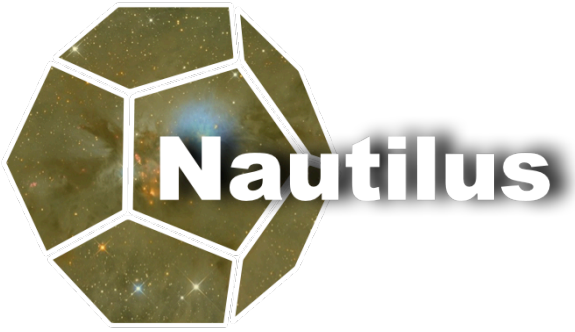

Nautilus Documentation

July 20, 2024

Christophe Cossou, Alexandre Mechineau, Maxime Ruaud, Valentine Wakelam

Contents

1 LICENSE and copyright

The pnautilus gas-grain code has been developed at the *Laboratoire d'astrophysique de Bordeaux* by a group of researchers led by Valentine Wakelam from the original code of Eric Herbst (Hasegawa et al. 1992). Copyright is protected by the following license:

When publishing results obtained with pnautilus (and any modified version of it), the authors should include the reference **Ruaud et al. Monthly Notices of the Royal Astronomical Society, Volume 459, Issue 4, p.3756-3767**. **If we provide the code for external users of the group, we do not provide any user support.** If users want help, then they can contact us, but this will be done in the context of a collaboration.

2 Historic of modifications

March 2022, V. Wakelam: Major modifications of nautilus:

- Changing the parameter files to remove some parameters and add a new one (is_chem_des to choose between Garrod and Minissale's formalism for the chemical desorption).
- For Minissale's formalism, there are two possible models: one for bare grains and one for water mantles. We added an automatic switch to switch from one to the other when the number of monolayers of molecules on grains is 4.
- Diffusion induced by cosmic-rays (processes added by a former student L. Reboussin (see Reboussin et al. 2014) was removed because it was not working any more).
- Modified rates from Caselli et al. (1992?) was also removed because it was obsolete and not working anyway.
- For the diffusion by tunnelling effect, we have simplified by only keeping 3 cases (see later in the document for explanations). We realized that one of the cases was not working.
- The sputtering of surface + mantle by cosmic-rays from Dartois et al. For the moment, the conservative yields for water ices has been included. This is the new type of reaction 77. The reactions need to be included in the chemical files.
- If the code is in 1D, then the cosmic-ray ionisation rate ζ is now read in 1D_static.dat.
- The directory example_simulations now possesses the recommended parameters and the latest chemical files.

3 Introduction

Whether you are a developer or a user, you do not have the same expectations about this manual.

Be sure to carefully read the section about the **installation** [§ 4].

Chemical networks are presented in [§ 8 on page 17]. In this section, you will be able to learn about the various types of reactions available and their format. A chemical network is given with the code, but you can use your own.

Input files are presented in [§ 9 on page 18], but the most important one, the only one you will modify daily (*parameters.in*) is dealt within [§ 7 on page 10].

Output files are presented in [§ 10 on page 19].

Simulation's information are displayed in *info.out* (see [§ 10.1 on page 19]).

[§ 11 on page 20] explain what are the Python scripts that you can use to plot useful information (mainly abundances) of your simulations (single plot or comparison between several runs).

One last section [§ 12 on page 21] presents technical specifications of the code and how to maintain it.

4 Installation

4.1 Prerequisites

You will need 3 tools to be able to use Nautilus:

- Git (To fetch the code)
- Python
- Fortran compiler (we have tested only gfortran, but others should work as well)

4.2 Getting the Git repository

To start, you need to fetch the latest version of nautilus using *git*.

```
git clone https://forge.oasu.u-bordeaux.fr/LAB/astrochem-tools/pnautilus.git Nautilus
cd Nautilus
```

At this point, we assume that your current directory will be `pnautilus`. This current directory will be referenced as **NAUTILUS_DIR**. You can get the complete path with *pwd*.

In your *.bashrc*, you can use the the content of the previous command such as:

```
export NAUTILUS_DIR=
```

On an existing repository, be sure to pull once in a while our *updates* if they exist.

```
git pull
```

We made multiple change on the git repository, if `git pull` don't produce the expected result, you may want to clone the repository to start from scratch.

4.3 Compilation

Now, we create a python environment to install the build tools and the nautilus scripts.

```
python -m venv .venv
source .venv/bin/activate
```

/.venv

 The virtual environment will be located at:

```
$NAUTILUS_DIR/.venv
```

Finally install the python package declared by nautilus:

```
pip install ".[build-tools]"
```

This will give you access to the post processing scripts and the required build tools.

To build the code, we use a tool named `CMake`, this tools does what is called an out-of-source build. It is the method of building the code in another directory than the one where the code is.

```
mkdir build && cd build
```

In this directory, we are going to choose the option that we want to use to compile/install the code.

```
cmake  -G Ninja \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_Fortran_COMPILER=gfortran \
      -DCMAKE_INSTALL_PREFIX=.. \
      ..
```

Simply the previous command tells `cmake` that you want to build the code in release mode, while using `gfortran` as a compiler. Next, we configure the installation prefix, i.e., where we want cmake to create the `bin` directory during the installation. Here, the bin directory will be in the nautilus directory. Finally, we specify where is the code that we want to build.

Now, build the code with the following:

```
cmake --build .
```

Then, to not have to search for the built binaries, we use cmake again to "install" our code.

```
cmake --install . -v
```

This will create a `bin` directory in the directory specified by `-DCMAKE\_INSTALL\_PREFIX`.

4.4 Binaries

The various binaries available in the directory bin/ are:

- *nautilus_code*
- *nautilus_outputs*
- *nautilus_rates*
- *nautilus_major_reactions*

 To be able to call from any directory the previous binaries, you will need to add to your **PATH** the path to the `bin` directory.

```
export PATH=<CMAKE_INSTALL_PREFIX>/bin:$PATH
```

Where `<CMAKE\_INSTALL\_PREFIX>` corresponds to the absolute path to the directory that you used to configure cmake.

`nautilus_code` is the main program, the one computing the evolution of the chemical scheme. It needs several input files described in [§ 9 on page 18]. This program will generate binary outputs that are not readable by default.

`nautilus_outputs` will read the binary outputs of `nautilus_code` to generate the corresponding ASCII outputs. More details on [§ 10.2 on page 19].

Using the ASCII outputs of `nautilus_outputs`, `nautilus_rates` generates two data files. 'rates.out' contains the fluxes of all reactions (rate coefficients times densities of the reactants) at each time. `rate_coefficients.out` contains the rate coefficients at each time.

To have a user-friendly interface to find the most crucial reactions depending on some parameters and a given species, use the program `nautilus_major_reactions`. More details on [§ 11.3 on page 20].

Remark : All the programs must be run in the directory of the nautilus simulation. First *nautilus_code*. Then *nautilus_outputs*. And then the two others if you want.

4.5 Python script

Multiple python scripts are provided, to use them you need to install the python package provided. Don't forget to activate this environment when you need to use those scripts.

```
pip install .
```

Remark : As always with Python, it's better to use a virtual environment. (`python -m venv .venv`, `source activate .venv/bin/activate`)

4.6 Examples

4.6.1 Typical install

This example follows the previous explanations

```
cd ~/Documents

git clone https://forge.oasu.u-bordeaux.fr/LAB/astrochem-tools/pnautilus.git
cd pnautilus

python -m venv .venv
source activate .venv/bin/activate
pip install .

mkdir build && cd build

cmake -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_Fortran_COMPILER=gfortran \
      -DCMAKE_INSTALL_PREFIX=.. \
      ..
cmake --build .
cmake --install . -v
```

At this point, you can add *~/Documents/pnautilus/bin* to your **PATH**. You can try the provided example as follows:

```
cd ~/Documents/pnautilus/example_simulation
# Then, run the code:
../bin/nautilus_code # This will run the code by specifying the path to the executable
nautilus_code # or,if the PATH is updated, no need to specify the path at all
```

4.6.2 Custom install

For this example, we want to use a different build type, use the intel compiler and set the installation prefix to */opt/nautilus*.

```
cd ~/Documents

git clone https://forge.oasu.u-bordeaux.fr/LAB/astrochem-tools/pnautilus.git
cd pnautilus
```

```
mkdir build && cd build

cmake -DCMAKE_BUILD_TYPE=Debug \
      -DCMAKE_Fortran_COMPILER=ifx \
      -DCMAKE_INSTALL_PREFIX=/opt/nautilus \
      ..
cmake --build .
cmake --install . -v
```

At this point, you can add */opt/nautilus/bin* to your **PATH**. You can try the provided example as follows:

```
cd ~/Documents/pnautilus/example_simulation
# Then, run the code:
../bin/nautilus_code # This will run the code by specifying the path to the executable
nautilus_code # or,if the PATH is updated, no need to specify the path at all
```

5 Starting with Nautilus

5.1 Generic information

5.1.1 Python scripts help

From all the python scripts that come with the code, you can see all parameters and sometimes a few examples by typing:

```
script_name help
```

5.1.2 Comments in input files

For all input files, the comment character is " !" and can be either at the beginning of a line, or anywhere else. Meaningful character must be comprised inside the 80 first characters of a given line.

5.1.3 Main parameter file

parameters.in is the main parameter file, and is rewritten by the code itself each and every time you run the code.

5.2 Useful tools

5.2.1 Cleaning a simulation folder

The script *nautilus-clean.sh* helps you delete all output files to have a clean simulation folder.

To clean the current working directory, launch:

```
nautilus-clean.sh
```

6 Two and three phases model

Up to now, Nautilus was a 2-phase model only. It means that the surface chemistry was considered the same for all adsorbed species. The two phases where then: gas and grain surface. Maxime Ruaud has extended Nautilus to 3-phase: gas, surface and mantle using the same approach as Hasegawa and Herbst [1993]. The main differences are:

- Species can only absorb on and desorb from the surface.
- The surface is the two first monolayers.
- Species can move slowly in the mantle and fast on the surface.
- Species can swap from the surface to the mantle and conversely.

- The competition between diffusion and reaction has been included (allowing for species to stay longer to react).

In the current version, both 2-phase and 3-phase have the competition between diffusion and reaction included. You can use the 3-phase version. The mantle is chemically active. To have the 2 phases on the surface, a new kind of species has been added: KX species.

7 Parameters_in

For generic information, see [§ 5.1.2 on the previous page].

parameters.in has the particularity to be re-written each time you launch a Nautilus simulation. This ensures several things:

- Parameters can be input in random ways, the code will sort them by categories
- New parameters, with default values will be added, to ensure retro-compatibility.
- A ***.bak** file is created before overwriting **parameters.in**, just in case it erases something important (in comments for instance).

7.1 Automatic test before computation

In *parameters.in*:

```
preliminary_test = 1
```

When set to 1, the parameter *preliminary_test* will allow you to test thoroughly the chemical network and print information in the file *info.out*.

Remark : This parameter should be set to 0 only in the case of an intensive campaign of simulations using the very same network, to avoid wasting computation time doing the same tests. But in any other cases, it is recommended to leave it activated, because it only takes around 1 second at the beginning of the simulation.

The tests currently made are:

- Check that grain species judging from their indexes are indeed grain species.
- Check that all species have production AND destruction reactions (error if none, warning if only one).
- Check that each reaction is balanced in prime elements and in charges.
- Display a warning for each reaction having alpha=0 (first parameter for reaction rate formula).
- Check that $T_{\min} < T_{\max}$ for each reaction.
- Check that each gas neutral species has a grain equivalent (excluding **GRAIN0** and **XH**).
- Check that each gas neutral species has an adsorption reaction (ITYPE=99) (excluding **GRAIN0** and **XH**).
- Check that each grain species has at least one reaction for each of the following types: 15, 16, 66, 67 (desorption reactions).
- Check that all index ranges associated with a reaction type join themselves to cover all the index range of all reactions.

For reaction with the same ID:

- Check that they have the same reactants and products
- Check that temperature ranges do not overlap. It does not check that the T ranges are complementary.

7.2 Simulation parameters

```
is_3_phase = 0 ! 0: 2 phase, 1: 3 phase
```

This switch is used to switch from the 2-phase to the 3-phase mode (see section 6).

```
start_time =  1.000E+00
```

In years, the first output time of the simulation (all simulation starts from $T = 0$).

```
stop_time =  1.000E+01
```

End of the simulation in years (and also the last output time).

```
nb_outputs =    3
```

Total number of outputs (including *start_time* and *stop_time*). This number will be used when *output_type* is **log** or **linear**.

```
output_type = log
```

Define the type of output you want. Possible values are **linear**, **log**, and **table**.

- **linear**: The spacing between the different output times will be linear
- **log**: The different output times will be log-spaced.
- **table**: The different output times are read from the file *structure_evolution.dat*. The parameter *nb_outputs* is then completely ignored.

```
relative_tolerance =  1.000E-04
```

Relative tolerance of the solver.

```
minimum_initial_abundance =  1.000E-40
```

Default minimum initial fraction abundance applied to species whose abundance is not specified in *abundances.in*.

7.3 Time evolution of the physical structure

```
is_structure_evolution = 1
```

If set to 1, the physical structure will evolve with time. This evolution will be read from the file *structure_evolution.dat* that must exist in the simulation folder. The times are read in this file by default and do not need to be regularly spaced.

This file will have the following format:

```
! time    log(Av)    log(n)    log(T)
! (yr)   log(mag)  log(cm-3)  log(K)
0.000e+00 -1.231e+00 1.813e+00 1.698e+00
2.360e-01 -1.233e+00 1.758e+00 1.712e+00
```

We define respectively time, visual extinction, gas density and gas temperature.

Optionally, one can add a 5-th column to define also grain temperature:

```
! time    log(Av)    log(n)    log(Tg)     log(Td)
! (yr)   log(mag)  log(cm-3)  log(K)      log(K)
0.000e+00 -1.231e+00 1.813e+00 1.698e+00 1.500e+00
2.360e-01 -1.233e+00 1.758e+00 1.712e+00 1.510e+00
```

If so, the parameter *grain_temperature_type* must be set to:

```
grain_temperature_type = table_evolv
```

The grain temperature can, however, be also set to the other cases (fixed, gas or computed).

If one does not start the file with time = 0 yr, then the physical conditions of the first line are used at time = 0. While constructing your file *structure_evolution.dat*, one has to keep in mind that the physical parameters will be changed at the time indicated in the file while in the output the physical parameters indicated at a specific time are the ones used to compute the chemical composition. Let's assume the following case:

```
! time    log(Av)    log(n)    log(T)
! (yr)   log(mag)  log(cm-3)  log(K)
1.000e+01  2.000e+00 4.300e+00 1.000e+00
1.000e+02  2.000e+00 5.000e+00 1.500e+00
```

At 100 yr, the model will change the physical conditions and will use log(n)=5 and log(T)=1.5 to compute the chemical composition. But in the output, the physical parameters that will be written at 100 yr will be log(n)=4.3 and log(T)=1.0 because those are the ones that have been used to compute the chemical composition.

 See [§ 7.4] for incompatibilities.

7.4 1D simulations

You can make simulation in 1D:

```
structure_type = 1D_no_diff
```

The 1D physical structure is read in the input file *1D_static.dat* where you define gas density, gas temperature, visual extinction, and the dust temperature with the units indicated below. Format of the file:

```
! Distance [AU] ; H Gas density [part/cm^3] ; Gas Temperature [K] ; Visual Extinction [mag] ;
! Dust Temperature [K] ; 1/abundance of grains ; AV/NH conversion factor ;
! radius of grains (cm) ! zeta_H2 (s-1)
3.353267e+04 5.545700e+02 1.413000e+01 1.850000e+00 0.000000e+00 1.446000e+01 0.000000e+00 0.000000e+00 0.000000e+00 1.07021
3.252265e+04 6.316200e+02 1.418000e+01 1.910000e+00 0.000000e+00 1.441000e+01 0.000000e+00 0.000000e+00 0.000000e+00 1.03494
3.151263e+04 7.222000e+02 1.421000e+01 1.960000e+00 0.000000e+00 1.437000e+01 0.000000e+00 0.000000e+00 0.000000e+00 1.00723
3.050261e+04 8.292200e+02 1.423000e+01 2.030000e+00 0.000000e+00 1.431000e+01 0.000000e+00 0.000000e+00 0.000000e+00 9.70798
```

The format of spacing between numbers does not need to be respected. In the input **parameters.in** file, you have to specify the number of lines by changing the parameter: **spatial __ resolution**. The spatial points do not have to be equally spaced. The code uses the physical structure provided and does not interpolate or extrapolate. To allow for the computation of self-shielding (see sections 7.5 and 3.6), the structure should start from the nearest point of the UV source.

The dust temperature is ruled by *grain_temperature_type* (see [§ 7.6 on the next page] for more details). If something else than table_1D is set, then the dust temperature read in 1D_static.dat is overwritten. Table_evolv is not compatible with this mode.

If the parameter is_dust_1D is set to one in the input **parameters.in** file, some of the grain parameters are read in *1D_static.dat*. Column 6 gives the inverse of the abundance of grains. This is used by the code to compute the number density of grains at each spatial point. Column 7 gives the AV to NH conversion factor (that can depend on the radius of grains and the gas to dust mass ratio). This is used in the code to convert the NH column density into AV for self-shielding (this parameter may change if you are changing the gas to a dust mass ratio). Column 8 gives the radius of the grains in cm. This option was introduced in the code to simulate different gas-to-dust mass ratios and different radius of grains for disk applications. If is_dust_1D is set to 0 then only one value for the gas-to-dust mass ratio and one size of grains is assumed for all the points and given in 1D_static.dat. The last column gives the cosmic-ray ionisation rate. In 1D, ζ needs to be given here.

 1D simulations are not compatible with time evolution read from a data file (see [§ 7.3 on the preceding page]). You must choose one or the other.

7.5 Self-shielding

UV photons are absorbed by dust grains at all wavelengths. Some species can also significantly absorbe at specific wavelength and then decrease their own photodissociation rates and even the ones of others. This is what we call self-shielding [see for instance van Dishoeck and Black, 1988]. How is it taken into account in Nautilus?

In the code, the following self-shielding approximations are provided: H_2 and CO self-shielding from Lee et al. [1996], CO self-shielding from Visser et al. [2009], and N_2 self-shielding from Li et al. [2013]. In the prescription by Lee et al. [1996], the H_2 self-shielding depends on the H_2 column densities, whereas the one for CO depends on the H_2 and CO column densities as well as the visual extinction. For Visser et al. [2009], the CO self-shielding depends on the CO and H_2 column densities, whereas for Li et al. [2013], the N_2 self-shielding depends on H_2 and N_2 column densities.

In $0D$ and for the first point of the 1D structure (the nearest point from the UV source), the photodissociation rates are computed depending on the local abundances and visual extinction provided in the file **parameters.in**. The species column densities are computed at each output time by the formula:

$$\mathrm{N_i} = \frac{\mathrm{A_v}}{5.34\mathrm{x}10^{-22}}\mathrm{X_i}$$

$\mathrm{N_i}$ is the species column density in cm^{-2}, $\mathrm{A_v}$ is the visual extinction, $5.34x10^{-22}$ is the factor of conversion of visual extinction to total H column density [in cm^2, see for instance Wagenblast and Hartquist, 1989], and $\mathrm{X_i}$ is the local abundance of the species.

For the other points of the 1D structure, the species column densities between the UV source and the considered point is computed. For each spatial point x, the column densities $\mathrm{N_i}$ are computed using:

$$\mathrm{N_i(x)} = \mathrm{N_i(x-1)} + \mathrm{n_H(x)} * (\mathrm{d(x)} - \mathrm{d(x-1)}) * \mathrm{X_i(x)}$$

with $\mathrm{n_H(x)}$ the H density at the point x (in cm^{-3}, d the distance in cm read in the **1D_static.dat** file (so d(x) - d(x-1) is the size of your cell), and $\mathrm{X}_i(x)$ the abundance of your species at the point x.

7.6 Grain temperature

You have four ways of defining grain temperature in the code. The parameter *grain_temperature_type* can have the following values:

fixed The grain temperature is fixed throughout the simulation. *initial_dust_temperature* defines this fixed value;

gas The grain temperature is equal to the gas temperature, no matter what.

computed The grain temperature is calculated following an energy equilibrium with the gas in the structure

table_evolv The grain temperature is interpolated from the 5-th column of the *structure_evolution.dat* file. *is_structure_evolution* must be set to 1.

table_1D The grain temperature is read in the 1D_static.dat file for 1D structures (the 5-th column).

computed Calculates the grain temperature from uv_flux and visual extinction by radiative equilibrium. See Ruaud et al. [2018] for explanations.

7.7 Switches

- To activate (or not) accretion on dust grains and grain surface reactions:

```
is_grain_reactions = 1
```

Remark : In that mode, the H_2 adhoc formation (see below) is by default on.

- To force ad-hoc formation of H_2 on grain surface:

```
is_h2_adhoc_form = 1
```

Remark : The ad-hoc formation of H_2 on grain surface assumes that each accretion event of two H atoms leads to the formation of H_2. In this prescription, when the ad-hoc H_2 formation is activated, 50% of the adsorbed H are available for grain reactions (other than H_2 formation) and 50% for the formation of H_2. You may want in some cases to force it when the grain temperature is too high to allow for the H_2 to form. Nautilus does not include all possible formation processes of H_2 on the dust.

- To activate (or not) the photodesorption of ices:

```
is_photodesorb = 1
```

Remark : The default treatment of photodesorption assumes that all-species desorb with the same yield that is provided in the chemistry files. These yields can be changed.

- To activate (or not) the Eley-Rideal and complex induced reaction mechanisms [Ruaud et al., 2015]:

```
is_er_cir = 1
```

- To activate (or not) the self-shielding of H_2, CO and N_2 related to visual extinction:

```
is_absorption_h2 = 1
```

Different types are:

0 : H_2 self-shielding is disabled
1 : H_2 self-shielding from Lee et al. [1996]

```
is_absorption_co = 1
```

Different types are:

0 : CO self-shielding is disabled
1 : CO self-shielding from Lee et al. [1996]
2 : CO self-shielding from Visser et al. [2009]

```
is_absorption_n2 = 1
```

Different types are:

0 : N_2 self-shielding is disabled
1 : N_2 self-shielding from Li et al. [2013]

- To activate (or not) grain tunneling diffusion and choose the type of grain tunneling diffusion:

```
grain_tunneling_diffusion = 0
```

Different types are:

0 : Diffusion through thermal hopping for all species [Hasegawa et al., 1992]

1 : Quantum tunneling diffusion rate for H and H_2 only if faster than thermal hopping [Hasegawa et al., 1992]

2 : Quantum tunneling diffusion rate for all species depending on their mass. With the large binding energy of O on ices proposed by new experiments, the oxygen does not move on the surface anymore. Some other experiments have proposed that O can diffuse by tunneling effect. So if you are using a O binding of 1600K, put this parameter to 4. See Wakelam et al. [2017] for details and references on the experiments.

- To choose whether or not we can modify some abundances (this doesn't work in 1D, because of diffusion !):

```
conservation_type = 0
```

Different types are:

0 : Only electrons conserved

1 : element #1 conserved + electrons

2 : element #1 and #2 conserved + electrons

n : element #1...#n conserved + electrons

Remark : This was added to assure the conservation of charges and elements during the calculations. I guess this was included in the past because of numerical problems. the test that is done is simply to change the abundance of electrons or atoms according to the sum of charges and/or elements (over all species). CHECK WHAT ARE ELEMENTS 1 AND 2. In practice, the default value for this switch is 0.

7.8 Gas parameters

```
initial_gas_density =  2.000E+04
```

initial gas density in particule/cm^3 is the total density of protons: $n_H = n(H) + 2n(H_2)$.

```
initial_gas_temperature =  1.000E+01
```

initial gas temperature in K

```
initial_visual_extinction =  1.500E+01
```

initial visual extinction in magnitude

```
cr_ionisation_rate =  1.300E-17
```

cosmic ray ionization rate in s^{-1}. A standard value is $1.3 \cdot 10^{-17}$.

```
x_ionisation_rate =  0.000E+00
```

Ionisation rate due to X-rays in s^{-1}. **This is not yet used in the code yet.**

```
uv_flux =  1.000E+00
```

Scale factor for the UV flux, in unit of the reference flux. By choosing 1, you will use the nominal value.

7.9 Grain parameters

`initial_dust_temperature = 1.000E+01`

initial dust temperature in K, used when *grain_temperature_type* is **fixed**.

`initial_dtg_mass_ratio = 1.000E-02`

Total mass of dust divided by total mass of gas (dimensionless).

`sticking_coeff_neutral = 1.000E+00`

sticking coefficient for neutral species. Exceptions are made for for H and H2 sticking coefficient from Chaabouni et al. [2012].

`sticking_coeff_positive = 0.000E+00`

sticking coefficient for positive species. This parameter will not be used unless sticking reactions by cations on grains are added to the chemistry.

`sticking_coeff_negative = 0.000E+00`

sticking coefficient for negative species. This parameter will not be used unless sticking reactions by anions on grains are added to the chemistry.

`grain_density = 3.000E+00`

mass density of grain material in g/cm^3

`grain_radius = 1.000E-05`

grain radius in cm.

`diffusion_barrier_thickness = 1.000E-08`

Thickness of the barrier in cm that a surface species need to cross while undergoing quantum tunneling to diffuse from one surface site to another. This is used in the formalism [Hasegawa et al., 1992, see equation 10 (parameter a)]. In case of grain_tunneling_diffusion = 2, we advise to used a slightly larger value of 2.500E-08 to limit the impact of the diffusion.

`surface_site_density = 1.500E+15`

density of sites at the surface of the grains in number/cm^2.

`diff_binding_ratio_surf = 4.000E-01`

Ratio (adimensioned) used to compute the DIFFUSION_BARRIER from the BINDING_ENERGY if not known for surface species. For the 2 phase model, only this value is used.

`diff_binding_ratio_mant = 8.000E-01`

Ratio (adimensioned) used to compute the DIFFUSION_BARRIER from the BINDING_ENERGY if not known for mantle species. Used for the 3 phase model.

`chemical_barrier_thickness = 1.000E-08`

Parameter (in cm) used to compute the probability for a surface reaction with activation energy to occur through quantum tunneling. This is the thickness of the energy barrier [Hasegawa et al., 1992, See equation 6].

`cr_peak_grain_temp = 7.000E+01`

Peak grain temperature in K when struck by a cosmic ray.

`cr_peak_duration = 1.000E-05`

duration [s] of peak grain temperature

```
Fe_ionisation_rate =  3.000E-14
```

(cosmic) Fe-ion–grain encounter [$s^{-1}grain^{-1}$] for 0.1 micron grain. For cosmic photo desorptions, only Fe-ions are efficient to heat grains.

```
vib_to_dissip_freq_ratio =  1.000E-02
```

(dimensionless) For the RRK (Rice Ramsperger-Kessel) desorption mechanism. Ratio of the vibration frequency (proper energy of a species when it is created on a grain) to the dissipation frequency (energy needed by the molecule to be evaporated from the grain surface). This ratio help to determine if a species evaporate after its formation on the grain surface. Since the dissipation frequency is usually unknown, this ratio is a free parameter. A common value is 1%.

```
ED_H2 =  2.300E+01
```

H2 binding energy over itself. Used for the desorption encounter mechanism. in K.

8 Chemical network

8.1 Reaction files

The files concerned are: *gas_reactions.in*, *grain_reactions.in* and *activation_energies.in*

A typical reaction line is:

```
1 !     Reactants                    ->      Products
                                                    A           B           C
      xxxxxxxxxxxxxxxxxxxx ITYPE Tmin    Tmax formula ID xxxxx
2 C+          CH2OH                    -> CH2+        HCO
                                                 7.500E-10 -5.000E-01  0.000E+00 0.00e
      +00 0.00e+00    NA  4      10     280   3   6098 1  1
```

The reaction displayed here is:

$$C^+ + CH_2OH \rightarrow CH_2{}^+ + HCO$$

The temperature range is $T \in [10; 280]$ K. The type of reaction is 4. The ID of the reaction is 6098. The formula used to compute reaction rate is 3, with the 3 parameters $A = 7.5 \cdot 10^{-10}$, $B = -0.5$ and $C = 0$. The corresponding formula can be found on the KIDA website (`https://kida.astrochem-tools.org/help.html`). Other columns are ignored, as the "xxx" emphasised in the legend line associated.

Each species name is encoded with 11 characters.

All reaction files have the same format. Depending on the evolution of the code, the number of reactants (`MAX_REACTANTS`) or products (`MAX_PRODUCTS`) may vary (increase), so these files must be modified to take that into account.

The following global variables (in the source code) are here to tell the code that these numbers have changed.

```
1 MAX_REACTANTS = 3 !< The maximum number of reactants for one reaction.
2 MAX_PRODUCTS = 5 !< The maximum number of products for one reaction.
3 MAX_COMPOUNDS = MAX_REACTANTS + MAX_PRODUCTS !< Total maximum number of compounds for
      one reaction (reactants + products)
```


Pay attention to the fact that some things might need manual modifications in the code. If this number changes, `get_jacobian(N, T, Y, J, IAN, JAN, PDJ)` must be actualized, since each reactant and product has its own variable, a new one must be created for the new column possible.

8.2 Reaction types

Chemical reactions can be of several types.
Here is the list:

0 Gas phase reactions with GRAINS

1 Photodissociation/ionisation with cosmic rays (CR)

2 Gas phase photodissociations/ionisations by secondary UV photons generated by CR

3 Gas phase photodissociations/ionisations by UV

4-8 Bimolecular gas phase reactions - several possible formula

10-11 H_2 formation on the grains when `IS_GRAIN_REACTIONS=0`

> **Remark** : Only one reaction for each of the two types (10 and 11).

14 Grain surface reactions

15 Thermal evaporation

16 Evaporation induced by cosmic-ray stochastic heating

17-18 Photodissociations by Cosmic rays induced UV photons on grain surfaces

19-20 Photodissociations by UV photons on grain surfaces

30 Eley-Rideal (low temperature)

31 Reactions of complexes: JX... Y -> JXY

40 Swapping reactions J -> K

41 Swapping reactions K -> J

66 Photodesorption by external UV

67 Photodesorption by UV induced by cosmic rays

77 Cosmic-ray sputtering

99 Adsorption on grains

For Photodissociations, there are two types for each process because: photodissociations on the surfaces are based on their equivalent in the gas. In the can, there can be photoionisations. We do not consider ions on the surfaces because grains are usually negatively charged so that any cation produced on the surface would recombine quickly. So for the equivalent of photoionisations on the surface would have the rates of photoionisations in the gas-phase but the products of the electronic recombination. ITYPES 18 and 20 are the grain equivalent of the gas-phase photoionisations.

For Cosmic-ray sputtering, see Wakelam et al. [2021] for details.

9 Input files

All input files have the same ***.in** extension. An example simulation, containing all necessary input files is provided in the sub-folder **example_simulation**.

The main parameter file is *parameters.in* (see [§ 7 on page 10]). In this file, you set many switches for the surface processes and the physical parameters in 0D.

abundances.in gives the initial abundances for a set of species that the user chooses. Default minimum values are applied to the species not present in this file (this value is set by the parameter *minimum_initial_abundance* in *parameters.in* (see [§ 7 on page 10]).

element.in gives information about prime elements (base elements used to construct molecules) existing in the simulation (name and mass in Atomic mass unit).

There are 2 parameter files listing all reactions in a given phase (gas or grain):

- *gas_reactions.in*
- *grain_reactions.in*

and 2 parameter files for species present in a given phase (gas or grain) reactions:

- *gas_species.in*
- *grain_species.in*

Remark : *gas_reactions.in* is in the same format as the kida.uva gas-phase network and only contains gas-phase reactions while *grain_reactions.in* mostly contains reactions for grain surfaces (as well as adsorption and desorption reactions) + a few gas-phase reactions for species not included in kida.uva.

activation_energies.in provides activation energies for some surface reactions.
surface_parameters.in provides parameters for species on the surfaces: masses, binding energies, etc.
In the directory , there are several versions of these input files that have been updated over the time.

10 Output files

 All output files have the same ***.out** extension.

Output files are:

- *info.out*: Various information about the simulation (see [§ 10.1] for more details)
- *species.out*: The list of species and their corresponding index
- *elemental_abundances.out*: The prime elements abundance and mass at the beginning of the simulation. A ***.tmp** version displays the same info, but at the last output.
- *abundances.*.out*: In binary format (unformatted), abundances of all species, each file for a different output time. *nautilus_outputs* read these files to generate ASCII files (see [§ 10.2] for more details).
- *rates.*.out*: In binary format (unformatted), rates of all reactions, each file for a different output time. *nautilus_rates* and *nautilus_major_reactions* analyse these files.
- *col_dens.*.out*: In ascii format, column densities of H, H_2, CO and N_2 at each time step used for the self-shielding.
- *abundances.tmp*: The abundance of all species at the last output in ASCII.

10.1 Information : info.out

In the file *info.out*, the ID reference of the current version of nautilus, used to run the simulation is printed, as well as other useful information about the state of Nautilus and how the simulation was executed.

Information and warnings about the coherence of the chemical network are also printed here.

10.2 Abundances

Files are named *abundances.000001.out* and so on, for each output time. Outputs are stored in binary format. Binary format is not detailed here. If you want details about variables in there, please refer to the Fortran routine **write_current_output** in the file **input_output.f90**

To get ASCII files from the binaries, one must compile the designed program (in the Git repository):

```
Makefile.py output
```

Then, in your simulation folder, type:

```
nautilus_outputs
```

(this assume you modified your PATH, but absolute path also work)

This program will generate **.ab* files in a subfolder **ab** of the simulation folder. If in 1D, a file space.ab will store the spatial points. Indeed, each species file will now have one column for time, and one column per spatial point.

The program will also generate **.struct* files in a subfolder **struct** of the simulation folder, one file per spatial point.

11 Graphic display

Python scripts were created to help the user display useful information about their simulations.

11.1 Plot abundances

The script *nautilus-plot-abundances* allow you to display the time evolution of the abundances of one or more species:

```
nautilus-plot-abundances species=CO,H20
```

You can zoom in a given period of time (**tmin**, **tmax** or both):

```
nautilus-plot-abundances species=CO,H20 tmin=1e4 tmax=1e6
```

Even if you can modify and store the result in the graphic windows displayed, a default version is automatically written in **abundances.pdf**

Check the other options and detailed examples via:

```
nautilus-plot-abundances help
```

11.2 Compare abundances

The script *nautilus-compare-abundances* allow you to display the time evolution of the abundances of one or more species for all subfolders of the current working directory, assuming each one is a simulation:

```
nautilus-compare-abundances species=CO,H20
```

If there are a lot of subfolders, you can select those you want by:

```
nautilus-compare-abundances species=CO,H20 dir=simu1,simu2
```

All options existing for *nautilus-plot-abundances* apply here.

Even if you can modify and store the result in the graphic windows displayed, a default version is automatically written in **compare_abundances.pdf**

Check the other options and detailed examples via:

```
nautilus-compare-abundances help
```

11.3 Evolution of main reactions for a given species

The script *nautilus-trace-species* allow you to display the time evolution of the main production and destruction reactions for a given species:

```
nautilus-trace-species species=CO2
```

You need to run in the same simulation folder the program:

```
nautilus_trace_major
```

That will generate 4 files:

- **trace_prod_CO2.percentage** data file for the python script (production reactions)
- **trace_prod_CO2.reaction** Tells the exact reaction corresponding to the ID displayed in the python script (production reactions)
- **trace_dest_CO2.percentage** data file for the python script (destruction reactions)

- **trace_dest_CO2.reaction** (destruction reactions)

Even if you can modify and store the result in the graphic windows displayed, a default version is automatically written in **major_reactions_CO2.pdf**

Check the other options and detailed examples via:

```
nautilus-trace-species help
```

12 For developers

Constants and global variables are defined in the module **global_variable.f90**. **nautilus_main.f90** contains all the main routines.

The four main programs are:

- *nautilus* in the source file **nautilus.f90** (compilation: `cmake --build . --target nautilus_code`)
- *nautilus_outputs* in the source file **nautilus_outputs.f90** (compilation: `cmake --build . --target nautilus_o`
- *nautilus_rates* in the source file **nautilus_rates.f90** (compilation: `cmake --build . --target nautilus_rates`)
- *nautilus_major_reactions* in the source file **nautilus_major_reactions.f90** (compilation: `cmake --build . --target nautilus_major_reactions`).

One last program exists to do some unitary tests on **Nautilus** (see [§ 12.1] for more details).

A lot of routines are handled by pointers. This allows us to change easily from one routine to another in function of the parameters. Thus, some routine names (in **call**) might not exist *as is*. They are defined in **global_variable.f90**. For each pointer, an interface is defined that constrains the architecture of each subroutine he can point to.

12.1 Unitary tests

 This section has not been ported to `cmake`.

A **fortran** program *unitary_tests* (**unitary_tests.f90**) was specifically designed to test some routines of the code separately. This is the main reason why a **nautilus_main.f90** file was created because we need to access these routines separately from the *nautilus* program.

To run the unitary tests, use the Python script designed for that (he will compile the source code too):

```
unitary_tests.py
```

The code will display some information and ask you what test you want to display graphically (and generate the corresponding **.pdf**):

```
0 : av_interpolation
1 : density_interpolation
2 : gas_temperature_interpolation
3 : grain_temperature_interpolation
4 : test_av_read
5 : test_density_read
6 : test_gas_temperature_read
What test do you want to display? (0-6 ;
'all' treat them all ; 'l' display list again)
```

The principle of this code is to do some tests, store the results in data files, generate Gnuplot script files associated. To get the plots, you only have to generate it by:

```
gnuplot script_name.gnuplot
```

All files generated by the program are stored in the “test” subfolder. You can generate the .pdf manually if you are familiar with Gnuplot, and view them separately.

Remark : It is up to you to write new routines in **unitary_tests.f90** and mimic the way I wrote the previous one to test new functionality of the code.

Please keep in mind that you need to add a `call new_routine()` in the main program, just before `contains` to ensure your routine is executed.

12.2 How to write documentation with Doxygen

12.2.1 General information

Doxygen comments generally have a marker and the description. The character to declare the marker is `@`, but one needs to start the commented line by `>` to declare there is something here.

If this is an inline comment, with the code on the left, comments are like this:

```
the code !< the doxygen description
```

just to say that the description refers to the code on the left.

To continue a description on another comment line, you can double the comment character:

```
!> @brief I describe something
!! on several lines.
```

Remark : This will not make two lines in the generated documentation though. This is only a way to avoid never-ending lines with thousands of characters.

12.2.2 For a module

Start the module file with:

```
!*****************************************************************************
! MODULE: Module Name
!*****************************************************************************
!
!> @author
!> Module Author Name and Affiliation
!
! DESCRIPTION:
!> @brief Brief description of what can be done in this module.
!! This description can be on several lines.
!! \n\n Do not forget the symbol "\n" to create a new line.
!
!*****************************************************************************
```

12.2.3 For a subroutine

Before the definition of the routine, add the following text:

```
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
!> @author
!> Routine Author Name and Affiliation.
!
! DESCRIPTION:
!> @brief Brief description of routine.
!! Flow method (rate of change of position) used by integrator.
!! Compute \f$ \frac{d\lambda}{dt} , \frac{d\phi}{dt},  \frac{dz}{dt} \f$ }
!
!%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In the rest of the routine, remember to add comments for input and outputs of the routine:

```
implicit none

! Inputs
real(double_precision), intent(in) :: delta_t !<[in] description

! Outputs
integer, intent(out) :: istate !<[out] Description of the variable
!! that can be continued on another line.

```

```
10 ! Inputs/Outputs
11 real(double_precision), intent(inout) :: time !< [in,out] description
12 !! \n Continuation line.
13
14 ! Locals
15 real(double_precision) :: t !< The local time, starting from 0 to delta_t
```

12.3 How to generate Doxygen documentation

To generate **Doxygen** documentation, type the following command inside your build directory:

```
cmake --build . --target doc
```

The doc is then available inside your build directory at `doc_doxygen`.
Moreover, *Doxygen*is configured at **doc/doxygen.conf**.

Bibliography

H. Chaabouni, H. Bergeron, S. Baouche, F. Dulieu, E. Matar, E. Congiu, L. Gavilan, and J. L. Lemaire. Sticking coefficient of hydrogen and deuterium on silicates under interstellar conditions. A&A, 538: A128, February 2012. doi: 10.1051/0004-6361/201117409.

T. I. Hasegawa and E. Herbst. Three-Phase Chemical Models of Dense Interstellar Clouds - Gas Dust Particle Mantles and Dust Particle Surfaces. MNRAS, 263:589, August 1993.

T. I. Hasegawa, E. Herbst, and C. M. Leung. Models of gas-grain chemistry in dense interstellar clouds with complex organic molecules. ApJS, 82:167–195, September 1992. doi: 10.1086/191713.

H.-H. Lee, E. Herbst, G. Pineau des Forets, E. Roueff, and J. Le Bourlot. Photodissociation of H_2_ and CO and time dependent chemistry in inhomogeneous interstellar clouds. A&A, 311:690–707, July 1996.

X. Li, A. N. Heays, R. Visser, W. Ubachs, B. R. Lewis, S. T. Gibson, and E. F. van Dishoeck. Photodissociation of interstellar N_2. A&A, 555:A14, July 2013. doi: 10.1051/0004-6361/201220625.

M. Ruaud, J. C. Loison, K. M. Hickson, P. Gratier, F. Hersant, and V. Wakelam. Modelling complex organic molecules in dense regions: Eley-Rideal and complex induced reaction. MNRAS, 447:4004–4017, March 2015. doi: 10.1093/mnras/stu2709.

M. Ruaud, V. Wakelam, P. Gratier, and I. A. Bonnell. Influence of galactic arm scale dynamics on the molecular composition of the cold and dense ISM. I. Observed abundance gradients in dense clouds. A&A, 611:A96, April 2018. doi: 10.1051/0004-6361/201731693.

E. F. van Dishoeck and J. H. Black. The photodissociation and chemistry of interstellar CO. ApJ, 334: 771–802, November 1988. doi: 10.1086/166877.

R. Visser, E. F. van Dishoeck, and J. H. Black. The photodissociation and chemistry of CO isotopologues: applications to interstellar clouds and circumstellar disks. A&A, 503:323–343, August 2009. doi: 10.1051/0004-6361/200912129.

R. Wagenblast and T. W. Hartquist. Non-equilibrium level populations of molecular hydrogen. II - Models of the Zeta OPH cloud. MNRAS, 237:1019–1025, April 1989.

V. Wakelam, J.-C. Loison, R. Mereau, and M. Ruaud. Binding energies: New values and impact on the efficiency of chemical desorption. *Molecular Astrophysics*, 6:22–35, March 2017. doi: 10.1016/j.molap.2017.01.002.

V. Wakelam, E. Dartois, M. Chabot, S. Spezzano, D. Navarro-Almaida, J. C. Loison, and A. Fuente. Efficiency of non-thermal desorptions in cold-core conditions. Testing the sputtering of grain mantles induced by cosmic rays. A&A, 652:A63, August 2021. doi: 10.1051/0004-6361/202039855.