Astrochem Documentation

Release 0.7-beta

Sébastien Maret

September 23, 2015

Contents

Contents:

CHAPTER 1

Astrochem documentation manual

1.1 What is Astrochem?

Astrochem is a code to compute the abundances of chemical species in the interstellar medium, as function of time. It is designed to study the chemistry in a variety of astronomical objects, including diffuse clouds, dense clouds, photodissociation regions, prestellar cores, protostars, and protostellar disks [1]. Astrochem reads a network of chemical reactions from a text file, builds a system of kinetic rates equations, and solves it using a state-of-the-art stiff ordinary differential equations (ODE) solver. The Jacobian matrix of the system is computed implicitly, so the resolution of the system is extremely fast: large networks containing several thousands of reactions are usually solved in a few seconds. In addition, Astrochem may be run in parallel on multi-cores and or multi-CPU computers. A variety of gas phase process are considered, as well as simple gas-grain interactions, such as freeze-out and desorption via several mechanisms (thermal desorption, cosmic-ray desorption and photo-desorption). The computed abundances are written in a HDF5 file, and can be plotted in different ways with the tools provided with Astrochem. Chemical reactions and their rates are written in a format which is meant to be easy to read and to edit. A tool to convert chemical networks from the OSU and KIDA databases into this format is provided. Astrochem is written in C, and its source code is distributed under the terms of the GNU General Public License (GPL).

This manual documents Astrochem. It is organized as follows. The model is described in section *The model*. The various chemical processes considered in the code and the underlying assumptions are discussed in this section. Astrochem usage is described in section *Using Astrochem*. This section is written as a tutorial: the different steps needed to computed abundances with Astrochem for a simple case are described. Section *Input and source files* gives a comprehensive description of Astrochem input and source model files, while section *Chemical networks* presents the file format used for chemical networks in Astrochem, as well as the different networks that are provided with the code. *Running Astrochem in parallel* explains how to run Astrochem in parallel. *Calling Astrochem from another code* details how to use the C and Python APIs. Finally, Appendix *Astrochem authors* gives a list of Astrochem authors, and Appendix *Contributing to Astrochem* explains how you can contribute to the project.

1.2 The model

Abundances in the interstellar medium are usually described by a system of kinetic equations. For example, let us consider the HCO^+ ion. In the dense interstellar medium, this ion is mainly formed through the reaction:

$$H_3^+ + CO \rightarrow HCO^+ + H_2$$

with a rate k_1 (in $cm^3\,s^{-1}$ units). HCO^+ it is destroyed principally through the following reaction:

$$HCO^+ + e^- \rightarrow H + CO$$

[1] For the moment, only spherical objects with an external radiation field are supported.

with a rate constant k_2. If we neglect other formation and destruction channels, the derivative of the HCO^+ density – i.e. the number of HCO^+ per volume unit that we note $n(HCO^+)$ in the following – writes as:

$$\frac{dn(HCO^+)}{dt} = k_1\, n(H_3^+)\, n(CO) - k_2\, n(HCO^+)\, n(e^-)$$

Similar kinetic equations can be written for all species; therefore to compute the abundances as a function of time, one need to solve a system of n_s equations for a set of initial conditions, with n_s the number of species in the chemical network. These are ordinary differential equations (ODE), that may be solved by different techniques (Euler method, Runge-Kutta method, etc.). One difficulty is that the ODE system is usually *stiff*, because of the different timescales considered in the code; for example, neutral-neutral reactions are typically several orders of magnitude slower than a ion-neutral reactions. Another characteristic of the system is that it is *sparse*; many species do not react together, resulting in a large number of zeros in the Jacobian matrix of the system [2].

Astrochem reads a set of reactions from a text file (see *Chemical networks* for a description a chemical network files), build-up the system of kinetic equation and solve them using the CVODE solver from the SUNDIALS library. CVODE uses the Backward Differentiation formula (BDF method) with a variable step-size, which is suitable for stiff systems. The resulting system of linear equations is solved using the Newton iteration. The Jacobian matrix of the system is computed explicitly to speed-up the computations. In addition, Astrochem may be run in parallel on multi core/CPU computers (see *Running Astrochem in parallel*). Astrochem includes a number of gas-phase chemical processes as well as gas-grain interactions that we describe in the following.

1.2.1 Gas-phase processes

Gas-phase reactions

Most gas-phase reactions have rates that can be described by an Ahrrenus law:

$$k = \alpha \left(\frac{T}{300}\right)^{\beta} \exp\left(-\frac{\gamma}{T}\right) \tag{1.1}$$

where T is the gas temperature, and α, β and γ are the rate constants. Usually γ corresponds to the energy barrier of the reaction, expressed in Kelvins. It is generally equal to zero for ion-neutral reactions, and equal or greater than zero for neutral-neutral reactions. The units of k depends on the order of the reaction: for a two body reaction, which is of the second order, these are $\mathrm{cm^3\,s^{-1}}$.

Astrochem reads the α, β and γ constants from the chemical network file (see *Network file format* for a description of the network file format). The formation rate of each products of a given reaction is then computed by multiplying the densities of the reactants by k. Similarly the destruction rate of each reactant is computed by multiplying the densities of the reactants by k. Reactions in Astrochem may have up to three reactants, and four products.

Cosmic-ray ionization

Cosmic-ray particles can ionize molecules and atoms. This may happen in a direct or indirect fashion. In the first case, the molecule (in general H_2) is ionized by a direct interaction with the cosmic-ray particle. In the second case, the particle first ionizes H_2, forming H_2^+ and an electron. The electron then recombines with H_2^+ and emit a UV photon.

[2] The Jacobian matrix is a n_s^2 elements square matrix, whose elements are defined as

$$J_{i,j} = \frac{\partial \dot{n(\mathrm{x_i})}}{\partial n(\mathrm{x_i})}$$

with:

$$\dot{n(\mathrm{x_i})} = \frac{\mathrm{d}n(\mathrm{x_i})}{\mathrm{d}t}$$

This *secondary* UV photon may then ionize other molecules or atoms. Astrochem assumes that the rate for these (either direct of indirect) cosmic-ray ionization reactions scale with the H_2 ionization rate ζ, such as:

$$k = \alpha\,\zeta \tag{1.2}$$

The value of ζ is read from the input file (see *Input file*). Typical values are comprised between 10^{-17} and $10^{-17}\,\mathrm{s}^{-1}$. Note that in this case the units of k are s^{-1} because cosmic-ray ionization reactions are of the first order.

Photo-ionization and photo-dissociation

UV photons from nearby stars may also dissociate and ionize molecules and atoms. For sources with a plane-parallel or spherical symmetry, the ionization or dissociation rate may be written as:

$$k = \alpha \exp\left(-\gamma A_v\right)\,\chi \tag{1.3}$$

where A_v is the visual extinction in magnitude, and χ is the external UV flux in units of the standard Draine interstellar radiation field (Draine, 1978).

This formulation implicitly assumes that the external radiation field has the same spectral shape than the the ISRF. In addition the *self-shielding* of species that dissociate through a line process is neglected.

1.2.2 Gas-grain interactions

H_2 formation on grains

In the interstellar medium H_2 is mainly formed on dust grains . The process is complex and involves the absorption of an H atom in a grain site, the tunneling of the H atom from one site to the other, and the reaction with another H to form H_2. The energy released during the reactions causes the evaporation of the H_2 molecule, which returns to the gas phase. Astrochem uses a simple treatment of this process. We assume that each H atom that strikes a grain forms H_2 with a given efficiency. Under this assumption, the formation rate of H_2 on the grains is given by:

$$\frac{\mathrm{d}n(\mathrm{H_2})}{\mathrm{d}t} = k\,n(\mathrm{H})$$

with:

$$k = \alpha \left(\frac{T}{300}\right)^{\beta} \tag{1.4}$$

The value of k may be estimated by assuming that the efficiency of the process is close to 1 (i.e. that each atom H that strikes a grain forms an H_2). The rate coefficient is then simply 1/2 of the collision rate between H atoms and grains. For 0.1 μm olivine grains and gas-to-dust mass ratio of 100, we obtain a value of $\sim 10^{-17}\,\mathrm{s}^{-1}$ at 10 K. This is close to the value of $5 \times 10^{-17}\,\mathrm{s}^{-1}$ determined observationally by Jura (1974). However, because of the numerous uncertainties associated with the formation of H_2, in Astrochem the rate is not computed in this fashion. Instead we use the α and β values from the network file, and compute it with the equation above.

It is important to note that although the formation of H_2 is a two body reaction – if we forget about the grain that only works as a catalyst – this reaction has a first order kinetics: the formation rate of H_2 depends on $n(\mathrm{H})$ and not on $n(\mathrm{H})^2$. Because of this, the reaction has its own type number, 0 (see *Reaction type numbers*). At present the formation of H_2 on grains is the only grain surface reaction that is considered in Astrochem.

Electron attachment and ion recombination on grains

Warning: Starting from version 0.x, the electron attachement and ion recombination on grains are computed in a different fashion (see below). Networks used with previous versions of Astrochem need to be updated accordingly.

Electron can hit grains and charge them. Cations may also hit grains and recombine. For example, let us consider the following reactions:

$$\mathrm{grain} + e^- \rightarrow \mathrm{grain}^-$$

$$\mathrm{C}^+ + \mathrm{grain}^- \rightarrow \mathrm{C} + \mathrm{grain}$$

The formation rate of negatively charged grains writes as:

$$\frac{dn(\mathrm{grain}^-)}{\mathrm{d}t} = k_1\, n(\mathrm{grain})\, n(\mathrm{e}^-)$$

while the recombination rate of C^+ on negatively charged grains is:

$$\frac{dn(\mathrm{C}^+)}{\mathrm{d}t} = -k_2\, n(\mathrm{grain}^-)\, n(\mathrm{C}^+)$$

For neutral grains, the electron attachement rate k_1 is given by the following expression (Semenov, Wiebe & Henning, 2004)

$$k = S\,\pi r_d^2\, v_{th} \tag{1.5}$$

with:

$$S = 1.329 \times \exp\left(-\frac{T_d}{20}\right) \tag{1.6}$$

and:

$$v_{th} = \left(\frac{8 k_B T_d}{\pi m_e}\right)^{1/2} \tag{1.7}$$

Here S is the sticking coefficient (comprised between 0 and 1) of electrons on the grains, v_{th} is the thermal velocity of the electrons, T_d is the grain temperature, and m_e is the electron mass. Note that the sticking coefficient is assumed to decrease exponentially with the temperature, so that for $T_d 20$, it is close to 0.5, and is essentially 0 at higher dust temperatures.

For charged grains, the expression above is multiplied by a factor correcting for the long-distance Coulomb attraction:

$$C_{ion} = 1 + \frac{e^2}{k_B\, r_d\, T} \tag{1.8}$$

where e is the electron charge (in statcoulombs), and r_d is the grain radius.

For neutral grains, the cation recombination rate k_2 is computed from the following expression:

$$k = \alpha\,\pi r_d^2\, v_{th} \tag{1.9}$$

where α is the (dimensionless) branching ratio for dissociative recombinations. For negatively charged grains the expression above is multiplied by C_{ion}.

Note that Astrochem considers singly charged (either positively or negatively) charged grains only; multiply charged grains are neglected.

Depletion

Molecules may accrete on dust grains and freeze-out (a process often called depletion). The formation rate of e.g. ices CO on the grains through this process is given by:

$$\frac{\mathrm{d}n(\mathrm{CO_{ice}})}{\mathrm{d}t} = k\, n(\mathrm{CO})$$

with:

$$k = S\,\pi r_d^2\, v_{th}\, n_d \tag{1.10}$$

and:

$$v_{th} = \left(\frac{8k_B T_d}{\pi m}\right)^{1/2} \tag{1.11}$$

Here S is the sticking coefficient of the molecule on the grain, n_d is the total grain density (neutral + charged) and m is the mass of the accreting species (Bergin et al., 1995).

Because no grain destruction or formation mechanisms are considered in Astrochem, n_d does not varies with time. It is therefore computed from the initial abundances of neutral and charged grains given in the input file (see *Initial abundances*). The grain size r_d is also read from this file. Both S and m are read from the network file.

Thermal desorption

Once frozen on the dust grains, molecules may evaporate through thermal or non-thermal processes. The formation rate of gaseous CO by CO ices thermal evaporation is:

$$\frac{\mathrm{d}n(\mathrm{CO})}{\mathrm{d}t} = k\, n(\mathrm{CO_{ice}})$$

where k is given by the Polanyi-Wigner equation:

$$k = \nu_0 \exp\left(-\frac{E_B}{T_d}\right) \tag{1.12}$$

with:

$$\nu_0 = \left(\frac{2N_S E_B}{\pi^2 m}\right)^{1/2}$$

Here ν_0 is the characteristic vibrational frequency of the desorbing species, E_B is the binding energy of the desorbing species on the grain surface expressed in Kelvins and N_S is the number of sites per unit surface assumed to be $3 \times 10^{15}\,\mathrm{cm}^{-2}$ (Hasegawa et al., 1992). The values of E_b and m are both read from the network file.

Cosmic-ray desorption

As mentioned above, ices may also evaporate by non-thermal processes. For example, cosmic-rays may desorb molecules from grains, either by creating hot-spots on the grain surfaces, or by heating the whole grains Leger et al. (1985). Because the energy deposited in a grain varies as Z^2, cosmic-ray desorption in mainly caused by heavy cosmic-ray ions, such as Fe. Leger et al. (1985) suggested that desorption by spot-heating dominates over desorption by whole-grain heating for grains smaller than 2.5 μm. However, recent molecular dynamics simulations indicate that for 0.1 μm grains the whole grain heating contribution is small (Bringa and Johnson, 2004).

Because of the uncertainties on this process, two different treatments are implemented in Astrochem. First, cosmic-ray desorption rates can be computed following Hasegawa and Herbst (1993), who assume that desorption occurs

mostly through whole-grain heating; when impacting grains, heavy cosmic-ray ions are assumed to impulsively heat the grains to a peak temperature of 70 K, at which most of the desorption occurs. The rate is then similar to that of thermal desorption:

$$k = f\,\nu_0 \exp\left(-\frac{E_B}{70}\right) \tag{1.13}$$

where f is the fraction of the time spent by a grain in the vicinity of 70 K between two cosmic-ray heating events, assumed to be 3.16×10^{-19} (Hasegawa and Herbst ,1993).

Alternatively, the cosmic-ray desorption rate of any specie can be given explicitly in the network file. This allows for the use of the cosmic-ray desorption rates that have been computed and/or measured for some species (e.g. H_2O and CO; Bringa and Johnson, 2004) . The user can specify in the network file which treatment to use for each species (see *Physical meaning of the rate constants used in chemical networks*). Note that no scaling of k with the cosmic ray ionization rate is performed.

Photo-desorption

Photo-desorption (i.e. desorption by UV photons) is another non-thermal desorption process. UV photons can originate in the ISRF, or in the ionization of H_2 by cosmic-rays followed by recombination (secondary UV photons). At present only photo-desorption from ISRF UV photons is implemented in Astrochem.

The photo-desorption rate of CO is for example (Oberg et al., 2009a, b):

$$\frac{\mathrm{d}n(\mathrm{CO})}{\mathrm{d}t} = k$$

with:

$$k = \chi\, I_{\mathrm{ISRF,FUV}} \exp\left(-2A_v\right)\, \pi r_d^2\, n_d\, Y_{\mathrm{PD}} \tag{1.14}$$

Here $I_{\mathrm{ISRF,FUV}}$ is the standard interstellar radiation field in the FUV (assumed to be $1.7 \times 10^8\,\mathrm{photons\,cm^2\,s^1}$; Draine, 1978), and Y_{PD} is the photo-desorption yield, i.e. the number of molecules ejected per incident photon. The latter is given by:

$$Y_{\mathrm{PD}} = Y_0\left[1 - \exp\left(-x/l\right)\right]$$

where x is the ice thickness of the considered species expressed in monolayers (ML), l is the diffusion length in ML, and Y_0 is the photo-desorption yield for thick ices (i.e. $x \gg l$). Typical values for Y_0 and l are $10^{-3}\,\mathrm{molecules\,photon^{-1}}$ and 2 ML, respectively [3]. The density of e.g. CO ices is given by:

$$x = \frac{n(\mathrm{CO_{ice}})}{N_s\,\pi r_d^2\, n_d}$$

It is interesting to note that for thick ices, photo-desorption is zeroth order process: the desorption rate does not depends on the amount of e.g. CO ices on the grains. This because UV photons can penetrate only the first ices monolayers; the bulk of ice is not affected. On the other hand, for thin ices (i.e. $x \ll l$) the desorption rate become linearly proportional to the ice thickness, and therefore on the ice abundance. Consequently for thin ices, photo-desorption is a first order process.

Astrochem follows the ice thickness of each species as a function of time. The desorption rate is then computed from the above equations, using the values of χ, r_d^2 and n_d from the input file, the A_v from the source file, and the Y_0 and l from the network file.

[3] For some species, Y_0 depends weakly on the dust temperature . This effect is not implemented in Astrochem.

1.3 Using Astrochem

In this section we present a simple example of Astrochem usage. We propose to use Astrochem to study the formation of the HCO^+ ion in a dense interstellar cloud. We suppose that the cloud is isodense and isothermal, and that it is shielded from the ISRF, so that photo-processes can be ignored. For a sake of simplicity, we also neglect the freeze-out of molecules on dust grains. In the following, we describe the various steps needed to vsolve this problem.

1.3.1 Describing the problem

In order to describe our problem, we first need create an input file that contains the various parameters the code. This file has several sections, that set the physical parameters (e.g. the cosmic ionization rate), the solver parameters (e.g. the initial and final time in the computation), the initial abundances, and a list of species we want in output. Some of these parameters are optional; if they are not specified in the input file, Astrochem will use a default value that should be suitable for most problems. Here is a what the input file for our example problem looks like (for a comprehensive description of the parameters in input files and their default value, see *Input file*):

```
[files]
source = source.mdl
chem = osu2009.chm
# Physical paramaters
[phys]
chi = 1.0
cosmic = 1.3e-17
# Solver parameters
[solver]
ti = 1e-6
tf = 1e7
# Initial abundances
[abundances]
H2      = 0.5
He      = 0.14
N       = 2.14e-5
O       = 1.76e-4
C(+)    = 7.30e-5
S(+)    = 8.00e-8
Si(+)   = 8.00e-9
Fe(+)   = 3.00e-9
Na(+)   = 2.00e-9
Mg(+)   = 7.00e-9
P(+)    = 2.00e-10
Cl(+)   = 1.00e-9
F       = 6.68e-9
e(-)    = 7.31012e-5
# Output
[output]
abundances = H3(+),e(-),CO,HCO(+)
trace_routes = 1
```

The various sections of the file are indicated by keywords within brackets. Lines starting with # are comments. The first section (`[files]`) indicates the name of the file describing our source (`source`), and the chemical network to use (`chem`). The following section (`[phys]`) sets the physical parameters of the source. Here we set the UV radiation field in Draine units (`chi`) to 1.0, and the cosmic ionization rate (`cosmic`) to $1.3 \times 10^{-17}\,\mathrm{s}^{-1}$. The solver parameters are set in following section (`[solver]`). `ti` and `tf` are the initial and final time for the calculation respectively. Both are expressed in years. The `[abundance]` section sets the initial abundances; abundances that are not specified are set to zero. The last section (`[output]`) sets parameters relative to the output of the code. `abundances` sets the name of the species for which we want to create output containing the abundances as a function

of time.. `trace_route` is an optional parameter that allow to trace the various formation and destruction routes of these species.

In addition to the input file, we need to provide a file describing our source. The file corresponding to our problem looks like this (for more information on source model files, see *Source file*):

```
# Source model file example
# cell number, Av [mag], nH [cm^-3], Tgas [K], Tdust [K]
#
0   20.0    1e+04   10.0    10.0
```

As for the input file, lines that starts with a # are comments. The file contains one line for each cell of our source. In this simple example, our source is isodense and isothermal, and therefore there is only one cell in the our source file. A more realistic source with a temperature and density gradient would be sampled in more cells.

Each line corresponding to a cell has five columns. The first column is the index of the cell, the second one is the visual extinction A_v (in magnitudes), the third one is the number density (in cm^{-3}). and the fourth and the fifth are the gas and dust temperature respectively (in Kelvins). Note that we have adopted a large visual extinction (20 magnitudes) because we neglect photo-processes

1.3.2 Running Astrochem

Astrochem is run from the command line, and takes the name of the input file as an argument:

```
% astrochem input.ini
Reading input from input.ini.
Reading source model from source.mdl.
Reading reactions network from osu2009.chm... done.
Found 6046 reactions involving 468 species.
Computing abundances in cell 0...
Done with cell 0.
Writing abundances in output files... done.
Writing formation/destruction in output files... done.
%
```

Astrochem produces one output file in HDF5 format named `astrochem_output.h5`. This file contains different datasets: a list a output species names, a list of the time steps, a matrix of abundances of the various species, and if the `trace_route` parameter is set to 1, a group of matrices containing the formation and destruction routes.

1.3.3 Plotting abundances

Astrochem comes with a program that make plots of the abundances computed by Astrochem. The program, named Plabun, allows to plot the abundances of one or several species as of function of time in a given cell. For example, the following command plots the CO, H_3^+, e^- and HCO^+ abundances:

```
% plabun --xrange=1,1e7 --yrange=1e-12,1e-4 astrochem_output.h5 CO H3(+) e(-) HCO
```

In the example above, we have set the x-axis range from 1 to 10^7 yr

and the y-axis range from 10^{-12} to 10^{-4} with the `--xrange` and `--yrange` options, respectively. The command above produces the plot shown on Fig. *Abundances as a function of time for the example problem*. Plabun has a number or other options, including a legacy mode to read old output format `.abun`; see `man plabun` for a complete list.

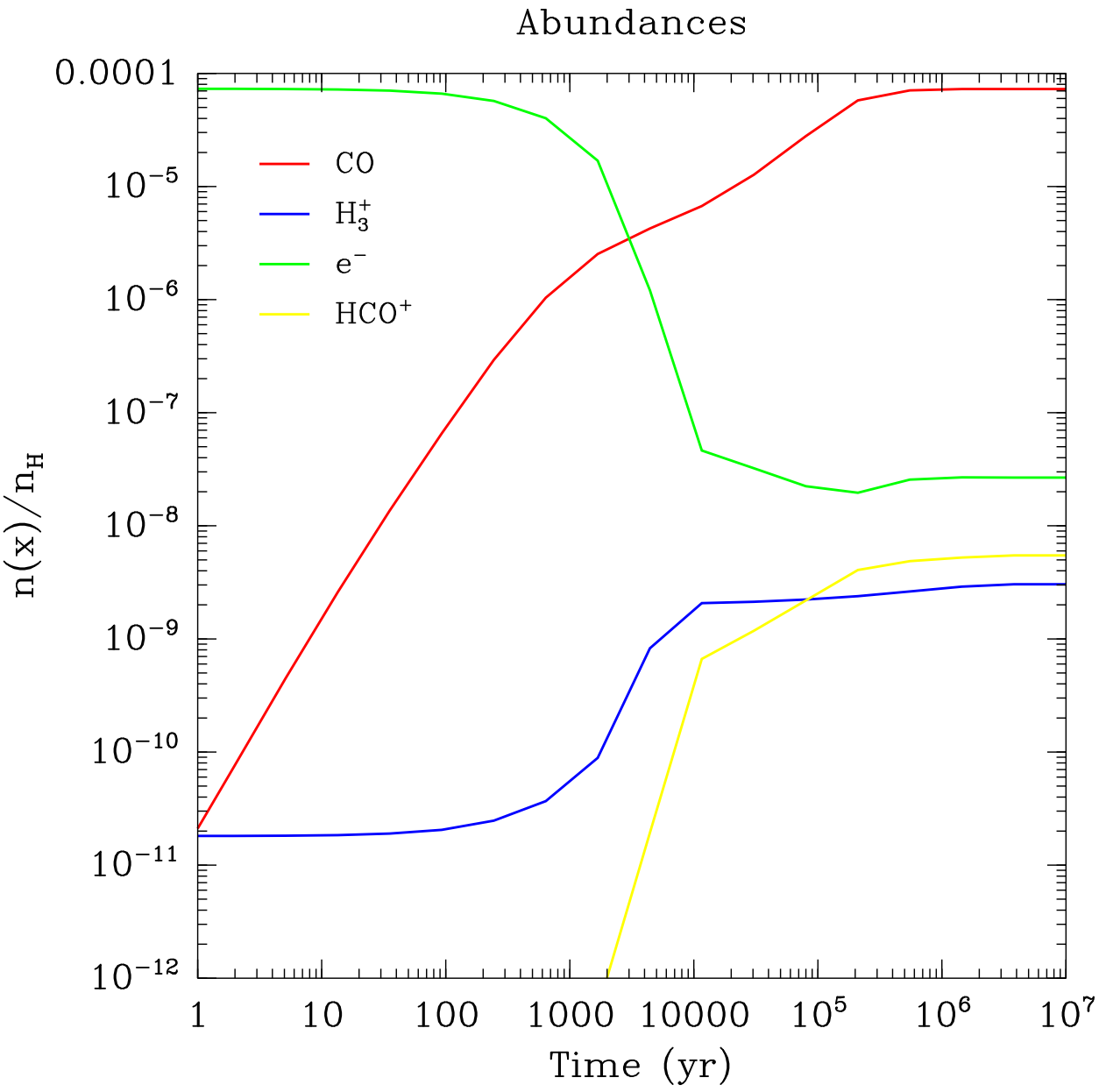

Fig. 1.1: Abundances as a function of time for the example problem

1.3.4 Identifying the main formation and destruction channels of a species

It is often useful to identify the main formation and destruction routes of a given species, for example to check that the rate of these main reactions have been determined accurately (e.g. from experiments) or are rather uncertain. This also allows to understand how the various species of a chemical network are linked together.

As already mentioned, if one turns on the `trace_route` option in the input file, Astrochem creates a dataset in the `astrochem_output.h5` file that contains the main formation and destruction routes of the species listed with `output` option. Of course these routes may change with time; therefore Astrochem saves the sixteen most important formation routes as well as the sixteen most important destruction routes at each time step and position.

A command allows to plot the main formation and destruction rate of a given species as a function of time or position. For example, one can plot the main formation and destruction routes of HCO^+ with the following command (see `man plroute` for a complete list of commands and options):

```
% plroute astrochem_output.h5 HCO(+)
```

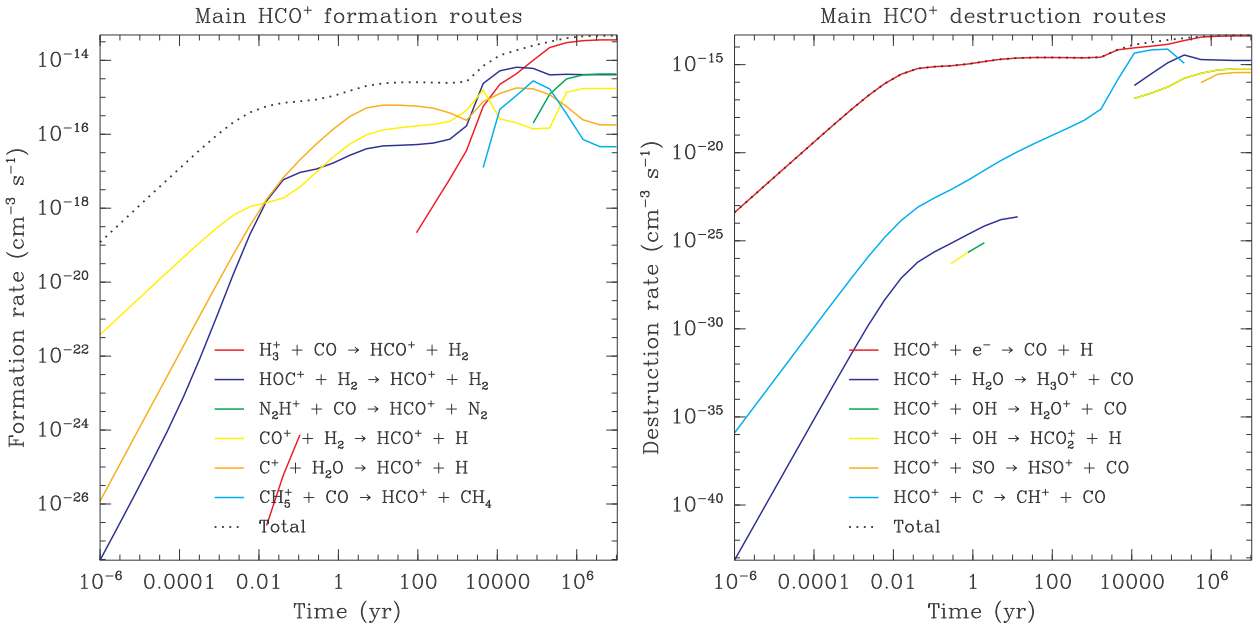

Fig. 1.2: Main HCO^+ formation and destruction routes as function of time for the example problem.

which produces the plot shown on Fig. *Main formation and destruction routes as function of time for the example problem.*.

The left panel of the plot shows the formation rate of HCO^+ (in $\mathrm{cm^{-3}\,s^{-1}}$) through the six most important formation channels [4], together with the total formation rate. Conversely, the right panel shows the destruction rate of the same species through the six most important destruction channels as well as the total destruction rate. On the left panel, we see that for $t > 10^5$ yr, the formation of HCO^+ is dominated by the reaction of CO with H_3^+. On the other hand, at any time in the simulation the destruction of HCO^+ is dominated by the dissociative recombination with electrons.

Note that because Astrochem keeps tracks of the sixteen most important formation or destruction routes at any time only, some gaps may appear in the plot. This can been seen for the reaction in red on the left panel of the plot for times between roughly 0.1 and 10 years. In this time range this reaction is not one the sixteen most important formation reaction, so Astrochem did not keep track of it in this time range.

[4] The formation and destruction reactions are ordered by the value of the integral of their formation/destruction rate over time. i.e. their total contribution to the formation/destruction of the species. The `--percent` option allows to display the relative contribution, expressed in percent, to the formation/destruction of the species.

1.4 Input and source files

1.4.1 Input file

As mentioned already, the various parameters of Astrochem are read from an input file. Although this is not mandatory, the file usually has the `.ini` file extension. The file has several sections that are delimited by a keyword within brackets (e.g. `[files]`). Each section has a number of parameters that we describe in the following.

Files

This section of the file starts with the `[files]` keyword, and specifies which file Astrochem should use for the source description (`.mdl` file) and chemical network (`.chm` file). The parameters allowed in this section are:

`source` The name of the file describing the source.

`network` The name of the chemical network file. Astrochem searches for this file in the current directory first. If it is not found, it searches for the file in Astrochem's data installation directory (`/usr/local/share/astrochem` by default).

Physical parameters

This section of the file starts with the `[phys]` keyword and specifies the physical parameters of the problem. These parameters are:

`chi` The external UV radiation field, expressed in Draine unit (χ).

`cosmic` The cosmic ray ionization rate of molecular hydrogen expressed in s^{-1} (ζ). The default value is 1.3×10^{-17}.

`grain_size` The grain radius in microns (r_d). The default value is 0.1.

`grain_gas_mass_ratio` The grain-to-gas mass ratio. The default value is 0 (no grains).

`grain_mass_density` The grain mass density, expressed in $\mathrm{kg\,m^{-3}}$. The default value is $3000\,\mathrm{kg\,m^{-3}}$, which corresponds to olivine grains.

Solver parameters

This section of the file starts with the `[solver]` and specifies the ODE solver parameters. These parameters are:

`ti` The initial time for the computation, expressed in years. The default value is 10^{-6}.

`tf` The final time for the computation, expressed in years. The default value is 10^{7}.

`abs_err` The solver absolute error (or tolerance) on the computed abundances. The default value is 10^{-20}.

`rel_err` The solver relative error on the computed abundances. The default value is 10^{-6}.

A note on tolerances: Astrochem adjusts the internal time step so that the relative error on any abundance is always lower that `rel_err`, unless the given abundance is lower that `abs_err`. Because errors on the abundances at each time step may add-up, we recommend to chose these errors quite conservatively. The default values should be suitable for most problems.

Initial abundances

This section specifies the initial abundances in the computation. Each line should contain a specie name followed by a equal sign and the initial abundance with respect to H nuclei. The initial abundances of species that are not listed in this section are assumed to be zero.

Note: Starting from version 0.7, the grain abundance is computed from the grain parameters (see *Physical parameters*). Setting the grain abundance explicitly in this section is deprecated.

Output

This section specifies what file Astrochem should create at the end of the computation. These parameters are:

`output` A list of species for which Astrochem creates an output file containing the abundance as a function of time and position. Species names must be separated by a comma. The `all` keyword may be used to have all species of the network in output.

`suffix` A suffix to append to the name of the species before the file extension (`.h5`) of the output file. This is useful when you want to run Astrochem for a number of different input files all located in the same directory; this way the results of a given simulation will not be overwritten by the results of others. A leading underscore is added to this suffix.

`time_steps` The number of time steps in output the files [5]. The default value is 32. Note that this parameter only affects the number of time steps in the *output* file. The internal time step size is set by the ODE solver in order to reach the specified absolute and relative errors on the abundances.

`trace_routes` This parameter is used to toggle the computation of the major formation and destruction routes for all species listed with the `output` parameter. If `trace_route` is set to 1, Astrochem will create a file containing the formation/destruction rate and reaction number of the 16 most important formation/destruction reaction for each specie, as a function of time and position (i.e. cell number). As for abundance files, file names for formation and destruction routes are formed with the species name possibly followed by a suffix (see below) and the `.rout` file extension.

1.4.2 Source file

Astrochem reads the physical parameters (density, temperature, visual extinction) of the astronomical source in a source file. This file can be of two formats, depending on whether the source physical parameters vary as a function of time or not.

Time-independent physical parameters

This type of source file is used to describe astronomical source in which the physical parameters do not change with time, or change on timescales longer than the chemical timescales. Although the source may have in principle any dimension, Astrochem is, for the moment, designed to study 1D spherical sources in an external radiation field only (such as a dense cloud or a prestellar core in the ISRF). Future versions will allow to study 2D sources with axisymmetrical geometries, such as protoplanetary disks.

In order to construct a time-independent source model file, one needs to sample the astronomical source in a number of spherical cells (or shells) at different source radius. What constitutes a good sampling depends on the source. Often density profiles of astronomical sources are well described by a power laws, so it is usually a good idea to sample the source in a number of logarithmically spaced cells. Of course, the larger number of cells, the longer computational

[5] Output files contain abundances for `time_steps` time values that are logarithmically sampled between `ti` and `tf`.

time. However, Astrochem may be run in parallel on multi-core computers in order to reduce the computational time (see section *Running Astrochem in parallel*).

Each line of the file corresponds to a different cell, while each column corresponds to a different parameter. These parameters are, from the leftmost to the rightmost columns:

1. The cell index. This is an integer that is used to identify each cell. The first index should be 0. Other indexes in the file should be in increasing order. All cells should have a different index.
2. The visual extinction in the cell, expressed in magnitudes.
3. The H nuclei density in the cell, expressed in cm^{-3}.
4. The gas temperature in the cell, expressed in K.
5. The dust temperature in the cell, expressed in K.
6. The radius corresponding to the cell, expressed in astronomical units (AU). This optional parameter is used for bookkeeping only; Astrochem ignores it.

Columns may be separated by any number of white spaces or tabs. Comments may written in the source file; comment lines must start with a # sign.

Time-dependent physical parameters

This type of source file is used to describe an astronomical source in which the physical parameters vary as a function of time, for example a protostar that undergoes gravitational collapse. In this case, the physical parameters (density, temperature), may come from theoretical prescriptions or from numerical simulations.

In order to construct this kind of model, one need to sample the source in a number of different cells that follow the dynamical evolution of the object as a function of time. These could correspond to the *smooth particles* from a SPH simulation, or the *buoy particles* from an adaptive mesh refinement (AMR) (magneto)hydrodynamical simulation. The source needs to be properly sampled both spatially and temporally. Indeed, Astrochem assumes that the physical parameters in each cell remain constant within each time step [6]. Therefore this time step should be sufficiently small so that the physical parameters do not vary significantly between two time steps.

Time-dependent source models have two sections, which are separated by keywords in brackets. The first one contains the times, expressed in years, and starts with the `[times]` keyword.

```
# Source model file example for a time-dependant source structure
[times]
     0     1.00e-06
     1     1.27e-06
     2     1.60e-06
     3     2.03e-06
     4     2.57e-06
(...)
   126     7.90e+06
   127     1.00e+07
```

The number on the first column is the time index, which must start at 0. The second column contains the time, expressed in years. In this example, the time on the first line is 10^{-6} years. This corresponds to the time at the end of the computation of the first time step; in other words, the time step `#0` is between $t = 0$ and $t = 10^{-6}$ years. Likewise, the time step `#127` is between 7.9×10^6 and 10^7 years, at which time the computation ends.

The second part of the file contains the physical parameters of each cell at each time step. This section must start with the `[cells]` keyword, followed by the physical properties of the first cell.

[6] This dynamical time step is different from the chemical time step. The later is adjusted internally by Astrochem to ensure a sufficient precision on the computed abundances. The former is provided by the user.

```
# cell number, time index, Av [mag], nH [cm^-3], Tgas [K], Tdust [K]
[cells]
     0         0    20.00     1.00e+04    10.00    10.00
     0         1    20.00     1.04e+04    10.00    10.00
     0         2    20.00     1.08e+04    10.00    10.00
(...)
     0       127    20.00     1.00e+06    10.00    10.00
     1         0    20.00     1.00e+05    10.00    10.00
(...)
     1       127    20.00     1.00e+07    10.00    10.00
```

The columns in this section are:

1. The cell index, starting at 0.
2. The time index.
3. The visual extinction in the cell, expressed in magnitudes.
4. The hydrogen density, in cm^{-3}.
5. The gas temperature, in Kelvins.
6. The dust temperature, in Kelvins.

Each line correspond to a different time step. In this example, the first cell (`#0`) has a density of $10^4\,\mathrm{cm}^{-3}$ and a temperature of 10 K during the first time step (`#0`), i.e. between $t = 0$ and $t = 10^{-6}$ years. During the last time step (`#127`), this cell has a density of $10^6\,\mathrm{cm}^{-3}$ and a temperature of 10 K.

Any number of cells may be given in the file. The physical parameters of the second cell (`#1`), directly follows that of the first one. For example, the second cell (`#1`) has a density of $10^4\,\mathrm{cm}^{-3}$ and a temperature of 10 K during the same time step (`#0`). Note that all cells must have the same time steps.

1.5 Chemical networks

Astrochem reads the chemical reactions and their rate coefficient from a chemical network file. This file should have `.chm` extension. Several networks are distributed with Astrochem, but the user may also write its own network. In the following, we describe the networks files that are distributed with Astrochem, as well as the format of these files. Finally, we explain how networks in other formats can be converted to Astrochem format.

1.5.1 Networks provided with Astrochem

The following networks are provided with Astrochem:

`osu2009.chm` This network file contains the reactions and rates from the Ohio State University (OSU) astrochemistry database, that is maintained by Eric Herbst. It corresponds to the January 2009 version of the database. This network contains 6046 reactions and 468 species, including anions.

`osu2008.chm` This network file contains the September 2008 version of OSU database. It includes 4457 reactions and 452 species (no anions).

These networks can be found in the network directory of the source distribution. When installing Astrochem, they are copied in the data installation directory (`/usr/local/share/astrochem` by default).

1.5.2 Network file format

Astrochem network file format is meant to be easily read and edited. Therefore chemical reactions in this files are written as you would write them on a piece of paper. Here is an example of a (incomplete) network file:

```
# A few reactions extracted from osu2008.chm
H     + H          -> H2                4.95e-17  5.00e-01  0.00e+00  0    1
H2    + cosmic-ray -> H2(+)  + e(-)      9.30e-01  0.00e+00  0.00e+00  1   39
H3(+) + CO         -> HCO(+) + H2        1.61e-09  0.00e+00  0.00e+00  2 1756
H3(+) + e(-)       -> H      + H    + H  4.36e-08 -5.20e-01  0.00e+00  9 3746
CO    + uv-photon  -> C      + O        3.10e-11  0.00e+00  2.54e+00 13 4297
(...)
```

As for input and model files, lines that starts with the # character are comments. Each line of the file corresponds to a different reaction. Lines have two parts: the chemical equation, and a list of five numbers that correspond to the rate constants, reaction type and reaction number.

The chemical equation is composed of one, two or three reactants, and one, two, three or four products. Each reactants and products are separated by a white space, a + sign, and another white space. To disentangle the + sign between reactants from the ones corresponding to ions, ion charge must be put in parenthesis: for example, the HCO^+ ion must be written as `HCO(+)` in the network file. Reactants and products are separated by a white space, an arrow (->), and another white space.

In general, Astrochem computes the formation rate of each product (or the destruction rate of each reactant) through a given reaction by multiplying the reaction rate by the product of the reactants. For example, for the reaction:

```
H3(+) + e(-)       -> H      + H    + H  4.36e-08 -5.20e-01  0.00e+00  9 3746
```

the destruction rate of H_3^+ and e^- is computed as:

$$\frac{dn(H_3^+)}{dt} = \frac{dn(e^-)}{dt} = -k\, n(H_3^+)\, n(e^-)$$

while the formation rate of H is computed as:

$$\frac{dn(H)}{dt} = k\, n(H_3^+)\, n(e^-)$$

In other words, single body reactions (e.g. cosmic-ray ionization, or UV ionization) are assumed to have a first order kinetics, two body reactions are assumed to have a second order kinetics, etc. However, there are two exceptions to this rule. First the formation of H_2

on the grains is assumed to be of the first order (see *formation on grains*). Second, UV photodesorption is assumed to have a zeroth order kinetics when the ice thickness is large enough (see *Photo-desorption*).

Several reactants and products are not *bona fide* chemical species, but are just meant to make the reading of the network easier. These *pseudo-species* are `cosmic-ray` (for cosmic-ray direct or indirect ionization or desorption reactions), `uv-photon` (for photo-ionization, photo-dissociation and photo-desorption reactions) and `photon` (for radiative association reactions). All of these are ignored by Astrochem.

The five numbers following the products are the three rate constants (that we note a, b and c respectively), the reaction type, and the reaction number. The reaction type is a signed integer that identifies the kind of the reaction (e.g. ion-molecule, dissociative recombination, etc.). Table *Reaction type numbers* lists the various reaction types together corresponding type numbers [7]. The reaction number is an integer that identify each reaction in a unique fashion: every reaction must have a different number. Reaction numbers must start at 1, but they are not necessarily contiguous. For example you may want to identify gas-phase reactions by numbers between 1 and 6046, and gas-grain reactions by numbers starting at 10000.

[7] The reaction types adopted in Astrochem follows closely that used in the OSU database.

Table 1.1: Reaction type numbers

Type number	Reaction type
0	H_2 formation on grains
1	Cosmic-ray ionization or cosmic-ray induced photo-reactions
2	Ion-molecule reactions, Charge exchange reactions
3	Negative ion - neutral species reactions
4	Radiative association
5	Associative ejection
6	Neutral + Neutral $\rightarrow$ ion + electron
7	Neutral-Neutral chemical reactions
8	Neutral-Neutral radiative association
9	Dissociative recombination
10	Radiative recombination
11	Positive ion - Negative ion recombination
12	Electron attachment
13	Photo-ionization, Photo-dissociation
20	Freeze-out on grains
21	Thermal desorption
22	Cosmic-ray induced desorption
23	Photo-desorption
24	Electron attachment on neutral grains
25	Electron attachment on positively charged grains
26	Cation recombination on negatively charged grains
27	Cation recombination on neutral grains

Astrochem computes the rate of each reaction from the a, b and c rate constants. The physical meaning of these constants depends on the type of the reaction. Table *Physical meaning of the rate constants used in chemical networks* gives physical meaning of these for each reaction type, as well as the equation that is used to computed the rate. Because reaction rates generally do not vary with time, Astrochem computes them only once for each cell. The only exception is the rate of UV photo-desorption reactions, that depends on the ice thickness; therefore the rate for these reactions is computed for each solver internal time step.

Table 1.2: Physical meaning of the rate constants used in chemical networks

Type number	Equation	a	b	c
0	(1.4)	α	β	-
1	(1.2)	α	β	-
2-12	(1.1)	α	β	γ
13	(1.3)	α	-	-
20	(1.10)	S	m/m_{H}	
21	(1.12)	-	m/m_{H}	E_b
22	(1.13)	k [8]	m/m_{H}	E_b
23	(1.14)	Y_0	-	l
24	(1.5)	-	-	-
25	(1.5) [9]	-	-	-
26	(1.9)	S	m/m_{H}	
27	(1.9) [11]	S	m/m_{H}	

1.5.3 Convert networks to Astrochem format

Astrochem comes with a tool called Chmconvert that converts network files to Astrochem native format (`.chm`). Chmconvert supports the OSU database and the KIDA formats. The file format is determined from the file extension, which must be `.osu` for Ohio State University database files, and `.kida` for KIDA. Networks can be converted as follows (see `man chmconvert` for more information):

```
% chmconvert -o osu2009.chm osu2009.osu
```

The `-o` option is used to select the name of the output file. If not specified, the network is copied to the standard output. Two networks provided with Astrochem (`osu2008.chm` and `osu2009.chm`; see *Chemical networks*) are automatically generated from the OSU files using this tool.

1.6 Output files

Note: Starting from version 0.7, the abundances are the formation/destruction routes are stored in a single binary file. This file can be converted into the legacy formats (`.abun` and `.rout` for the abundances and routes, respectively) using the `converttolegacy` utility.

Astrochem results are stored in binary a file named `astrochem_output.h5`. The file format is based on HDF5, a hierarchical data format that allows to store large datasets efficiently. The data can be easily accessed from Python using the Astrochem Python module. For example, the CO abundance as a function of time from the output file can be read as follows:

```
>>> from astrochem import tools
>>> species = tools.listspecies("astrochem_output.h5") # species in the file
['H3(+)' 'e(-)' 'C(+)' 'CO' 'HCO(+)']
>>> time, abun = tools.readabun("astrochem_output.h5", 'CO')
>>> print time
[  1.00000000e-06   2.62636353e-06   6.89778538e-06   1.81160919e-05
   4.75794431e-05   1.24960914e-04   3.28192787e-04   8.61953566e-04
   2.26380341e-03   5.94557071e-03   1.56152301e-02   4.10112707e-02
   1.07710506e-01   2.82886943e-01   7.42963951e-01   1.95129342e+00
   5.12480588e+00   1.34596032e+01   3.53498111e+01   9.28414545e+01
   2.43835410e+02   6.40400427e+02   1.68192432e+03   4.41734470e+03
   1.16015530e+04   3.04698957e+04   8.00250228e+04   2.10174801e+05
   5.51995432e+05   1.44974067e+06   3.80754602e+06   1.00000000e+07]
>>> print abun[:,0] # print the CO abundance in the first cell
[  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   1.53148001e-19   2.69657101e-18
   4.50311276e-17   6.70402913e-16   8.05604252e-15   6.97572674e-14
   4.15800484e-13   2.15041377e-12   1.20634392e-11   7.31990562e-11
   4.48849346e-10   2.59982365e-09   1.35763909e-08   6.55299735e-08
   2.93246622e-07   1.03837261e-06   2.52591748e-06   4.25450305e-06
   6.71696964e-06   1.26719213e-05   2.79093860e-05   5.76522869e-05
   7.08298745e-05   7.27426681e-05   7.27526266e-05   7.27520464e-05]
```

The formation/destruction routes may also be accessed from Python.

1.7 Running Astrochem in parallel

Astrochem may be run in parallel for sources containing more than one cell. In this kind of source, if diffusion and advection are neglected, each cell is independent from others: abundances in a given cell are not affected by

abundances in other cells. Therefore abundances may be computed in several cells simultaneously.

Parallelism in Astrochem is implemented using the OpenMP standard. Basically, OpenMP is a set of compiler directives that allows a program to be run in parallel on a shared-memory parallel computer, such as a multi-core and/or multi-CPU machine. Astrochem forks in several threads that are run simultaneously on several cores and/or CPUs. Each thread corresponds to a computation in a different cell. When run in parallel, Astrochem execution time scales almost linearly with the inverse of the number of cores or CPUs. For example, for a 64 cells source, Astrochem should run almost eight times faster on a quad-core dual-CPU (i.e. eight cores in total) than on single-core single-CPU computer.

The compilation of the parallel version of Astrochem can be turned on by specifying the `--enable-openmp` option to the configure script, e.g. (see the `INSTALL` file in the source distribution for more information):

```
./configure --enable-openmp
```

The configure script will attempt to detect if your compiler supports the OpenMP standard [10]. Then one need to set the `OMP_NUM_THREADS` environment variable to the number of threads to be run in parallel. It is recommended to set this variable the number of cores on the machine (e.g. 8 for a eight core computer). With the bash shell, this is done as follows:

```
export OMP_NUM_THREADS=8
```

It is also a good idea to sample the source in a number of cell that is a multiple of the number of threads (e.g. 64 cells for 8 threads) so that all cores 100% of the time during the computation.

1.8 Calling Astrochem from another code

Astrochem can be called from another code through an application programming interface (API). This allows to compute the chemical evolution of a gas cell within another code, typically an hydrodynamic or magneto-hydrodynamic (MHD) simulation. In practice, Astrochem functions that are part of the API are included in a dynamic library, against which other codes may be linked. Both the C and Python languages are supported.

1.8.1 From C

Note: In this section, we give an example on how to call the Astrochem library from a C code. For a full description of the API, see Astrochem C API reference.

A typical call of Astrochem library can be decomposed in several steps. First, one must select a chemical network file, which is done as follows:

```
#include <libastrochem.h>

int verbose = 0;
char *chem_file = "osu2009.chm";
net_t network;
read_network (chem_file, &network, verbose);
```

This creates a network structure of type *net_t*. In principle, one should check the return code of the *read_network()* function to make sure that no error occurred while reading the network file. This is not done in this example for a sake of clarity.

The second step is to set the physical parameters of the model. This is done by defining a structure of type *phys_t*, and by setting the different elements of this structure:

[10] Most compilers (including GCC, starting from version 4.2) support OpenMP.

```
phys_t phys;
phys.cosmic = 1e-17;
phys.chi = 0;
phys.grain_abundance = 0;
```

The parameters are the same than in input file. Please refer to *Input file* for details. As for input files, parameters that are not set explicitly are set to their default value.

The third step is to allocate a work array that will contain the abundances of all species at a given time. We also need to set the initial abundances:

```
const char* species[]  = {"CO", "HCO(+)", "e(-)"};
const double initial_abundances[] = {1e-4, 1e-9, 1e-9};
double *abundances;
alloc_abundances (&network, &abundances);
set_initial_abundances (species, 3, initial_abundances, &network, abundances);
```

Obviously, the species in the `char*` array must be present in the network.

The fourth step is to set the initial density, visual extinction and temperature of the gas cell:

```
double density = 1000;
double av = 20;
double temperature = 10;

cell_t cell;
cell.nh = &density;
cell.av = &av;
cell.tgas = &temperature;
cell.tdust = &temperature;
```

The fifth step is to initialize the solver, and to set the absolute and relative error:

```
double abs_err = ABS_ERR_DEFAULT;
double rel_err = REL_ERR_DEFAULT;
astrochem_mem_t astrochem_mem;

solver_init (&cell, &network, &phys, abundances , density, abs_err, rel_err, &astrochem_mem );
```

The `astrochem_mem` variable is a structure of type *`astrochem_mem_t`* that contains the various parameters and work arrays of the solver.

The sixth step is to call the solver to *advance time,* i.e. to compute the abundances up to a given time:

```
double time = 0;
time += 1e-6;
solve (&astrochem_mem, &network, abundances, time, verbose);
```

The *`solve()`* function updates the abundance array; after a successful call, the array contains the abundances of all species at a given time. The choice of the time step is left to the user. It should be sufficiently small so that the physical properties of the cell do not change much, and can approximated as being constant.

This step is repeated an number of times, until the dynamical simulation finishes. Between two calls, the cell properties needs to be updated with the new values of the density, temperature, etc. that are computed in the dynamical code:

```
time += 1e-6;
density = 2000;
temperature = 15;
solve (&astrochem_mem, &network, abundances, time, verbose);
```

In this exemple, `cell.nh`, `cell.tgas`, `cell.tdust` and `cell.av` are pointers to `density`, `temperature`, etc, so we can simply update the values of `density` and `temperature`.

The seventh and final step is to free the arrays and structures that are used by Astrochem:

```
solver_close (&astrochem_mem);
free_abundances (abundances);
free_network (&network);
```

1.8.2 From Python

Note: In this section, we give an example on how to run Astrochem from Python. For a full description of Astrochem Python module, see Astrochem Python module.

Calling Astrochem from Python is simpler than from C. First, one need to import the `astrochem.wrapper` and `numpy` modules:

```
from astrochem.wrapper import *
import numpy
```

The input physical parameters (see *Physical parameters*) are set using the `phys` object:

```
p = phys()
p.cosmic = 1e-17
p.chi = 0
```

Note that parameters that are not set explicitly are set to their default value. Initial abundances are set with a dictionary.

```
initial_abundances = {"CO": 1e-4, "HCO(+)": 1e-9, "e(-)": 1e-9}
```

Obviously, the species in the dictionnary must be present in the network. The density, visual extinction and temperature of the source are set with the `cell` object:

```
density = 1000
av = 20
tgas = 10
tdust = 10
c = cell(av , density, tgas,  tdust)
```

The next step is to initialize the solver:

```
verbose = 0
abs_err = 1e-15
rel_err = 1e-6
s = solver(c,  "network.chm", p , abs_err, rel_err, initial_abundances, density, verbose)
```

Then actual computation can be done by *advancing time,*, which is typically done within a loop:

```
time = 0
for i in range(1, 128):
    time+=1e-6
    try:
        abundances = s.solve(time)
    except ArithmeticError as e:
        raise "Something went wrong: %s" % e
```

The `solve()` function updates the internal abundance array and then return it as a dictionary.

1.9 Astrochem authors

The following authors have contributed to Astrochem:

Sébastien Maret wrote Astrochem.

Ted Bergin wrote an original version of this code in Fortran. Although the present version of Astrochem has been written from scratch, many ideas on its implementation were borrowed from the original Fortran version.

1.10 Contributing to Astrochem

Astrochem is hosted by GitHub, a web-based hosting service for software development that uses the Git revision control system. From the Astrochem project page on GitHub , you can download Astrochem source code, either as a tar file, or using Git. Latest releases source are available from this page as well. Bugs and issues may also be reported there.

You can help the development of Astrochem in a number of ways: by testing the package and reporting any bugs that you find; by making improvements that you develop available to others; by working on the problems or items listed on GitHub.

CHAPTER 2

Astrochem C API reference

This document gives a comprehensive list of the *types* and *functions* that are defined in Astrochem C API. These types and functions are defined in the *libastrochem.h* header file. For an example of how to use the C API, see *calling Astrochem from C*.

2.1 Types

astrochem_mem_t
Astrochem memory structure

cell_t
Structure containing the parameters of a gas cell

double **av**
Visual extinction in the cell, in magnitudes

double **nh**
H nuclei density in the cell, in cm^{-3}

double **tgas**
Gas temperature in the cell, in K

double **tdust**
Dust temperature in the cell, in K

net_t
Structure containing a chemical network

int **n_species**
Number of species in the network

int **n_alloc_species**
Number of allocated species in the network structure

species_t ***species**
Structure containing the species in the network

int **n_reactions**
Number of reactions

react_t ***reactions**
Structure containing the reactions in the network

phys_t

Structure containing the physical parameters

double **chi**
External UV radiation field, in Draine units

double **cosmic**
Cosmic ray ionization rate of molecular hydrogen, in s^{-1}

double **grain_size**
The grain radius, in micron

double **grain_abundance**
The grain abundance

double **grain_gas_mass_ratio**
The grain-to-gas mass ratio

double **grain_mass_density**
The grain mass density in $\mathrm{kg\,m}^{-3}$

2.2 Functions

int **alloc_abundances**(const *net_t* network*, double** *abundances*)
Allocate an array to store the abundances for all species in a network

Parameters

- **network** (net_t*) – Network structure
- **abundances** (*double***) – Pointer on the abundance array

Returns EXIT_SUCCESS if the allocation was successful, EXIT_FAILURE otherwise

void **free_abundances**(double* *abundances*)
Free the array containing the abundances

Parameters

- **abundances** (*double***) – Pointer on the abundance array

int **set_initial_abundances**(const char** *species*, int *n_initialized_abundances*, const double* *initial_abundances*, const *net_t* network*, double* *abundances*)
Set the initial abundances

Parameters

- **species** (*char***) – Array containing the species name
- **n_initialized_abundances** (*int*) – Number of initial abundances
- **initial_abundances** (*double**) – Array containing the initial abundances
- **network** (net_t*) – Network structure
- **abundances** (*double**) – Array containing the abundances of all species

Returns 0

int **solver_init**(const *cell_t* cell*, const *net_t* network*, const *phys_t* phys*, const double* *abundances*, double *density*, double *abs_err*, double *rel_err*, *astrochem_mem_t* astrochem_mem*)
Initialize the solver

Parameters

- **cell** (cell_t*) – Cell structure

- **network** (net_t*) – Network structure
- **phys** (phys_t*) – Physical parameters structure
- **abundances** (*double**) – Array containing the abundances of all species
- **density** (*double*) – Initial density, in cm^{-3}
- **abs_err** (*double*) – Solver absolute error on the abundances
- **rel_err** (*double*) – Solver relative error on the abundances
- **astrochem_mem** (astrochem_mem_t*) – Astrochem memory structure

Returns 0 if the initialization was successful, -1 otherwise.

int **solve** (*astrochem_mem_t* astrochem_mem*, const *net_t** *network*, double* *abundances*, double *time*, const *cell_t** *new_cell*, int *verbose*)

Solve the system of ODE

This function solve the system of ODE up to a given time, and update the abundance array. If the physical parameters in the gas cell have changed since the last call, a pointer to cell structure must be passed to the function; if not, a null pointer must be passed instead.

Parameters

- **astrochem_mem** (astrochem_mem_t*) – Astrochem memory structure
- **network** (net_t*) – Network structure
- **abundances** (*double**) – Array containing the abundances of all species
- **time** (*double*) – Time, in seconds
- **new_cell** (cell_t*) – New cell structure if the physical parameters have changed since the last call
- **verbose** (*int*) – Verbosity (0 for quiet, 1 for verbose)

Returns 0

void **solver_close** (*astrochem_mem_t* astrochem_mem*)

Close the solver

Parameters

- **astrochem_mem** (astrochem_mem_t*) – Astrochem memory structure

int **read_network** (const char* *chem_file*, *net_t** *network*, const int *verbose*)

Read a chemical network

This function reads a chemical network in Astrochem format (.chm) and creates a network structure containing all the reactions.

Parameters

- **chem_file** (*char**) – Network filename
- **network** (net_t*) – Network structure
- **verbose** (*int*) – Verbosity (0 for quiet, 1 for verbose)

Returns EXIT_SUCCESS after a successful call, EXIT_FAILURE otherwise

void **free_network** (*net_t * network*)

Free a chemical network structure

Parameters

- **network** (net_t*) – Network structure

CHAPTER 3

Astrochem Python Module reference

This documents gives a description of Astrochem Python module. The module itself contains two packages: the *tool package*, which allows to work on chemical networks and astrochem output files and *wrapper package* , which allows to run Astrochem from Python. For an example on how to do this, see *calling Astrochem from Python*

3.1 Tools (`astrochem.tools`)

Various tools for Astrochem.

`tools.`**`converttolegacy`**(*filename*, *specie*)
Convert a hdf5 output file specific species to .abun and .rout legacy format.

Parameters

- **`filename`** (*str*) – Path to the output file
- **`species`** (*str*) – Name of specie to read abundance and route of

`tools.`**`listspecies`**(*filename*)
Print available species from an hdf5 output file and return it in a array

Parameters **`filename`** (*str*) – Path to output file.

Returns Species list

Return type list if str

class `tools.`**`network_reader`**(*reactions*)
Chemical network reader class.

`__repr__`()
Returns the string representation of a network.

Returns The string representation of the network.

Return type str

`duplicate_react_numbers`()
Find reactions with the same reaction number.

Returns List of duplicated reaction_numbers.

Return type list of str

`duplicate_reactions`()
Find duplicate reactions.

Returns List of reaction_numbers of duplicated reactions.

Return type list of int

static **fromfile** (*f*, *fileformat*)
Read a network from a file.

This function reads a chemistry network from a file and creates a network instance. Supported formats are chm, osu and kida.

Parameters

- **f** (*file*) – Network file
- **fileformat** (*str*) – Network file format ("chm", "osu" or "kida")

Returns Network

Return type *network_reader*

getreact (*number*)
Returns the reaction with a given number.

Parameters **number** (*int*) – The reaction number.

Returns The reaction found.

Return type *reaction*

Raises `ValueError` – If no reaction with this number if found.

tofile (*f*, *renumber=False*)
Write network in a file.

Parameters

- **f** (*file*) – Network file handle
- **renumber** (*bool, optional*) – Renumber reactions (default False)

class `tools.`**reaction** (*reactants*, *products*, *alpha*, *beta*, *gamma*, *rtype*, *rnumber*)
Chemical reaction class.

reactants
list of str

List of reactants.

products
list of str

List of products.

alpha
float

Reaction constant.

beta
float

Reaction constant.

gamma
float

Reaction constant.

rtype
int

Reaction type.

rnumber
int

Reaction number.

__eq__(*other*)
Compares the reaction with another.

This method compares two reactions. The reactions are supposed to be equal if both the reactants and products are equal, regardless of the reaction rates.

Parameters **other** (*reaction*) – Reaction instance to compare with.

Returns True if the reactions are equal, False otherwise.

Return type bool

__repr__()
Returns the string representation of a reaction.

Returns The string representation of the reaction.

Return type str

totex()
Returns a reaction in TeX format.

Returns TeX formated reaction string.

Return type str

tools.**readabun**(*filename*, *specie*)
Read abundances for a specific specie from an hdf5 output file and return arrays of time and abundances

Parameters

- **filename** (*str*) – Path to the output file
- **specie** (*str*) – Name of specie to read abundance of

Returns

- **timesteps** (*list of float*) – List of timesteps
- **abundance** (*list of float*) – List of abundances

tools.**readabunlegacy**(*filename*)
Read an abund file and return arrays of time and abundances.

Parameters **filename** (*str*) – Path to the abund file.

Returns

- **timesteps** (*list of floats*) – List of timesteps
- **abundances** (*list of floats*) – List of abundances

tools.**readfilesattrs**(*filename*)
Read chem_file and source_file attributes from an hdf5 output file

Parameters **filename** (*str*) – Path to output file.

Returns

- **chemfile** (*str*) – chem_file attribute
- **sourcefile** (*str*) – source_file attribute

`tools.`**`readrout`**(*filename*, *specie*)

Read a rout from a hdf5 output file and return arrays of time, shell number, formation/destruction rates.

Parameters

- **`filename`** (*str*) – Path to the output file.
- **`specie`** (*str*) – Name of specie to read route of.

Returns

- **timesteps** (*list of float*) – List of timesteps.
- **shells** (*list of float*) – List of shell numbers.
- **formation_reac** (list of *reaction*) – List of formation reactions.
- **formation_rate** (*list of float*) – List of formation rates.
- **destruction_reac** (list of *reaction*) – List of destruction reactions.
- **destruction_rate** (*list of float*) – List of destruction rates.

`tools.`**`readroutlegacy`**(*filename*)

Read a rout file and return arrays of time, cell number, formation/destruction rates

Parameters **`filename`** (*str*) – Path to the output file.

Returns

- **timesteps** (*list of float*) – List of timesteps.
- **shells** (*list of float*) – List of shell numbers.
- **formation_reac** (list of *reaction*) – List of formation reactions.
- **formation_rate** (*list of float*) – List of formation rates.
- **destruction_reac** (list of *reaction*) – List of destruction reactions.
- **destruction_rate** (*list of float*) – List of destruction rates.

3.2 Wrapper (`astrochem.wrapper`)

Python wrapper for libpyastrochem.

class `wrapper.`**`cell`**(*av*, *nh*, *tgas*, *tdust*)

Cell class.

`av`

float

Visual extinction in magnitudes.

`nh`

float

Hydrogen density in cm^-3.

tgas
float

Gas temperature in K.

tdust
float

Dust temperature in K.

class wrapper.**network**(*chem_file*, *verbose*)
Network class.

chem_file
str

File containing a network to load.

verbose
int

Verbose if 1, Quiet if 0.

class wrapper.**phys**
Physical parameters to use in chemical reaction solver.

chi
float

Chi physical property.

cosmic
float

Cosmic physical property.

grain_abundance
float

Grain Abundance physical property.

grain_size
float

Grain Size physical property.

class wrapper.**solver**(*cell*, *chem_file*, *phys*, *abs_err*, *rel_err*, *initial_abundances*, *density*, *verbose*)
Chemical reaction solver.

cell
cell

Chemical cell class to use in solver.

chem_file
str

Chemical network file string to load network from and use in solver.

phys
phys

Physical properties class to use in solver.

abs_err
float

Absolute acceptable error to use in solver.

rel_err
float

Relative acceptable error to use in solver.

initial_abundances
dict

Initial abundances (format {Species:Value}).

density
float

Density to use in solver.

verbose
int

verbose if 1, quiet if 0.

solve (*time*, *new_cell*)
Solve chemical reaction for a certain time

Parameters

- **time** (*float*) – Time to solve the system at
- **new_cell** (*cell*) – Cell class to use in solver, optionnal

t

w

Symbols

A

B

C

D

F

G

I

L

N

P

R

S

T

V

W