
SD1D Documentation

Release 0.1.0

The BOUT++ team

Jun 09, 2023

CONTENTS

1	Getting started	3
1.1	Examples	4
1.2	Non-uniform mesh	5
2	Outputs	7
3	Plasma model	9
3.1	Heat conduction	9
4	Boundary conditions	11
4.1	Upstream: Symmetry	11
4.2	Downstream: Sheath	11
5	Neutral model	13
5.1	Diffusive model	13
5.2	Neutral fluid model	14
6	Sources and transfer terms	15
6.1	Transfer channels	15
6.2	Recycling	16
7	Atomic cross sections	19
8	Numerical methods	21
8.1	Advection terms $\nabla \cdot (\mathbf{b}V_{ }f)$	21
8.2	Artificial viscosity	22

This is a model of a 1D fluid, assuming equal ion and electron temperatures, no electric fields or currents.

GETTING STARTED

First get a copy of development branch of BOUT++. You can download a tarball from <https://github.com/boutproject/BOUT-dev>, but it is strongly recommended you use Git:

```
$ git clone https://github.com/boutproject/BOUT-dev.git
```

Configure and make BOUT-dev, including SUNDIALS. This is available from <http://computation.llnl.gov/projects/sundials>, and is needed for preconditioning to work correctly.

```
$ cd BOUT-dev
$ ./configure --with-sundials
$ make
```

The user manual for BOUT++ is in subdirectory of BOUT-dev called “manual”, and contains more detailed instructions on configuring and compiling BOUT++. This will build the core library code, which is then used in each model or test case (see the examples/ subdirectory)

Next download a copy of SD1D into the BOUT-dev/examples subdirectory. This isn’t strictly necessary, but it makes the “make” command simpler (otherwise you add an argument BOUT_TOP=/path/to/BOUT-dev/ to make)

```
BOUT-dev/examples/$ git clone https://github.com/boutproject/SD1D.git
BOUT-dev/examples/$ cd SD1D
BOUT-dev/examples/SD1D $ make
```

Hopefully you should see something like:

```
Compiling sd1d.cxx
Compiling div_ops.cxx
Compiling loadmetric.cxx
Compiling radiation.cxx
Linking sd1d
```

Here the main code is in “sd1d.cxx” which defines a class with two methods: `init()`, which is run once at the start of the simulation to initialise everything, and `rhs()` which is called every timestep. The function of `rhs()` is to calculate the time derivative of each evolving variable: In the `init()` function the evolving variables are added to the time integration solver (around line 192). This time integration sets the variables to a value, and then runs `rhs()`. Starting line 782 of `sd1d.cxx` you’ll see the density equation, calculating `ddt(Ne)`. `Ne` is the evolving variable, and `ddt()` is a function which returns a reference to a variable which holds the time-derivative of the given field.

BOUT++ contains many differential operators (see `BOUT-dev/include/difops.hxx`), but work has been done on improving the flux conserving Finite Volume implementations, and they’re not yet in the public repository. These are defined in `div_ops.hxx` and `div_ops.cxx`.

The atomic rates are used in `sd1d.cxx` starting around line 641, and are defined in `radiation.cxx` and `radiation.hxx`.

To run a simulation, enter:

```
$ ./sd1d -d case-01
```

This will use the “case-01” subdirectory for input and output. All the options for the simulation are in case-01/BOUT.inp.

The output should be a whole bunch of diagnostics, printing all options used (which also goes into log file BOUT.log.0), followed by the timing for each output timestep:

Sim Time	RHS evals	Wall Time	Calc	Inv	Comm	I/O	SOLVER
0.000e+00	1	1.97e-02	1.9	0.0	0.2	21.6	76.3
5.000e+03	525	1.91e-01	89.0	0.0	0.6	1.1	9.3
1.000e+04	358	1.30e-01	88.8	0.0	0.6	1.4	9.2
1.500e+04	463	1.68e-01	89.2	0.0	0.6	1.3	8.9
2.000e+04	561	2.02e-01	89.6	0.0	0.6	1.1	8.7
2.500e+04	455	1.65e-01	89.2	0.0	0.6	1.2	9.1

The simulation time (first column) is normalised to the ion cyclotron frequency (as SD1D started life as part of a turbulence model), which is stored in the output as “Omega_ci”. So each output step is $5000 / \text{Omega_ci} = 104.4$ microseconds. The number of internal timesteps is determined by the solver, and determines the number of times the `rhs()` function was called, which is given in the second column. If this number starts steadily increasing, it’s often a sign of numerical problems.

To analyse the simulation, the data is stored in the “case-01” subdirectory along with the input. You can use IDL or Python to look at the “Ne”, “NVi”, “P” variables etc. which have the same names as in the `sd1d.cxx` code. See section 6 for details of the output variables and their normalisation. The evolving variables should each be 4D, but all dimensions are of size 1 except for the time and parallel index (200). Please see the BOUT++ user manual for details of setting up the Python and IDL reading (“collect”) routines.

1.1 Examples

1.1.1 Case 1: Without heat conduction (Euler’s equations)

Removing heat conduction reduces the system to fluid (Euler) equations in 1D. Note that in this case the boundary condition (equation [eq:sheath_speed]) is subsonic, because the adiabatic fluid sound speed is

$$c_s = \left(\frac{\gamma p}{n} \right)^{1/2} \quad \gamma = 5/3$$

In this case the sources of particles and energy are uniform across the grid.

1.1.2 Case 2: Localised source region

The same equations are solved, but here the sources are only in the first half of the domain, applied with a Heaviside function so the sources abruptly change.

1.1.3 Case 3: Heat conduction

We now add Spitzer heat conduction, the $\kappa_{||e}$ term in the pressure equation. This coefficient depends strongly on temperature, and severely limits the timestep unless preconditioning is used. Here we use the CVODE solver with preconditioning of the electron heat flux. In addition to improving the speed of convergence, this preconditioning also improves the numerical stability.

1.1.4 Case 4: Recycling, neutral gas

The plasma equations are now coupled to a similar set of equations for the neutral gas density, pressure, and parallel momentum. A fixed particle and power source is used here, and a 20% recycling fraction. Exchange of particles, momentum and energy between neutrals and plasma occurs through ionisation, recombination and charge exchange.

1.1.5 Case 5: High recycling, upstream density controller

This example uses a PI feedback controller to set the upstream density to $1 \times 10^{19} \text{m}^{-3}$. This adjusts the input particle source to achieve the desired density, so generally needs some tuning to minimise transient oscillations. This is controlled by the inputs

```
density_upstream = 1e19
density_controller_p = 1e-2
density_controller_i = 1e-3
```

The input power flux is fixed, specified in the input as 20MW/m²:

```
[P]      # Plasma pressure P = 2 * Ne * T
powerflux = 2e7 # Input power flux in W/m^2
```

The recycling is set to 95%

```
frecycle = 0.95
```

NOTE: This example is under-resolved; a realistic simulation would use a higher resolution, but would take longer. To increase resolution adjust ny:

```
ny = 200      # Resolution along field-line
```

Rather than 200, a more realistic value is about 600 or higher with a uniform mesh. An alternative is to compress grid cells closer to the target by varying the grid spacing dy.

1.2 Non-uniform mesh

An example of using a non-uniform grid is in `diffusion_pn`. The location l along the field line as a function of normalised cell index y , which goes from 0 at the upstream boundary to 2π at the target, is

$$l = L \left[(2 - \delta y_{min}) \frac{y}{2\pi} - (1 - \delta y_{min}) \left(\frac{y}{2\pi} \right)^2 \right]$$

where $0 < \delta y_{min} < 1$ is a parameter which sets the size of the smallest grid cell, as a fraction of the average grid cell size. The grid cell spacing δy therefore varies as

$$\delta y = \frac{L}{N_y} \left[1 + (1 - \delta y_{min}) \left(1 - \frac{y}{\pi} \right) \right]$$

This is set in the BOUT.inp settings file, under the mesh section:

```
dy = (length / ny) * (1 + (1-dymin)*(1-y/pi))
```

In order to specify the size of the source region, the normalised cell index y at which the location l is a given fraction of the domain length must be calculated. This is done by solving for y in equation [eq:nonuniform_l].

$$y_{xpt} = \pi \left[2 - \delta y_{min} - \sqrt{(2 - \delta y_{min})^2 - 4(1 - \delta y_{min}) f_{source}} \right] / (1 - \delta y_{min})$$

which is calculated in the BOUT.inp file as

```
y_xpt = pi * ( 2 - dymin - sqrt( (2-dymin)^2 - 4*(1-dymin)*source ) ) / (1 - dymin)
```

where `source` is the fraction f_{source} of the length over which the source is spread. This is then used to calculate sources, given a total flux. For density:

```
source = (flux/(mesh:source*mesh:length))*h(mesh:y_xpt - y)
```

which switches on the source for $y < y_{xpt}$ using a Heaviside function, then divides the flux by the length of the source region $f_{source}L$ to get the volumetric sources.

OUTPUTS

Output quantities are normalised, with the normalisation factors stored in the output files

Table 2.1: Normalisation quantities

Name	Description	Units
Nnorm	Density	m^{-3}
Tnorm	Temperature	eV
Cs0	Speed	m/s
Omega_ci	Time	1/s
rho_s0	Length	m

The following variables are stored in the output file if they are evolved:

Name	Description	Normalisation
Ne	Plasma density	Nnorm [m^{-3}]
NVi	Plasma flux	Nnorm \times Cs0 [$m^{-2}s^{-1}$]
P	Plasma pressure	$e \times$ Nnorm \times Tnorm
Nn	Neutral density	Nnorm [m^{-3}]
NVn	Neutral flux	Nnorm \times Cs0 [$m^{-2}s^{-1}$]
Pn	Neutral pressure	$e \times$ Nnorm \times Tnorm

The following rates and coefficients are also stored:

Note that the R term is energy which is lost from the system, whilst E is energy which is transferred between plasma and neutrals. For all transfer terms (S, F, R) a positive value means a transfer from plasma to neutrals.

To diagnose atomic processes, turn on `diagnose = true` in the input settings (this is the default). Additional outputs contain the contributions from each atomic process. They have the same normalisation factors as the corresponding (S, F, R) term.

Name	Description
Srec	Sink of plasma particles due to recombination
Siz	Sink of plasma particles due to ionisation (negative)
Frec	Sink of plasma momentum due to recombination
Fiz	Sink of plasma momentum due to ionisation
Fcx	Sink of plasma momentum due to charge exchange
Fel	Sink of plasma momentum due to elastic collisions
Rrec	Radiation loss due to recombination
Riz	Radiation loss due to ionisation (inc. excitation)
Rzrad	Radiation loss due to impurities
Rex	Radiation loss due to electron-neutral excitation
Erec	Sink of plasma energy due to recombination
Eiz	Sink of plasma energy due to ionisation
Ecx	Sink of plasma energy due to charge exchange
Eel	Sink of plasma energy due to elastic collisions

PLASMA MODEL

Equations for the plasma density n , pressure p and momentum $m_i n V_{||i}$ are evolved:

$$\begin{aligned}\frac{\partial n}{\partial t} &= -\nabla \cdot (\mathbf{b} V_{||} n) + S_n - S \\ \frac{\partial}{\partial t} \left(\frac{3}{2} p \right) &= -\nabla \cdot \mathbf{q} + V_{||} \partial_{||} p + S_p - E - R \\ \frac{\partial}{\partial t} (m_i n V_{||}) &= -\nabla \cdot (m_i n V_{||} \mathbf{b} V_{||}) - \partial_{||} p - F \\ j_{||} &= 0 \\ T_i &= T_e = \frac{1}{2} \frac{p}{en} \\ \mathbf{q} &= \frac{5}{2} p \mathbf{b} V_{||} - \kappa_{||e} \partial_{||} T_e\end{aligned}$$

Which has a conserved energy:

$$\int_V \left[\frac{1}{2} m_i n V_{||i}^2 + \frac{3}{2} p \right] dV$$

The heat conduction coefficient $\kappa_{||e}$ is a nonlinear function of temperature T_e :

$$\kappa_{||e} = \kappa_0 T_e^{5/2}$$

where κ_0 is a constant. See section 8 for details.

Operators are:

$$\partial_{||} f = \mathbf{b} \cdot \nabla f \quad \nabla_{||} f = \nabla \cdot (\mathbf{b} f)$$

3.1 Heat conduction

Spitzer heat conduction is used

$$\kappa_{||e} = 3.2 \frac{n e^2 T \tau_e}{m_e} \simeq 3.1 \times 10^4 \frac{T^{5/2}}{\ln \Lambda}$$

which has units of W/m/eV so that in the formula $q = -\kappa_{||e} \nabla T$, q has units of Watts per m² and T has units of eV. This uses the electron collision time:

$$\tau_e = \frac{6\sqrt{2}\pi^{3/2}\epsilon_0^2\sqrt{m_e}T_e^{3/2}}{\ln \Lambda e^{2.5}n} \simeq 3.44 \times 10^{11} \frac{T_e^{3/2}}{\ln \Lambda n}$$

in seconds, where T_e is in eV, and n is in m^{-3} .

Normalising by the quantities in table 1 gives

$$\hat{\kappa}_{||e} = 3.2 \hat{n} \hat{T}_e \frac{m_i}{m_e} \tau_e \Omega_{ci}$$

where hats indicate normalised (dimensionless) variables.

BOUNDARY CONDITIONS

4.1 Upstream: Symmetry

Symmetry boundary conditions are applied at the upstream side, corresponding to zero flow through the boundary.

$$\partial_{||}n = 0 \quad \partial_{||}p = 0 \quad \partial_{||}T_e = 0 \quad V_{||} = 0 \quad nV_{||} = 0$$

Since the boundary is half-way between grid points, this is implemented as

$$\begin{aligned} n_0 &= n_1 \\ p_0 &= p_1 \\ T_{e,0} &= T_{e,1} \\ V_{||,0} &= -V_{||,1} \\ nV_{||,0} &= -nV_{||,1} \end{aligned}$$

4.2 Downstream: Sheath

Boundary conditions are applied to the velocity and the heat flux:

- At the left boundary a no-flow condition is applied:

$$\begin{aligned} V_{||} &= 0 \\ \partial_{||}T_e &= 0 \end{aligned}$$

- At the right boundary is a sheath boundary:

$$\begin{aligned} V_{||} &\geq v_s \\ \partial_{||}T_e &= 0 \end{aligned}$$

where the inequality is implemented by switching from a Dirichlet to a Neumann boundary if $V_{||} > v_s$ in front of the boundary.

The critical speed into the sheath, v_s is sensitive to assumptions on the thermodynamics of the sheath, taking the form: [1]

$$v_s = \left(\frac{e(T_e + \gamma T_i)}{m_i} \right)^{1/2}$$

where T_e is the electron temperature (in eV), T_i is the ion temperature, γ is the ratio of specific heats. For isothermal flow $\gamma = 1$, for adiabatic flow with isotropic pressure $\gamma = 5/3$, and for one-dimensional adiabatic flow $\gamma = 3$. Here we are assuming $T_e = T_i$ and $\partial_{||}T_e$ so take the isothermal case. This therefore becomes:

$$v_s = \left(\frac{p}{n} \right)^{1/2}$$

Note: If the sheath velocity is subsonic, then waves can propagate in from the boundary. Their domain of dependence is outside the simulation domain, so these waves can cause numerical instabilities.

Several boundary conditions are available for the density and pressure, including free boundaries and Neumann (zero gradient). These are controlled by settings `density_sheath` and `pressure_sheath`. Density can have the following values:

0. Free boundary, linearly extrapolating the value from inside the domain

$$n_{-1} = 2n_{-2} - n_{-3}$$

1. Neumann (zero gradient)

$$n_{-1} = n_{-2}$$

2. Constant flux

$$n_{-1/2} = n_{-2}v_{-2}J_{-2} / (v_s J_{-1/2})$$

where the Jacobian factors J account for a changing flux tube cross-section area.

Pressure can have the following values:

0. Free boundary, linearly extrapolating the value from inside the domain

$$p_{-1} = 2p_{-2} - p_{-3}$$

1. Neumann (zero gradient)

$$p_{-1} = p_{-2}$$

2. Constant energy flux $\frac{5}{2}pv + \frac{1}{2}nv^3$

$$5p_{-1/2} = (5p_{-2}v_{-2} + n_{-2}v_{-2}^3) / v_s - n_{-1/2}v_s^2$$

NEUTRAL MODEL

The number of equations solved is controlled by the following parameters in the input file:

```
[NVn]
evolve = true # Evolve neutral momentum?

[Pn]
evolve = true # Evolve neutral pressure? Otherwise Tn = Te model
```

Neutral density is always evolved, so turning off evolution of momentum and pressure (setting both of the above to false) reduces the neutral model to a simple diffusion model (next section). By turning on the momentum equation

5.1 Diffusive model

In the simplest neutral model, neutral gas is modelled as a fluid with a density n_n which diffuses with a diffusion coefficient D_n :

$$\frac{\partial n_n}{\partial t} = \nabla \cdot (D_n \nabla n_n) + S - n_n / \tau_n$$

The temperature of the neutrals is assumed to be the same as the ions $T_n = T_i$. Diffusion of neutrals depends on the neutral gas temperature, and on the collision rate:

$$D_n = v_{th,n}^2 / (\nu_{cx} + \nu_{nn})$$

where $v_{th,n} = \sqrt{eT_n/m_i}$ is the thermal velocity of a neutral atom; $\nu_{cx} = n\sigma_{cx}$ is the charge-exchange frequency, and $\sigma_{nn} = v_{th,n}n_n a_0$ is the neutral-neutral collision frequency where $a_0 \simeq \pi (5.29 \times 10^{-11})^2 \text{ m}^2$ is the cross-sectional area of a neutral Hydrogen atom. In order to prevent divide-by-zero problems at low densities, which would cause D to become extremely large, the mean free path of the neutrals is limited to 1m.

An additional loss term is required in order to prevent the particle inventory of the simulations becoming unbounded in detached simulations, where recycling no longer removes particles from the system. This represents the residence time for neutral particles in the divertor region, which in [Togo 2013] was set to around 10^{-4}s .

5.2 Neutral fluid model

A more sophisticated neutrals model can be used, which evolves the neutral gas momentum and energy:

$$\begin{aligned}\frac{\partial n_n}{\partial t} &= -\nabla \cdot (\mathbf{b}V_{||n}n_n) + \nabla \cdot (D_n \nabla n_n) + S - n_n/\tau_n \\ \frac{\partial}{\partial t} \left(\frac{3}{2} p_n \right) &= -V_{||n} \partial_{||} p_n + \nabla \cdot (\kappa_n \nabla T_n) + \nabla \cdot (D_n T_n \nabla n_n) + E \\ \frac{\partial}{\partial t} (m_i n V_{||n}) &= -\nabla \cdot (m_i n V_{||n} \mathbf{b}V_{||n}) - \partial_{||} p + F\end{aligned}$$

where κ_n is the neutral gas heat conduction coefficient. This is assumed to be

$$\kappa_n = n_n v_{th,n}^2 / (\nu_{cx} + \nu_{nn})$$

i.e. similar to D_n for the diffusive neutral model, but with a factor of n_n .

Note that if the diffusion term D_n is retained in the neutral density (n_n) equation, then a corresponding term is needed in the pressure (p_n) equation. To remove these terms, set `dneut` to zero in the input options, which will set $D_n = 0$.

The density diffusion term should not be included if the momentum is evolved, and so is switched off if this is the case. The continuity equation for n_n is exact once the flow is known, so the diffusive flux should be contained in the flow velocity $V_{||n}$. To see where this comes from, assume an isothermal neutral gas:

$$\begin{aligned}\frac{\partial n_n}{\partial t} &= -\nabla \cdot (\mathbf{b}V_{||n}n_n) + S - n_n/\tau_n \\ \frac{\partial}{\partial t} (m_i n V_{||n}) &= -\nabla \cdot (m_i n V_{||n} \mathbf{b}V_{||n}) - eT_n \partial_{||} n_n + F\end{aligned}$$

Dropping the inertial terms reduces the momentum equation to

$$eT_n \partial_{||} n_n = F = \nu m_i n_n (V_{||i} - V_{||n})$$

where ν is a collision frequency of the neutrals with the ions, due to charge exchange, recombination and ionisation (i.e. $\nu_{cx} + \nu_{nn}$ as used in the calculation of diffusion coefficient D_n). This gives an equation for the neutral flow velocity:

$$V_{||n} = V_{||i} - \frac{eT_n}{m_i n_n \nu} \partial_{||} n_n = \frac{1}{n_n} \frac{v_{th,n}^2}{\nu} \partial_{||} n_n$$

where $v_{th} = \sqrt{eT_n/m_i}$ is the neutral thermal speed, as used in the calculation of D_n . This gives a flux of neutrals

$$n_n V_{||n} = n_n V_{||i} - D_n \partial_{||} n_n$$

Hence the diffusive flux is included in the balance between pressure gradients and friction in the momentum equation.

SOURCES AND TRANSFER TERMS

External sources are

- S_n = Source of plasma ions
- S_p = Source of pressure, related to energy source $S_E = \frac{3}{2}S_p$

In the simulations carried out so far, these source functions are both constant between midplane and X-point, and zero from X-point to target.

6.1 Transfer channels

There are several transfer channels and sinks for particles, energy and momentum due to rates of recombination, ionisation, charge exchange, electron-neutral excitation, and elastic collisions with units of $\text{m}^{-3}\text{s}^{-1}$:

$$\begin{aligned}
 \mathcal{R}_{rc} &= n^2 \langle \sigma v \rangle_{rc} & (\text{Recombination}) \\
 \mathcal{R}_{iz} &= n n_n \langle \sigma v \rangle_{iz} & (\text{Ionisation}) \\
 \mathcal{R}_{cx} &= n n_n \langle \sigma v \rangle_{cx} & (\text{Charge exchange}) \\
 \mathcal{R}_{el} &= n n_n \langle \sigma v \rangle_{el} & (\text{Elastic collisions})
 \end{aligned}$$

where n is the plasma density; n_n is the neutral gas density; σ_{cx} is the cross-section for charge exchange; σ_{rc} is the cross-section for recombination; and σ_{iz} is the cross-section for ionisation. Each of these processes' cross-section depends on the local density and temperatures, and so changes in time and space as the simulation evolves.

- S = Net recombination i.e neutral source (plasma particle sink). Calculated as Recombination - Ionisation:

$$S = \mathcal{R}_{rc} - \mathcal{R}_{iz}$$

- R = Cooling of the plasma due to radiation, and plasma heating due to 3-body recombination at temperatures less than 5.25eV.

$$\begin{aligned}
 R &= (1.09T_e - 13.6\text{eV}) \mathcal{R}_{rc} & (\text{Recombination}) \\
 &+ E_{iz} \mathcal{R}_{iz} & (\text{Ionisation}) \\
 &+ (1\text{eV}) \mathcal{R}_{ex} & (\text{Excitation}) \\
 &+ R_{z,imp} & (\text{Impurity radiation})
 \end{aligned}$$

The factor of 1.09 in the recombination term, together with factor of 3/2 in E below, is so that recombination becomes a net heat source for the plasma at $13.6/2.59 = 5.25\text{eV}$. E_{iz} is the average energy required to ionise an atom, including energy lost through excitation.

If excitation is not included (`excitation = false`) then following Togo *et al.*, E_{iz} is chosen to be 30eV. If excitation is included, then E_{iz} should be set to 13.6eV.

- E = Transfer of energy to neutrals.

$$\begin{aligned}
 E = & \quad \frac{3}{2} T_e \mathcal{R}_{rc} \quad (\text{Recombination}) \\
 - & \quad \frac{3}{2} T_n \mathcal{R}_{iz} \quad (\text{Ionisation}) \\
 + & \quad \frac{3}{2} (T_e - T_n) \mathcal{R}_{cx} \quad (\text{Charge exchange})^{**} \\
 + & \quad \frac{3}{2} (T_e - T_n) \mathcal{R}_{el} \quad (\text{Elastic collisions})^{**}
 \end{aligned}$$

(**) Note that if the neutral temperature is not evolved, then $T_n = T_e$ is used to calculate the diffusion coefficient D_n . In that case, T_n is set to zero here, otherwise it would cancel and leave no CX energy loss term.

- F = Friction, a loss of momentum from the ions, due to charge exchange and recombination. The momentum of the neutrals is not currently modelled, so instead any momentum lost from the ions is assumed to be transmitted to the walls of the machine.

$$\begin{aligned}
 F = & \quad m_i V_{||} \mathcal{R}_{rc} \quad (\text{Recombination}) \\
 - & \quad m_i V_{||n} \mathcal{R}_{iz} \quad (\text{Ionisation}) \\
 + & \quad m_i (V_{||} - V_{||n}) \mathcal{R}_{cx} \quad (\text{Charge exchange}) \\
 + & \quad m_i (V_{||} - V_{||n}) \mathcal{R}_{el} \quad (\text{Elastic collisions})
 \end{aligned}$$

All transfer channels are integrated over the cell volume using Simpson's rule:

$$S = \frac{1}{6J_C} (J_L S_L + 4J_C S_C + J_R S_R)$$

where J is the Jacobian of the coordinate system, corresponding to the cross-section area of the flux tube, and subscripts L , C and R refer to values at the left, centre and right of the cell respectively.

6.2 Recycling

The flux of ions (and neutrals) to the target plate is recycled and re-injected into the simulation. The fraction of the flux which is re-injected is controlled by `frecycle`:

```
frecycle = 0.95    # Recycling fraction
```

The remaining particle flux (5% in this case) is assumed to be lost from the system. Note that if there are any external particle sources, then this fraction must be less than 1, or the number of particles in the simulation will never reach steady state.

Of the flux which is recycled, a fraction `fredistribute` is redistributed along the length of the domain, whilst the remainder is recycled at the target plate

```
fredistribute = 0.8 # Fraction of recycled neutrals redistributed evenly along length
```

The weighting which determines how this is redistributed is set using `redist_weight`:

```
redist_weight = h(y - pi) # Weighting for redistribution
```

which is normalised in the code so that the integral is always 1. In these expressions y is uniform in cell index, going from 0 to 2π between the boundaries. The above example therefore redistributes the neutrals evenly (in cell index) from half-way along the domain to the end.

When neutrals are injected, some assumptions are needed about their energy and momentum

- When redistributed, neutrals are assumed to arrive with no net parallel momentum (so nothing is added to NV_n), and they are assumed to have the Franck-Condon energy (3.5eV currently)
- When recycled from the target plate, neutrals are assumed to have a parallel momentum away from the target, with a thermal speed corresponding to the Franck-Condon energy, and is also added to the pressure equation.
NOTE: This maybe should be one or the other, but not both...

ATOMIC CROSS SECTIONS

Cross sections are approximated with semi-analytic expressions, obtained from E.Havlickova but of unknown origin. For the purposes of calculating these cross-sections, any temperatures below 1eV are set to 1eV. The charge exchange cross-section is approximated as:

$$\sigma_{iz} = \begin{cases} 10^{-14} T^{1/3} & \text{if } T \geq 1\text{eV} \\ 10^{-14} & \text{if } T < 1\text{eV} \end{cases}$$

with units of $[\text{m}^3/\text{s}]$. Ionisation is calculated as

$$\sigma_{cx} = \begin{cases} 5.875 \times 10^{-12} \cdot T^{-0.5151} \cdot 10^{-2.563/\log_{10} T} & \text{if } T \geq 20\text{eV} \\ 10^{-6} \cdot T^{-3.054} \cdot 10^{-15.72 \exp(-\log_{10} T) + 1.603 \exp(-\log_{10}^2 T)} & \text{if } 1\text{eV} < T < 20\text{eV} \\ 7.638 \times 10^{-21} & \text{if } T \leq 1\text{eV} \end{cases}$$

Recombination rates are calculated using a 9×9 table of coefficients so is not reproduced here.

Plots of these cross-sections are shown in figure 1. There are a few anomalies with this: charge exchange always has the highest cross-section of any process, and ionisation has a jump at 20eV. The ionisation and charge exchange rates do not depend on density, but recombination does so a typical range of values is shown.

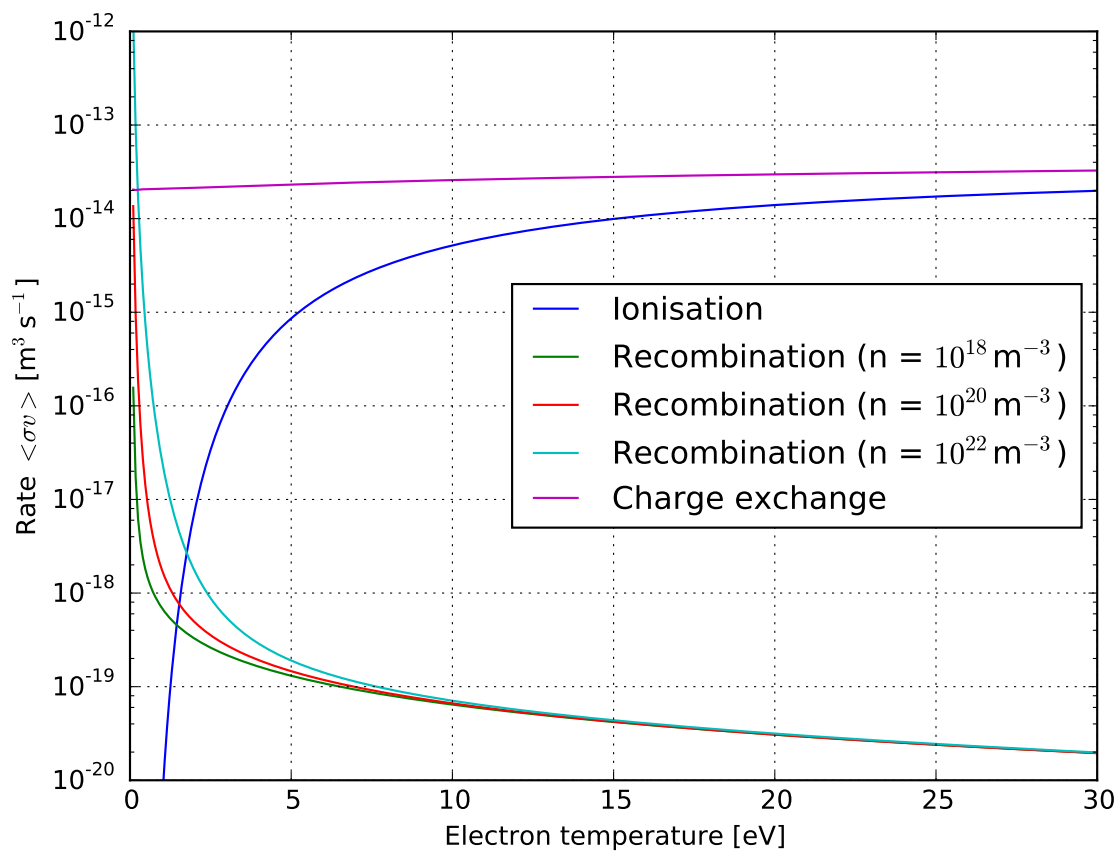


Fig. 7.1: Cross-sections [Thanks to E.Havlickova and H.Willett]

NUMERICAL METHODS

All variables are defined at the same location (collocated). Several different numerical methods are implemented, to allow testing of their accuracy and robustness.

8.1 Advection terms $\nabla \cdot (\mathbf{b}V_{\parallel}f)$

8.1.1 Flux splitting, MinMod limiter

The default method uses a combination of HLL-style flux splitting and MinMod slope limiting. Terms of the form $\nabla \cdot (\mathbf{b}f)$ are implemented as fluxes through cell boundaries:

$$\nabla \cdot (\mathbf{b}Vf)_i \simeq \frac{1}{J\Delta y} [F_{i+1/2} - F_{i-1/2}]$$

where F is the flux. This is calculated by linearly interpolating the velocity to the cell edges

$$V_{i+1/2} = \frac{1}{2} (V_i + V_{i+1})$$

The field being advected, f , is reconstructed from the cell centre values f_i onto cell edges f_i^L and f_i^R :

$$f_i^L = f_i - \frac{1}{2}s \quad f_i^R = f_i + \frac{1}{2}s$$

where the slope s is limited using the MinMod method:

$$s = \begin{cases} 0 & \text{if } \text{sign}(f_{i+1} - f_i) \neq \text{sign}(f_i - f_{i-1}) \\ f_{i+1} - f_i & \text{if } |f_{i+1} - f_i| < |f_i - f_{i-1}| \\ f_i - f_{i-1} & \text{otherwise} \end{cases}$$

In order to handle waves travelling both left and right, flux splitting handles characteristics moving left differently from characteristics moving right. In general this is problem dependent and computationally expensive, so here we adopt a simple approximation similar to an HLL splitting². We assume that the fastest waves in the system travel with speed a (the sound speed) with respect to the flow, so that there are waves travelling with $V + a$ and $V - a$. If the flow speed is supersonic then these waves are only in one direction, but for subsonic flows there is a flux in both directions. The fluxes between cells are calculated using:

$$F_{i+1/2} = \begin{cases} f_i^R V_{i+1/2} & \text{if } V_{i+1/2} > a \\ f_{i+1}^L V_{i+1/2} & \text{if } V_{i+1/2} < -a \\ f_i^R \frac{1}{2} (V_{i+1/2} + a) + f_{i+1}^L \frac{1}{2} (V_{i+1/2} - a) & \text{otherwise} \end{cases}$$

Hence for subsonic flows the flux becomes $V_{i+1/2} \frac{1}{2} (f_i^R + f_{i+1}^L) + \frac{a}{2} (f_i^R - f_{i+1}^L)$, where the second term is a diffusion. When the solution is smooth, $f_i^R \simeq f_{i+1}^L$, the numerical method becomes central differencing and the diffusion goes to zero as Δx^2 . Oscillatory solutions introduce dissipation, and the method becomes increasingly upwind as the flow becomes sonic.

² A. Harten, P. D. Lax, and B. van Leer, "On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws", SIAM Review, 25(1), pp. 35-61, 1983

8.1.2 Nonlinear fluxes

When advecting quantities which are a nonlinear combination of variables, such as $nV_{||}$, conservation properties can be slightly improved by using the following interpolation³⁴⁵:

$$(fg)^R = \frac{1}{2} (f^R g^C + f^C g^R)$$

where superscript C denotes cell centre, and R right hand side. This method is implemented, using MinMod interpolation for each variable.

8.1.3 Central differencing

Central difference schemes have an advantage over upwind schemes, in that they do not need to take account of wave speeds. The simple central differencing scheme produces large unphysical oscillations, due to the decoupling of odd and even points in collocated schemes, but can (usually) be stabilised by adding dissipation. It is implemented here for comparison with other schemes.

8.1.4 Skew symmetric central differencing

A simple modification to the central differencing scheme improves numerical stability, coupling nearby points⁶⁷ The idea is to split the divergence terms into a “skew-symmetric” form

$$\nabla \cdot (\mathbf{b}V_{||}f) = \frac{1}{2} [\nabla \cdot (\mathbf{b}V_{||}f) + V_{||}\mathbf{b} \cdot \nabla f + f\nabla \cdot (\mathbf{b}V_{||})]$$

Each of the terms on the right are then discretised with standard 2nd-order central differences. This method can avoid the need for additional dissipation, or be stabilised with a smaller viscosity than the simple central differencing method.

8.2 Artificial viscosity

Artificial viscosity (`viscos` input) is implemented as a diffusion of momentum in index space, so that the diffusion coefficient varies as Δy^2 .

$$\frac{\partial}{\partial t} (nV_{||})_i = \dots + \nu [(V_{i+1} - V_i) J_{i+1/2} - (V_i - V_{i-1}) J_{i-1/2}] / J_i$$

where J is the Jacobian, subscript i indicates cell index, and $J_{i+1/2} = (J_i + J_{i+1}) / 2$.

³ F.N.Felten, T.S.Lund “Kinetic energy conservation issues associated with the collocated mesh scheme for incompressible flow” J.Comp.Phys. 215 (2006) 465-484

⁴ F.N.Felten, T.S.Lund “Critical comparison of the collocated and staggered grid arrangements for incompressible turbulent flow” Report ADP013663

⁵ Y.Morinishi et al. “Fully Conservative Higher Order Finite Difference Schemes for Incompressible Flow” J.Comp.Phys. 143 (1998) 90-124

⁶ S.Pirozzoli “Stabilized non-dissipative approximations of Euler equations in generalized curvilinear coordinates” J.Comp.Phys. 230 (2011) 2997-3014

⁷ A.E.Honein, P.Moin “Higher entropy conservation and numerical stability of compressible turbulence simulations” J.Comp.Phys. 201 (2004) 532-545