_kh ( per, sma, k, h, inc, lambda, orbit )
    REAL(C_DOUBLE), VALUE, intent(in) :: per, sma, k, h, inc, lambda
    TYPE(C_PTR), VALUE, intent(in) :: orbit
    INTEGER(C_INT) :: arome_set_orbit_kh
end function arome_set_orbit_kh
```
function f90arome_set_orbit_kh (per, sma, k, h, inc, lambda, orbit)  
  DOUBLE PRECISION, intent(in) :: per, sma, k, h, inc, lambda  
  INTEGER*8, intent(in) :: orbit  
  INTEGER :: f90arome_set_orbit_kh

This function sets the elliptical elements of the orbit. It can be called an arbitrary number of times. Only the last call is taken into account. With this set of elliptical parameters \((k, h)\), the origin of time is assumed to be the passage at the ascending node \(t_{\text{node}}\). This function should be preferred with respect to arome_set_orbit_eo since \(k, h\), and \(t_{\text{node}}\) are independent while \(e, \omega\), and the passage at the pericenter are not.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \(per\) orbital period in days.
- \(sma\) semi-major axis in stellar radius.
- \(k\) \(e \cos \omega\).
- \(h\) \(e \sin \omega\).
- \(inc\) orbital inclination w.r.t the plane of the sky in degrees.
- \(lambda\) projected spin-orbit angle in degrees.
- \(orbit\) orbit structure.

5.7.2.4 arome_get_orbit_transit_time

double arome_get_orbit_transit_time (t_orbit *orbit)  
function arome_get_orbit_transit_time (orbit) BIND(C)  
  TYPE(C_PTR), VALUE, intent(in) :: orbit  
  REAL(C_DOUBLE) :: arome_get_orbit_transit_time

function f90arome_get_orbit_transit_time (orbit)  
  INTEGER*8, intent(in) :: orbit  
  DOUBLE PRECISION :: f90arome_get_orbit_transit_time

This function computes and returns the instant of midtransit modulo the orbital period. The origin of time depends on the function used to set the orbital elliptical elements. If the orbit has been set using arome_set_orbit_eo, the origin of time is the passage at the pericenter, otherwise, if the orbit has been set using arome_set_orbit_kh, the origin of time is the passage at the ascending node.

The arguments are:

- \(orbit\) orbit structure.
5.7.2.5 arome_get_orbit_xyz

```c
int arome_get_orbit_xyz( t_orbit *orbit, double t, double *x, double *y, double *z )
```

```fortran
function arome_get_orbit_xyz( orbit, t, x, y, z ) BIND (C) [Fortran 2003]
    TYPE(C_PTR), VALUE, intent(in) :: orbit
    REAL(C_DOUBLE), VALUE, intent(in) :: t
    REAL(C_DOUBLE), intent(out) :: x, y, z
    INTEGER(C_INT) :: arome_get_orbit_xyz
end function arome_get_orbit_xyz
```

```fortran
function f90arome_get_orbit_xyz( orbit, t, x, y, z ) [Fortran 77/90/95]
    INTEGER*8, intent(in) :: orbit
    DOUBLE PRECISION, intent(in) :: t
    DOUBLE PRECISION, intent(out) :: x, y, z
    INTEGER :: f90arome_get_orbit_xyz
end function f90arome_get_orbit_xyz
```

This function computes the coordinates \((x, y, z)\) of a planet on the orbit \(orbit\) at the time \(t\). The origin of time \(t_0\) depends on the function used to set the orbital elliptical elements. If the orbit has been set using \(arome_set_orbit_eo\), the origin of time is the passage at the pericenter, otherwise, if the orbit has been set using \(arome_set_orbit_kh\), the origin of time is the passage at the ascending node.

In the case where the dates are more easily defined with respect to the midtransit time, the function \(arome_get_orbit_transit_time\) can be used to compute the midtransit time \(t_{trans}\). Then, \(t + t_{trans}\) is such that \(t_0 = 0\).

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \(orbit\) orbit structure.
- \(t\) date at which the coordinates will be computed.
- \(x\) output \(x\) coordinate in stellar radius.
- \(y\) output \(y\) coordinate in stellar radius.
- \(z\) output \(z\) coordinate in stellar radius.

5.7.2.6 arome_mget_orbit_xyz

```c
int arome_mget_orbit_xyz( t_orbit *orbit, double t[], double n, double x[], double y[], double z[] )
```

```fortran
function arome_mget_orbit_xyz( orbit, t, n, x, y, z ) BIND (C) [Fortran 2003]
    TYPE(C_PTR), VALUE, intent(in) :: orbit
    REAL(C_DOUBLE), dimension(n) :: t
    INTEGER(C_INT), VALUE, intent(in) :: n
    REAL(C_DOUBLE), dimension(n) :: x, y, z
    INTEGER(C_INT) :: arome_mget_orbit_xyz
end function arome_mget_orbit_xyz
```
function f90arome_mget_orbit_xyz (orbit, t, n, x, y, z)  
  INTEGER*8, intent(in) :: orbit
  DOUBLE PRECISION :: t(n)
  INTEGER, intent(in) :: n
  DOUBLE PRECISION :: x(n), y(n), z(n)
  INTEGER :: f90arome_mget_orbit_xyz

This function computes the coordinates (x, y, z) of a planet on the orbit \textit{orbit} for all time \textit{t}. The origin of time \textit{t}_0 depends on the function used to set the orbital elliptical elements. If the orbit has been set using \texttt{arome_set_orbit_eo}, the origin of time is the passage at the pericenter, otherwise, if the orbit has been set using \texttt{arome_set_orbit_kh}, the origin of time is the passage at the ascending node.

In the case where the dates are more easily defined with respect to the midtransit time, the function \texttt{arome_get_orbit_transit_time} can be used to compute the midtransit time \textit{t}_\text{trans}. Then, \textit{t} + \textit{t}_\text{trans} is such that \textit{t}_0 = 0.

On exit, it returns a non-zero value if an error occurs, otherwise the return value is 0.

The arguments are:

- \textit{orbit} orbit structure.
- \textit{t} dates at which the coordinates will be computed.
- \textit{n} number of dates.
- \textit{x} output x coordinates in stellar radius.
- \textit{y} output y coordinates in stellar radius.
- \textit{z} output z coordinates in stellar radius.

5.7.2.7 \texttt{arome_free_orbit}

void arome_free_orbit (t_orbit *orbit)  
[C]

subroutine arome_free_orbit (orbit) BIND(C)  
  TYPE(C_PTR), VALUE, intent(in) :: orbit
[Fortran 2003]

subroutine f90arome_free_orbit (orbit)  
  INTEGER*8, intent(in) :: orbit
[Fortran 77/90/95]

This function frees allocated memory by the function \texttt{arome_new_orbit}.  

Appendix A Release notes

- Version 1.0.0
  Initial release.