# ZDPLASKIN v. 1.3

## Zero-Dimensional PLASma KINetics solver

## USER'S GUIDE

**University of Toulouse, LAPLACE**

**CNRS-UPS-INP, Toulouse, France**

**March 2012**

## ABOUT

ZDPlasKin is a Fortran 90 module designed to follow the time evolution of the species densities and gas temperature in non-thermal plasmas with an arbitrarily complex chemistry. A Boltzmann equation solver (BOLSIG+) incorporated in ZDPlasKin provides values of electron transport and rate coefficients when the electron energy distribution function is non-Maxwellian.

ZDPlasKin is now available as freeware and can be downloaded from the following address: http://www.zdplaskin.laplace.univ-tlse.fr.

---

## SYSTEM REQUIREMENTS

ZDPlasKin has been tested and runs well on Microsoft Windows, Mac OS X or Linux OS. A Fortran 90 compiler is required. We have tested ZDPlasKin with the following compilers: Intel Fortran, Lahey/Fujitsu Fortran, Compaq Fortran and gFortran (available as freeware).

---

## WHO CAN USE ZDPlasKin?

The user must be able to write simple Fortran code, either adapting the example master codes or starting from scratch and making calls to ZDPlasKin following the call structures defined in the user's guide.

A typical user with some knowledge of Fortran can learn to use the basic features of ZDPlasKin in about one day. Making use of all the advanced features requires more experience and familiarity with the options available in the ZDPlasKin library routines.

Several demo versions of ZDPlasKin in executable format are available for downloading. These are intended to illustrate the kinds of problems that can be solved and provide an overview of the output options.

---

## PHYSICAL MODEL

The time evolution of the species densities in a plasma can be formulated as a set of coupled rate equations. In ZDPlasKin, the time evolution of $[N_i]$, density of species i = 1 ... $i_{max}$, as represented in eq. **(1)**, is determined numerical integration, starting with some

initial conditions defined by the user. The source terms $\boldsymbol{Q_{ij}}$ corresponding to the contributions from different processes j = 1 ... $j_{max}$ are constructed automatically from information provided by the user in the input data file. The source terms corresponding to reaction **(2)**, for example, are constructed using with reaction rate $\boldsymbol{R_j}$ **(3)**. The source terms are shown explicitly in eq. **(4)**.

$$\frac{d[N_i]}{dt} = \sum_{j=1}^{j_{max}} Q_{ij}(t) \qquad (1)$$

$$a\mathrm{A} + b\mathrm{B} \quad [\ +\delta\varepsilon\ ] \quad \rightarrow \quad a'\mathrm{A} + c\mathrm{C} \quad [\ +\delta\varepsilon\ ] \qquad (2)$$

$$R_j = k_j[\mathrm{A}]^a[\mathrm{B}]^b \qquad (3)$$

$$Q_\mathrm{A} = (a' - a)R,\ Q_\mathrm{B} = -bR,\ Q_\mathrm{C} = cR \qquad (4)$$

$$\frac{N_{gas}}{\gamma - 1}\frac{dT_{gas}}{dt} = \sum_{j=1}^{j_{max}} \pm\delta\varepsilon_j \cdot R_j + P_{elast} \cdot [N_e] \qquad (5)$$

The evolution of the gas temperature is optionally taken into account. When included, it is determined from the solution of the heat transport equation in the form given in **(5)** (adiabatic isometric approximation) with known specific gas heat ratio $\boldsymbol{\gamma}$. The second term in the right side is Joule heating due to the electron current and corresponds to elastic election-neutral collisions. It is computed using the BOLSIG+ solver. Joule heating due to the ion currents is assumed negligible, however, it can be easily implemented into the user's master code.

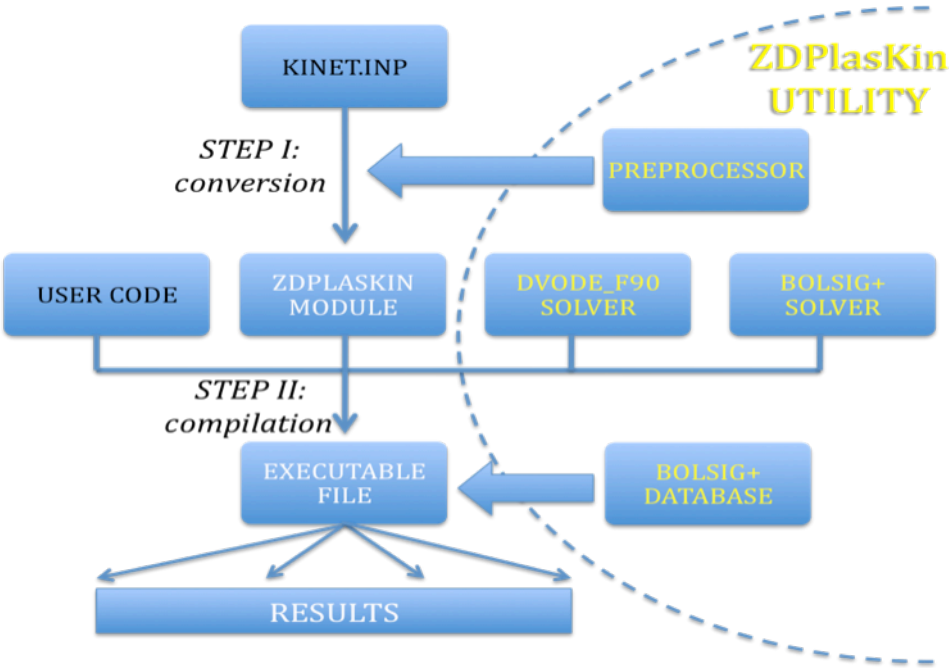
---

## GENERAL STRUCTURE

A 2-step structure was chosen to minimize computational overhead and enhance execution speed.

**Step I**: The preprocessor converts the input text file into a customized Fortran module with the user-supplied input data for the plasma chemistry incorporated directly into the code.

**Step II**: The user must provide a short master code to call ZDPlasKin library routines that perform the time integration and update electron transport and rate coefficients using BOLSIG+. Output files are written and diagnostic routines are called from the master code. The master code must be compiled and linked with the ZDPlasKin Fortran routines.




---

## STRUCTURE OF INPUT DATA FILE

The input data file consists of four different sections as described below. The input file is case non-sensitive and the lines starting with # are treated as a comment. Every section starts with the **NAME** of the section and ends with **END** and consists of one or a few lines. The following section order is recommended **ELEMENTS** - **SPECIES** - **BOLSIG** - **REACTIONS**. The max allowed length of a line is 256 symbols.

| SECTION NAME | DESCRIPTION |
|---|---|
| ELEMENTS | A list of chemical elements to be considered in the problem. |
| SPECIES | A list of species to be considered. |
| BOLSIG (optional) | A list of species for BOLSIG+, a Boltzmann equation solver, that provides values of electron transport and rate coefficients for both Maxwellian and non-Maxwellian electron energy distribution function. |
| REACTIONS | A list of reactions with corresponding constant rates. |

### Section ELEMENTS

Contains a list of "element" names to be used in ZDPlasKin. The element names are ASCII format and separated by one or more spaces. The element names must start with

a letter and must not contain a **^**, **"**, **=** or **!** symbol. The max allowed length of an element name is 124 symbols. Element **E** is always supposed to be electrons.

Example:

```
ELEMENTS
Ar  E
END
```

N.B.: Here you can declare either the normal elements of Mendeleev's table (plus the electrons) or declare some virtual elements with other names to differentiate them. Virtual elements are a flexible tool for simulating various processes within this 0D approach (e.g. diffusion losses can be approximated by an equivalent volume loss of species **X** producing a species **X(w)** which represents particles having diffused to the walls. In this case **X(w)** must be declared as a species.)

## Section SPECIES

Contains the list of species used in following sections **REACTIONS** and **BOLSIG**. The species must not contain a **"**, **=** or **!** symbol. The max allowed length of a species name is 124 symbols. Species **E** is always treated as electrons. Excited states must be preceded in parentheses or marked with an asterisk(s) after the species name, for example, N2(A) or N2*. The positive/negative ions are marked as **^+** or **^-** after the species name, for example, N4^+.

A species name can be organized using one or a few elementary blocks in the following order: ... [*element name*][*number of elements*][*any of* *, (), [], and {}][^+ *or* ^-] ... . N2(A)^+O2(a)^-H2(X[m]) if an example of complex species.

The species names must be composed of the element names declared above.

Example:

```
SPECIES
E  Ar  Ar*  Ar^+  Ar2^+
END
```

## Section BOLSIG

This is an optional section that contains the list of species to be used in the calculation of the electron energy distribution function in the BOLSIG+ solver; all the species must be present in section **SPECIES**. This will tell the solver to call BOLSIG+ and to use the cross sections for the listed species in the solution of the Boltzmann equation. The max allowed number of species in this section is 32.

Available options in the section are:

- **set electron-electron collisions *NE/NGAS***, where NE/NGAS electron-to-gas fraction beyond which electron-electron collisions are taken into account in the Boltzmann solver (the default value is $10^{-5}$ and the maximum value is 1.0). Note,

this value can be changed during code execution using ZDPlasKin_set_config routine;

- **set min field *EMIN***, **set max field *EMAX***, **set max points *IMAX***, and **set accuracy *RTOL*** to manage the caching mechanism of BOLSIG+ solver (default values are $10^{-1}$ Td, $10^{3}$ Td, $10^{3}$, and $10^{-3}$, respectively);
- **set dbfile *FILENAME*** to select the database filename containing the cross sections (default is **bolsigdb.dat**). ***FILENAME*** must be in capital letters for UNIX systems;
- one or multiple **DENSITY *<A>* += *<B>*** which adds all set of electron collisional processes of species ***<A>*** to the set of processes for species ***<B>***. Typically, it can be used for avoiding electron cross-section database doubling for similar species, for example, the ground state and vibrational states of a molecule.

We implemented a caching mechanism for the results of BOLSIG+ solver in order to increase computational affectivity of the package. By default, an exponential field mesh is generated: $EN(i) = EMIN \cdot \exp(A \cdot i)$, $i = 0 \ldots IMAX$, where $A = \ln(1 + RTOL)$ constant is chosen to satisfy relation $EN(i+1)/EN(i) = 1 + RTOL$. In this case, the maximum number of points is equal to $IMAX = A^{-1} \cdot \ln(EMAX / EMIN)$. BOLSIG+ solver is used to compute required rates only once for every $EN(i)$ point if other parameters (gas temperature, fractional composition, etc) have not changed too much (relative changes are less than *RTOL*). For a filed below *EMIN* the code computes rates for this minimum field and extrapolates the results linearly. This caching mechanism can be switched off by setting *IMAX* to 0.

Example:

```
BOLSIG
Ar  Ar*
set electron-electron collisions 1.0D-3
set max points 0
set dbfile MYDB.DAT
END
```

In some cases, the set of species in **BOLSIG** section is reduced with respect to the full species set in section **SPECIES**. It means that some of species are not used when BOLSIG+ solver defines electron transport parameters and reaction rates. A warning is issued as soon as the reduced density (normalized by the total gas density) of such "missing" species exceeds some critical level (1E-3 by default). This can affect the accuracy of calculations.

## Cross section data

Starting from version 1.2b we do not more distribute electron cross sections. Please use the following site for downloading: http://www.lxcat.laplace.univ-tlse.fr. All download from this site data is compatible with ZDPlasKin package. Before upgrading cross-section data or making any changes of this data file please consult the page HOW TO USE.

Users are encouraged to reference the original data in publications. Users are also strongly encouraged to check the data carefully by comparing with the original references. Please let us know if you see any discrepancies.

## Section REACTIONS

The main section contains a list of reactions in format **$R_1$ + $R_2$ + ... => $P_1$ + $P_2$ + ...** followed by several spaces, a sign **!** and a reaction rate with a length of up to 130 symbols. There are no limits for the number of reactants or products per reaction; neither is there a limit for the total number of reactions in the input file.

Reaction and product terms **R** and **P** can be one from

- the species defined in section **SPECIES**; a short version is also possible, for example, 2 N2 to replace N2 + N2;
- **xxx_K** or **xxx_eV** record with **xxx** equal to a non-negative value of energy released (in right, products, side) or energy absorbed (in left, reactants, side) for endothermic and exothermic reactions, respectively;
- **ANY_SPECIES**, **ANY_NEUTRAL**, **ANY_ION_POSITIVE**, or **ANY_ION_NEGATIVE** to indicate any, neutral, positive or negative (except electrons) species, respectively.

The reaction rate following the **!** symbol can any of the following:

- an expression in Fortran 90 [$s^{-1}$, $cm^3\ s^{-1}$, $cm^6\ s^{-1}$, ...];
- **BOLSIG+ A -> B**, to indicate that corresponding constant rates must be taken from BOLSIG+ solver. An expression in Fortran can be used just before the BOLSIG+ record.

Example:

```
REACTIONS
E + Ar => 2E + Ar^+   !  1.0d-7
E + Ar => 2E + Ar^+   !          BOLSIG+ Ar -> Ar^+
E + Ar => 2E + Ar^+   !  0.1 * BOLSIG+ Ar -> Ar^+
END
```

It is possible to use brief form for similar reactions using group syntax. A group **@G**, wherein G is any alphabetical symbol, will be enrolled according to supplied group list. Group list has to present in the following line in form **@G = A B C ...**, i.e. a list of substitutions delimited with one or many spaces. Multiple groups can be used simultaneously under conditions of equal number of elements in every group.

Example:

```
REACTIONS
E + Ar^+ + @M  =>  Ar + @M   !  1.0d-7 * @R
  @M = Ar        Ar*        Ar**
  @R  = 1.0d0    1.0d-1     1.0d-2
END
```

The lines starting with **$** will be copied to the output Fortran module "as is"; the user can put here any Fortran structures, define variables and functions of external parameters

- **Time**, the time moment [s]
- **Tgas**, the gas temperature [K]
- **EN**, the reduced electric field [Td]
- **OmegaN**, the reduced angular frequency [cm3 $s^{-1}$]
- **Te**, the electron temperature [K]
- **Vdr**, the electron drift velocity [cm $s^{-1}$]
- **De**, the electron diffusion coefficient [$cm^2$ $s^{-1}$]
- **ANY_SPECIES**, **ANY_NEUTRAL**, **ANY_ION_POSITIVE**, or **ANY_ION_NEGATIVE** [$cm^{-3}$] to indicate any, neutral, positive or negative (except electrons) species, respectively.

Variables **Te**, **Vdr** and **De** are not even defined until **BOLSIG** section is configured in input file.

Example:

```
REACTIONS
$ double precision :: my_rate
$ my_rate = 1.0D-10 * Tgas/300.d0 * Te
E + Ar -> 2E + Ar^+   !   my_rate
END
```

**N.B.:** We note here that ZDPlasKin package operates with direct reaction processes only and does not construct a list of inverse processes at the moment.

---

## PREPROCESSOR

The **PREPROCESSOR** program (included in ZDPlasKin package) translates the input data file into a Fortran code. The semantic analysis used by the **PREPROCESSOR** checks that each element, species and reaction is unique, and that the charge and elementary composition is balanced in every reaction. Error messages are returned to the user if these conditions are not respected in the input data.

---

## ZDPlasKin MODULE

ZDPlasKin module is generated by **PREPROCESSOR** utility and then can be used in a master code. It gives a set of subroutines that will be used for plasma-chemical simulations.

Basically, the minimum user code must look like following:

```
program test
use ZDPlasKin                       !   to use module ZDPlasKin
call ZDPlasKin_init()               !   initialization of the module
call ZDPlasKin_set_conditions()     !   set conditions
call ZDPlasKin_set_density()        !   set initial densities
call ZDPlasKin_timestep()           !   main call - time integration
end program test
```

## A brief review of main routines

A call to the subroutine **init()** initializes the module and must be made before calling any of the other ZDPlasKin functions.

The main data structures are accessible from the master code, either directly as public variables and/or through function calls. For example, species densities are stored in the **density(i)** array, and the user can use this array in the master code to directly set (or get) values. Alternatively, the user can set (or get) values of the species densities by calling the **set(get)_density("X",dens)** subroutine. ZDPlasKin allows the user to work with species names **"X"** rather than a corresponding array index. If needed, the array index **i** of species **X** is available from the **species_name(:)** array or by calling the **get_species_index( "X",i)** subroutine.

The parameters specifying the conditions of calculations can be set or accessed (get) with **set(get)_conditions()** routines. Required parameters are the gas temperature and the reduced electric field; optional parameters include the specific heat gas ratio. A logical switch is used to signal that the evolution of the gas temperature should be included in the calculation.

The principal subroutine supplied for the time integration of the rate equations is **timestep()**, which calls DVODE_F90, an implicit and usually very efficient and accurate integration routine. VODE is a package of subroutines for the numerical solution of the initial-value problem for systems of first-order ordinary differential equations. The package uses the fixed-leading-coefficient Backward Differentiation Formula (BDF) method for stiff systems. An alternate time integration routine is the **timestep_explicit()** subroutine which is an explicit, first-order accurate, Euler solver. The latter is recommended only for very specific applications (such as an integration with a small fixed timestep).

Two further routines **get_density_total()** and **get_rates()** provide output useful for analysis: the total density of all species - neutral and charged particle species - total charge density, species source terms, reaction rates and a matrix of source terms, respectively.

Subroutine **write_file()** writes an output list of species and reactions as well as the matrix of reaction source terms to the corresponding files.

The only configuration subroutine in the module is **set_config()**. It allows setting the absolute and relative errors for the DVODE solver, or it can be used to manage the silent operating mode (no screen output) and the regime of statistic acquisition.

## The list of defined public variables

| | | |
|---|---|---|
| integer, parameter :: | species_max | ! number of species |
| integer, parameter :: | reactions_max | ! number of reactions |
| integer, parameter :: | species_length | ! max length of species names |
| integer, parameter :: | reactions_length | ! max length of reaction lines |
| double precision :: | density(species_max) | ! table of densities, cm-3 |
| integer :: | species_charge(species_max) | ! species charge = +1 / 0 / -1 |
| character(species_length) :: | species_name(species_max) | ! table of species names |
| character(reactions_length) :: | reaction_sign(reactions_max) | ! table of reaction lines |
| logical :: | lreaction_block(reactions_max) | ! selective reaction blocking |
| integer, parameter :: | species_electrons | ! index of electrons (if defined) |
| character(32), allocatable :: | qtplaskin_user_names(:) | ! user-defined variables for QtPlaskin GUI output |
| double precision, allocatable :: | qtplaskin_user_data(:) | |

## The list of public subroutines

### subroutine ZDPlasKin_init()

Initialization, must be called before any use of ZDPlasKin functions.

### subroutine ZDPlasKin_set_density(str,DENS, LDENS_CONST)

Sets density **DENS** for species **str**, optional flag **LDENS_CONST** fixes the density at a given level (no time integration of this species; default value is **FALSE**). This subroutine can be called at any time, not only for the initialization.

| | | |
|---|---|---|
| *character(*),intent(in)* | *:: str* | *! species* |
| *double precision, optional, intent(in)* | *:: DENS* | *! cm$^{-3}$* |
| *logical, optional, intent(in)* | *:: LDENS_CONST* | *! true/false* |

### subroutine ZDPlasKin_get_density(str,DENS, LDENS_CONST)

Gets density **DENS** for species **str**, optional flag **LDENS_CONST**gets information about the fixed density.

| | | |
|---|---|---|
| *character(*), intent(in)* | *:: str* | *! species* |
| *double precision, optional, intent(out)* | *:: DENS* | *! cm$^{-3}$* |
| *logical, optional, intent(out)* | *:: LDENS_CONST* | *! true/false* |

### subroutine ZDPlasKin_get_species_index(str,i)

Gets index of species **str** for direct use of the density( **i** = 1 : species_max ) array.

| | | |
|---|---|---|
| *character(*), intent(in)* | *:: str* | *! species* |
| *integer, intent(out)* | *:: i* | *! array index* |

### subroutine ZDPlasKin_set_conditions()[1]

Sets gas temperature, reduced electric field, reduced angular frequency, electron temperature, specific heat gas ratio (default value is **1.4**) and gas heating mode (**FALSE**). Optional logical key **SOFT_RESET** makes soft reset of DVODE solver keeping all user data and configurations unchanged. This can be useful if a fast change of input parameters is occurred (much faster than the typical time of density changes). It is recommended not do this regularly since it can have a significant impact on the efficiency of the package. Using **ELEC_TEMPERATURE** assumes Maxwellian energy distribution for electrons; the value of reduced field is calculated in this case to satisfy electron energy balance.

```
double precision, optional, intent(in)   :: GAS_TEMPERATURE      ! K
                                         :: REDUCED_FIELD        ! Td
                                         :: REDUCED_FREQUENCY    ! cm^3 s^-1
                                         :: ELEC_TEMPERATURE     ! K
                                         :: SPEC_HEAT_RATIO      ! 1
logical,  optional, intent(in)           :: GAS_HEATING          ! true/false
                                         :: SOFT_RESET           ! true/false
```

### subroutine ZDPlasKin_get_conditions()

Gets gas temperature **GAS_TEMPERATURE**, reduced electric field **REDUCED_FIELD**, reduced angular frequency **REDUCED_FREQUENCY**, electron temperature **ELEC_TEMPERATURE**, drift velocity **ELEC_DRIFT_VELOCITY** and diffusion **ELEC_DIFF_COEFF** coefficients, electron reduced collisional frequency **ELEC_FREQUENCY_N**, electron reduced total **ELEC_POWER_N**, elastic **ELEC_POWER_ELASTIC_N** and inelastic **ELEC_POWER_INELASTIC_N** powers. The electron temperature is defined as 2/3 of average energy determined from Boltzmann calculation. Electron energy distribution function can be returned in table form **ELEC_EEDF**(1:2,1:jmax) where ELEC_EEDF(1,:) and ELEC_EEDF(2,:) return the energies and the values of the EEDF, respectively; jmax = 128 is recommended. Variables **ELEC_*** are only available if **BOLSIG** section presents in input file.

```
double precision, optional, intent(out)  :: GAS_TEMPERATURE      ! K
                                         :: REDUCED_FIELD        ! Td
                                         :: REDUCED_FREQUENCY    ! cm^3 s^-1
                                         :: ELEC_TEMPERATURE     ! K
                                         :: ELEC_DRIFT_VELOCITY  ! cm s^-1
                                         :: ELEC_DIFF_COEFF      ! cm^2 s^-1
                                         :: ELEC_FREQUENCY_N     ! cm^3 s^-1
                                         :: ELEC_POWER_N         ! eV cm^3 s^-1
                                         :: ELEC_POWER_ELASTIC_N ! eV cm^3 s^-1
                                         :: ELEC_POWER_INELASTIC_N ! eV cm^3 s^-1
                                         :: ELEC_EEDF(:,:)       ! eV - eV-3/2
```

### subroutine ZDPlasKin_timestep(time,dtime)

Central subroutine. Makes time integration from **time** to **time + dtime** using the DVODE90 solver. The densities and conditions must be set before using this subroutine. Here it is advised not to put time steps too big; $10^{-6}$ s is often a good maximum value. We also note that any changes in the species densities in user's

---

[1] *Here and below, a subroutine with optional in/out variables can be used as follows, for example: call ZDPlasKin_set_conditions(GAS_TEMPERATURE = my_temperature).*

master code cause additional memory re-allocation procedures in DVODE90 solver; in this case the present subroutine can be less efficient than explicit Euler method (see below).

*double precision, intent(in)* *:: time* *! s*
*double precision, intent(inout)* *:: dtime* *! s*

### subroutine ZDPlasKin_timestep_explicit(time,dtime,error_max,density_min)

Alternative routine; makes time integration from **time** to **time + dtime** using explicit Euler method. Variables **error_max** and **density_min** - maximum relative density change and the minimum absolute density - control the number of internal steps keeping Δ[N] / ( [N] + **density_min** ) < **error_max** for any internal timestep integration. Optional number of internal steps is used to call ZDPlasKin_timestep() routine instead if the required number of timesteps exceeds **SWITCH_IMPLICIT**.

*double precision, intent(in)* *:: time* *! s*
*double precision, intent(inout)* *:: dtime* *! s*
*double precision, intent(in)* *:: error_max* *! 1*
*:: density_min* *! $cm^{-3}$*
*double precision, optional, intent(in)* *:: SWITCH_IMPLICIT* *! number of steps*

### subroutine ZDPlasKin_get_density_total()

Gets total density of all, neutral species, positive and negative ions and total charge density (in elementary charges).

*double precision, optional, intent(out)* *:: ALL_SPECIES* *! $cm^{-3}$*
*:: ALL_NEUTRAL* *! $cm^{-3}$*
*:: ALL_ION_POSITIVE* *! $cm^{-3}$*
*:: ALL_ION_NEGATIVE* *! $cm^{-3}$*
*:: ALL_CHARGE* *! $cm^{-3}$*

### subroutine ZDPlasKin_get_rates()

Get species source terms $\mathbf{Q_i}$ defined in equation (4), reaction rates $\mathbf{R_j}$ (3) and matrix of reactions source terms (should be used after the main subroutine ZDPlasKin_timestep() to update values). Variables **MEAN_*** mean time averaging starting from the last call ZDPlasKin_set_config( **STAT_ACCUM** = .true. ); they cannot be used without this call.

*double precision, optional, intent(out)* *:: SOURCE_TERMS(species_max)* *! $cm^{-3}\,s^{-1}$*
*:: REACTION_RATES(reactions_max)* *! $cm^{-3}\,s^{-1}$*
*:: SOURCE_TERMS_MATRIX(species_max, reactions_max)* *! $cm^{-3}\,s^{-1}$*
*:: MEAN_DENSITY(species_max)* *! $cm^{-3}$*
*:: MEAN_SOURCE_TERMS(species_max)* *! $cm^{-3}\,s^{-1}$*
*:: MEAN_REACTION_RATES(reactions_max)* *! $cm^{-3}\,s^{-1}$*
*:: MEAN_SOURCE_TERMS_MATRIX(species_max, reactions_max)* *! $cm^{-3}\,s^{-1}$*

### subroutine ZDPlasKin_set_config()

Sets internal configuration with **ATOL** absolute and **RTOL** relative errors, the **SILENCE_MODE** flag for silent operating mode (reduced screen output), the regime of statistic acquisition **STAT_ACCUM** (see ZDPlasKin_write_file and ZDPlasKin_get_rates). **QTPLASKIN_SAVE** allows auto saving of results in QtPlaskin compatible format. Every operation with statistic acquisition in this routine makes a reset of accumulated results. The electron-to-gas fraction beyond which electron-electron collisions are taken into account in the

Boltzmann solver can be changed **BOLSIG_EE_FRAC**. Logical flag **BOLSIG_IGNORE_GAS_TEMPERATURE** allows BOLSIG+ solver to do not recalculate the electron rates when the gas temperature changes.

```
double precision, optional, intent(in)   :: ATOL                          ! cm^-3
                                         :: RTOL                          ! 1
                                         :: BOLSIG_EE_FRAC                ! 1
logical, optional, intent(in)            :: SILENCE_MODE
                                         :: STAT_ACCUM
                                         :: QTPLASKIN_SAVE
                                         :: BOLSIG_IGNORE_GAS_TEMPERATURE
```

### subroutine ZDPlasKin_write_file()

Saves species and reactions lists in the files **FILE_SPECIES** and **FILE_REACTIONS**, respectively, and the matrix of reaction source terms **FILE_SOURCE_MATRIX** (the matrix can be written only if the **STAT_ACCUM** flag is activated in the previous subroutine. This flag can be activated at any time, and then calling this subroutine to write the matrix file will write the average source terms for each species and for each reaction). This routine uses Fortran flow with unit = 5 by default that can be changed by optional **FILE_UNIT** variable.

```
character(*), optional, intent(in)   :: FILE_SPECIES
character(*), optional, intent(in)   :: FILE_REACTIONS
character(*), optional, intent(in)   :: FILE_SOURCE_MATRIX
integer, optional, intent(in)        :: FILE_UNIT
```

### subroutine ZDPlasKin_write_qtplaskin(time, LFORCE_WRITE)

Saves pre-defined and user-specific data in qt_*.txt files in a format compatible with QtPlaskin graphical interface. The route writes output data if some level of relative change of densities or other parameters from the previous output is reached ($10^{-2}$ by default). Optional **LFORCE_WRITE** logical variable allows controlling output frequency.

```
double precision, intent(in)         :: FILE_SPECIES
logical, optional, intent(in)        :: LFORCE_WRITE
```

### subroutine ZDPlasKin_reset()

Resets all data and configuration except BOLSIG+ loaded data, used as a part of initialization subroutine. This has to be done if many runs are programmed in the same code (for example, if you want to make one run with a given pressure, then another one with a different pressure, you have to reset ZDPlasKin).

## The list of internal subroutines

Normally, the user would not use any of the following routines.

**subroutine ZDPlasKin_bolsig_rates()**

**subroutine ZDPlasKin_stop()**

**subroutine ZDPlasKin_fex()**

**subroutine ZDPlasKin_jex()**

**subroutine ZDPlasKin_reac_rates()**

**subroutine ZDPlasKin_reac_source_matrix()**

---

## DATA SAVE AND VISUALIZATION OF RESULTS

In principal, users have to organize an output of data using standard FORTRAN capabilities. We encourage, however, saving data in QtPlaskin-compatible format.

An adaptive auto save is triggered on calling just after initialization

```
call ZDPlaskin_set_config(QTPLASKIN_SAVE=.true.)   !   auto save in QtPlaskin format
```

In this case, ZDPlasKin saves all necessary data in qt_*.txt files after every timestep. Manual data save in QtPlaskin format is possible using

```
call ZDPlaskin_write_qtplaskin(time)   !   manual save in QtPlaskin format
```

The list of saved data includes temporal dependences of key parameters (reduced electric field, gas and electron temperatures, electron current density and electric power dissipation per unit volume), density of species, reaction source rates and reaction-specific production rates of species for sensitivity analysis.

User-specific data can be optionally saved using **qtplaskin_user_names(:)** and **qtplaskin_user_data(:)** arrays. These predefined arrays have to be allocated and updated before making timestep integration. For example:

```
call ZDPlasKin_set_config(QTPLASKIN_SAVE=.true.)
allocate(qtplaskin_user_data(2), qtplaskin_user_names(2))
qtplaskin_user_names(1) = 'my parameter'
qtplaskin_user_names(2) = 'one more my parameter'
do while( time < time_end )
   qtplaskin_user_data(1) = ...
   qtplaskin_user_data(2) = ...
   call ZDPlasKin_timestep(time,dime)
enddo
```

Generated in this way output can then be analyzed using QtPlaskin graphical interface developed by the TRAPPA group at the Instituto de Astrofísica de Andalucía, CSIC in Granada, Spain. Please use the following web site for downloading source or complete applications for Windows and Mac OS: http://www.trappa.es/content/software.

---

# COMPILATION

In order to satisfy module dependences it is highly recommended to compile source files in the following order: *dvode_f90_m.F90 - zdplaskin_m.F90 - user_code.F90*, and to use the BOLSIG+ binary library during the linking stage. This advises applied to both command line and project-based compilations. Although this version of ZDPLASKIN has been tested with several freeware and commercial F90 and F95 compilers, the developers make no guarantees as to accuracy. Please check section FAQs before contacting the authors.

**IMPORTANT**: The **PREPROCESSOR** translates the input data file into a Fortran code, and thus any changes in the input data file require that the user re-run the **PREPROCESSOR** and re-compile the generated Fortran code.

**Note to Windows users**: Linking to dynamic library bolsig.dll is usually handled by linking to an import library *bolsig.lib*. Three different pairs of *dll/lib* BOLSIG+ libraries are supplied to the package: *bolsig.dll/lib* file contains all caps cdecl functions (ZDPLASKIN_BOLSIG_XXX) which is native format for Intel Fotran compiler, bolsig_g.dll/lib contains lowercase cdecl (zdplaskin_bolsig_xxx_) and recommended to use with Lahey/Fujitsu Fortran and gFortran, bolsig_s contains all caps stdcall (_ZDPLASKIN_BOLSIG_XXX@8) and can be used with Compaq Fortran.

### Intel Fortran compiler (checked for version 10.x)

…> ifort [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig.lib (windows)
…> ifort [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig.dylib (mac os)
…> ifort [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig.so (linux)

### gFortran (checked for version 4.x)

…> gfortran [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig_g.dll (windows)
…> gfortran [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig.dylib (mac os)
…> gfortran [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig.so (linux)

### Lahey/Fujitsu Fortran 95 (checked for version 5.x)

…> lf95 [options] dvode_f90_m.F90 zdplaskin_m.F90 user_code.F90 bolsig_g.lib (windows)

### Do not forget to set library search path

…> export DYLD_LIBRARY_PATH=.:$DYLD_LIBRARY_PATH (mac os)
…> export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH (linux)

---

# FREQUENTLY ASKED QUESTIONS

### How should I get started using ZDPlasKin?

1) Try first using the simple 2-reaction test case. Execute the preprocessor with the kinet.inp input data file corresponding to the 2-reaction test case. Then compile the generated Fortran code with the master code we provide for the test case.

Execute the code and make sure that your numerical results are identical to those in our test case.

2) Do the same test with one of the more complicated examples provided at ZDPlasKin web site.
3) Then start adapting the master codes provided with these examples until you are familiar with the various subroutines and options available.
4) Put together your own input data file and you should be ready to use ZDPlasKin!

### What upgrades are planned for ZDPlasKin?

ZDPlasKin is not a commercial project and it does not have any financial support at the moment, so its future depends on feedback we will have from you. Please send us your comments.

---

## COPYRIGHT STATEMENT