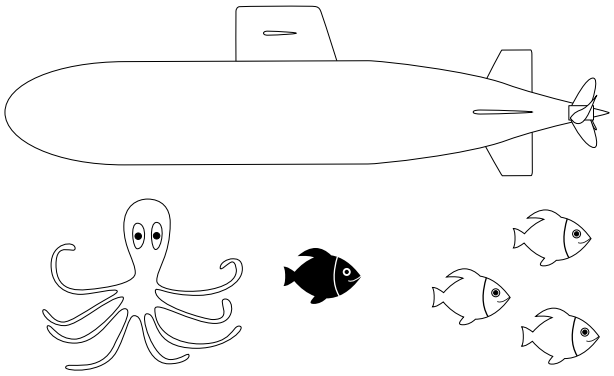

Nautilus Documentation

December 2, 2015

Christophe Cossou, Maxime Ruaud, Valentine Wakelam

Contents

1 TODO

- Create again a Doxygen documentation on the web site

2 Starting with Nautilus

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

[§ 2] explains the basics, mainly how to get the repository, how to compile the simulation code, how to use the various tools. Be sure to read carefully the section about **Installation** [§ 2.3].

[§ 5 on page 16] explains the chemical network, its format, and the various types of reactions available. A chemical network is given with the code, but you can use your own.

Input files are presented in [§ 2.6 on page 7], but the most important one, the only one you will modify daily (*parameters.in*) is dealt with in [§ 4 on page 8].

Output files are presented in [§ 6 on page 17]. Simulations information are displayed in *info.out* (see [§ 6.1 on page 18]).

[§ 7 on page 18] explain what are the Python scripts you can use to plot useful information (mainly abundances) of your simulations (single plot or comparison between several runs).

One last section [§ 8 on page 19] present technical specifications of the code and how to maintain it.

2.1 Generic informations

2.1.1 Python scripts help

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

```
script_name.py help
```

2.1.2 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.

2.1.3 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.

2.2 Getting the Git repository

First, you need to get the Git repository from *Framasoft* before actually using the code.

I assume you only want to use it, without ever putting on the distant repository anything. This way, you don't even need to have an account and a password.

In the parent directory where you want to put a **nautilus** folder containing the Git repository, type:

```
git clone https://git.framasoft.org/Wakelam/nautilus.git
```

Be sure to run once in a while the command to get distant *updates* if they exists:

```
git pull
```

2.3 Installation

Once you have the Git repository, I advice you to run the script *configure_nautilus.py* :

```
configure_nautilus.py
```

This script will add useful shortcuts in the *.bash_profile*. When updating your Git repository, the bash commands will also be updated automatically into the *.bash_profile*.

 Nautilus is a file browser in GNU/Linux (Gnome) environments. Thus, You must be *very* careful. If you use:

```
nautilus
```

in a terminal, you will not run the nautilus code, but rather the file browser (which you do not want, especially in a remote server, I assure you).

The improvements brought by this script are:

- You can go to the Nautilus folder by typing:

  ```
  cd $nautilus
  ```

- You can launch the code, the output code and the rates code by these three commands, regardless of the folder you currently are:

  ```
  nautilus_code
  nautilus_outputs
  nautilus_rates
  nautilus_major_reactions
  ```

- You have now useful information about Git repositories directly into the Bash prompt.
- all scripts in the folder **scripts**/ can be run from any folder. This include:

  ```
  nautilus-clean.sh # Cleaning a simulation folder
  nautilus-plot-abundances.py # plot abundances of a simulation
  nautilus-compare-abundances.py # compare two simulations or more.
  ```

Remark : All of this will have little interest for you if you don't use 'Bash' as the interpreter for you terminal.
You can also check the options of this script. Especially, you can undo the installation.

2.4 Compilation

The script *Makefile.py* allow you to compile the code (see [§ 2.1.1 on the previous page] for infos about Python scripts). The default compiler is *gfortran*.
If you want to just compile the code[1], type:

```
Makefile.py
```

Remark : Errors are in ***.log** files associated with the module incriminated. Warnings are stored in a generic file *compilation.log*.

 By default, all warnings issued by *ODEPACK* routines are masked (to increase readability, since nobody can modify this thing from scratch). An option **opkd** can force their display.

If you want to compile *only* one binary, use respectively those command:

```
Makefile.py nautilus
Makefile.py output
Makefile.py rates
Makefile.py major
```

One particular option exist:

[1] By default, all the binaries are compiled

```
Makefile.py test
```

This option **test** use compilation options that allow comparison of the code between different versions and is to be used for *compare_simulations.py* (see [§ 8.2 on page 20]).

Remark : Mainly, this option avoid too hard optimization that can result in slightly different results when the code evolve (not because the maths changes but because of optimization only).

Finally, one option exist so that you can compile one particular source file of the current working directory:

```
Makefile.py name=source_code.f90
```

The Binary name will be `source_code`

 If you change compilation options, pay attention to use the option **force** to ensure all the modules will be compiled using these options.

2.5 Binaries

The various binaries available are:

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

`nautilus_code` is the main program, the one computing the evolution of the chemical scheme. He needs several input files described in [§ 2.6]. 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 [§ 6.2 on page 18].

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 [§ 7.3 on page 19].

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.

2.6 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 [§ 4 on the following page]). 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 [§ 4 on the next page]).

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.

2.7 Usefull tools

2.7.1 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
```

2.7.2 Installation script

The script *configure_nautilus.py* will configure the bash profile to add several shortcuts (You only have to run the command once). More details are available in [§ 2.3 on page 5].

3 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 phases: gas, surface and mantle using the same approach as **?**. The main differences are:

- Species can only adsorbe 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, only the surface is chemically active. A paper is in preparation. DO NOT USE THIS VERSION OF THE CODE YET SINCE THERE MAY STILL BE CHANGES. To have the 2 phases on the surface, a new kind of species has been added: KX species. The full model will be put online later.

4 Parameter file : parameters.in

For generic informations, see [§ 2.1.2 on page 5].
parameters.in has the particularity to be re-written each time you launch a Nautilus simulation. This ensure 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).

4.1 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 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 WHAT IT MEANS)
- Check that all species have production AND destruction reactions (error if none, warning if only one).
- Check that each reaction is balanced in prime element and in charge.
- 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 have 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 of 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.

4.2 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 3).

```
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*.

4.3 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
```

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. Lets 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 [§ 4.4 on the facing page] for incompatibilities.

4.4 1D simulations

You can make simulation in 1D:

```
structure_type = 1D_diff or 1D_no_diff
```

You have to choose between 1D with diffusion (`1D_diff`) and 1D without species diffusion (`1D_no_diff`). **BE CAREFUL THAT THE 1D_DIFF DOES NOT WORK FOR THE MOMENT.**

The 1D physical structure is read in the input file *1D_static.dat* where you define gas density, gas temperature, visual extinction, diffusion coefficient 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] ;
! Diffusion coefficient [cm^2/s] ; Dust Temperature [K] ; 1/abundance of grains ; AV/NH conversion factor ;
! radius of grains (cm)
1.66096e+02   1.24576e+05   1.73205e+01   0.00000e+00   0.00000e+00   1.73205e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.63480e+02   1.41119e+05   1.73099e+01   3.45122e-03   0.00000e+00   1.73099e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.60865e+02   1.59773e+05   1.72782e+01   7.35865e-03   0.00000e+00   1.72782e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.58249e+02   1.80819e+05   1.72256e+01   1.17808e-02   0.00000e+00   1.72256e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.55633e+02   2.04581e+05   1.71523e+01   1.67840e-02   0.00000e+00   1.71523e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.53018e+02   2.31428e+05   1.70589e+01   2.24439e-02   0.00000e+00   1.70589e+01   7.56765e+11   6.25000e-22   1.00000e-05
1.50402e+02   2.61783e+05   1.69459e+01   2.88461e-02   0.00000e+00   1.69459e+01   7.56765e+11   6.25000e-22   1.00000e-05
```

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. This will not work with the diffusion. To allow for the computation of self-shielding (see sections 4.5 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 [§ 4.6 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 give 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. The last column 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.

Comment added by Christophe that needs to be modified when we take care of this: Diffusion of the structure is not done in the code. Only diffusion of the species abundances is done. Diffusion is expected to be in the z direction, assuming the structure is a disk. Other structures are possible but need to be implemented as other values for the parameter structure_type This allow you to use any sort of model or diffusion process. You only have to provide a data sample of the spatial points and evolution of the different values. To fix one parameter, you only have to put the same value in the corresponding column.

 1D simulations are not compatible with time evolution read from a data file (see [§ 4.3 on the facing page]). You must choose one or the other.

4.5 Self-shielding

UV photons are absorbed by dust grains at all wavelengths. Some species can also significantly absorbe at specific wavelength and then decrease the 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 times by the formula:

$$N_i = \frac{A_v}{5.34x10^{-22}} X_i$$

N_i is the species column density in cm^{-2}, A_v is the visual extinction, $5.34x10^{-22}$ is the factor of conversion of visual extinction to total H column density [Wagenblast and Hartquist, 1989, in cm^2, see for instance[], and 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 N_i are computed using:

$$N_i(x) = N_i(x-1) + n_H(x) * (d(x) - d(x-1)) * X_i(x)$$

with $n_H(x)$ 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 $X_i(x)$ the abundance of your species at the point x.

4.6 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).

4.7 Switches

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

```
is_grain_reactions = 1
```

- To activate (or not) 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 assume that each accretion events 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.

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

```
is_photodesorb = 1
```

> **Remark** : Special treatment are considered for molecules such as CO_2, CO, H_2O, CH_3OH and N_2 [Öberg et al., 2009a,b,c]. The default yield is 1×10^{-3} for the others atoms/molecules.

- To activate (or not) the CRID (Cosmic Rays Induced Diffusion) [Reboussin et al., 2014]:

```
is_crid = 1
```

- 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 : Thermal for H, H_2 [Hasegawa et al., 1992]
1 : Quantum tunneling diffusion rate [s^{-1}] [Watson, 1976]
2 : Quantum tunneling diffusion rate [s^{-1}] [Hasegawa et al., 1992]
3 : Choose fastest

- To choose whether or not we can modify some rates:

```
modify_rate_flag = 1
```

Different types are:

-1 : H+H only
1 : modify only H
2 : modify H and H_2
3 : modify all atoms

Remark : The modified rates have been proposed by Caselli et al. [1998] to take into account the discrete nature of interstellar grains in the accretion limit case (when we have less than one mean species on the grains). Stantcheva et al. [2001] proposed a simplification of the modified rates and this is what is included in the code. For surface reactions without any activation barrier, the following test is done: if the diffusion rate is larger than both the accretion and evaporation rates then the diffusion rate is set to the maximum rate between accretion and evaporation of this species. For reactions with activation barrier (which are by definition slow rates), the following is done: if the rate coefficient of the reaction is larger than the evaporation and accretion rates of both species then the rate coefficient is set to the larger between the accretion and evaporation of either species. The reference is supposed to be Caselli et al. [2002]. WE HAVE FOUND SOME PROBLEMS WITH THESE MODIFIED RATES. IT MAY BE BETTER TO USE NOTHING INSTEAD OF THESE MODIFIED RATES IN DENSE CONDITIONS. MORE WORK WILL HAVE TO BE DONE IN THE FUTURE.

- 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 is 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.

4.8 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 $1.3 \cdot 10^{-17}$ (TODO ref ?).

`x_ionisation_rate = 0.000E+00`

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

`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.

4.9 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

`sticking_coeff_positive = 0.000E+00`

sticking coefficient for positive species

`sticking_coeff_negative = 0.000E+00`

sticking coefficient for negative species

`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)].

`surface_site_density = 1.500E+15`

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

`diff_binding_ratio_surf = 5.000E-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.

5 Chemical network

5.1 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 (http://integration-kida.obs.u-bordeaux1.fr/help.html). Other columns are ignored, as the "xxx" emphasize 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 theses files must be modified to take that into account.

The following global variables (in the source code) are here to tell to the code that theses number 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 change, `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.

5.2 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 stocastic 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

98 storage of H_2S under a refractory form (commented in the code)

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 photoioniozations. 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 photoionizations on the surface would have the rates of photoionizations in the gas-phase but the products of the electronic recombination. ITYPES 18 and 20 are the grain equivalent of the gas-phase photoionizations.

6 Outputs

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

Output files are:

- *info.out*: Various information about the simulation (see [§ 6.1 on the next page] for more details)
- *species.out*: The list of species and their corresponding index
- *elemental_abundances.out*: The prime elements abundances and mass at the beginning of the simulation. A ***.tmp** version display the same infos, 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 theses files to generate ASCII files (see [§ 6.2 on the following page] 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 theses 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 abundances of all species at the last output in ASCII.

6.1 Informations : 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 coherence of the chemical network are also printed here.

6.2 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 have an alias, but absolute path also work)

This program will generate **.ab* files in a sub-folder **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 sub-folder **struct** of the simulation folder, one file per spatial point.

7 Graphic display

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

7.1 Plot abundances

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

```
nautilus-plot-abundances.py species=CO,H2O
```

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

```
nautilus-plot-abundances.py species=CO,H2O 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 detailled examples via:

```
nautilus-plot-abundances.py help
```

7.2 Compare abundances

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

```
nautilus-compare-abundances.py species=CO,H2O
```

If there is a lot of sub-folder, you can select those you want by:

```
nautilus-compare-abundances.py species=CO,H2O dir=simu1,simu2
```

All option existing for *nautilus-plot-abundances.py* applies 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.py help
```

7.3 Evolution of main reactions for a given species

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

```
nautilus-trace-species.py 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.py help
```

8 For developpers

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: `Makefile.py`)
- *nautilus_outputs* in the source file **nautilus_outputs.f90** (compilation: `Makefile.py output`)
- *nautilus_rates* in the source file **nautilus_rates.f90** (compilation: `Makefile.py rates`).
- *nautilus_major_reactions* in the source file **nautilus_major_reactions.f90** (compilation: `Makefile.py major`).

One last program exist to do some unitary tests on **Nautilus** (see [§ 8.1] for more details).

A lot of routines are handled by pointers. This allow us to change easily from one routine to another in function of the parameters. Thus, some routine names (in **call**) might not exists *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.

8.1 Unitary tests

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 "tests" sub-folder. You can generate the .pdf manually if you are familiar with Gnuplot, and of course, view them separately.

Remark : It's up to you to write new routines in **unitary_tests.f90** and mimic the way I wrote the previous one to tests 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.

8.2 Check before commit

You can have two types of modifications in the code, those that modify the outputs, and those that do not.

I created a Python script *compare_simulations.py* that can help you ensure the code do not change the results between two versions. It's up to you to check your modifications when the outputs are different though.

The basic use of this script is as follow (in the code main directory):

```
Makefile.py test && compare_simulation.py
```

If you want to update the "reference" version of the code, because outputs changed and you want a new reference, type:

```
compare_simulation.py rev=HEAD
```

rev can accept any commit reference, **HEAD**, **HEAD^** and a hashtag will work.

8.3 How to write documentation with Doxygen

8.3.1 General informations

Doxygen comments generally have a marker and the description. The character to declare the marker is `@`, but one need 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, just 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.

8.3.2 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.
!
!********************************************************************************
```

8.3.3 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, do not forget 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.

! Inputs/Outputs
real(double_precision), intent(inout) :: time !< [in,out] description
!! \n Continuation line.

! Locals
real(double_precision) :: t !< The local time, starting from 0 to delta_t
```

8.4 How to generate Doxygen documentation

To generate **Doxygen** documentation, type the following command in the root Git repository:

```
doxygen doxygen.conf 2>doxygen.log
```

doxygen.conf is the parameter file of *Doxygen*. Errors will be redirected in the file **doxygen.log**.


The documentation (in the folder **html**/) is synchronized with the server repository. Thus, you may generate documentation only after major changes in the *Doxygen* documentation, to avoid too many changes for too few improvements.

8.5 Bash profile

The file *.nautilus_profile* in the Git repository contains Bash commands to give user friendly tools to use *Nautilus* in command line. By running *configure_nautilus.py*, a call to this file will be done in the *.bash_profile*. See [§ 2.3 on page 5] for more details.

Bibliography

P. Caselli, T. I. Hasegawa, and E. Herbst. A Proposed Modification of the Rate Equations for Reactions on Grain Surfaces. ApJ, 495:309–316, March 1998. doi: 10.1086/305253.

P. Caselli, T. Stantcheva, O. Shalabiea, V. I. Shematovich, and E. Herbst. Deuterium fractionation on interstellar grains studied with modified rate equations and a Monte Carlo approach. Planet. Space Sci., 50:1257–1266, October 2002. doi: 10.1016/S0032-0633(02)00092-2.

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.

K. I. Öberg, R. T. Garrod, E. F. van Dishoeck, and H. Linnartz. Formation rates of complex organics in UV irradiated CH_3OH-rich ices. I. Experiments. A&A, 504:891–913, September 2009a. doi: 10.1051/0004-6361/200912559.

K. I. Öberg, H. Linnartz, R. Visser, and E. F. van Dishoeck. Photodesorption of Ices. II. H_2O and D_2O. ApJ, 693:1209–1218, March 2009b. doi: 10.1088/0004-637X/693/2/1209.

K. I. Öberg, E. F. van Dishoeck, and H. Linnartz. Photodesorption of ices I: CO, N_2, and CO_2. A&A, 496:281–293, March 2009c. doi: 10.1051/0004-6361/200810207.

L. Reboussin, V. Wakelam, S. Guilloteau, and F. Hersant. Grain-surface reactions in molecular clouds: the effect of cosmic rays and quantum tunnelling. MNRAS, 440:3557–3567, June 2014. doi: 10.1093/mnras/stu462.

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.

T. Stantcheva, P. Caselli, and E. Herbst. Modified rate equations revisited. A corrected treatment for diffusive reactions on grain surfaces. A&A, 375:673–679, August 2001. doi: 10.1051/0004-6361:20010859.

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.

W. D. Watson. Interstellar molecule reactions. *Reviews of Modern Physics*, 48:513–552, October 1976. doi: 10.1103/RevModPhys.48.513.