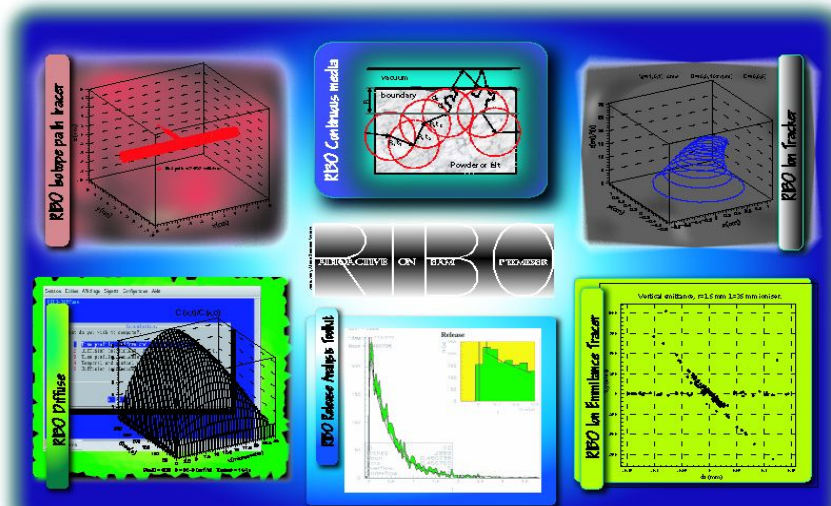


RADIOACTIVE ION BEAM OPTIMISER, RIBO. USER MANUAL.

Mario Santana Leitner



Contents

Index	ii
List of Figures	v
Preface	vi
1 Monte Carlo code. RIBO user manual.	1
1.1 RIBO, a MC code for isotope release optimization. Overview. . .	1
1.2 Setup.	3
1.2.1 Description of files.	3
1.2.2 Installation.	4
1.3 Input file.	5
1.3.1 <i>Surfaces</i> card.	6
1.3.2 <i>Cells</i> card.	10
1.3.3 Birth of particles, the <i>Source</i> card.	12
1.3.4 The <i>Tally</i> (end) card.	18
1.3.5 <i>Histogram</i> card.	21
1.4 Effusion with moving walls (valves).	21
1.5 Executing RIBO	23
1.5.1 The Isotope RElease Simulator, IRES	23
1.5.2 Express Execution with a batch file.	23
1.5.3 Runtime options.	24
1.6 Debugging running errors.	35
1.6.1 Introduction.	35
1.6.2 Typical error situations/messages	39
1.7 Making 3D model views.	40
1.8 User defined subroutines	41
1.8.1 User defined collisions, <i>customcollision.f</i>	41

1.8.2	User defined desorption in powder, <i>customcollision.f</i>	46
1.8.3	User defined source distributions, <i>customsource.f</i>	48
1.8.4	Setting up the electro-magnetic fields, <i>readEfield</i> and <i>Bn.f</i>	48
1.8.5	User defined output printing files <i>userprint.f</i>	52
1.8.6	Other user defined routines	54
1.9	Output file.	55
1.9.1	Fitting of events to a release function.	56
1.9.2	Computation of release fractions.	59
2	Three-dimensional diffusion module <i>Diffuse3D</i>.	61
2.1	Brief physical introduction	61
2.2	Instructions of use	63
2.2.1	[<i>data</i>]/ <i>Cond.dat</i> file	64
2.2.2	Activating 3-D diffusion calculations	65
2.3	User defined functions and subroutines	66
2.3.1	Customizing the starting distribution of Concentration or Temperature, <i>CSTART.f</i>	67
2.3.2	Defining source and / or sink terms, <i>gener.f</i>	68
2.3.3	User defined mass/heat diffusion coefficients, <i>Dijkt.f</i>	69
2.3.4	Customized heat radiation (and convection) function, <i>radia.f</i>	71
2.3.5	Customized the printout values, <i>userPRINT3D.f</i>	73
2.4	Examples of use	75
3	Examples of <i>RIBO</i> use.	76
3.1	Examples.	76
3.1.1	First example. Geometry issues.	76
3.1.2	Second example. Bigger files.	78
3.1.3	Third example, <i>test.inp</i>	79
3.1.4	Example of use of the 3D-Diffusion, 3D GRID: <i>heat3D.t.</i> . .	96
4	<i>Diffuse</i>. A diffusion emulator.	107
	Reference List	111

List of Figures

3.1	A simple example consisting on a bulb full of He at 300 K connected to a pipe of 1 cm diameter with a bend of 90°	76
3.2	Surfaces and cells for the first example. Auxiliary surfaces are needed to define the correct subspaces.	77
3.3	Sketch of a GANIL-SPIRAL type target [1, 2].	78
3.4	The target and ion source of example 3.	80
3.5	The spherical W target and the Ta screen.	97
4.1	Screen captures of the interactive DIFFUSE program.	110

Preface

Currently the nuclear chart includes around 3000 nuclides, distributed as β^+ , β^- and α -emitters, stable and spontaneously fissioning isotopes. A similar amount of unknown nuclei belongs to the so-called *terra incognita*, the uncertain region contained also within the proton, neutron and (fast) fission driplines and thereby stable against nucleon emission. The exploration of this zone is to be assisted by the use of radioactive ion beams (RIB) and could provide a new understanding of several nuclear properties. Moreover, besides pointing at crucial questions such as the validity of the shell model, the dilute matter and the halo structure, challenging experiments outside nuclear physics are also attended, e.g., explanations of the nucleosynthesis processes that may justify why the matter in the universe has evolved to present proportions of elements, and which represents a major challenge to nuclear physics.

These, together with other fascinating research lines in particle physics, solid state physics and medicine, demand utterly exotic and intense ion beams for which a global optimization of all relevant phenomena in beam formation has to be coherently conducted. As a response to this request, a Monte Carlo simulation code has been written, to integrate diffusion and effusion under various

pressure flows and conditions, including the transport through continuous media and enabling diffractive and surface dependent effects, emulating ionization in surface and plasma ion sources and, finally, reproducing the movement of ions under electro-magnetic fields.

Chapter 1

Monte Carlo code. RIBO user manual.

1.1 RIBO, a MC code for isotope release optimization. Overview.

The Radioactive (or Rare) Ion Beam Optimizer, RIBO, is a scientific Monte Carlo simulation program focused on the optimization of radioactive ion beam production. It tracks the paths of atoms through ISOL targets, from generation (not included, this step should be calculated with codes like MCNPX [3], MARS [4, 5], FLUKA [6, 7]...) to ionization and extraction. It includes the following models:

- Diffusion from slabs, fibers or powder.
- Diffusion in 3D structures, with custom options. **NEW FEATURE**
- Heat transfer in 3D multi-body set-ups, including conduction, radiation, cooling, heat deposition ... **NEW FEATURE**
- Effusion in the molecular flow and the intermediate regime. Cosine law.
- Effusion through porous media.

- Effusion under multiple path-passage conditions, and surface-crossing sense detection **NEW FEATURE**
- Absorption (condensation) to the walls. Definition of cold spots.
- Adsorption-desorption on the walls (temporary retention). Frenkel sampling **NEW FEATURE**
- Specular effusion.
- Custom collisions. **NEW FEATURE**
- Effusion through crystal systems (vibrations)¹.
- Effusion through systems with moving walls (valves).
- Surface ionization.
- Plasma ionization.
- Ion recombination.
- Ion transport in electric and magnetic fields. Emittance plots.
- Enhanced compilation, simulation speed increased up to 40 % **NEW FEATURE**

Other models are in consideration and could be implemented as a result of the common effort of a community of prospective users. Developments should be centralized through RIBO's web-page:

www.cern.ch/ribo

¹Under development.

Moreover, users are invited to customize their RIBO distribution by modifying the provided routines (in directory [*sources*]), explained in page 1.8:

- Source of Radioisotopes: *customsource.f*
- Customized collision law: *customcollision.f*
- Special desorption in powder or fiber: *powderdesorp.f*
- Magnetic field map: *Bn.f*
- Electric field map: *readEfield.f*
- User-defined on-line printing: *userprint.f*
- User-defined functions for 3D-diffusion: *custom3D.f*

In addition, a number of applications that help writing the input file and to analyze the output data² are being developed (and stored in [*tools*]).

1.2 Setup.

1.2.1 Description of files.

The RIBO code is distributed with a number of FORTRAN files, data libraries and text files. The present distribution spans over the following files:

- DIFFUSE PACKAGE For diffusion calculations only (fancy options).
 - . D.f, diffuse.e, diffuse.o, diffuse.sh, functC.f, license.pdf, README, timeN.f

²A set of PAW-based applications is under development.

- **MONTE CARLO** For typical (simple) diffusion and all effusion calculations.

data Cond.dat, init.dat, plion.dat, sion.dat, Urz.dat, valves.dat, workf.dat,
README

docs effusion.pdf (dvi, tex), MANUAL, ...

batchinputs Several runtime batch files.

targets Several input files.

objects diffuse3D.o, init.o, pIION.o, powder.o, space.o, explicitEM.o,
main.o, povray.o, readUrz.o, surfION.o

tools 3D-view.sh, convgeom.f, make.sh, trajectory.sh, rotate.f, translate.f

sources Bn.f, customcollision.f, customsource.f, emittance.f, math.f, read-
Efield.f, userprint.f, custom3D.f

. license, README, version

Moreover, for the graphical options (*Diffuse.sh*, *3D-RIBO.sh* and *trajectory.sh*) the two free programs are required:

1. Physics Analysis Workstation [8].
2. Persistence of Vision, Povray [9].

1.2.2 Installation.

Execute the make.sh script in [*tools*] if you need to recompile. Otherwise no installation is required.

Then you should enable execution permission on the executable file that has been created:

```
> chmod +x RIBO
```

Finally you should edit your bash profile to include the path of the executable file or you can run it from the installed directory simply typing:

```
> ./RIBO
```

Depending on the shell and configuration, typing 'bash' or 'sh' before 'RIBO' may also work.

Once installed, in order to use the program, an input file has to be created. The following instructions explain how to do this. Alternatively, you can test the installation with the input files stored in *[targets]*. Give the name of the file and RIBO will search for in the directory *[targets]*, if failed, in *[inputs]*, and otherwise in the directory from which you are executing RIBO. Results will be stored in an output file whose name will be of your choice. Before doing a first test, it should be reminded that naming an output file with an existing file name will overwrite the old one.

1.3 Input file.

The input file contains the information of the geometric arrangement of the target, the starting properties and nature of atoms and the end conditions. Choices about the physical models to be employed and determination of the output modes are decided interactively at run time.

The input file is organized in four (or five) cards³ each of them grouping different sets of information. Every card is initiated by a key word which must not be

³This term of the computing jargon, comes from the days when punch cards were inserted in early computers to transmit a set of data.

changed (it is case sensitive) followed by a line that explains the inputs to come. The explaining line can be edited but not deleted. The next lines contain the core information of the card, like the coefficients of the equations of the surfaces or the logic of the cells. Extra line spacing within a card or between cards is authorized since the program skips blank spaces. Concerning the columns of text, there are no constraints on the horizontal spacing of elements within a line. However, an even arrangement of elements in columns helps to clarify the input file and to find possible mistakes.

The first two cards define the entire geometry of the system (walls and volumes) through which particles will effuse. This means that outer elements where atoms cannot reach shall not be described (e.g. if particles effuse through a tube, then only the inner bore is given as input; the outer surface defining the core is omitted). First comes the card **Surfaces**, corresponding to the walls of the system. Second, the card **Cells** describes the elementary cells that are enclosed by the given surfaces.

Third, the **Source** card, which contains all information about the initial state of the atoms (position, speed, mass) and fourth the card **Tally**, which gathers the end conditions of the simulation.

Usually, time and velocity histograms are processed from the raw output data. However, in some cases it may be useful to get directly a histogram of the release times. This can be solicited by a fifth card called *Histogram*.

1.3.1 *Surfaces* card.

This card contains all the information of the walls (surfaces) that bound the effusion paths of atoms. In some Monte Carlo codes [6, 7] these are known as *bodies*.

The program works internally with the equations of quadrics⁴, which are introduced in the *Surfaces* card. The first lines look like this:

```
Surfaces
n RC T X2 Y2 Z2 XY XZ YZ X Y Z C
third line
fourth line
...
```

The third, fourth, fifth ..., (n+2)th line have the information of the surfaces 1, 2, 3 ..., n. In each of these lines the first entry corresponds to the surface number, 1, 2, 3 ... Numbering should be done in consecutive jumps of 1 unity starting with 1. The second entry contains simultaneously two parameters of the surface nature, the roughness index (RC) - only relevant if Phong [10] reflections are active ([11] chapter 3.2.3) - is stored in the integer part and the absorption probability in the decimal part⁵; the temperature (T) - in Kelvin - of the surfaces is stored in the third entry, and the nine remaining numbers fully define any surface of second degree (quadric). The family of *quadrics* comprises planes, cylinders, cones, spheres, hyperboloids... With this base of surfaces almost any arbitrary shape can be approached with a moderate consumption of memory and computation time. Smooth bends of tubes could have been modeled in a single surface: a toroid. However, these belong to fourth degree surfaces (quartics), for which 33 parameters are needed. This is too costly and rather unmanageable, which means that such surfaces must be implemented in pieces.

*Toroids have to be
approached by
several cylinders*

A plane with an equation $x = 1$ could be implemented as (geometric entries):

```
0000001001
```

⁴ $c_{x^2} \cdot x^2 + c_{y^2} \cdot y^2 + c_{z^2} \cdot z^2 + c_{xy} \cdot x \cdot y + c_{xz} \cdot x \cdot z + c_{yz} \cdot y \cdot z + c_x \cdot x + c_y \cdot y + c_z \cdot z = C$

⁵e.g. 200.05 would mean that the surface is not diffusive (RC=200 ~ Aluminum) and that its absorption coefficient is 0.05 (5 % of the impacting atoms condense to the surface).

The last number corresponds to the independent term (C). Naturally, any proportional equation would be equivalent, e.g:

```
000000-2500-25
```

A sphere centered at the origin of radius R would be $x^2 + y^2 + z^2 = R^2 = C$, thus:

```
111000000 R^2
```

If the sphere is centered at x_0, y_0, z_0 then, the equation transforms to $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$, hence:

```
111000-2x0-2y0-2z0 (R^2 - x0^2 - y0^2 - z0^2)
```

Similarly, ellipsoids and cylinders can be implemented. Moreover, arbitrarily oriented figures may be obtained by applying the rotation equations over the coordinates. For instance, an ellipsoid with an axis \mathbf{a} in the x, y plane forming an angle α with respect to the x coordinate can be introduced by applying a rotation α_z ; \hat{x} is replaced by $\cos(\alpha) \cdot x + \sin(\alpha) \cdot y$ and \hat{y} by $\sin(\alpha) \cdot x - \cos(\alpha) \cdot y$. Therefore:

$$a^2 \cos(\alpha) \cdot x + \sin(\alpha) \cdot y^2 + b^2 \sin(\alpha) \cdot x - \cos(\alpha) \cdot y^2 + c \cdot z^2 = 1$$

*Unlimited number
of objects*

It should be remarked that the number of surfaces accepted by RIBO is unlimited, but, naturally, if simplifications conduct to less elements, then the CPU power requirements will be lower and results will be obtained faster.

At this point some prospective users might be thinking that the level of complication involved in writing the input file is fairly high. Responding to the demands of the first groups that have been exposed to these explanations, a routine called **convgeom** has been written to assist in the generation of the *Surfaces* card.

When executing **convgeom**, the user is asked to choose a surface type, it can be a plane (P), a sphere (S), a Cylinder (C), an ellipsoid (SQ), a cone (K) or a general quadric equation (GQ). Depending on this first choice, RIBO then asks to specify

the radius, or the position of the plane, or the orientation of the plane/cylinder, or the radii of the ellipsoid. Then, the user can decide to rotate the surface by specifying the two Euler angles (or equivalently the two angles of the new \hat{x} axis) and to make a translation. Finally the temperature of the surface is introduced and then the user can continue to define the second surface, and so on. The program writes into the input file the Surfaces card, including the headers.

For example:

```
> convgeom
  what is the input file name?
INPUT
  surface number 1
  P: plane ,S: Sphere ,C: Cylinder ,K: cone ,SQ: ellipse ,GQ: gen
P
  YZ (PX) , XZ (PY) , XY (PZ) , AX+BY+CZ=D (P)
PX
  position?
1
  eq:   0.  0.  0.  0.  0.  0.  1.  0.  0.  1.
  rotation? give polar angles of new x vector
  (polar ,azimuthal)=(90,0) ==> no rotation
90 45
  rotation   90.  45.
  (alpha ,beta)=   0.  0.  0.  0.  0.  0.  0.707107  0.707107  0.  1.
  translation? give x0 ,y0 ,z0
1 1 0
  (x0 ,y0 ,z0)=   1.  1.  0.
  X(x0 ,y0 ,z0)=   0.  0.  0.  0.  0.  0.  0.707107  0.707107  0.  2.414214
  what is the temperature of the surface [K]?
2000
  add surface? (Y/N)
Y

  surface number 2
  P: plane ,S: Sphere ,C: Cylinder ,K: cone ,SQ: ellipse ,GQ: gen\
C
```



```

Radius=?
2
CX, CY, CZ?
CX
eq:  0.  1.  1.  0.  0.  0.  0.  0.  0.  4.
rotation? give polar angles of new x vector
(polar ,azimuthal)=(90,0) ==> no rotation
90 30
rotation  90.  30.
X(alpha ,beta)=  0.250000  0.750000  1. -0.866025  0.  0.  0.  0.  0.  4.
translation?give x0,y0,z0
0 0 0
X0 0.250000  0.750000  1. -0.866025  0.  0.  0.  0.  0.  4.
(x0,y0,z0)=  0.  0.  0.
X(x0,y0,z0)=  0.250000  0.750000  1. -0.866025  0.  0.  0.  0.  0.  4.
what is the temperature of the surface[K]?
1500
add surface? (Y/N)
N

```

The generated input file would look like:

Surfaces											
n	RC	T	x2	y2	z2	xy	xz	yz	...		
1	0.5	2000.	0.	0.	0.	0.	0.	0.	0.707106	0.707106	0. 1.
2	0.5	1500.	0.2500	0.749999	1.	-0.8660	0.	0.	0.	0.	0. 4.

It would now remain to define the cells.

1.3.2 *Cells card.*

This card includes the logic expressions that assemble the previously defined surfaces into the delimiting elements that enclose *cells* (these are referred to as **regions** in e.g. FLUKA [6] or GEANT4 [12]). Surfaces may extend infinitely, and therefore bounds are required to define real elements, thus the mechanism of cells. It should be stressed that cells are **finite** subspaces. This card has the following structure:

```

Cells
n S1 S2 S3 ...
third line
fourth line
...

```

The third, fourth, fifth... $(n+2)^{th}$ lines correspond to the definition of the cells, 1,2,3 ...n, whose first element is precisely the cell number. The following columns define the cell volume in terms of the bounding surfaces. The MC code understands the cells as an intersection of the subspaces divided by a collection of surfaces; the boolean union operation is not defined; regions that require such an operator have to be split into several cells.

*Boolean logics
based on intersec-
tions*

As an example of a simple cell, cell 1 comprises the volume over the plane 1, under the plane 2 and inside the sphere 3 then. The corresponding third line in the cells card is:

```
1 1 -2 -3
```

The recommended methodology consists in sketching the geometry and numbering the surfaces and cells onto this drawing. In some cases the concept *under* or *over* is not clear - like with oblique planes - then the guiding concept is the normal (unitary) vector at the surface; if the cell is to the side of the normal of the surface then the sign is positive, otherwise it is negative. This may again seem to lead to dubious cases. It should then be reminded that the gradient of the surface at a given point unambiguously defines the normal vector.

*Sign defined by
the gradient*

Remarks:

† The MC code does NOT impose any restriction to the maximum number of cells of a given problem nor on the number of limiting surfaces of a given cell.

- † The cells definitions have to be completed with zeros up to the maximum cell degree.
- † The zeros in the cells definitions have to be put at the end.
- † Cells are 'convex' elements; They are exclusively defined with the boolean intersection operator, and not with the union one.

The effort done with **convgeom** is directed to achieve the importation of geometry files whose format is compliant to other MC codes (e.g. MCNPX [3]). This shall enable to write the input files under those formats and to benefit from the plotting options offered by them.

1.3.3 Birth of particles, the *Source* card.

Predefined sources, *Source* card (input file card)

The *Source* card has up to 1+14 entries that describe the source atoms, their starting position and the velocity distribution. The first entry is a character that encodes the geometric distribution shape of the generated atoms. The following vector of 14 numbers gives details of the source: mass number (A) ($S(1)$), temperature ($S(2)$) [K], semi-angle $-\alpha[^\circ]$ - of aperture of the "luminous cone" ($S(3)$) with respect to the central direction of emission ($S(4)$, $S(5)$, $S(6)$), the birth coordinates centroid ($S(7)$, $S(8)$, $S(9)$) [cm], and details regarding the shape of the distribution. Coordinates from S(7) to S(14) are explained below:

Maximum 15 entries for the source

1. **Point source.** Only three geometric parameters are needed ([cm]): $S(7) = x_0$, $S(8) = y_0$, $S(9) = z_0$.
2. **Spherical source.** Like the point source with a radius: $x_0, y_0, z_0, S(10) = R$.

3. **Box source.** Particles are sampled within a parallelepiped centered at x_0, y_0, z_0 , with sides of *full* lengths ([cm]) $S(10) = L_x, S(11) = L_y, S(12) = L_z$ and oriented in space with the angles ($^\circ$) $S(13) = \theta, S(14) = \varphi$.
4. **Target generation.** The starting position is sampled inside a cylinder centered at x_0, x_0, x_0 , of radius R , *full* length ([cm]) $S(11) = L_x$, Gaussian radial dispersion ([cm]) $S(12) = \sigma$, and orientation angles [$^\circ$] of the target cylinder with respect to \hat{x} of $S(13) = \theta, S(14) = \varphi$.

Some remarks have to be made at this stage:

- † A Cylinder category has not been included because random cylindrical birth distributions are a special case of **Target** with *sigma* = 0.
- † All angles -aperture cone and axis orientation- should be given in degrees.
- † The atom temperature (in Kelvin) expresses the energy (velocity) of the atoms. If it is unknown then a good value is that of the wall temperatures since thermalization should fully have taken place after a few hundred collisions.
- † The central angle of emission is indeed a velocity (unitary) directing vector. However, normalization is **not** required, it is done internally, e.g., 2 1 1 (not normalized) can be given instead of 0.8165 0.4082 0.4082 (normalized but not fully precise).
- † For isotropic generation $\alpha = 180^\circ$, for focused beams $\alpha = 0$. In no case it can be omitted.
- † Dimensions are expected, like elsewhere, in cm. If the axis of the cylinder is \hat{x} , then $(\theta, \varphi) = (90^\circ, 0)$. If the cylinder has its axis in \hat{y} then, $(\theta, \varphi) =$

$(90^\circ, 90^\circ)$, if it is in \hat{z} , then $(\theta, \varphi) = (0, *)$. Note that these rotations only affect the geometry of generation and not the velocity vectors. This means, in particular, that angles like $(-90^\circ, 0)$ or $(0, -90^\circ)$ would also be valid for the two examples just shown. In these trivial examples this does not matter, but, for more complicated cases the parity property avoids errors of 'sign'.

† In total, for the point source 1+6+3 data are needed, 1+6+4 for spheres and 1+6+8 for the rest.

An example of Source card: In this example a Target source is used to simulate the generation of A = 95 at energy 2000 K, with isotropic birth direction (180°), in a cylinder centered at 0, 8.5, 0 of radius R = 3 and full length L = 13 cm. Isotopes are born according to a Gaussian law centered at the axis with sigma = 0.5 cm. The axis of the cylinder is oriented parallel to \hat{y} .

Source															
<i>type</i>	Mass	T(K)	Alpha	<i>nx</i>	<i>ny</i>	<i>nz</i>	<i>x</i>	<i>y</i>	<i>z</i>	R	L	sigma	<i>thita</i>	<i>phi</i>	
T	95	2000	180	0	1	0	0	8.50	0	3	13	0.5	90	90	

A simulation for a given source gives way to various sets of events (collisions, flight paths...). If these events are sorted into histograms, the resulting distributions can be added and subtracted to those of other simulations. This permits to reproduce almost any source by performing additions and subtractions from the elementary sources. Thus, e.g. the flight path of atoms born in a cylindric ring can be obtained by subtracting the pondered flight path distribution of a small cylindric source to that of a bigger one. Normalization has to take into account the volume of the respective sources and the number of histories⁶. As a first step, all

⁶In Monte Carlo jargon, each individual repetition is called *history*, and the whole group of histories is called *simulation*, whose *size* will be the number of histories.

histograms could be normalized to unity and then they could be weighted proportionally to the spatial volume of the sources. Analogue procedures are possible for complicated velocity spectra.

Particle birth within a cell (runtime option).

Another possibility to work with more elaborate sources is to force generation inside a given cell. This option is offered at runtime:

```

.....
| ... source limited to a cell? give cell number. |
|   celln > 0 ==> generation limited to volume |
|         defined by celln                       |
|   celln = 0 ==> do not constrain to a cell    |
|   celln < 0 ==> just generate a geometry plot |
.....

```

A basic delimiting source is needed every time. The primary container source, chosen between *Point*, *Sphere*, *Target* and *Box* should include the entire source cell. If the introduced cell number is '0' then the method remains inactive. If it is negative, then RIBO will not do any simulation, it will just generate a plot of the geometry (see 1.7). Otherwise particles will be sampled exclusively inside the selected cell. This method can be universally used, e.g., an infinite sphere containing a dodecahedron, but in order to perform efficient simulations the volume of the primary source ought to match the cell dimensions as closely as possible (this is similar to the concept of **the rejection sampling technique** [13]). For instance, sampling the birth of atoms homogeneously inside the volume of a $\frac{1}{4}$ -sector of a cylinder - cell **2** in the example below - can be carried out first by defining a source cylinder *C* of the same radius and then by using the cell delimiter command. The efficiency of the method corresponds to the fraction of the cylinder sector (here it is 25 %).

Surfaces												
n	RC	T	X2	Y2	Z2	XY	XZ	YZ	X	Y	Z	C
1	0.5	298	0	0	0	0	0	0	1	0	0	0
2	0.5	298	0	0	0	0	0	0	1	0	0	10
3	0.5	298	0	1	1	0	0	0	0	0	0	1

4	0.5	298	0	0	0	0	0	0	0	0	1	0	0
5	0.5	298	0	0	0	0	0	0	0	0	0	1	0
<i>Cells</i>													
n	S1	S2	...										
1	1	-2	-3	4	5								
2	1	-2	-3	4	-5								
3	1	-2	-3	-4	5								
4	1	-2	-3	-4	-5								
<i>Source</i>													
Type	M	T	Alpha	nx	ny	nz	x	y	z	L	s	th	phi
C	40	298	18	1	0	0	5	0	0	10	0	0	0
<i>Tally</i>													
S	Nmax	Tmax	Tpmax										
2	1000	110	101										

... The cell-delimited source command is more powerful than the composition (additions or subtractions) of simulations and can simulate more complicated distributions; e.g., a ring can be simulated by both ways but a cylinder sector is only produced by the cell restriction, as just shown.

Customized sources (programmable option).

The instructions given up to this point already offer a fairly high control of the geometry distribution of the starting atoms and also give the possibility to choose the initial speed, the direction and the semi-aperture angle of an isotropic emission cone. However, the user may want to define fancier distributions: correlation of velocity and position coordinates, speed distributions, position dependent weights... the possibilities are infinite. Responding to these potential needs the open source routine "customsource.f" found in the directory SOURCES allows to create user define source distributions. This subroutine deals with the following (I/O) variables: x,y,z,ux,uy,uz,tp,SOURCE, where "SOURCE" is the vector that stores the parameters introduced in the input file in the card 'Source'. Thus the user can specify whichever dependency g(x,y,z,ux,uy,uz,tp,SOURCE). For instance, for a discrete generation profile over "y" where the probabilities are:

**	<i>pill</i>	<i>position_y</i>	<i>production</i>	<i>probability</i>	<i>Probability</i>
**	1	2.08	2.85	0.19	0.19

**	2	4.22	2.63	0.17	0.36
**	3	6.36	2.41	0.16	0.51
**	4	8.49	2.20	0.14	0.66
**	5	10.61	1.98	0.13	0.78
**	6	12.73	1.76	0.11	0.90
**	7	14.83	1.55	0.10	1.00

10

Then the user should write the following instructions in “customsource.f”:

```

a=rand( zero )
IF ( a .gt. 0.9 ) THEN
  y=14.83
ELSE IF ( a .gt. 0.78 ) THEN
  y=12.73
ELSE IF ( a .gt. 0.66 ) THEN
  y=10.61
ELSE IF ( a .gt. 0.51 ) THEN
  y= 8.49
ELSE IF ( a .gt. 0.36 ) THEN
  y= 6.36
ELSE IF ( a .gt. 0.19 ) THEN
  y= 4.22
ELSE
  y= 2.08
END IF

```

11

12

13

WARNING!! Remember that any changes in the source files will take place only after having recompiled. The compilation instruction appears in the README file on the RIBO distribution and can be performed with the script [tools]/make.sh

Reading source events from a file.

The user can also read values from the file 'init.dat' in the DATA directory of the RIBO distribution. This file must have been previously generated by another run with RIBO or by any other program, e.g. FLUKA. The file init.dat can have whatever format you wish as long as the reading instructions in “customsource.f” are coherent with the data in the file. The variables to play with are the 6-space

coordinates + starting time. For example, to read x, y, z, ux, uy, uz, tp from `init.dat`, just uncomment the following line in “`customsource.f`” and recompile.

```
*      read(9,*)x,y,z,ux,uy,uz,tp
```

1.3.4 The *Tally* (end) card.

The end conditions are preceded by a line with the word *Tally*. There are four inputs, the first and last are relative to the individual histories and the two intermediate fix the end conditions for the global simulation. Up to now, an end *Tally* would look like this (still valid):

Simple *Tally* card.

Tally			
S	Nmax	Tmax	Tpmax
3	6000	750	10

The text and the case of the first line are unmodifiable, the second line is indicative and should not be omitted although it can be edited at the user taste. The first number in the last line indicates the surface used as detector for the atoms¹⁴. When atoms reach this surface, the history is completed and a new atom is simulated. The fourth number alternatively terminates the current history if the individual flight time reaches a certain threshold¹⁵. Setting a low threshold can be useful to have a fast scan of a rapid release peak, sparing the long simulations of the tails of the release distributions. For release fitting purposes, however, the threshold should be high, in order to avoid annoying normalization issues.

¹⁴Several detectors and forced paths can be used with the *Complex Tally card*, described just after.

¹⁵Unlike the other threshold times, this one is not CPU time, but physical flight time.

The second number displays the number of histories (size of the simulation) and the third one the maximum elapsed time. The simulation is finished as soon as any of these two events takes place: maximum time OR maximum number of atoms (histories).

WARNING: Make sure that you don't use tabulators in the Tally card

Complex Tally card.

In addition to the options provided in *Simple Tally Card*, the user can specify several *sequences* each of them composed of 1 or more conditions (every condition is a surface crossing). When ALL the conditions within any of the sequences are accomplished then the particle reaches the end. The user can do the following:

1. Specify to have various end surfaces (that is to say, several trivial *sequences* each composed by a single condition). This is logic **OR**, introduced by means of the character |).

e.g. 10 **OR** 20 → (10 | 20) 6000 750 10.

The particle will stop when it first reaches 10 or 20 or when its elapsed time exceeds 750 s or when the total CPU time for the simulation is greater than 6000 s.

2. Follow only the paths that first go through/hit a surface **and** then through another one (and so on).

e.g. first through surface 10 **AND** then through 20 → (10 20) 6000 750 10.

Note that the order counts.

3. Specify the sense¹⁶ when crossing a surface (\pm).

e.g. end only when it crosses 10 positively \rightarrow +10 6000 750 10.

4. Demand several crossings before end (particular case of option 2.).

e.g. Stop after crossing *surface 20* 3 times

\rightarrow (20 20 20) 6000 750 10.

Some remarks should be made at this point:

† Parenthesis are needed to delimit the boolean definition of end surfaces.

† (S1 | S2) Nmax Tmax Tpmax is equivalent to

(S2 | S1) Nmax Tmax Tpmax

† (-S1 | +S2) Nmax Tmax Tpmax is equivalent to

S1 Nmax Tmax Tpmax

† (S1 S2) Nmax Tmax Tpmax is NOT equivalent to

(S2 S1) Nmax Tmax Tpmax

† (S1 S2 | S2 S1) Nmax Tmax Tpmax means that both S1 and S2 have to be crossed, but the order is irrelevant.

† (S2 | S1 S2) Nmax Tmax Tpmax will never score the second sequence because the first has priority (the condition is read before).

† (S1 S2 | S2) Nmax Tmax Tpmax will score sequence 2 (crossing of surface 2) only when sequence 1 has not been fulfilled (thus, only when S1 has not been previously crossed).

† **WARNING: Make sure that you don't use tabulators in the Tally card**

¹⁶The sign criteria is the gradient to the surface

1.3.5 *Histogram card.*

† **WARNING: This card is OBSOLETE! Please remove this entry from your input files and from the batchscript runtime options**

1.4 Effusion with moving walls (valves).

In some circumstances a few walls in the system are not static all the time. This is the case when a valve is present in the system. In order to deal with these objects RIBO reads over the file `[data]/valves.dat` and obeys the instructions thereby provided. This file contains a table where every line describes the opening and closing sequences of a wall that communicates two contiguous cells. These walls would normally behave as virtual boundaries in the sense that they limit different regions (so they belong to the boolean logic definition) but particles do not *bounce* with them but rather they *cross* them.

The user must therefore specify the closing time (“start_time”) together with the ramp time to close (fast valves may close in about 0.005 s), the respective opening time (“end_time”) and opening ramp time, the surface number of the boundary that is going to lose its transparency and the area where it is acting, that is, the two cells that it is communicating/isolating.

For example, the ”gate” 1 that communicates cell(1) with cell(2) and cell(1) with cell(3) can be closed in the interval $t=(0-10)$ s between cell(1) and cell(3) and then open up there and close between cell(1) and cell(2).

t: 0-10 s		t: 10-... s	
_____		_____	
:			
(2) 1:		(2) 1	

```

|_____|:      |      |_____|      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|_____|      |      |_____|      |

```

====>

```

|_____|:      |      |_____|      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|      |      |      |      |      |
|_____|      |      |_____|      |

```

The corresponding definition would be:

start_time	t_ramp	end_time	t_ramp	surface	cell1	cell2
0	0	10	0	1	1	3
10	0	0	0	1	1	2

REMARKS:

- † If *end_time* is smaller than *start_time* then the program assumes that the wall will no longer open after *start_time*.
- † The surface that acts as a valve needs to be defined in the input file and it must divide two existing regions.
- † The user must specify between which cells the wall will be acting (the order is irrelevant).
- † The user can define up to 10 moving wall events.
- † A wall can appear in several lines if there are several events concerning that wall.
- † If the *end_time* is smaller than the *start_time* then the program assumes that in fact, the end time is infinite.

1.5 Executing RIBO

1.5.1 The Isotope RElease Simulator, IRES

RIBO can be run on-line from a server at www.targisol.csic.es for simplified geometries and through a GUI front-end that permits easy operation of the code. The server contains a wide compendium of release data (diffusion coefficients and sticking times) which is used in combination with this on-line version. The server administrator requires registration.

1.5.2 Express Execution with a batch file.

One of the most useful tools is the execution through a batch file. This file permits to execute the program routinely without having to reintroduce the interactive options.

When RIBO is executed all runtime options are automatically recorded in a file whose default name is 'batch'. If the user then wishes to re-simulate the system, it will be enough to type the *.exe file with '<batch' at the end, which means that all interactive data will be taken from the batch file.

The potential of this methodology is vast; a user could edit the batch file, change some input parameters (probably also the output file name) and save it as batch1, then repeat for different parameters for batch2... Finally, a script like:

```
{  
RIBO < batch1  
RIBO < batch2  
:  
}
```

or even a script with a loop, could produce an enormous amount of data.

Before going deep into the runtime options, you may want to test your RIBO distribution by running one of the batch files stored in the folder `[batchinputs]`

1.5.3 Runtime options.

At run time the user interacts with the program in order to define the input and output file names, the type of output, the models to be used and the options that shape out the results of the simulations.

In the first place the user introduces the file name of the **input file** and of the *output file*¹⁷. The input file name needs no particular extension, usually *.t is used, getting at the fact that the file describes a *target*, but *.inp or any other choices are also accepted. **WARNING:** The string of the input file should not be longer than 20 characters.

```
Name of the input file?  
rectangle.t
```

The name of the **output file** again needs no particular extension, *.out is quite intuitive (*.o should be avoided as it may lead to confusion between the output files and the object (assembled) files). **WARNING:** The string of the output file should not be longer than 20 characters.

```
Name of the output file?  
(Beware it will overwrite the existing file)  
results/rectangle.o
```

The code used to ask whether a histogram was to be made.

† This option is now obsolete!!

¹⁷RIBO searches for the input file in `[targets]`, `[inputs]`, and in running directory, with this priority order.

If the user wishes to intersect the domain covered by *Source* (see 1.3.3) with the volume restricted by a given cell, it is then asked to specify the cell identifier. Otherwise, the value '0' will impose no restriction.

```
| ... source limited to a cell? give cell number. |
| celln > 0 ==> generation limited to volume |
| defined by celln |
| celln = 0 ==> do not constrain to a cell |
| celln < 0 ==> just generate a geometry plot |
```

The following module activates the ionization mode and termination mode. Depending on the choice additional parameters will be demanded.¹⁸

```
| SPECIAL EVENTS, TERMINATION |
| Please choose among these options: |
| 1: Crossing of end surface (1st card in "Tally") |
| No ionizations in the system. |
| 2: Atoms can be ionized in a PLASMA ion source |
| Histories (of atoms and ions) end when |
| they cross the end_surface detector |
| 3: Atoms can be ionized in a SURFACE ionizer |
| Histories (of atoms and ions) end when |
| they cross the end_surface detector |
| 4: Atoms can be absorbed in the walls. Trajec- |
| tories end at absorption or when crossing |
| the end surface |
| So, what do you want, 1, 2, 3 or 4? |
```

As a function of the previous choice four options are then possible:

¹⁸A fourth option including laser ionization could be available in the future. The fifth option is not really an ionization mechanism, it is used to map the distribution of radioactive atoms stuck in the walls as a consequence of prolonged sticking.

1. If RIBO is run without ionization, then no more data will be required at this point and the program will continue to collect the runtime parameters.
2. If ionization is activated, then the code will ask into how many cells the ion source expands. Please, consult the example at 3.1.3 to learn more.

*Configuring the
ion source*

- If *Plasma* ionization has been chosen, then the RIBO code will start by asking which are the indexes of all the ionizing cells, then it will demand the the flux of electrons and the program **plION** will take charge of the calculation of the electron impact direct ionization cross section subsequently asking for the energy of the electron beam, the species to be ionized and the sought ion state¹⁹.
- If *Surface* ionization is chosen, then the program will ask for the cell index of the first ion cell and for its corresponding surface ionizers, then it will do the same for the second ion cell and so on. Next a program called **surfION** will be launched and it will ask which is the number of the surface that corresponds to the ionizer , and then it will demand the atomic numbers of the projectile and of the substrate, or, in their default, the work function, their mass number and the type of compound (Boride, Carbide, Element. . .).

3. If absorption has been activated, then, after each collision to the walls, the code will check if the particle condenses in the surface. In order to do so, a uniform random number is compared to the absorption probability on the surface, which should have previously been introduced in the input file,

*Activating absorp-
tion in the walls*

¹⁹The database (plion.dat) should be consulted to see if the wished ion is tabulated.

in the decimal part of the second input of each surface (the integer part corresponds to the roughness parameter) as explained in 1.3.1.

From the fan of phenomena implemented in RIBO, after having decided about ionization, the user is asked to specify which of the remaining steps will be included in the simulation.

```
SELECT MODE:
 1: Diffusion .
 2: Effusion .
 3: Diffusion+Effusion .
recommended 2
 4: Conductance calculator (Clausing Coefficient).
 5: Diffusion3D. — ALPHA VERSION !!
 6: Heat transfer (cond.+rad.). — ALPHA VERSION !!
```

1: Diffusion This option uses the second Fick law, analytically integrated from the 1st law for simple cases like foils, cylinders (fiber) or spheres (powder) in uniform conditions. More elaborated descriptions based in finite integration of 1st law are provided by option 5 (see also chapter 2. or by the program `Diffuse`, explained on page 107. Note also that even if diffusion is not explicitly asked for (like in option 2), the output file computes the total release efficiency for effusion and diffusion, for a range of diffusion time constants (refer to section 1.9.2 on page 59).

If the user only wants to simulate Diffusion (this cannot be the case if Ionization is activated), then an ulterior option asks whether it will be in-grain or inter grain diffusion or both.

```
Diffusion/Powder. effusion/Both [D/P/B]?
```

In any case, one can obtain average parameters only or delay distributions²⁰.

²⁰Required if time histograms are to be plotted.

```

CHOOSE THE OUTPUT MODE:(recommended 13)
1: Only average figures
2: Diffusion delay distribution

```

2: Effusion The recommended and standard option is '2' because diffusion can be simulated separately with **diffprof**. If only effusion is chosen (in addition or not to ionization), then the alternatives thereby available are those shown in example 3, 3.1.3.

3: Diffusion+Effusion This option combines the first two. The advantage is that the final release times (diffusion+effusion) are obtained without need of convolving an analytical diffusion formula with a fitted histogram of effusion. The drawback is that diffusion is sampled, and therefore it includes a stochastic, artificial error.

4: Conductance calculator (Clausing Coefficient) This option estimates the conductance between two sections in terms of the Clausing coefficient. For technical reasons, the surfaces cannot coincide with the *endsurface* defined in the *Tally* card.

```

1: Only average figures
Introduce the beginning and ending surfaces

```

*Extracting con-
ductances for
analytic vacuum
calculations*

5: Diffusion3D. – ALPHA VERSION !! This uses the combinatorial geometry to define a 3D grid for a finite-difference finite-method integration of Fick's first Law. This allows to compute diffusion from arbitrary geometries, including distinct regions with different diffusing coefficients and other fancy effects. Chapter 2 introduces this module.

6: Heat transfer (cond.+rad.). – ALPHA VERSION !! This is like option 5. but for heat diffusion (conduction). The different with respect to 5, is that in the surface of the objects instead of *atomic desorption, heat radiation* takes place. This option could be useful for coarse heat calculations of your target.

The next alternative concerns the physic model to be used for the reflection of atoms from the walls. Specular reflection (S) may be used to represent reflections of light in systems of mirrors, Lambertian reflections (B) follow the cosine model and thermalize the energy of the projectile to that of the surface, and exact reflections (D) include information of the crystal lattice and of its vibrations. The last choice, (C) custom, activates the user routine `[sources]/customcollisions.f`, explained in detail in 1.8.1 on page 41.

Reflection models

```

| ..... COLLISION MODEL I ..... |
| What model for the treatment of the collisions? |
| S: Specular (E<<) |
| B: B=Knudsen-Lambert (recommended cosine law) |
| D: Debye, semi-classic (under development) |
| C: Custom |
| ..... |
| BACK: Go back to previous menu |
|-----|
| For option "C", Custom, modify the subroutine |
| [sources]/customcollision.f at your convenience |
| and recompile using [tools]/make.sh |
|-----|

```

During runtime the user can decide between having the energy of the reflected particles sampled from the Maxwell-Boltzmann distribution or fixing its energy

to the average of the distribution. The second option saves CPU time while preserving precise results if the number of collisions is high enough (central theorem of the limit for normal distributions), but it may give way to notorious statistical errors in situations where few collisions take place.

```

| ..... COLLISION MODEL II ..... |
| Use average energy or sample from coll. law |
|   Y: Use average energy (recommended) |
|   N: Sample from collision law (slower) |
| ..... |
| BACK: Go back to previous menu |

```

Including temporary sticking

It has been shown that the number of collisions suffered by each particle in its path to the exit of the system (or before ionization) can be included among other output numbers. Nonetheless, in some circumstances it may be interesting to specify sticking in the release times. The user can provide a positive t_s , which will be used as an exact fix number, or a negative number ($t_s < 0$), which will serve to sample a sticking time upon every collision from the exponential law $p(t) = \frac{\exp(\frac{-t}{|t_s|})}{|t_s|}$. The same comment about the CPU-speed vs. accuracy done for energy sampling applies here (it is only worthwhile for a small number of collisions). Note that the user-routine `[sources]/customcollision.f` can cope with more complex schemes, e.g. desorption time depends on the surface and on the position. In particular, this allows to define **cold spots**.

```

| Sticking time[s]? |
| > 0 ==> every collision delays exactly ts [s] |
| = 0 (recommended). The output file includes a |
| post-analysis with several hypotetic ts |
| Use 0 for noble gases |

```

```

| < 0 ==> sample from law P(t)=exp(-t/ts)/ts |
| It slows down calculations. Only necessary |
| if the number of collisions is low |
|-----|
| For ts=ts(X,surface), see examples 4, 5 in the |
| user routine customcollision.f |
|-----|

```

The following step decides if the module for continuous media shall be used. If there is no continuous porous material (only slabs or empty system, etc.) then the tell MESH that you have 0 regions filled with powder or fiber. Otherwise, input *Selecting target filling* the number of regions where powder or fiber is present.

```

... Reading the "source" settings ...
-----
|                                     |
|          M M EEEEE  SSSS  H  H    |
|        MMM E      S    H  H    |
|       M M M EEEE   SSS  HHHHH    |
|          M M E      S  H  H    |
|          M M EEEEE  SSSS  H  H    |
|-----|
|  —  STARTING MODULE FOR e/dIFFUSION IN POWDER  —  |
| How many cells contain powder? e.g. 2 |
| 2 |

```

If a number bigger than zero is given (in the example 2) then the program will pose some additional questions. First it will ask which are the region numbers of the cells filled with target material.

```

what is the cell number of the powder cell # 1
1
what is the cell number of the powder cell # 2
2

```

In the example the first region is number 1 and the second is number 2. Then more information is requested. In particular, MESH will ask what is the average flight path of atoms in the Fiber or Powder. It will also ask for the probe spheres that should be used as macro steps for faster calculation.

```

mean free path in powder/fiber[um]?
  e.g. UC powder 15, ZrO2 fiber 250
50
sphere probe radius[um](step path)? e.g. 800
  recommended: 6 x mean free path
300
sampling a limiting residence time in powder...
DONE, tmax_sphere_of_powder[ms]= 0.0183047683
Characterization of macrocollisions in powder:
  Generating time and angle probability functions
  ... this may take some minutes ...

```

Next, the pressure inside the system is introduced in order to enable collisions (if any) between gas atoms. Typically simulations are done for perfect molecular flow, thus, at nil pressure. *residual pressure*

```

| ..... COLLISION WITH RESIDUAL NUCLEI ..... |
| Residual pressure [torr]? (0.75 torr = 100 Pa) |
| P <= 0 ==> molecular flow (ideal vacuum)      |
| P > 0 ==> collisions with residual gas         |
| (if you do not know it but you know the mean  |
| free path type anything > 0 now)              |

```

If no more input is given, the program will estimate the mean free path between atom collisions from statistical considerations. Alternatively, if the mean free path is known, it should be provided.

```

|
|   RRRR   EEEEE   SSSS   GGGG   AAA   SSSS   |
|   R  R  E     S     G     A  A  S   |
|   RRRR   EEEE    SSS   G GG   AAAAA   SSS   |
|   R  R  E           S  G  G  A  A     S   |
|   R  R  EEEEE  SSSS   GGG   A  A  SSSS   |
|-----|
| - STARTING MODULE FOR INTERACTIONS BETWEEN ATOMS - |
|-----|
| ..... COLLISION WITH RESIDUAL NUCLEI II ..... |
| INTRODUCE A NUMBER n, IF ... |
| n > 0 ==> n = ATOM DIAMETER [pm] |
| - Some indicative values [pm] are: |
| He:62;Ne:76;Ar:142;Kr:176;Xe:216;Rn:240;N2:374 |
| - Consider also diameter  $\tilde{=} 2 \times 1.4 \times A^{\{1/3\}}$  |
| n < 0 ==> |n| = -MEAN_FREE_PATH [cm] in "vacuum" |
| n = 0 ==> RIBO will estimate the MEAN_FREE_PATH |
|-----|

```

In case of ionic transport, the emittance is computed with the program *emittance.f*. The central axis has to be modified to match the particularities of each case. The output is written in *emit.map*.

```

*****
****   EEEEE  M  M  II  TTTT  TTTT  AAA  N  N  CCCC  EEEEE  ****
****   E      MM MM  II  T      T      A  A  NN  N  C      E      ****
****   EEEE   M M M  II  T      T      AAAAA  N N N  C      EEEE   ***
****   E      M  M  II  T      T      A  A  N  NN  C      E      ****
****   EEEEE  M  M  II  T      T      A  A  N  N  CCCC  EEEEE  ****
*****
*           The RIBO project , MARIO SANTANA LEITNER 2000–2006
*-----*
SUBROUTINE emittance(epsilon ,X3,U3)
  real*8 X3(3),U3(3)           ! i
  real*8 epsilon(2)           ! o
  real*8 R(2),Pi               ! a
*****
*           CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM
*
*   USAGE:  print emittance maps in a given cross section f
*

```



```

*
*
*   VARIABLES:
*
*       INPUT
*
*   X3(3)  ————  Absolute position (x,y,z)[cm]
*
*   U3(3)  ————  Velocity {ux,uy,uz} of the ion [m/s]
*
*       AUXILIARY
*
*   R(2)   ————
*
*   Pi     ————  3.141592...
*
*       OUTPUT
*
*   epsilon(2) ————  Emittance in perpendicular directions
*
*   R(2)   ————  Transverse position of the beam axis [cm]
*
*
*****
*
*       Pi=3.14159257
*
*   Modify this according to the exit axis and position
*
*       R(1)=X3(1)-0.0
*       R(2)=Z3(3)-1.0
*
*   Safeguard condition
*
*       IF (U3(3).eq.0.0) THEN
*           write(6,*)'Warning:extraction axis perpendicular to velocity'
*           U3(3)=1E-12
*       END IF
*
*       write(4, '(1X,F8.4,F9.5,F8.4,F9.5)')R(1),U3(1)/U3(3),R(2)
*
*   $           ,U3(2)/U3(3)
*
*       epsilon(1)=epsilon(1)+10*R(1)*1000*atan(U3(1)/U3(3))/Pi
*       epsilon(2)=epsilon(2)+10*R(2)*1000*atan(U3(2)/U3(3))/Pi
*
*   ENDS$

```

At this point the program has all necessary elements to pursue simulations, eventually including in-grain diffusion in slabs, particles or fibers, inter-grain diffusion through powders or fibers, effusion in molecular or intermediately pressurized systems, with mirror like walls, diffusive walls or crystals, and ionization in plasma chambers or surface ionizers. After preprocessing some messages will appear in

the screen and simulations will start. The connectivity matrix will be stored in a file called CCONM and the results will be stored in the output file.

1.6 Debugging running errors.

1.6.1 Introduction.

Running a Monte Carlo code is similar to programming: every little thing that is overlooked contains a potential bug and it will most likely induce an error at some point. The time required to detect and to fix the bug will exceed the amount of work needed to avoid such flaws from the beginning. This general recommendation concerns specially the implementation of the geometry. It is strongly advised to take some time to do a sketch of the system as it will be modeled, drawing and labeling each surface and marking the cells. Eventually this step may already help to rise some questions about the optimality of the target and often new configurations are immediately suggested. Moreover, the sketch helps to attain a logic numbering of surfaces and of cells, this will in turn aid to write the input file and also to introduce future modifications. Once the input file completed, it should be reread, cross checking with the sketch, counting the number of cells and surfaces, verifying that no space is undefined or multiply defined. If the file has been correctly tabulated, a fast glance will spot typing errors from the irregularities in the columns of data.

As what concerns the systematics, another advice is to make several stages before reaching the full complexity of the problem. This enables to progressively correct mistakes; for instance, first one can implement the system of tubes with a simple source (point source), then, after debugging, insert the target material (foils

or powder...), and, only after proper running, implement a more complex source, etc. If this were done in one go, it would be harder to disentangle the individual causes of the overall errors.

If a major error has been produced the program will terminate, complaining about some input/output error. If this happened it could be due to any of these causes:

- Some of the lines needed have been forgotten (like the card names or the explanatory lines).
- A coefficient in the cells definition has been omitted, typically some zero, or the roughness coefficient, or the independent term.
- All cells do not have the same number of elements; some cell has not been completed with zeros.
- The source type is not compatible with the number of source parameters (missing orientation angles or particle mass, or temperature ...).
- An additional line has been written somewhere (line breaking is not authorized).

Once the input file is 'digested' by the program, it is preprocessed and a message summarizes the geometry, telling the number of surfaces, the number of cells and the maximum number of surfaces that contour a cell.

```
----- number of cells: 2
----- highest number of walls in a cell: 6
----- number of bodies: 7
.....
```

The MC code then assembles the geometry, thereby networking the cells that have a common interface, and then it saves the result into an array. By doing so, each time that RIBO verifies whether a particle migrates from the current cell to any neighboring region (this happens after every collision), the amount of checks needed is reduced to the number of connecting cells, listed in the connectivity matrix (computed only one, at the beginning). That matrix, printed separately in the file **CCONM** condenses a lot of information of the geometry of the system and therefore it aids to cross-check the *Cells* and the *Surfaces* cards of the input file. The file contains a column with several groups of integer numbers; every group starts by the cell number and it is followed by those cells that have a common interface with that given cell²⁹. The CCONM matrix of the followed example has only two groups (there are only two cells):

```
1
2

2
1
```

The first group says that cell number 1 (first line) is connected to cell number 2 (second line), the second group says that cell number 2 (third line) is connected to cell number 1 (fourth line). This case is quite trivial but it helps to underline two properties:

- The connectivity matrix is symmetric.

²⁹This does by far not mean that these cells are actually touching.

- All groups should at least have two elements so that no cell is isolated from the rest³⁰. Every row of the connectivity matrix must have at least two numbers different from zero.

If the source of particles were entirely located outside the system or if the cell source contained some errors, then a message on the screen would clearly warn the user:

```
Error in source card, source out of cell domain?
```

This may happen due to an incorrect implementation of the source or to some flaw in the definition of the geometry of the system.

In some cases the errors of geometry are detected by a routine of RIBO that casts a message on the screen like:

```
GEOMETRY ERROR!  
check cell 2 or surface 4
```

However, many errors are not traced by the code; a vast geometry case compiler is still a pending task for future upgrades of the program.

For the time being one more tool is available for debugging purposes. At runtime, option 7 gives the opportunity to track a particle from birth to termination. This tracking can be made effective in three subsequent choices:

```
Choose one option  
1: coordinates x,y,z  
2: cell history  
3: surface history
```

Normally this is enough to detect persistent geometry mistakes.

³⁰If that were the case an error message would show on screen and in the CCONM file.

In addition to all this, the user can change the end surface and choose closer surfaces (e.g. instead of having the gage at the end of a complex tubular system, it can be first put at the end of the beginning section and then pushed forward to next section, etc.) to ease the detection of the errors of geometry.

1.6.2 Typical error situations/messages

Source error

```
...Reading the "source" settings...
invalid number: incomprehensible list input
apparent state: unit 1 named inputfile.t
last format: list io
lately reading direct formatted external IO
Aborted
```

Reason: † Too few parameters in Source card.

Input file error: surfaces not read.

```
READING INPUT FILE
temp.t
...
1) reading surfaces...
list in: end of file
apparent state: unit 1 named temp.t
last format: list io
lately reading direct formatted external IO
Aborted
```

Reason: † You wrote 'cells' instead of 'Cells'.

Input file error: apparent error in source.

```
Post-processing geometry...
DONE
RUNNING...
x,y,z 0.302196309 0.00917545272 -0.146321036
Error in source card, source out of cell domain?
```

Reason:

† You forgot some comment line, e.g. after 'Surfaces', or after 'Cells'.

† You defined a source that falls completely off the geometry.

Input file reading error

```
... Reading the "source" settings ...
invalid number: incomprehensible list input
apparent state: unit 1 named inputfile.t
last format: list io
lately reading direct formatted external IO
Aborted
```

Reason:

† You did not exactly spell 'Surfaces'.

† You misspelled the name of the input file.

Error in the cells definitions

```
reading geometry ...
... storing surfaces (bodies)...
... storing cells (regions)...
invalid number: incomprehensible list input
apparent state: unit 1 named temp.t
last format: list io
lately reading direct formatted external IO
Aborted
```

Reason: † Missing elements in the region definition.

1.7 Making 3D model views.

Making a 3D view of the target geometry is one of the fastest options to find out bugs in the geometry and to become aware of the target proportions. This is now possible through *3D-RIBO* and *Povray*. The first program executes RIBO and halts it when the Povray compliant geometry file (*.pov) is created. Next *Povray* is called and the image file (*.tga) is created.

```
> 3D-view input.t 1
```

The flat *l* specifies that the image file will be a thumbnail. 2 or 3 would produce images with higher resolution.

The intermediate *.pov file is a text document with the combinatorial geometry in Povray format. This file can be edited to include fancy features, e.g. roughness and special optical effects. Then, 3-D can be run by specifying the proper file extension.

```
> 3D-view input.pov 1
```

1.8 User defined subroutines

RIBO includes a growing number of routines that the user can change and customize. It goes without saying, that important user developments will be gladly accepted and acknowledged, in this way, RIBO will grow and the whole community will benefit.

Presently these are the routines available in [*sources*]:

1.8.1 User defined collisions, *customcollision.f*

This routine is called if the user answers 'C' in the runtime question about the collision model.

```

SUBROUTINE CustomColl(X3, grad3 ,T,A, surn , rc , ts , tde ,U3, COL6)
  integer*4 surn                                ! i
  real*8 X3(3) , grad3(3) , rc ,T,A, ts        ! i
  real*8 U3(3) ,COL6(6) , tde                  ! i/o
  real*8 SU3(3) ,SV3(3) ,SW3(3)              ! a
  real*8 phi , alpha , cosphi , n , zero , k , V3scaV3 , norm , a1 , b1 , c1 , v , t0    ! a
  external rand
*****
*

```


*	WRITTEN BY:	*	32
*	MARIO SANTANA LEITNER, 2006	*	
*		*	
*		*	
*	CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM	*	33
*	USE: transport problems with special physics for the collisions	*	
*		*	
*	HIGHLIGHTS:	*	
*	– customized law, as a function of impinging angle	*	
*	– desorption time sampled from Frenkel Law	*	34
*	– simulation of COLD spots	*	
*	– desorption function of temperature (position)	*	
*		*	
*	ACTIVATE USE THROUGH option 'C' at RUNTIME to the QUESTION:	*	
*	'What kind of treatment of the collisions?'	*	35
*	'S=Specular (E<<),B=Knudsen-Lambert ,D=Debye ,C=Custom	*	
*		*	
*	VARIABLES:	*	
*	INPUT	*	
*	X3(3) ——— Absolute position at collision point (x,y,z) [cm]	*	36
*	grad(3) ——— Surface gradient (normal) at collision point	*	
*	always pointing inwards and normalized.	*	
*	T ——— Starting energy of the atom [k] (source card)	*	
*	A ——— Atomic mass [uam] (source card of input file)	*	
*	surn ——— Surface number (from input file)	*	37
*	rc ——— First parameter of the surface after surface	*	
*	index number (e.g. roughness coefficient)	*	
*	ts ——— Average sticking time per collision (if it is ~	*	
*	constant all-over, it is normally define as 0	*	38
*	in the runtime options. The output file then	*	
*	includes a sensitivity analysis for values	*	
*	other than 0.	*	
*	I/OUT	*	
*	U3(3) ——— Incoming/Outgoing velocity, ux,uy,uz of the atom	*	
*	[m/s]	*	39
*	COL6(6) ——— Counter for the type of collision for each history	*	
*	1: Maxwell type 1	*	
*	2: Maxwell type 2	*	

```

*           3: Mirror-like *
*           4: Debye (semi-classic) *
*           5: Custom 1 *
*           6: Custom 1 *
*
* tde ——— Particle desorption time [s]. Normally desorption *
*           time in the collisions is taken into account *
*           at the end of each history as "ts*COLL". *
*           If you use a special desorption time distri- *
*           bution, (examples 4, 5), update the value of *
*           tde (as shown in those examples). *
*
*           AUXILIARY *
* SU3(3),...----- Base of auxiliary vectors to build up reflection *
* phi, alpha...--- Auxiliary angles to build reflected direction *
*           ... *
*
*           EXTERNAL *
* rand ——— Seed for random generation *
*
* NOTES: *
* - Check available mathematical functions in math.f *
* - Surface ionization is done separately *
* - *
*
* *****
*
*
*-----*
* EXAMPLE 1, snell reflection if rc>0.5, otherwise perpendicular *
*-----*
*
* zero = 0. 0
*
* IF (rc.gt.0.5) THEN
*
* k = V3scaV3(grad3,U3) ! k = grad3(1)*U3(1)+grad3(2)*U3(2)+grad3(3)*U3(3) < 0
*
* U3(1) = U3(1) - 2 * k * grad3(1) !
*
* U3(2) = U3(2) - 2 * k * grad3(2)
*
* U3(3) = U3(3) - 2 * k * grad3(3)
*
* COL6(5) = COL6(5) + 1.0
*
* ELSE
*
* U3(1) = grad3(1)
*
* U3(2) = grad3(2)

```

```

*      U3(3) = grad3(3)
*      COL6(6) = COL6(6) + 1.0
*      END IF
*
*-----*
*      EXAMPLE 2, If the incidence angle is within a cone of 60 \deg around
*      the surface normal, then reflect isotropically within that cone,
*      otherwise use mirror-like reflections
*-----*
*
*      cosphi = V3scaV3(U3, grad3)/(norm(U3)*norm(grad3))
*      ! ||cos(phi)=a * b / |a|*|b| < 0
*      IF (cosphi.lt.0.5) THEN                                ! (phi>60\deg)
*      k = V3scaV3(grad3,U3) ! k = grad3(1)*U3(1)+grad3(2)*U3(2)+grad3(3)*U3(3)
*      U3(1) = U3(1) - 2 * k * grad3(1)
*      U3(2) = U3(2) - 2 * k * grad3(2)
*      U3(3) = U3(3) - 2 * k * grad3(3)
*      COL6(5) = COL6(5) + 1.0
*      ELSE                                                    ! (|phi|>60\deg)
*      CALL isotropic(grad3,U3,60)                            ! defined in math.f
*      COL6(6) = COL6(6) + 1.0
*      END IF
*
*-----*
*      EXAMPLE 3, Poth (or Phong) model.
*      The particle is reflected with a cosine^n law around the
*      mirror-like reflected direction.
*      n depends on the roughness
*-----*
*
*      zero=0.0
*      n = exp(rc*5.887)
*      ! ||rc-->0 ==> very rough
*      cosphi = V3scaV3(U3, grad3)/(norm(U3)*norm(grad3))
*      ! ||cos(phi)=a * b / |a|*|b| < 0
*      *C----- Sample the alpha from "cos(alpha)**n" by REJECTION
*      alpha = asin(rand(zero)*cosphi)
*      ! ||alpha uniform between 0 and 90-phi
*      DO WHILE (rand(zero).ge.(cos(alpha)**(n-1.0)))

```

48

49

51

52

53

54

```

*      alpha = asin(rand(zero)*cosphi)
*      END DO
*----- Now sample the azimuthal angle, beta by INVERSE transformation
*      beta=rand(zero)*2*3.14159265
*----- Now build up local base and get exit vector U3
*      CALL surfaceBase(SU3,SV3,grad3)
*      a1 = SIN(alpha) * COS(beta)
*      b1 = SIN(alpha) * SIN(beta)
*      c1 = COS(alpha)
*      U3(1) = a1 * SU3(1) + b1 * SV3(1) + c1 * grad3(1)
*      U3(2) = a1 * SU3(2) + b1 * SV3(2) + c1 * grad3(2)
*      U3(3) = a1 * SU3(3) + b1 * SV3(3) + c1 * grad3(3)
*      CALL renorm(U3)
*      COL6(5)=COL6(5)+1.0
*
*-----*
*      EXAMPLE 4, if incidence angle > 60 ==> mirror-like, no sticking
*      Otherwise, cosine law reflection + maxwell-Boltzmann thermali-
*      zation + sticking time ~ exp(-t/t0), t0=1E-5s
*-----*
*
*      zero=0.0
*      cosphi = V3scaV3(U3, grad3)/(norm(U3)*norm(grad3))
*      ! |cos(phi)=a * b / |a|*|b|
*      IF (cosphi.lt.0.5) THEN
*          ! (phi>60\deg)
*          k = V3scaV3(grad3,U3) ! k = grad3(1)*U3(1)+grad3(2)*U3(2)+grad3(3)*U3(3)
*          U3(1) = U3(1) - 2 * k * grad3(1)
*          U3(2) = U3(2) - 2 * k * grad3(2)
*          U3(3) = U3(3) - 2 * k * grad3(3)
*          COL6(5) = COL6(5) + 1.0
*      ELSE
*          ! (phi>60\deg)
*          CALL cosineLaw(grad3,U3)
*          ! existing function
*          CALL Boltzmann(U3,T,A,mode)
*          ! existing function
*----- Now adding updating total desorption time per particle, tde:
*      zero=0.0
*      tde = tde + ts * log(1/rand(zero)) ! ts provided at runtime, e.g. 1E-6 [s]
*      COL6(6) = COL6(6) + 1.0
*      END IF
*-----*

```

```

* EXAMPLE 5, Sample the VELOCITY vector (angle, speed) with the cosine *
* Law and and M-B thermalization. *
* USE customized STICKING, as a function of position, X3, surface *
* and with a distribution function (do not just use average value) *
* of the type :  $\exp(-t/ts)/ts$ ,  $ts=1E-5s$  *
* The dependence with position is, e.g.:  $t0 = ts * (1 + 1.25*z)$  *
* The surface (surn = 5) is a COLD SPOT where:  $t0 = ts * 100.0$  *
*-----*
*
* zero=0.0
* CALL cosineLaw(grad3,U3) ! existing function
* CALL Boltzmann(U3,T,A,mode) ! existing function
*----- Now adding updating total desorption time per particle, tde:
* IF (surn.eq.5) THEN
* t0 = 100.0 * ts
* ELSE
* t0 = ts * ( 1 + 1.25 * X3(3))
* END IF
* tde = tde + t0 * log(1/rand(zero)) ! Frenkel Law
* COL6(1) = COL6(1) + 1.0 ! CosineLaw type counter
** WARNING: use a positive ts, with a negative time you
** would be using twice the exponential law!
END

```

63

64

65

66

1.8.2 User defined desorption in powder, *customcollision.f*

Very often the number of collisions in the powder is very large and therefore desorption becomes important for most isotopes. If the mean sticking time in the powder differs from that elsewhere or if it changes from one region of powder to another, the user can customize the user routine [*sources*]/*powderdesorp.f* in order to meet the particular requirements.

```

SUBROUTINE powderdesorption(COLPOW, celln, tdep, ts)
  integer*4 celln ! i
  real*8 COLPOW, ts ! i
  real*8 tdep ! i/o
  external rand
*****

```

67

```

*
*
*   WRITTEN BY:
*
*           MARIO SANTANA LEITNER, 2006
*
*
*-----*
*
*           CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM
*
*   USE:  add customized desorption time within a powder cell
*
*   NOTE: normally at the end of the history, the amount ts * (COLP)
*           is added if tdep is zero
*
*           don't try to sample from distributions (i.e. Frenkel) because
*           here we check after a bunch of collisions (actually when the
*           particle exits the powder cell), and NOT after each
*           collision. This will not imply any big error (central limit
*           theorem)
*
*   HIGHLIGHTS:
*
*           - simulation of COLD spots
*
*
*   THIS ROUTINE IS ALWAYS READ THROUGH
*
*
*   VARIABLES:
*
*           INPUT
*
*   celln  ——— Cell (region) (from input file)
*
*   COLPOW ——— Number of collisions in the powder cell in the
*
*                   last passage through
*
*   ts    ——— average sticking time per collision
*
*                   (given at runtime)
*
*           I/OUT
*
*   tdep  ——— Particle elapsed desorption time spent in powder
*
*                   domains [s]
*
*           EXTERNAL
*
*   rand  ——— Seed for random generation
*
*
*-----*
*
*
*   EXAMPLE 1, cells 1, 3, 5 contain powder, the first one is warmer than
*   the second and that one is warmer than the third

```

```

*-----*
*      IF ( celln.eq.1) THEN
*          tdep = tdep + 1.0*ts
*      ELSE IF ( celln.eq.3) THEN
*          tdep = tdep + 2.0*ts
*      ELSE IF ( celln.eq.5) THEN
*          tdep = tdep + 3.0*ts
*      END IF
*
END

```

76

1.8.3 User defined source distributions, *customsource.f*

This routine, called at particle generation is explained in 1.3.3, on page 16.

1.8.4 Setting up the electro-magnetic fields, *readEfield* and *Bn.f*

These routines are called when ions are created and explicit transport is requested at runtime:

```

----- ELECTROMAGNETIC ION TRANSPORT -----

You are dealing with ions, which "see" the EM-
fields, external or internal (plasma fields ...)
IF you DO NOT know these fields, but you have a
rough idea about the extraction efficiency of the
ions from the source, you can give this number
as well as the estimated extraction time [ms]
e.g: 0.3 1E-3 would mean that 30 % of the
initially ionised atoms, make it to the outlet
as ions. The rest recombine. The process takes
about 1 microsecond in all.

IF you know these fields, you can edit explicitly
transport the ions, by editing the functions:
readEfield.f, Bn.f functions, emittance.f
AND answering -1 0 now.
.....
Ion extract. efficiency [0-1], extract. time [ms]?
-1 0

```

Thus, by answering “-1 0” this mode is activated. The first routine, `[sources]/readEfield.f` serves to determine the electric field vector as a function of time, space . . . , either analitically or by reading it from an external binned file.

```

*
*****
*
*           readEfield is a subroutine of the RIBO code
*
*****
*
*****
*   RRRR   EEEEE   AAA   DDDD   EEEEE   FFFF   I   EEEEE   L   DDDD   *
*   R   R   E     A   A   D   D   E     F     I   E     L     D   D   *
*   RRRR   EEE     AAAAA   D   D   EEEE   FFFF   I   EEEE   L     D   D   *
*   R   R   E     A   A   D   D   E     F     I   E     L     D   D   *
*   R   R   EEEEE   A   A   DDDD   EEEEE   F     I   EEEEE   LLLL   DDDD   *
*****
*           THIS ROUTINE READS ELECTRIC FIELD DATA (SI) FROM A FILE
*
*           and/or from constants
*
*****
*
*   DEPENDENCIES:
*
*
*           checkCELL----->-----|-----|-----|
*
*           -----|-----|-----|-----|-----|
*   readEfield |-----> El ----|-----|-----|
*           ~~~~~|-----|-----|-----|-----|
*           |-----> computedt ----|-----|
*   Bn ----|-----|-----|-----|-----|-----|-----|
*           |-----> B ----|-----|-----|
*           |----->-----|-----> electrans ----|
*           |-----|-----|-----|-----|
*           backupstep --|-----|
*
*****
*
*   VARIABLES:
*
*
*   *) x,y,z: absolute coordiantes , [x,y,z]=m
*   *) E(3): electric field vector (SI)
*

```

Defining Electric Fields

77

78

79

80

81

82

83


```

*****
*
*      SUBROUTINE readEfield(x,y,z,E)
*      real*8 x,y,z,E(3)
*
*
*      *****          example          *****
*      EXTRACTION FIELD + PLASMA RADIAL FIELD
*
*-----*
**  A field E={Er,Ey} penetrates a tube up to Length-Upen = 4.5 cm = y
**  The tube has a radius Rmax = 2 cm
**  We have a map of the electrostatic potential U for a grid with
**  17 radial divisions and 68 longitudinal divisions —> Ury(17,68)
**  There is a second radial field due to plasma effects Up
**  this field behaves as Up(r)=Up*exp(-r/lambda)/(lambda*0.01)
*-----*
*      real*8 Ury(17,68),r,Rmax,Er
*      real*8 Ex,Ey,Ez,Up,lambda
*      integer*4 i,j
***
*      r=x*x+y*y
*      Ymax=5
*      Rmax=2
*      Udep=0.5
*      IF (y.ge.(Ymax-Udep).and.r.lt.Rmax) THEN
*          open(UNIT=1,FILE='data/Urz.dat',STATUS='OLD')
*          read(1,*)Ury
*          close(1)
*          Ey=30.0*100.0*(Ury(i,j)-Ury(i,j+1))/(Udep/67.0)
*          Er=30.0*100.0*(Ury(i,j)-Ury(i+1,j))/(Rmax/15.0)
*      END IF
*** need to define Up and lambda (see Koe2000)
*      Er=Er+(Up*exp(-r/lambda)/(lambda*0.01))
*      Ey=Ey
*** conversion to rectangular coordinates
*      Ex=Er*x/(abs(r+1E-10))
*      Ez=Er*z/(abs(r+1E-10))
*      E(1)=Ex
*      E(2)=Ey

```

```

*          E(3)=Ez                                     *          91
*****
*
*          E(1)=0.0
*          E(2)=0.0
*          E(3)=0.0
*
*          END

```

As for the Magnetic field, ...

```

*****
*          Bn is a function of the RIBO code          *          93
*****
*          CUSTOMIZE THIS FUNCTION TO FIT YOUR CONDITIONS
*          *****
*          BBBB                                     *
*          B  B                                     *
*          BBBB                                     *
*          B  B                                     *
*          BBBB                                     *
*          *****
*
*          DEPENDENCIES:
*
*          Bn ----| (O)
*          |----> B ----| (S)
*          |----> explicitEM ----| (S)
*          |----> main.f (S)
*
*          *****
*
*          MEANING:
*
*          *) Bn is the nth component of the magnetic field
*          Thus , B={B1,B2,B3}
*          [Bn] = T
*
*          *****
*
*          VARIABLES:
*
*          *) X0(4) {space , time} four-vector
*          [X0]={m,m,m, s}

```

*Defining Electric
Fields*

93

94

95

96

97

98

```

*
*****
FUNCTION Bn(X0, n)
  real*8 X0(4), Bn
  integer*4 n
  Bn=0.0
  IF (n.eq.1) THEN
    Bn=0.0
  ELSE IF (n.eq.2) THEN
    Bn=0.0
  ELSE IF (n.eq.3) THEN
    Bn=0.0
  *
    Bn=1.0+X0(3)/10
  *
  END IF
END

```

99
100
101

To obtain emittance plots, modify the emittance plane and settings in the routine `[sources]/emittance.f`. This function is presented in section 1.5.3.

1.8.5 User defined output printing files *userprint.f*

This routine is called at the end of each particle history. It allows the user to print in whatever relevant information in units 21-25¹⁰²

```

SUBROUTINE userprint( ofile , td , tp , ts , tPo , tde , tdep , COL, COLP, COL6, U3 ,
  X3, X03, ocell , celln , ion , ads , survec , endvec )
  character*20 ofile ! i
  integer*4 ocell , celln , ion , ads , survec (11) , endvec (11) ! i
  real*8 X03(3) , X3(3) , U3(3) ! i
  real*8 COL, COLP, COL6(6) , td , ts , tde , tdep , tPo , tff , tp , tT ! i
  real*8 V3scaV3 , norm , eve ! a
  external rand ! e
*****
*
* WRITTEN BY:
*
* MARIO SANTANA LEITNER , 2006
*

```

103
104

¹⁰²The data will be stored in the files `21.out` to `25.out` in the directory from which RIBO is run.

```

*
*-----*
*          CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM          *
*          USAGE:  print values after each history, eg. x,y,z,teff *
*
*          VARIABLES:
*
*          INPUT
*
*          ofile  -----  Output file
*          ocell  -----  Cell where particle was born
*          celln  -----  Cell where particle trajectory ends
*          ion    -----  Ionic state: 0 = neutral, 1 = ionized
*          ads    -----  Adsorption: 0 = none, 1 = particle was adsorbed
*          endvec(11) --- Tells which of the paths (you can put up to 11)
*                          has been first completed. In normal cases
*                          (a single end surface) this is not relevant
*          survec(11) --- Tells which is the end surface
*          X03(3)  -----  Absolute position at starting point (x0,y0,z0)[cm]
*          X3(3)   -----  Absolute position at end point (xf,yf,zf)[cm]
*          U3(3)   -----  End velocity {ux,uy,uz} of the atom [m/s]
*          COL     -----  Total number of collisions outside powder/particle
*          COLP    -----  Total number of collisions inside powder/particle
*          COLP6(6) ----- Collisions of type i per particle
*          td      -----  Diffusion release time [s]
*          ts      -----  Sticking time per collision [s/coll]
*          tde     -----  Particle elapsed desorption time spent in surfaces
*                          other than those of the powder [s]
*          tdep    -----  Particle elapsed desorption time spent in powder
*                          cells [s]
*          tPo     -----  Flight time in the powder [s]
*          teff    -----  Flight time outside the powder [s]
*          tp      -----  Total effusion time:
*                           $tp = teff + tPo + ts * (COL+COLP)$ 
*          tT      -----  Total release time per particle:
*                           $tT = tp + td$ 
*
*          AUXILIARY
*          eve     -----  Event index:
*                           $eve = ion + 2 * ads$ 
*          ...
*
*          EXTERNAL

```

```

*      rand  —————  Seed for random generation      *
*
*  OUTPUT UNITS
*    — UNITS 21–25 are available for the user
*
*****
*
*
*-----*
**  EXAMPLE 1: write to unit 21 the final and initial radius in a  *
**    cylinder oriented with axis z
*-----*
*
*      write (21, FMT= '(1X, 2F9.6) ' ) sqrt (X3(1)**2+X3(2)**2),
*      $      sqrt (X03(1)**2+X03(2)**2)
*
*-----*
**  EXAMPLE 2: write to unit 22 the number of collisions of each type *
*-----*
*
*      write (21, FMT= '(1X, 8F9.2) ' ) COL, COLP, COL6
*      $      sqrt (X03(1)**2+X03(2)**2)
*
END

```

† **NOTE:** A special routine exists to individualize the printing for 3D diffusion problems. Please consult page 73.

1.8.6 Other user defined routines

The RIBO distribution includes some other user customizable FORTRAN functions and subroutines. Indeed, the directory `[sources]` contains also the open routines for the 3-D diffusion calculations of 3D GRID, explained in chapter 2, and printed in 2.3.

1.9 Output file.

The output file contains information about the run, average numbers and variables linked with every history. Its shape depends on the requests made by the user during execution, but in general terms these elements can be out-marked:

- *Heading*. It includes authoring information and specifies the input file name and path.
- *Individual scores*. This section may contain the global and fractioned release time and total number of collisions for every simulated atom. This usually constitutes the core goal of the simulations since it permits to reconstruct the intrinsic release functions.
- *Preconfigured histogram*. This histogram (unnormalized to bin size) appears if the card Histogram is included in the input file.
- *Average figures of the release speed*. This group of data includes the average release time, the relative time consumption in the diffusion and effusion phases, the amount of in-grain versus inter-grain diffusion time. . . In presence of ionic fluxes, this output is divided in two groups, for ions and for neutral atoms. This allows assessing the impact of ionization in the effusion path and extraction efficiency.
- *Fitting of events* to a release function (see section 1.9.1)
- *Estimated release fractions* for given parameters and for conditions other than the ones specified, i.e. different sticking times and diffusion time constants (explained in section 1.9.2).

- *Average effusion time in each cell.* It permits to see which elements are slowing down effusion.
- Average number of collisions to each surface. This array provides clues on the potential chemical selectivity of each surface, or on its surface ionization power.
- *Statistics of the free flight:* average distance between two consecutive collisions, average flight path from birth to ionization or up to extraction. . .
- *Statistics of the effusion in a powder or fiber (if present):* average free flight path, number of collisions. . .
- Report of the effect of the residual gas: average free path between two collisions with a gas atom and average number of collisions per history.
- *Summary of the ionization scores:* ionization probability, estimate of the ionization efficiency and error margin.
- *Module integrating computation times.*
- *Authoring.*

Some modules deserve a dedicated explanation. For all the rest examples will illustrates a complete output file presenting almost all modules.

1.9.1 Fitting of events to a release function.

Unless otherwise specified, the code prints at run time, the starting (x,y,z) and final conditions (diffusion time td , effusion in powder tPo , effusion elsewhere tf , number of collisions in the powder $COLP$, number of collisions elsewhere COL ,

and ionic state) of each history. If the user wanted to build up a release distribution function out of the data, then the histogram option could be of help. However, the ideal histogram binning depends of the distribution itself (unknown at the beginning) and of the quality of the results (statistics), and therefore fitting the histogram to a smooth function may then become problematic.

Indeed, very often the ultimate goal is to obtain an analytic function that describes the release distribution probability (intrinsic release function), $R(t)$. Thus, it may be useful to deduce this function without having to go through the cumbersome procedure of producing a histogram of the data and fitting a curve to it.

RIBO contains a release function fitting functionality that is based on the so-called “statistical momenta” of the distribution. The method is the following. For every history the code updates a variable that contains the, e.g., average ‘flight time’, $\langle tf \rangle$. For every new particle n this figure can be updated in this way:

$$\langle tf \rangle_n = \frac{n-1}{n} \cdot \langle tf \rangle_{n-1} + \frac{tf_n}{n} \quad (1.1)$$

Thus, the code can compute $\langle tf \rangle$ without need of storing tf_1, tf_2, \dots, tf_n . The same can be done for the higher order momenta, $\langle tf^2 \rangle, \langle tf^3 \rangle \dots$ and for other magnitudes, like $tPo, tf + tPo, \dots$

Now, if we believe that the release function behaves simply as a decaying exponential $f_1(t) = C \cdot \exp(t/t_1)$, we can find the value of C and t_1 easily by computing the momenta of $f(t)$ and equalizing to those of the real distribution.

Moreover, we impose that the distribution is normalized. The parameters thereafter obtained are printed in the output file, for example, for the flight time¹¹⁷, a possible result can be:

```

1) T = FLIGHT_TIME_IN_VACUUM -----
M1      M2      M3      M4      M5
0.2379  0.3249  0.4003  0.4643  0.5182

Fit to T(i,t) = exp(-t/t1)/t1:
t1= 0.23786
error O(2)[%]: -7.17366
error O(3)[%]: -25.86393
Fit to T(i,t) ~ (1-exp(-t/t1))*exp(-t/t2):
t1= 0.04672
t2= 0.19999
error O(3)[%]: 7.83485
error O(4)[%]: -1.87617

```

The meaning of the results is:

† $M1, \dots, M5$ are the five momenta for the free flight time ($\sqrt[n]{\langle t_{flight}^n \rangle}$) obtained from the distribution. In this category the flight time spent within a powder or felt mesh is not considered (it is counted in “2) T = FLIGHT_TIME_IN_POWDER ...”).

*Statistical
momenta*

† A first fit to a simple decaying exponential $f_1(t_1)$ is done, the fit parameter being t_1 .

† `error O(2) [%]` is the relative error between the predicted M2 (from f_1) and the real one. Idem for `O(3)`

† Idem for the second function $f_2(t_1, t_2)$.

† Diffusion is not included here because its function is in principle known [14].

¹¹⁷The same is done for the *flight time in the powder/fiber* (if any) and for the *total effusion time*.

1.9.2 Computation of release fractions.

The individual delays for every history are printed in the output file, as well as the particle state (atom or ion, extracted or absorbed) and starting coordinates, so that the user can compute the the release fraction as a function of whichever parameter, e.g. diffusion time constant, sticking time, starting z coordinate ...

Moreover, the output file includes a set of tables with the estimated release fraction for the input parameters (diffusion coefficient, sticking time) for different half-lives as well as for a variety of other diffusion and sticking time constants.

Briefly, the way to obtain these values, consisted in recording on-line a covariance matrix with the statistical momenta of the *effusion flight time* ($tf + tPo$) and of the *number of collisions* ($COLP + COL$) (and crossed terms) up to degree 3. Then, for any sticking time t_s , a 2-exponential effusion release function like $f_2(t_1, t_2; t)$ could be fitted (let us now call it $E(t_1, t_2, t_s; t)$) in the way described in 1.9.1. The global release function $R(t)$, would result from folding diffusion¹¹⁸ and effusion:

$$R(t) = D(\tau_D, geometry; t) \otimes E(t_1, t_2, t_s; t) \quad (1.2)$$

And the release fraction for a half-life $T_{1/2}$ would be:

$$RF(T_{1/2}) = \int_0^t R(t) \cdot \exp(-t/(\ln(2) \cdot T_{1/2})) \cdot dt \quad (1.3)$$

This is equivalent to making the Laplace transform \mathcal{L} of RF:

$$RF(T_{1/2}) = \mathcal{L}(RF(t)) \circ \left(s = \frac{1}{\ln(2) \cdot T_{1/2}} \right) \quad (1.4)$$

¹¹⁸Which depends on the diffusion time constant τ_D and the geometry (foil, fiber, particle)

The key point is that the Laplace transform of a convolution of two functions ($D \otimes E$) is the product of the Laplace transforms of each function, so the global release fraction RF is the product of the diffusion release fraction DRF and the effusion release fraction ERF :

*The release
efficiency
of Diffusion
and Effusion*

$$RF(T_{1/2}) = DRF(T_{1/2}) \cdot ERF(T_{1/2}) \quad (1.5)$$

RIBO computes $DRF(\tau_D, geometry, T_{1/2})$ and $ERF(t_1, t_2, t_s; T_{1/2})$ from the analytic functions and it prints out the results in tables. For example:

		----- ts (s) -----						
tao_d[s]		0.10E+0	0.0E+0	0.1E-8	0.1E-7	0.1E-6	0.1E-5	0.1E-4

T_1/2[s]	DRF[%]	RF(ts1)	RF(ts2)	RF(ts3)	RF(ts4)	RF(ts5)	RF(ts6)	
0.1E-2	7.727	0.000	0.000	0.000	0.000	0.000	0.000	
0.3E-2	13.126	0.007	0.007	0.007	0.006	0.003	0.000	
0.6E-2	18.198	0.036	0.036	0.035	0.031	0.014	0.001	
0.1E-1	23.022	0.116	0.116	0.114	0.102	0.046	0.003	
0.3E-1	37.213	1.214	1.212	1.201	1.096	0.552	0.046	
0.6E-1	48.936	4.329	4.326	4.294	3.995	2.247	0.225	
0.1E+0	58.506	9.598	9.592	9.536	9.005	5.584	0.678	
0.3E+0	78.059	33.413	33.402	33.302	32.325	24.677	5.327	
0.6E+0	87.058	53.647	53.637	53.541	52.597	44.483	14.564	
0.1E+1	91.622	67.231	67.222	67.143	66.364	59.327	25.862	
0.3E+1	96.960	86.780	86.776	86.738	86.364	82.766	57.048	
0.6E+1	98.441	93.026	93.024	93.003	92.797	90.770	73.942	
0.1E+2	99.051	95.719	95.718	95.705	95.575	94.298	82.933	
0.3E+2	99.673	98.533	98.533	98.528	98.483	98.036	93.739	
0.6E+2	99.830	99.256	99.256	99.254	99.231	99.004	96.784	
0.1E+3	99.893	99.548	99.548	99.546	99.533	99.396	98.046	

3-D Diffusion module.

2.1 Brief physical introduction

The diffusion of atoms within a solid, and the conduction of heat both follow the Fick's Law:

$$\vec{J}_{\hat{n}} = a \cdot \vec{\nabla}_{\hat{n}} U \quad (2.1)$$

Where U is the concentration ($[\frac{par}{cm^3}]$) or the temperature ($[K]$) and $\vec{J}_{\hat{n}}$ is the flux of particles ($[\frac{par}{cm^2s}]$) or of heat power ($[\frac{J}{s}]$) in the direction determined by \hat{n} . The constant a corresponds either to the diffusion coefficient D ($[\frac{cm^2}{s}]$) or to the thermal conductivity k ($[\frac{W}{sK}]$).

RIBO divides the space in finite units of volume dV (boxels), the dimensions ($dV = dx_i^3$) of which are determined by the user. Each boxel will be labeled with its initial concentration (or temperature) and the cell number to which it belongs (if any). Time is discretized in time steps (the length is adjusted dynamically). Each boxel exchanges atoms (or heat) in the direction \hat{i} through the interface area $dA_i = dx_j \cdot dx_k$ that communicates to the next boxel. This is done by mass diffusion (or

1st Fick Law
solved in evenly meshed space

head conduction) flows, or by desorption (or radiation¹) if the communicating boxel belongs to vacuum.

For a given boxel i, j, k , the balance of U in the interval dt , expressed in Cartesian coordinates is:

$$dU = (g + d + s) \cdot dt \quad (2.2)$$

Where:

- g is the source term:

$$g = \begin{cases} \frac{\text{par}}{\text{cm}^3 \cdot \text{s}} & \text{for atomic diffusion} \\ \frac{J}{\text{cm}^3 \cdot \text{s}} & \text{for heat calculations} \end{cases} \quad (2.3)$$

- d is the diffusion or conduction flow term:

$$\sum_{i=x,y,z} \check{a} \frac{\{U_{i+} - 2U_i + U_{i-1}\}}{dx_i} \cdot dA_i \cdot \frac{dt}{dV} \quad (2.4)$$

The constant \check{D} is the mass or heat diffusion coefficient ($[\frac{\text{cm}^2}{\text{s}}]$). Note that the thermal diffusion coefficient is obtained in terms of other more familiar parameters:

$$\check{D} = \frac{k}{C_p \cdot \rho} = \frac{\text{conductivity} [\frac{\text{W}}{\text{cm} \cdot \text{s}}]}{\text{Specific Capacity} [\frac{\text{W} \cdot \text{s}}{\text{g} \cdot \text{K}}] \cdot \text{Density} [\frac{\text{g}}{\text{cm}^3}]} \quad (2.5)$$

The user is asked to split up the geometry in cells enclosed by quadrics. The mass diffusion / heat transfer properties of each cell must be defined by the user in the data file `[data]/Cond.dat` (see 2.2.1 on page 64).

¹Convection is not considered in this version, but it could be easily included.

- s is the surface term, which appears only in those boxels in the boundary between a cell and the vacuum. This term has the following meaning:

$$s = \begin{cases} d \cdot (1 - \exp(-dt/t_d)) \left[\frac{\text{par}}{\text{cm}^3 \cdot \text{s}} \right] & \text{atomic desorption for surfaces} \\ \varepsilon(T) \cdot \sigma_{SB} \cdot (T^4 - T_0^4) \cdot \frac{dA}{C_p \cdot \rho \cdot dV} \left[\frac{\text{K}}{\text{s}} \right] & \text{T = U, heat radiation} \end{cases} \quad \text{Boundary conditions} \quad (2.6)$$

Where, for atomic desorption t_d is the average desorption time ([s]), and for heat radiation, $\varepsilon(T)$ is the emissivity (0-1), σ_{SB} is the Stephan-Boltzmann constant ($5.6703 \cdot 10^{12} \left[\frac{\text{W}}{\text{K}^4 \text{cm}^2} \right]$), U is the temperature T ([K]) of the boxel and T_0 is the average temperature 'seen' by the boundary boxel.²

By applying the equation 2.2, the Concentration / Temperature in every boxel $C(i, j, k, 1)$ ³ can be monitored over time.

2.2 Instructions of use

In order to run a 3D (atomic or thermal) diffusion problem, the following steps have to be cleared.

1. Retrieve the conduction, desorption and/or radiation constants, transform them into S.I, except length units (in [cm]), and fill in the [data]/cond.dat file, as explained in 2.2.1.
2. Write the input file to describe the geometry of the system using combinatorial geometry (refer to sections 1.3.1, 1.3.2). Note that you must fill in all the cards and fields in the standard way for effusion calculations (*Surfaces, Cells, Source and Tally*), even if you are not going to need them all.

²The function *radia.f* is open for the user to customize the expression of the radiation as well as the temperature dependency of ε , $\varepsilon(T)$, see section 2.3.4 and example at page 96.

³ $C(i, j, k, 2)$ contains the cell index of the boxel $\{i, j, k\}$.

3. Adjust the source routines to match your specific requirements:
 - Customize the initial concentration (or temperature) distribution by modifying the file `[sources]/CSTART.f`, as explained in 2.3.1 (page 67).
 - If needed, introduce time-space-direction dependences in the diffusion parameter by editing the routine `[sources]/Dijkt.f`, printed in 2.3.3 (on page 69).
 - Edit the file `[sources]/gener.f` (page 68) to add time-space-region dependent sources (or sinks) of atom concentration (or heat).
 - For heat transfer calculations, modify the subroutine `[sources]/radia.f` (see sec 2.3.4, on page 71) to adjust the radiation function to your problem.
 - Customize the printing routine `[sources]/userPRINT3D.f` (see section 2.3.5) to print the necessary information.
4. Recompile the RIBO distribution with the script provided in `[tools]/make.sh`
5. Run RIBO and provide the right run-time options, discussed in 3.1.4.

2.2.1 `[data]/Cond.dat` file

The file `[data]/Cond.dat` contains the information about the diffusion parameter, desorption time, emissivity and volumetric specific capacity (at constant pressure). The first 6 lines of the file are explanatory and should not be removed. The code should find the valid information starting exactly in row 7. The **first column**

(i) contains the cell number (celln). The **columns 2, 3, 4** have the information on the diffusion parameter \check{D} ($\frac{cm^2}{s}$), either as the diffusion coefficient D , or the conductivity normalized to the specific capacity and to the density. In any case, the diffusion parameter can depend on the variable U (Concentration or Temperature). In every boxel (see eq. 2.2) the code will compute the diffusion parameter as:

$$\hat{D} = \hat{D}0 + \hat{D}U \cdot U + D\hat{U}U \cdot U^2 \quad (2.7)$$

Column 5 has the information on the average desorption time ([s]) for atomic diffusion or the emissivity, ε , for photon radiation. Special temperature dependence laws $\varepsilon(T)$ should be introduced in *[sources]/radia.f*. See 3.1.4.

Columns 6, 7 and 8 Contain the specific heat capacity in $[\frac{W \cdot S}{cm^3 \cdot K}]$. Three coefficients can be provided, like for \check{D} .

```
| 3-D DIFFUSION (MASS/HEAT) DATA RESPECT FORMAT !!
| D(T) [cm2/s] = D0 + DC * C + DCC * C^2; K(T) [W/cmK]= k0 + KT * T + KTT * T^2
| Cp(T) [J/Kcm^3] = Cp0 + CpT * T + CpTT * T^2; t_desorption [s]; eps = emissivity 0,1
+-----+
| i D0|K0 DC|KT DCC|KTT t_des|eps - | Cp0 - | CpT - | CpTT
+-----+
| 1 2.70 0.0 0.0 0.25 170.0 0.0 0.0
| 2 2.49 0.0 0.0 0.19 58.0 0.0 0.0
```

2.2.2 Activating 3-D diffusion calculations

The user should give the following answers to activate 3D diffusion calculations (see example in 3.1.4),

1. **inputfilename**
2. **outputfilename**

3. **cello** cell to which generation is limited. Type 0 if you don't need any restriction here.
4. **1**
5. **inmode 5** for atomic diffusion) or **6** for heat transfer.
6. **output printing mode 8** average numbers, **9** print full grid, **C** custom.
7. **Xmin Ymin Zmin** Low edge coordinates of the 3D window. Vacuum will occupy the outer space.
8. **DX DY DZ** Lengths of the sides of the 3D window box.
9. **NX NY NZ** Number of bins in each direction.
10. **Termination condition and value**

2.3 User defined functions and subroutines

The file `[sources]/custom3D.f` contains a set of functions and subroutines that can be edited to fit specific requirements in 3D diffusion problems. For the time being, the included objects are:

1. *CSTART.f* to specify the starting profile of concentration or temperature.
2. *gener.f* to define outer sources or sinks of atoms or heat.
3. *Dijkt.f* to customize the diffusion coefficient (spatial or time dependence).
4. *radia.f* to adapt the radiation function.
5. *userPRINT3D.f* to determine the entities to be printed.

2.3.1 Customizing the starting distribution of Concentration or Temperature, *CSTART.f*

```

FUNCTION CSTART(R, celln ,inmode)
  real*8 R(3),CSTART           ! i
  character*12 inmode         ! i
  integer*4 celln             ! i
  real*8 C0                   ! a
  4
  *****
  *          CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM          *
  *          USAGE: 3D diffusion problems                          *
  * The starting Concentration / Temperature may depend on the coordinates *
  *   e.g.  C0 = 273                                               *
  *          C0 = 273 * ( R(1) + R(2) )                            *
  * It can also depend on the material (celln), or on both        *
  * Variables:                                                    *
  *   - R(3): absolute coordinates, x,y,z [cm] of the volume boxel *
  *   - CSTART: Starting concentration / Temperature              *
  *   - celln: number of the cell                                 *
  *   - inmode: (5) ==> mass diffusion problem (6) ==> heat transfer *
  *****
  5
  6
  IF ( celln.ge.1) THEN
    IF (inmode.eq.'5') THEN
      C0 = 1.0
    ELSE IF (inmode.eq.'6') THEN
      C0 = 500.0 ! starting temperature
    END IF
  ELSE           ! outer space
    IF (inmode.eq.'5') THEN
      C0 = 0.0
    ELSE IF (inmode.eq.'6') THEN
      C0 = 298.0
    END IF
  END IF
  CSTART = C0
  7
  8
  END
  9

```

2.3.2 Defining source and / or sink terms, *gener.f*

```

FUNCTION gener(R,C0,t,PAR,Dbq,ncell,celln,inmode)
  integer*4 ncell,celln                                ! i
  character*12 inmode                                  ! i
  real*8 R(3),C0,t,PAR(5),Dbq(7,ncell)                ! i
  real*8 dV,CpRho                                     ! a
  real*8 gener                                         ! o
*****
*   Customize this function to include heat or mass sources   *
*   that depend on the position X, or/and time, or/and       *
*   the concentration, celln, emissivity, ...               *
*****
*   VARIABLES:                                           *
*   R(3) = coordinates (x,y,z)[cm] DO NOT MODIFY VALUE!    *
*   C0 = Concentration [par/cm^3] | Temperature [K]        *
*   t = elapsed time [s]                                    *
*   PAR(1) = dV = dx * dy * dz [cm^3] Useful to normalize generation *
*   PAR(2) = concentration/temperature at point X [cm^-3 | T] *
*   PAR(3) = celln                                         *
*   PAR(4) = t_desorption/emissivity [s | W/cm^2*K^4]      *
*   PAR(5) = dt [s]                                       *
*   inmode = 5 mass diffusion | 6 heat transfer            *
*   -----                                                 *
*   Dbq(1) = D0 [cm^2/K^0 s] | k0 [W/cm K^1]               *
*   Dbq(2) = DC [cm^2/K^1 s] | kT [W/cm K^2]              *
*   Dbq(3) = DCC [cm^2/k^2 s] | kTT [W/cm K^3]             *
*   Dbq(4) = t_s [s] | Eps [W/K^4]                        *
*   Dbq(5) = - | Cp0*rho [W s/cm3 K^1]                     *
*   Dbq(6) = - | CpT*rho [W s/cm3 K^2]                    *
*   Dbq(7) = - | CpTT*rho [W s/cm3 K^3]                   *
*   ncell = number of cells of the geometry                *
*   .....                                                 *
*   CpRho = Cp*Rho [W s/K cm^3]                            *
*****
*   UNITS of gener                                         gener *
*   For MASS TRANSFER: dV                                   *
*   need : par / (cm^3 * s) -----> par / s               *
*   For HEAT TRANSFER: (dV/CpRho)                          *

```

```

*      need :  J / (cm^3 * s)  ----->  K / s      *
*  EXAMPLES:      *
*      1) constant generation of particles , 1E5 (par/cm3 s)      *
*      gener [par/s] = 1E5 * dV      *
*      2) constant heat generation: (1kW/cm^3)      *
*      ==> gener [K/s]  = 1000 * dV / (CpRho)      *
*****
*  NOTE:      *
*      The generation may depend on time and on X, ...      *
*****
      dV = PAR(1)  ! [cm^3]
      gener = 0.0
      IF (inmode.eq.'5') THEN      ! 3D mass diffusion
*      gener = 1E4 * dV      ! [par/s]
      ELSE IF (inmode.eq.'6') THEN ! 3D heat transfer
      CpRho = (Dbq(5, celln))
$      + (Dbq(6, celln) * T)
$      + (Dbq(7, celln) * T * T)
      IF (celln.eq.1) THEN
*  EXAMPLE 10 kW / cm^3 deposited in cell = 1:
      gener = 10000.0 / CpRho  ! [W/cm^3]-->[K/s]
      END IF
      END IF
      END

```

2.3.3 User defined mass/heat diffusion coefficients, *Dijkt.f*

```

FUNCTION Dijkt(I,X,t,N,C,Dbq,ncell,inmode)
      integer*4 N(3),I(3),ncell      ! i
      character*12 inmode      ! i
      real*8 C(300,300,300,2),X(3),Dbq(7,ncell),t      ! i
      integer*4 celln      ! a
      real*8 CorT,CpRho,k      ! a
      real*8 Dijkt      ! o
*****
*      CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM      *
*****

```

```

*                               written by MSL, 2006                               *
*****
*   This function computes the DIFFUSION coefficient | CONDUCTIVITY           *
*   for a given boxel (that corresponds to position X), and depending         *
*   on the CONCENTRATION | TEMPERATURE, parsed constants Dbq and cell       *
*   number                                                                    *
*   VARIABLES:                                                                  *
*   case          (MASS) transfer          (HEAT) transfer                    *
*   -----
*   inmode:       = 5                      |          =6                      *
*   CpRho:        |          Cp*Rho [W s/K cm^3]                             *
*   k:            |          conductivity [W/K cm]                           *
*   Dijkt: Diffusion coeff.(D)[cm^2/s] | k/(Cp*Rho) [cm^2/s]                 *
*   C...:        {concentration, celln} | {Temperature, celln}                 *
*   Dbq:         diffusion constants     | conductance/emiss. (Cond.dat)      *
*   t:           elapsed time [s]                                             *
*   N(3):        number of divisions in {x,y,z}                               *
*   I(3):        boxel identifier {i,j,k}                                     *
*   X(3):        position of center of boxel {x,y,z}                         *
*   ncell:       total number of regions in geometry                         *
*   -----
*   NOTES:
*   A quadratic law D(C) and k(T) are assumed, but other laws can be
*   implemented in this function
*****
*   The user may introduce fancy features now, for example:
*   1) Time dependency of diffusion coefficient
*       Dijkt = ... * sin(t)
*   2) X-dependency of diffusion coefficient
*   ...
*****
      celln = NINT(C(I(1),I(2),I(3),2))
      IF (inmode.eq.'5') THEN
        IF (celln.eq.0) THEN
          Dijkt = 0.0
        ELSE
          CorT = C(I(1),I(2),I(3),1) ! Concentration
          Dijkt = (Dbq(1,celln))
$          + (Dbq(2,celln) * CorT)

```

```

$          + (Dbq(3, celln) * CorT * CorT)
END IF
ELSE IF (inmode.eq.'6') THEN
  IF (celln.eq.0) THEN
    Dijkstra = 0.0
  ELSE
    CorT = C(I(1),I(2),I(3),1) ! Temperature
    k = (Dbq(1, celln))
$          + (Dbq(2, celln) * CorT)
$          + (Dbq(3, celln) * CorT * CorT)
    CpRho = (Dbq(5, celln))
$          + (Dbq(6, celln) * CorT)
$          + (Dbq(7, celln) * CorT * CorT)
    Dijkstra = k / CpRho
  END IF
END IF
END

```

2.3.4 Customized heat radiation (and convection) function, *radia.f*

```

FUNCTION radia(i,M,T,C,N,R,PAR,Dbq,dA,ncell)
  integer*4 i,ncell,N(3) ! i
  real*8 M(3),T,PAR(5),dA(3),Dbq(7,ncell),C(300,300,300,2),R(3) ! i
  real*8 emissivity,sigSB,eps,T0,dV,radia,CpRho,norm ! a
  integer*4 celln ! a
  *****
  * THIS SUBROUTINE checks if we are in a heat radiation problem *
  * (inmode=6) and, if so, it computes the radiated power Qr [W] *
  * as a function of the emittance (eps), the exposed area dA(i) *
  * and the temperature difference (T=C0,T0=external) *
  * VARIABLES: *
  * i = exposed area normal vector {i,j,k} *
  * M(3) = material of neighbouring cell (should be vacuum, M=0) *
  * T = Temperature of the boxel [K] *
  * C... = Temperature{i,j,k,celln} *
  * N(3) = number of divisions in {x,y,z} *

```

```

*      X(3) = position of center of boxel {x,y,z} *
*      PAR(1) = dV = dx * dy * dz [cm^3] Useful to normalize generation *
*      PAR(2) = concentration/temperature at point X [cm^-3 | T] *
*      PAR(3) = celln *
*      PAR(4) = t_desorption/emissivity [s | W/cm^2*K^4] *
*      PAR(5) = dt [s] *
*
*-----*
*      Dbq(1) = D0 [cm^2/K^0 s] | k0 [W/cm K^1] *
*      Dbq(2) = DC [cm^2/K^1 s] | kT [W/cm K^2] *
*      Dbq(3) = DCC [cm^2/k^2 s] | kTT [W/cm K^3] *
*      Dbq(4) =          t_s [s] | Eps [W/K^4] *
*      Dbq(5) =          - | Cp0*rho [W s/cm3 K^1] *
*      Dbq(6) =          - | CpT*rho [W s/cm3 K^2] *
*      Dbq(7) =          - | CpTT*rho [W s/cm3 K^3] *
*
*.....*
*      dA(3) = dA(i) is the perpendicular area for normal i [cm^2] *
*      CpRho = Cp*Rho [W s/K cm^3] *
*      T0 = ambient temperature [K] *
*      sigSB = stephan-Boltzmann constant = 5.6703E-12 [W/cm^2*K^4] *
*****
*      CUSTOMIZATION *
*      - replace eps by a function of temperature eps(T) *
*      - replace T0 by the ambient temperature or by the temperature *
*        of enfolding cells , e.g T0=C(10,4,9,1) *
*****
*      celln = NINT(PAR(3))
*      IF (M(i) .lt .1E-14) THEN
*          CpRho = (Dbq(5, celln))
*      $          + (Dbq(6, celln) * T)
*      $          + (Dbq(7, celln) * T * T)
*
*      dV = PAR(1) ! [cm3]
*      sigSB = 5.6703E-12 ! [W/K^4 cm^2]
*      emissivity = PAR(4) ! [-]
*      eps = sigSB * emissivity ! [W/K^4 cm^2]
*      T0 = 298.0 ! [K] ambient temperature
*****
*      EXAMPLE: heat3D.t *
*      - The solid W sphere sees the inner side of the outer Ta spheric *
*        shell. C(10,90,90,1) corresponds to the temperature in a point *

```

```

*      of the Ta inner side. The spheric symmetry allows to leave that *
*      same reference point for whatever position                      *
*      - The Ta spheric shell (celln=2) sees, in the inside, the W    *
*      sphere C(31,90,90,1) and, in the outside, the open air, T=298 *
*****
      IF (celln.eq.1) THEN      ! W shpere
          T0 = C(10,90,90,1)    ! Temperature of inner face of Ta shell
      ELSE IF (celln.eq.2) THEN ! Ta shell
          IF (norm(R).le.4.40) THEN ! inner face
              T0 = C(31,90,90,1) ! ==> ~ Surface temperature of the W sphere
          ELSE
              T0 = 298.0        ! ==> Sky temperature
          END IF
      END IF
      radia = eps * ((T0**4)-(T**4)) * dA(i) / (CpRho*dV)
*****
*      EXAMPLE 2  CONVECTION
*      - Note that you can also include convection
*      First define the variable "conv" and the parameter "constant"
*      - Uncomment these lines:
*      conv = constant * (T0 - T)
*      radia = eps * ((T0**4)-(T**4)) * dA(i) / (CpRho*dV) + conv
*****
      ELSE
          radia = 0.0
      END IF
  END

```

2.3.5 Customized the printout values, *userPRINT3D.f*

```

SUBROUTINE userPRINT3D(t,N,C,C0,nbin,Cmax,Imax,Rmax,cellmax,
$      Cmin,Imin,Rmin,cellmin,sum,sum0)
      integer*4 N(3),Imax(3),Imin(3),cellmax,cellmin      ! i
      real*8 C(300,300,300,2),C0(300,300,300,2),Cmax,Cmin,nbin ! i
      real*8 t,sum,sum0                                     ! i
      real*8 Rmax(3),Rmin(3),fmax,fmin,Cav,fav           ! i
      integer*4 i                                          ! a

```



```

logical fileNOTyetOPEN ! a
*****
*
*          CUSTOMIZE THIS FUNCTION TO FIT YOUR PROBLEM          *
*
*****
*
*          written by MSL, 2006
*
*****
*
*  VARIABLES:
*
*    - t:      Elapsed time [s]
*
*    - N(3):   Number of bins in each direction x,y,z
*
*    - C(300,300,300,2): two values for each voxel
*
*          C(300,300,300,=2): material index (region number)
*
*          C(300,300,300,=1): Concentration | Temperature
*
*    - C0(300,300,300,2): stores the values of C at t=0
*
*    - Cmax(Cmin): Highest(Lowest) Concentration | temperature
*
*    - Imax(Imin): Pointers {i,j,k} of the boxel with Cmax(Cmin)
*
*    - Rmax(Rmin): Coordinates of the boxel of Cmax(Cmin) {x,y,z} [cm]
*
*    - cellmax(cellmin): the boxel of Cmax(Cmin) belongs to celln...
*
*    - nbin:   number of boxels belonging to a cell
*
*    - sum:    sum of concentrations/temperatures at time t
*
*    - sum0:   sum of concentrations/temperatures at time 0
*
*****
*
*  EXAMPLE 1
*
*    - Print the evolution of the matrix C (lot of memory!!):
*
**
*
*    DATA fileNOTyetOPEN / .TRUE. /
*
*    IF ( fileNOTyetOPEN ) THEN
*
*      fileNOTyetOPEN = .FALSE.
*
*      OPEN(UNIT=26,FILE='26.out')
*
*      write(6,*)'Creating file unit 26'
*
*    END IF
*
**
*
*      write(6,*)C
*
*****
*
END $

```

2.4 Examples of use

Please, check the dedicated example discussed in chapter 3.1.4.

Examples of RIBO use.

3.1 Examples.

3.1.1 First example. Geometry issues.

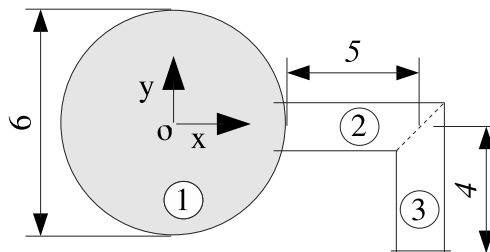


Figure 3.1: A simple example consisting on a bulb full of He at 300 K connected to a pipe of 1 cm diameter with a bend of 90° .

The first example consists on bulb full felt with He at 300 K and connected to a pipe of 1 cm diameter with a bend of 90° . Dimensions are shown in fig.3.1. It is indeed a simple geometry, but attention has to be paid to define the correct subspaces unambiguously. For that sake auxiliary planes are needed. In fig.3.2. surface 2 is an auxiliary plane that will help to define the second cell, if it was not used, the program could not logically decide between the real cell and the dotted one. In fact, it would allow transmission of particles both to the right and to the left. The same is valid for the crossing of pipes at the surface 4. The *Surfaces* card

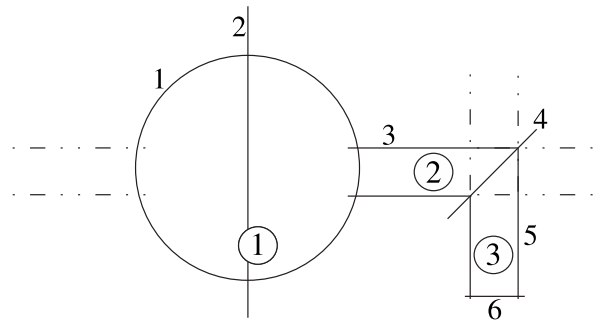


Figure 3.2: Surfaces and cells for the first example. Auxiliary surfaces are needed to define the correct subspaces.

would be something like this:

Surfaces												
n	rc	T	x2	y2	z2	xy	xz	yz	x	y	z	C
1	0.5	300	1	1	1	0	0	0	0	0	0	9.0
2	0.5	300	0	0	0	0	0	0	1	0	0	0.0
3	0.5	300	0	1	1	0	0	0	0	0	0	0.25
4	0.5	300	0	0	0	0	0	0	1	-1	0	8.0
5	0.5	300	1	0	1	0	0	0	-16	0	0	-63.75
6	0.5	300	0	0	0	0	0	0	0	1	0	-4.0

And the *Cells* card would read as:

Cells				
n	S1	S2	S3	S4
1	-1	0	0	0
2	1	2	-3	4
3	-4	-5	6	0

The *Source* card, for a homogeneous distribution in the sphere should be:

Source									
Type	M	T	Alpha	nx	ny	nz	x	y	z
S	4	300	180	1	0	0	0	0	0

3.1.2 Second example. Bigger files.

A more realistic example for the field of radioactive ion beams is that of a target made of thin foils. Fig.3.3 sketches a target with a SPIRAL Christmas-tree-like shape [1, 2], intended to dissipate the energy of the beam in steps. Surfaces 1, 2 and 3 are planes and 4 and 5 cylinders; they delimit the target container and the transfer line (cells 1 and 2, respectively). The sixth surface is a cone and the remaining surfaces 1 ÷ n are planes; these surfaces enclose the target slabs (of thickness d) and the spacing between them (δ between two consecutive front faces). The *Surfaces* card would look something like this (the symbols L1,

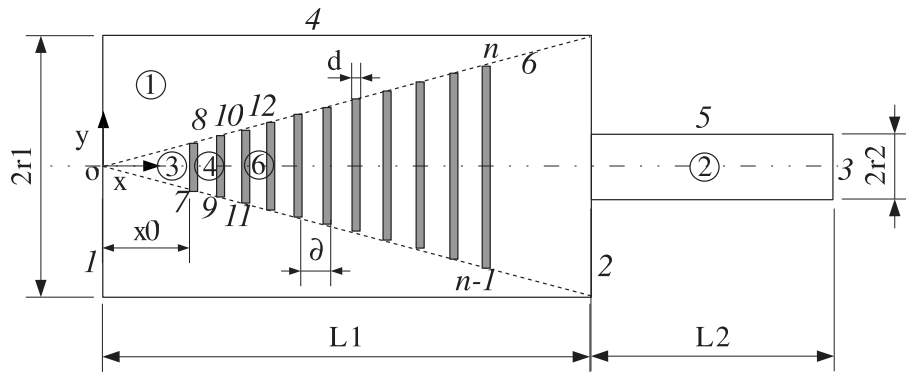


Figure 3.3: Sketch of a GANIL-SPIRAL type target [1, 2].

L2... should have to be replaced by numeric values):

Surfaces													
n	rc	T	x2	y2	z2	xy	xz	yz	x	y	z	C	
1	0.5	300	0	0	0	0	0	0	1	0	0	0	plane
2	0.5	300	0	0	0	0	0	0	1	0	0	L1	plane
3	0.5	300	0	0	0	0	0	0	1	0	0	L2	plane
4	0.5	300	0	1	1	0	0	0	0	0	0	r1^2	big cylinder
5	0.5	300	0	1	1	0	0	0	0	0	0	r2^2	small cylinder
6	0.5	300	$-(r1/L1)^2$	1	1	0	0	0	0	0	0	0	containing cone
7	0.5	300	0	0	0	0	0	0	1	0	0	x0	1st foil front
8	0.5	300	0	0	0	0	0	0	1	0	0	x0+d	1st foil back
9	0.5	300	0	0	0	0	0	0	1	0	0	x0+dx	2nd foil front
10	0.5	300	0	0	0	0	0	0	1	0	0	x0+d+dx	2nd foil back

⋮

3

4

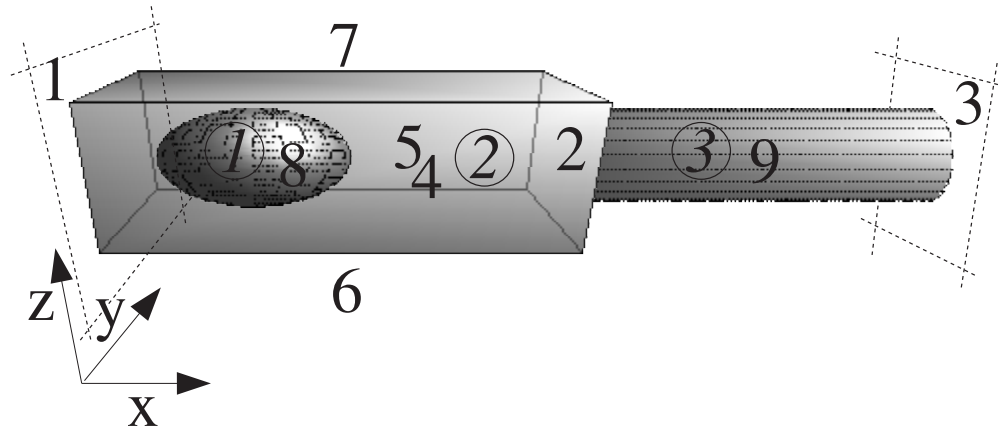


Figure 3.4: The target and ion source of example 3.

8	0.5	298	0.25	1	1	0	0	0	0	0	0	0.25	7
9	0.5	2000	0	1	1	0	0	0	0	0	0	0.249	
Cells													
#	S1	S2	..										
1	-8	0	0	0	0	0	0	0	0	0	0	0	8
2	8	1	-2	4	-5	6	-7						
3	2	-3	-9	0	0	0	0						
Source													
Type	M	T	Alfa	nx	ny	nz	x	y	z	R			
S	7	298	180	-1	0	0	0	0	0	0.999			9
Tally													
S	max	Tmax	Tpmax										
3	300	750	10										

The primary source that is used here is a sphere, the radius of which equals the maximum radius of the ellipsoid. The cell-limited option will be chosen at runtime.

Runtime options

The sequence of interactions with the program at runtime is screen printed below. Some comments are interpolated:

The ion source may span over multiple cells

```

...
-----
|   RRR           II           BBB           OO           |
|   R R           II           B B           O O           |
|   RRR  adio-    II  on  BBB  eam  O O  ptimi-    |
|   R R  active  II           B B           O O  ser    |
|   R R           II           BBB           OO           |
-----

                                2000-2006 MSL

                                WELCOME !!

*****
** Program Licensed exclusively to:
**   READER
*****
version 30-06-06
full version

Name of the input file?
test.inp
ifile=test.inp
Name of the output file?
(Beware it will overwrite this file!)
test.out
READING INPUT FILE test.inp           ...
  1) reading surfaces ...
  2) reading cells ...
----- number of cells: 3
----- highest number of walls in a cell: 7
----- number of bodies: 9
.....
...reading tally
3           300       750       10
Nmax,tmax,tpmax 300  750.  10.
reading geometry ...
...storing surfaces (bodies)...
    ...storing cells (regions)...
INPUT FILE HAS BEEN READ SUCCESSFULLY

```



```

| SOURCE CUSTOMIZATION: |
| - You can edit the file [sources]/customsource.f |
|   and then recompile with [tools]/make.sh |
| - Additionally you can use the data file init.dat |
|   by uncommenting "read..." in customsource.f |
| - And you can restrict generation to a cell... |
|   ...source limited to a cell? give cell number. |
|     celln > 0 ==> generation limited to volume |
|         defined by celln |
|     celln = 0 ==> do not constrain to a cell |
|     celln < 0 ==> just generate a geometry plot |
|-----|
1
|-----|
| SPECIAL EVENTS, TERMINATION |
| Please choose among these options: |
| 1: Crossing of end surface (1st card in "Tally") |
|   No ionizations in the system. |
| 2: Atoms can be ionized in a PLASMA ion source |
|   Histories (of atoms and ions) end when |
|   they cross the end_surface detector |
| 3: Atoms can be ionized in a SURFACE ioniser |
|   Histories (of atoms and ions) end when |
|   they cross the end_surface detector |
| 4: Atoms can be absorbed in the walls. Trajec- |
|   tories end at absorption or when crossing |
|   the end surface |
| So, what do you want, 1, 2, 3 or 4? |
|-----|
3
how many ionising cells? max is 3
1
cell number of ion cell number 1
3
how many ionising surfaces in cell 3
1
Ionizer surface number 1 in ion cell 1
9

```

After giving the name of the input and output file, we asked to restrict the source to cell number 1 (the ellipsoid) and activated simulations with a surface ioniser. The ionising surface is the cylinder 9, belonging to the cell 3, the extraction tube

```

-----
|
|   SSSS U  U RRR  FFFF II  OOOO N  N  |
| S      U  U R  R F    II OO  OO NN N  |
|   SSS  U  U RRRP FFF  II  O   O N N N  |
|     S UU UU R  R F    II OO  OO N NN  |
| SSSS   UUU  R  R F    II  OOOO N  N  |
|-----|
|
| This program computes the probability of sur- |
| face ionization after each single atom-surface |
| collision. Theory summarized in KOE2000 p.223> |
| is used. Parameters are the ionization work |
| function and ionic statistical weights, Wi,g0, |
| g+,g- for an atom Z. They are stored in the |
| outer database sion.dat. The substrate work- |
| function is stored in workf.dat. |
|
|   written by Mario Santana Leitner. CERN 2003 |
|-----|
Nmax1, Nmax2= 102 27
atom:
  element (Z)? (0 if you want to define parameters)
3
substrate:
  element (Z)? (0 if you want to define parameters)
74
  0: element ; 1: Boride ; 2: Carbide ; 3: Oxide ; 4: CeCompound
0
  gplus , gzero , gminus , Wf, Wi, Ae  1.  2.  1.  4.54  5.39  0.62
T?[K]
2000
positive surface ionization
alphaS= 0.00360563289

```

```

negative surface ionization
alphaS= 6.61641972E-11
betaS= 0.003592679
----- ELECTROMAGNETIC ION TRANSPORT -----

You are dealing with ions, which "see" the EM-
fields, external or internal (plasma fields, ...)
IF you DO NOT know these fields, but you have a
rough idea about the extraction efficiency of the
ions from the source, you can give this number
as well as the estimated extraction time [ms]
e.g: 0.3 1E-3 would mean that 30 % of the
initially ionised atoms, make it to the outlet
as ions. The rest recombine. The process takes
about 1 microsecond in all.
IF you know these fields, you can edit explicitly
transport the ions, by editing the functions:
readEfield.f, Bn.f functions, emittance.f
AND answering -1 0 now.
.....
Ion extract. efficiency [0-1], extract. time [ms]?
0.2 1E-3

```

SURFION starts and we type the atomic number of the effusing atoms and of the ioniser substrate. We select the temperature of the surface and its composition (pure chemical element), and SURFION determines that the most likely event is positive surface ionisation and prints out the corresponding individual probabilities. Then, the module for the IONIC TRANSPORT asks whether we want an explicit EM transport (then we would need to customize the routines readEfield, Bn.f... printed in section 1.8.4) or rather we will estimate the extraction with an average extracting efficiency and an associated extracted time. We opt for the second and introduce the two values.

```
SELECT MODE:
```

```

1: Diffusion.
2: Effusion.
3: Diffusion+Effusion.
recommended 2
4: Conductance calculator (Clausing Coefficient).
5: Diffusion3D. — ALPHA VERSION !!
6: Heat transfer (cond.+rad.). — ALPHA VERSION !!

```

2

```

| CHOOSE THE OUTPUT MODE: (standard is 3) |
| 1: Only average figures |
| 3: Print all relevant events (standard) |
| 4: Like 3 + individual desorption times |
| 5: Effusion time AND velocity direction |
| 6: Particle tracking options (x, cell, surf.) |
| 7: Only events in the POWDER/FIBER |
| C: Use [sources]/userPRINT.f routine |

```

3

```

| ..... COLLISION MODEL I ..... |
| What model for the treatment of the collisions? |
| S: Specular(E<<) |
| B: B=Knudsen-Lambert (recommended cosine law) |
| D: Debye, semi-classic (under development) |
| C: Custom |
| ..... |
| BACK: Go back to previous menu |

```

```

| For option "C", Custom, modify the subroutine |
| [sources]/customcollision.f at your convenience |
| and recompile using [tools]/make.sh |

```

B

```

| ..... COLLISION MODEL II ..... |
| Use average energy or sample from coll. law |

```

```

|   Y: Use average energy (recommended)   |
|   N: Sample from collision law (slower)  |
|   .....                               |
|   BACK: Go back to previous menu       |
|-----|
Y
|-----|
| ..... COLLISION MODEL III ..... |
| Sticking time[s]?                |
| > 0 ==> every collision delays exactly ts [s] |
| = 0 (recommended). The output file includes a |
| post-analysis with several hypothetical ts |
| Use 0 for noble gases                |
| < 0 ==> sample from law P(t)=exp(-t/ts)/ts |
| It slows down calculations. Only necessary |
| if the number of collisions is low      |
| .....                               |
| For ts=ts(X,surface), see examples 4, 5 in the |
| user routine customcollision.f        |
|-----|
0
...Reading the "source" settings...

```

We then select to run in mode 2 (standard option), to have the standard individual output 3, and choose the normal collisions (cosine law, thermalization without sampling and no initial sticking time).

```

...Reading the "source" settings...
-----
|
|           M  M  EEEEE  SSSS  H  H
|          MMMM E      S    H  H
|         M M M  EEEE   SSS  HHHHH
|          M  M  E      S  H  H
|          M  M  EEEEE  SSSS  H  H
|-----|
— STARTING MODULE FOR e/dIFFUSION IN POWDER —

```

```

How many cells contain powder? e.g. 2
1
what is the cell number of the powder cell # 1
1
v= 949.3955
mean free path in powder/fiber[um]?
  e.g. UC powder 15, ZrO2 fiber 250
15
sphere probe radius[um](step path)? e.g. 800
  recommended: 6 x mean free path
90
sampling a limiting residence time in powder...
DONE, tmax_sphere_of_powder[ms]= 0.0110181198
Characterization of macrocollisions in powder:
  Generating time and angle probability functions
  ... this may take some minutes ...
  ----- Most likely macro step -----
pmax= 7427. ( 0.7427 %)
bin: 1 6 12 6 tm[s], thitax , alphav , thitav [deg]
6.05996603E-07 123.75 345. 123.75
-----
DONE

```

Next, we specify that there is only one region containing powder, cell number 1. We give the mean free path in the powder and the size of the macro steps. MESH samples 100000 macro-steps to fill a 4-d grid that will serve to sample paths within the powder.

```

-----
| ..... COLLISION WITH RESIDUAL NUCLEI ..... |
| Residual pressure [torr]? (0.75 torr = 100 Pa) |
| P <= 0 ==> molecular flow (ideal vacuum) |
| P > 0 ==> collisions with residual gas |
| (if you do not know it but you know the mean |
| free path type anything > 0 now) |
-----
1E-4
0.0001
-----
|

```

```

| RRRR EEEEE SSSS GGGG AAA SSSS |
| R R E S G A A S |
| RRRR EEEE SSS G GG AAAAA SSS |
| R R E S G G A A S |
| R R EEEEE SSSS GGG A A SSSS |
|-----|
- STARTING MODULE FOR INTERACTIONS BETWEEN ATOMS -
|-----|
| ..... COLLISION WITH RESIDUAL NUCLEI II ..... |
| INTRODUCE A NUMBER n, IF ... |
| n > 0 ==> n = ATOM DIAMETER [pm] |
| - Some indicative values [pm] are: |
| He:62;Ne:76;Ar:142;Kr:176;Xe:216;Rn:240;N2:374 |
| - Consider also diameter  $\bar{r} = 2 \times 1.4 \times A^{1/3}$  |
| n < 0 ==> |n| = -MEAN.FREE.PATH [cm] in "vacuum" |
| n = 0 ==> RIBO will estimate the MEAN.FREE.PATH |
|-----|
0
Free mean path between collisions with gas [cm],l,T,d,P:
3587.8078 298. 44. 0.0001
DONE.
DONE
READING data/valves.dat file for moving walls
no valves active in the system
Post-processing geometry...
DONE
END OF INITIALIZATION. SIMULATION STARTS!!
RUNNING...
N,Nion,Nabs,<dist>[m] 50. 8 0 1.15077824
N,Nion,Nabs,<dist>[m] 100. 16 0 1.30160217
N,Nion,Nabs,<dist>[m] 150. 22 0 1.42556459
N,Nion,Nabs,<dist>[m] 200. 32 0 1.37736645
N,Nion,Nabs,<dist>[m] 250. 36 0 1.34756701
N,Nion,Nabs,<dist>[m] 300. 41 0 1.37057745
closing...
... fitting release function to statistical momenta
DONE. Results are stored in: test.out

```

Finally, RESGAS asks us for the residual pressure. Normally we would say 0. Here, however, we say it is $1E-4$ torr. We ask the code to compute the free mean path from the pressure. Then RIBO ends up the preparatory phase and starts the simulation. At the completion of a bunch of 50 particles it prints to screen the number of simulated particles, the number of extracted ions, the number of surface absorptions (here zero) and the mean effusion flight path.

Output file interpretation

The resulting output file displays the following information (comments are interpolated).

```

results after processing input file: test.inp
-----
preprocessing cpu time[s]= 327.
-----
|
| event = 0 neutral particle
|       = 1 ionised particle
|       = 2 neutral particle absorbed
|       = 3 ionised particle absorbed
|
| x0, y0, z0 = birth coordinates
| ocell = starting cell (region)
| td = diffusion time in bulk [s]
| tPo = effusion time in powder/felt [s]
| COLP = number of collisions in powder/felt
| COL = number of collisions elsewhere
| teff = Flight time outside the powder [s]
| tde = Total desorption time in vacuum [s]
| tdep = Total desorption time in powder [s]
| tp = Total effusion time [s]
|      tp = teff + tPo + tde + tdep
| tT = Total release time per particle [s]
|      tT = tp + td
| tdep = desorption time in the powder [s]
|
-----
event  x0    y0    z0    td    tP0    COLP    COL    teff    tT
-----
0      4.579  1.000  -0.009  0.00000  0.00228  144017.6  56.0  -0.00142  0.00086
0      4.284  0.399  1.000  0.00000  0.00006  3644.3  314.0  0.00540  0.00546
0      4.760  1.000  0.484  0.00000  0.00114  72076.3  76.0  0.00020  0.00133
* * *
0      4.946  -0.440  1.000  0.00000  0.00872  551768.6  142.0  -0.00623  0.00249
1      4.388  -0.472  -1.000  0.00000  0.00001  803.0  117.0  0.00211  0.00213
0      4.193  1.000  -0.402  0.00000  0.00357  225838.4  36.0  -0.00290  0.00067
0      3.952  -1.000  -0.951  0.00000  0.00008  4792.8  118.0  0.00198  0.00206
1      3.891  -0.548  -1.000  0.00000  0.00043  27154.5  132.0  0.00191  0.00234
1      4.316  0.185  1.000  0.00000  0.00420  265829.4  124.0  -0.00222  0.00198
* * *
* * *

```

The first part contains the table with the individual events. Outgoing particles are labeled with 1 (ions) or 0 (neutral atoms). The caption explains the different columns. e.g. by multiplying COL and COLP with an average sticking time one can get the total desorption time.


```

----- GENERAL FIGURES FOR NEUTRAL ATOMS -----
Average intrinsic delay time[s]= 0.0050
Slowest particle took 0.0338[s]
Average particle time consumption:
  0.0000[% Diffusion] 100.0000[% Effusion]
-----
AVERAGE DELAY in POWDER[s]: 0.0035
DIFFUSION DISTRIBUTED AS:
  0.0000% grain diffusion
 100.0000% intergrain
----- GENERAL FIGURES FOR IONISED ATOMS -----
Average intrinsic delay time[s]= 0.0049
Slowest particle took 0.0292[s]
Average particle time consumption:
  0.0000[% Diffusion] 100.0000[% Effusion]
-----
AVERAGE DELAY in POWDER[s]: 0.0035
DIFFUSION DISTRIBUTED AS:
  0.0000% grain diffusion
 100.0000% intergrain

```

This part of the output gives the effusion to diffusion and vacuum to powder time share. Since effusion was not explicitly activated (mode 1 or mode 3), 100 % of the release time is due to effusion. Obviously the average intrinsic delay time is bigger than the average delay in the powder (because it includes it).

```

-----
***** EFFUSION OF NEUTRAL ATOMS *****
-----
EFFUSION IN VACUUM (excluding powder/fiber):
TOTAL: 806.
Average free path[m]= 1.4589
Distance between collisions:
  average= 1.6666[cm]
  highest maximum= 6.8559[cm]
Average number of collisions= 87.5384615
100.%b1 0.%b2 0.%D 0.%S
  b1: Stuck particles. Emitted thermally
  b2: Surface stuck particles. Thermally emitted
  D: Inelastic scattering. Debye surface phonons
  S: Almost elastic scattering. Specular reflection
-----
EFFUSION IN POWDER:
Average path[m]= 3.3238
 69.496[%] of total path
Average collisions in the powder= 221586.217
 99.961[%] of total
Distance between collisions in powder:

```

```

average free mean path 14.99998[um]
-----
RESIDUAL GAS:
  Mean free path[m]:  3587.808
  Average # of collisions with residual atoms:    0.04
-----
*****  EFFUSION OF IONISED ATOMS  *****
-----
EFFUSION IN VACUUM (excluding powder/fiber):
TOTAL: 194.
Average free path[m]=  1.3579
Distance between collisions:
  average=  1.6671[cm]
  highest maximum=  6.7662[cm]
Average number of collisions=  81.4536082
100.%b1  0.%b2  0.%D  0.%S
  b1: Stuck particles. Emitted thermally
  b2: Surface stuck particles. Thermally emitted
  D: Inelastic scattering. Debye surface phonons
  S: Almost elastic scattering. Specular reflection
-----
EFFUSION IN POWDER:
Average path[m]=  3.3322
71.047[%] of total path
Average collisions in the powder=  222144.447
99.963[%] of total
Distance between collisions in powder:
average free mean path 14.99998[um]
-----
RESIDUAL GAS:
  Mean free path[m]:  3587.808
  Average # of collisions with residual atoms:    0.06

```

These two tables tell about the effusive path of the atoms and ions (when they were still neutral) in the tubes and inside the powder. It also prints the number of collisions with residual gas at the pressure that was specified at runtime

```

STATISTICAL DATA
-----

Moment(T, i)=(<T^i >)^{1/i} [s]

1) T = FLIGHT.TIME.IN.VACUUM -----
M1      M2      M3      M4      M5
0.0015  0.0021  0.0027  0.0033  0.0038

Fit to T(i, t) = exp(-t/t1)/t1:
t1=  0.00154
error O(2)[%]: -6.48439

```

```

error O(3)[%]: -12.50578
Fit to  $T(i, t) \sim (1 - \exp(-t/t_1)) * \exp(-t/t_2)$ :
t1= 0.00026
t2= 0.00133
error O(3)[%]: 14.89847
error O(4)[%]: 21.85320

2) T = FLIGHT_TIME_IN_POWDER -----
M1      M2      M3      M4      M5
0.0034  0.0058  0.0079  0.0099  0.0118

Fit to  $T(i, t) = \exp(-t/t_1)/t_1$ :
t1= 0.00343
error O(2)[%]: 29.08787
error O(3)[%]: 50.57320
Fit to  $T(i, t) \sim (1 - \exp(-t/t_1)) * \exp(-t/t_2)$ :
t1= -0.00136
t2= 0.00527
error O(3)[%]: -30.52552
error O(4)[%]: -44.10240

3) T = TOTAL_EFFUSION_TIME (FLIGHT+STICKING) ---
M1      M2      M3      M4      M5
0.0050  0.0069  0.0089  0.0108  0.0126

Fit to  $T(i, t) = \exp(-t/t_1)/t_1$ :
t1= 0.00498
error O(2)[%]: -2.79223
error O(3)[%]: -4.74298
Fit to  $T(i, t) \sim (1 - \exp(-t/t_1)) * \exp(-t/t_2)$ :
t1= 0.00031
t2= 0.00469
error O(3)[%]: 6.63791
error O(4)[%]: 8.99565

```

These are the momenta up to order 5 of the flight time in vacuum, in powder and the number of collision. With this information you can in principle build up many fitting functions. The code uses them to make fits based in exponential functions

```

RELEASE FRACTION COMPUTATIONS
-----

GIVEN CONDITIONS:
=====

  T_1/2    ERF[%]
-----
  0.1E-2    9.10
  0.3E-2    26.98

```

0.6E-2	43.95
0.1E-1	57.26
0.3E-1	80.48
0.6E-1	89.25
0.1E+0	93.27
0.3E+0	97.66
0.6E+0	98.82
0.1E+1	99.29
0.3E+1	99.76
0.6E+1	99.88
0.1E+2	99.93
0.3E+2	99.98
0.6E+2	99.99
0.1E+3	99.99

OTHER CONDITIONS:
=====

tao_d [s]	ts (s)						
	0.10E+0	0.0E+0	0.1E-8	0.1E-7	0.1E-6	0.1E-5	0.1E-4
T.1/2 [s]	DRF[%]	RF (ts1)	RF (ts2)	RF (ts3)	RF (ts4)	RF (ts5)	RF (ts6)
0.1E-2	7.727	0.703	0.766	0.000	0.000	0.000	0.000
0.3E-2	13.126	3.542	3.571	5.056	0.000	0.000	0.000
0.6E-2	18.198	7.999	7.934	8.056	0.000	0.000	0.000
0.1E-1	23.022	13.183	13.039	12.337	0.000	0.000	0.000
0.3E-1	37.213	29.948	29.730	28.073	29.377	0.000	0.000
0.6E-1	48.936	43.674	43.484	41.919	35.257	0.000	0.000
0.1E+0	58.506	54.570	54.417	53.112	45.217	0.000	0.000
0.3E+0	78.059	76.231	76.154	75.476	69.799	67.202	0.000
0.6E+0	87.058	86.027	85.983	85.590	81.998	66.568	0.000
0.1E+1	91.622	90.969	90.940	90.688	88.299	73.938	0.000
0.3E+1	96.960	96.729	96.718	96.628	95.740	88.315	84.216
0.6E+1	98.441	98.323	98.318	98.272	97.815	93.645	75.731
0.1E+2	99.051	98.980	98.977	98.949	98.672	96.048	80.286
0.3E+2	99.673	99.649	99.648	99.639	99.545	98.627	90.955
0.6E+2	99.830	99.818	99.818	99.813	99.766	99.301	95.061
0.1E+3	99.893	99.886	99.886	99.883	99.855	99.574	96.924

tao_d [s]	ts (s)						
	0.10E+1	0.0E+0	0.1E-8	0.1E-7	0.1E-6	0.1E-5	0.1E-4
T.1/2 [s]	DRF[%]	RF (ts1)	RF (ts2)	RF (ts3)	RF (ts4)	RF (ts5)	RF (ts6)
0.1E-2	2.481	0.226	0.246	0.000	0.000	0.000	0.000
0.3E-2	4.279	1.155	1.164	1.648	0.000	0.000	0.000
0.6E-2	6.020	2.646	2.624	2.665	0.000	0.000	0.000
0.1E-1	7.727	4.425	4.377	4.141	0.000	0.000	0.000
0.3E-1	13.126	10.564	10.487	9.902	10.362	0.000	0.000
0.6E-1	18.198	16.241	16.170	15.588	13.111	0.000	0.000
0.1E+0	23.022	21.473	21.413	20.900	17.793	0.000	0.000

0.3E+0	37.213	36.342	36.305	35.982	33.275	32.037	0.000
0.6E+0	48.936	48.357	48.332	48.111	46.092	37.418	0.000
0.1E+1	58.506	58.088	58.070	57.909	56.384	47.213	0.000
0.3E+1	78.059	77.872	77.864	77.791	77.076	71.099	67.799
0.6E+1	87.058	86.954	86.949	86.909	86.504	82.817	66.975
0.1E+2	91.622	91.557	91.554	91.528	91.271	88.844	74.264
0.3E+2	96.960	96.937	96.936	96.927	96.836	95.943	88.480
0.6E+2	98.441	98.429	98.428	98.424	98.377	97.919	93.738
0.1E+3	99.051	99.044	99.044	99.041	99.013	98.735	96.107
* * *							
* * *							

The exponential fitting functions are Laplace-Transformed and folded with the diffusion release efficiency for several diffusion parameters. The global release efficiency factor, as a function of the half-life and sticking coefficient [s] are printed in tables. Every table corresponds to a different diffusion time constant.

average effusion time in each cell [%]	
cell	<time share>
1	10.344
2	88.634
3	1.022

average number of events in each surface			
surface	<collisions>	<absorptions>	<eff[%]>
1	6.948	0.000	0.000
2	4.101	0.000	0.000
3	0.000	0.000	0.000
4	19.103	0.000	0.000
5	19.083	0.000	0.000
6	19.267	0.000	0.000
7	19.514	0.000	0.000
8	0.000	0.000	0.000
9	0.000	0.000	0.000

CONNECTIVITY		
ocell	→ ecell	share
1	3	1.0010

These 3 tables tell about the effusion time share in the different regions (1), the number of collisions and absorptions in each surface (2) (no particle collides with

last surface because they are forced to cross it) and the connectivity between the starting regions and the end region (3) (In this case they are all born in cell 1 because we forced that, and they all die when crossing surface 9, which belongs to cell 3.

```

-----
SURFACE IONIZATION
surface collision ionization probability:  0.003592
ionization efficiency= 19.40000[%]
relative error= 6.44565[%]
"IMPLICIT" ION TRANSPORT
Ion extraction efficiency:  0.2
(this factor is included in ionization eff)
Ion extraction time[s]:  0.001
(this factor is included in teff)
-----

```

In presence of ion sources, RIBO prints the ionization efficiency and the parameters linked to the transport of the ions.

```

-----
Number of histories[N]= 1000. CPU time[s]= 130.
CPU_time/history= 0.13[s/history]
CPU_time/collision= 0.00184250808[s/collision]
=====

```

Duration of the simulation.

Connectivity matrix.

In this case the matrix of candidate links between cells is very simple:

1
2
2
1
3
3
2

This means that the first cell is only connected to the second (it is indeed so because it is embedded in the latter one), that the second source is connected not only to the first but also to the third one and that the third cell is only linked to the cell number two.

Data analysis.

The results stored in the output file can be analyzed in various manners depending on the specific needs of the problem. Usually the individual parts are imported into a worksheet in order to plot histograms with the possibility to observe the influence of several parameters on the delay time. Whence, a first column of results may contain the diffusion time, and this may be modified proportionally as a function of η . In the same way, the effusion time vector may be adjusted to a particular speed of atoms (proportional to $T^{0.5} \cdot M^{-0.5}$) and the sticking time vector will be obtained by multiplying the array that stores the number of collisions by the individual sticking times.

The array of results shall be divided in equally sized groups in order to compute separate calculations, merge them and compare them. This procedure provides information about the variance of the results.

3.1.4 Example of use of the 3D-Diffusion, 3D GRID: heat3D.t.

The RIBO distribution includes an example of the use of the 3-D diffusion package. The input file `[targets]/heat3D.t` describes a simple case where a spherical target of W with a diameter of 3 cm is irradiated with a beam that deposits $10 \frac{kW}{cm^3}$ uniformly over the sphere. The sphere is under vacuum and exchanges heat with a spheric heat screen that encloses the W target. The screen is made of Ta,

it has an inner diameter of 4 cm and an outer diameter of 4.5 cm. The starting temperature of the W target is 500 K, while the shield is at room temperature. The example shows how to compute the evolution of the temperature of target and shield. Figure 3.5 describes the main parameters linked with this example:

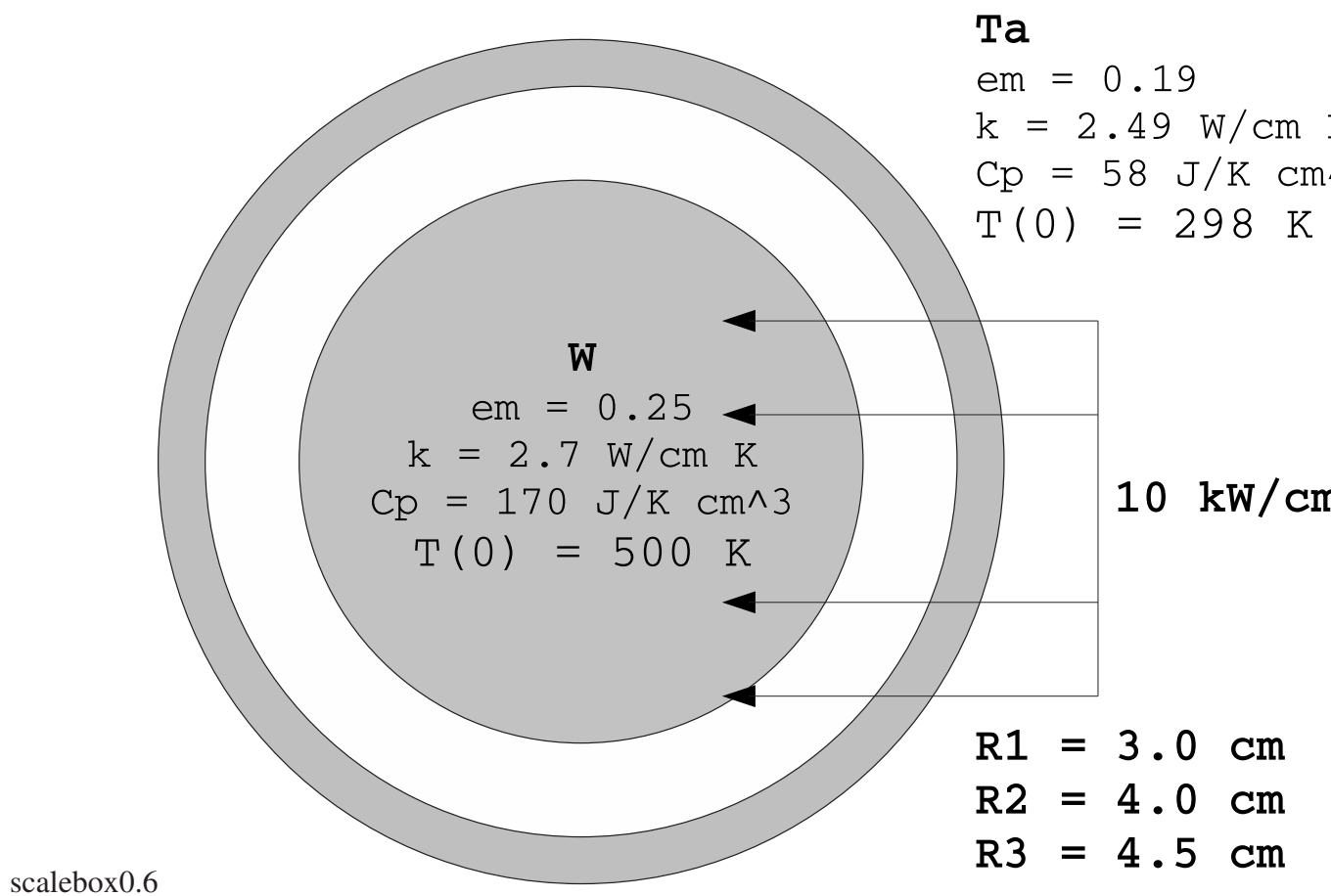


Figure 3.5: The spherical W target and the Ta screen.

Input file

The input file for this example is very simple. There are 3 quadrics (the three spheres that describe the radius of the target and the inner and outer radius of the shield) and 2 cells, one for the target and one for the shield.

The details of the problem are found in the *[targets]/heat3D.t*

Surfaces												
#	RC	T	X2	Y2	Z2	XY	XZ	YZ	X	Y	Z	C
1	0.5	298	1	1	1	0	0	0	0	0	0	9.00
2	0.5	298	1	1	1	0	0	0	0	0	0	16.00
3	0.5	298	1	1	1	0	0	0	0	0	0	20.25
Cells												
#	S1	S2	..									
1	-1	0										
2	2	-3										
Source												
Type	M	T	Alfa	nx	ny	nz	x	y	z	R		
S	90	298	180	-1	0	0	0	0	0	3		
Tally												
S	max	Tmax	Tpmax									
3	50000	750	10									

In the present version, the values of the Source card have no effect in the results for 3D-diffusion, but still, **it is compulsory to fill in all fields**. As for the Tally cards, for the moment only the third one (Tmax) is used in 3D-diffusion, having essentially the same meaning as for effusion simulations.

As for the data files, the simulation reads the specific capacity, conductivity and emissivity from *[data]/Cond.dat*. The example printed in 2.2.1 and included in the RIBO distribution already contains the values for W (cell 1) and Ta (cell 2). Note that this table can be refined by filling in the fields that show a dependence of first and second order with respect to the temperature (or concentration). Moreover, if the temperature of the system varies a lot, you may also want to introduce

the variation of the emissivity with the temperature. You can easily do that by editing the function *radia* in the `[sources]/custom3D.f` file.

Custom routines

The specific options ($10 \frac{kW}{cm^3}$, radiation function, starting temperatures...) are already implemented in the open source functions and subroutines in `[sources]/custom3D.f`. You are invited to look through them and identify the different choices.

Runtime options

† **NOTE:** The input values are stored in the batch file `[batchinputs]/heat3D.batch`.

```

|   RRR           II       BBB       OO           |
|   R R           II       B B       O O           |
|   RRR  radio-   II  on  BBB  eam  O O  ptimi-   |
|   R R  active  II       B B       O O  ser      |
|   R R           II       BBB       OO           |
+-----+
                                     2000-2006 MSL

                                WELCOME !!

*****
** Program Licensed exclusively to:
**   MANUAL READER
*****

version 30-06-06
full version

Name of the input file?
heat3D.t

```

```

ifile=heat3D.t
Name of the output file?
  (Beware it will overwrite this file!)
heat3D.out
READING INPUT FILE heat3D.t      ...
  1) reading surfaces...
  2) reading cells...

----- number of cells: 2
----- highest number of walls in a cell: 2
----- number of bodies: 3
.....
...reading tally
Nmax,tmax,tpmax 50000 750. 10.
reading geometry...
...storing surfaces (bodies)...
  ...storing cells (regions)...
INPUT FILE HAS BEEN READ SUCCESSFULLY

-----
| SOURCE CUSTOMIZATION: |
| - You can edit the file [sources]/customsource.f |
|   and then recompile with [tools]/make.sh |
| - Additionally you can use the data file init.dat |
|   by uncommenting "read..." in customsource.f |
| - And you can restrict generation to a cell... |
|   ...source limited to a cell? give cell number. |
|   celln > 0 ==> generation limited to volume |
|   defined by celln |
|   celln = 0 ==> do not constrain to a cell |
|   celln < 0 ==> just generate a geometry plot |
-----
1

```

This part is standard. As for any simulation, the code asks for the input file name and the output file name. Then, as usual, it gives the chance to restrict the initial distribution of particles (CSTART.f, 2.3.1) to one of the cells defined in the input file.

```

| SPECIAL EVENTS, TERMINATION |
| Please choose among these options: |
| 1: Crossing of end surface (1st card in "Tally") |
| No ionizations in the system. |
| 2: Atoms can be ionized in a PLASMA ion source |
| Histories (of atoms and ions) end when |
| they cross the end_surface detector |
| 3: Atoms can be ionized in a SURFACE ioniser |
| Histories (of atoms and ions) end when |
| they cross the end_surface detector |
| 4: Atoms can be absorbed in the walls. Trajec- |
| tories end at absorption or when crossing |
| the end surface |
| So, what do you want, 1, 2, 3 or 4? |
|-----|
1
SELECT MODE:
  1: Diffusion.
  2: Effusion.
  3: Diffusion+Effusion.
recommended 2
  4: Conductance calculator (Clousing Coefficient).
  5: Diffusion3D. — ALPHA VERSION !!
  6: Heat transfer (cond.+rad.). — ALPHA VERSION !!
6
|-----|
| CHOOSE THE OUTPUT MODE: (standard is 3) |
| 8: Only average figures |
| 9: Print full matrix evolution (SPACE!!) |
| C: Use the custom routine userPRINT3D.f |
|-----|
8

```

The first of the questions has no impact in our future decisions. We answer 1. The next question is crucial. We choose 6, because we are dealing with a heat transfer

problem. This triggers a preselected number of output formats, 8 (standard), 9 (full) and C, custom (please consult section 2.3.5).

```

-----
|
| 3333 DDDD GGGG RRRR II DDDD |
| 3 D D G R R II D D |
| 333 D D G GGG RRRR II D D |
| 3 D D G G R R II D D |
| 3333 DDDD GGGG R R II DDDD |
|-----|
| DEFINING 3D WINDOW FOR THE CALCULATIONS |
| 1) starting vertex xmin , ymin , zmin [cm]? |
| e.g: -10 -10 -5 |
|-4.5 -4.5 -4.5 |
| 2) what are the full widths x,y,z [cm]? |
| e.g: 10 10 5 |
|-----|
9.0 9.0 9.0
| 3) number of divisions in x,y,z? e.g: 10 10 20 |
|-----|
180 180 180

```

The module 3D takes over. The user must define the 3D window where heat transfer will act. Vacuum will replenish the remaining space. The first data input are the coordinates of the lower corner X_{min} , Y_{min} , Z_{min} . The 3D box will be defined by writing the 3 amplitudes ΔX , ΔY , ΔZ . Finally, the space will be evenly discretized in parallelepipeds when the number of divisions in each dimension is given. Here we are in a symmetrical case so the number of division in each dimension is the same.

† **WARNING:** The maximum dimension of the matrix is 300 300 300 !!

```
INITIALISING GRID... PLEASE WAIT
```

```
Number of boxels outside vacuum 1814368.
```

```

Maximum value at t = 0      500.000
Minimum value at t = 0      0.000
Average value at t = 0      398.7524
-----

| ..... END CONDITION ..... |
| What is the end condition for the 3 D problem? |
| (Give condition number and associated value) |
| - 1 tmax[s] e.g. 1 20 |
|     means "stop_when_elapsed_time=_20_s" |
| - 2 average/average(0) e.g. 2 0.2 |
|     means "stop_when_average_Temperature_ |
|         decreases_down_to_0.5_of_the_initial_value" |
| - 3 Tmin[K] e.g. 3 350 |
|     means "stop_when_minimum_Temperature_ |
|         goes_below,_or_grows_up_to_Tmin" |
| - 4 Tmax[K] e.g. 4 1700 |
|     means "stop_when_maximum_Temperature_ |
|         goes_below,_or_grows_up_to_Tmax" |
| - 5 #_of_steps e.g. 5 1000 |
|     means "stop_after_1000_time_steps" |
| - 6 |(dT/T)max/dt| e.g. 6 1E-6 |
|     means "stop_when_the_maximum_relative_T_ |
|         change_per_unit_time_goes_below_1E-6/s_ |
|         (biggest_change_<_1%_in_10000_s)" |
| ..... |
| NOTE: |
| - Whatever condition, the computation will be |
|   terminated if the computation time exceeds |
|   CPUmax [s] (4th entry in "Tally", input file |
| ..... |
|----- Read Cond.dat matrix, found matrix is: -----|
| 2.700  0.000  0.000  0.250 170.000  0.000  0.000 |
| 2.490  0.000  0.000  0.190  58.000  0.000  0.000 |
|-----|
| INITIALISING TIME STEPS... |
| RUNNING... |

```

```

time [ s ]      Cmax  IM      XM      YM      ZM
Cmin  Im        Xm        Ym        Zm  <C(t)/C0> <C(t)>
0.000000      500.0000  1    -2.9750  -0.3750  -0.0750
298.0000      2    -4.4750  -0.4250  -0.1750  1.0000  398.752
0.136000      508.0000  1    -2.9250  -0.3750  -0.0750
298.0000      2    -4.4750  -0.4250  -0.1750  1.0100  402.743
* * *
* * *
1.052380      561.9047  1    -1.9250  -0.0750  -0.0250
297.4828      2    -2.5250  -2.3750  -2.3750  1.0774  429.630
1.097014      564.5302  1    -1.8750  -0.0750  -0.0250
293.7514      2    -2.5250  -2.4250  -2.3750  1.0807  430.940
CALCULATION FINISHED!
DONE. Results are stored in: heat3D.out

```

Ribo initializes the 3D grid. It then asks for the termination condition. In this case the simulation will finish when the maximum temperature reaches 700 K, or when the computational time reaches $T_{max} = 750$ s

Output file interpretation

The resulting output file displays the following information:

```

results after processing input file:heat3D.t
-----
preprocessing cpu time[s]= 15.
-----
|      time = elapsed time [s]      |
|      Cmax = Maximum value (it can be multiple) |
|      IM = cell to which Cmax belongs |
| {XM,YM,ZM} = coordinates of one of the Cmax |
|      Cmin = Minimum value (it can be multiple) |
|      Im = cell to which Cmin belongs |
| {Xm,Ym,Zm} = coordinates of one of the Cmin |
|      <C(t)> = Average value of C |
|      <C(t)/C0> = Relative increase or decrease <C> |
-----

```

time [s]		Cmax	IM	XM	YM	ZM	Cmin	
Im	Xm	Ym	Zm	<C(t)/C0>	<C(t)>			
0.000000	500.0000	1	-2.9750	-0.3750	-0.0750	298.0000	2	
-4.4750	-0.4250	-0.1750	1.0000	398.752				
0.136000	508.0000	1	-2.9250	-0.3750	-0.0750	298.0000	2	
-4.4750	-0.4250	-0.1750	1.0100	402.743				
0.274078	516.1242	1	-2.2750	-1.5250	-1.1750	297.9584	2	
-3.0750	-1.9750	-1.6750	1.0202	406.794				
*	*							
*	*							
*	*							
1.046866	561.5804	1	-1.9250	-0.0750	-0.0250	293.8229	2	
-2.5750	-2.3750	-2.3750	1.0770	429.468				
1.052380	561.9047	1	-1.9250	-0.0750	-0.0250	297.4828	2	
-2.5250	-2.3750	-2.3750	1.0774	429.630				
1.097014	564.5302	1	-1.8750	-0.0750	-0.0250	293.7514	2	
-2.5250	-2.4250	-2.3750	1.0807	430.940				
<hr/>								
Number of steps= 50 CPU time[s]= 765.								
CPU_time/step= 15.3[s/step]								
CPU_time/sim_time= 693.922836[s-CPU/s_diffusion]								
=====								
Mario Santana Leitner								
ISOLDE-CERN, 2001-2006								

The output file for these type of calculations is rather simple to interpret. In output mode 8, the elapsed diffusion time is printed in the **first column**. The highest found temperature, the cell and the coordinates of a boxel with such temperature (there may be many with the same value) are printed in **columns 2-6**. The same information is displayed in **columns 7-11** for the minimum value. Finally, the **forlast column** tells about the ratio of the average Temperature at time t , and that

at time 0, and the **leftmost column** shows the evolution of the average temperature with time. The file ends with some numbers related to the computational speed.

The primary source that is used here is a sphere, the radius of which equals the maximum radius of the ellipsoid. The cell-limited option will be chosen at runtime.

RIBO-Diffuse. A diffusion toolkit.

DIFFUSE is a bash shell script that manages the core FORTRAN program DIFFUSE.F. It computes diffusion profiles analytically (through the infinite series provided by Fujioka [14] and the second law of Fick [15]) and, for one-dimensional geometries under variable and/or non-homogeneous conditions, numerically (first law of Fick). Additionally, it may compute the diffusion coefficient provided that a release fraction is known at any given time. More details can be found in [11], 2.5.1.

Running DIFFUSE is trivial because it is fully interactive. Fig.4 shows the first and second menu choices. In fig.4.1(a) options **1,3,5** permit to plot the drop of total concentration in the slab as a function of time. **5** additionally shows the space dependency (2-D graph). In any of the cases, it is necessary to install PAW [8] to have the functions plotted. Otherwise, the user can take the output data, written in the file *profile.dat* and use his favorite data analyzing program. **2** and **4** invert the diffusion function to extract the Diffusion coefficient from a fixed diffusion release situation.

Based on the chosen option, DIFFUSE poses the following questions:

Insert the number of space nodes

A good compromise between precision and speed of computation would be between 10 and 1000.

Insert the diffusion coefficient [cm²/s]

The units are specified, [$\frac{cm^2}{s}$]

Fractional concentration

Insert the remaining concentration.

e.g. 0.5 would mean that 50 % has diffused out.

Insert the lapse length [s].

It refers to the final time, or the time at which the fractional concentration is measured.

Time definition.

Insert the printing time grid (number intervals).

The time steps for the diffusion calculations are adjusted internally to keep the system stable, but the time steps are often much too small to have them all printed. Thus, the user specifies what should be the time binning.

For the present status of the program, the user needs to introduce the wished time dependencies directly in the corresponding subroutines.

```

SUBROUTINE timeN(N, i, m, t, dt, a)
      real*8 N(m,2), t, dt, tao, pulse, a, x
      integer*4 i, m
* decay law
      tao=0.097
*      tao=27.0
*      N(i,1)=N(i,1)*exp(-log(2.0)*dt/tao)
* pulsing source
      x=(a/m)*i

```

```

pulse=0.5
IF (mod(t,pulse).lt.1.0001*dt.and.t.gt.pulse) THEN
    N(i,1)=N(i,1)+functC(x,a)*1
END IF
END

```

2

For time dependencies, or

```

FUNCTION functC(x,a)
    real*8 x,a
    x=a-x
    functC=1*exp(-x*10.0/a)*(1-exp(-x*1.0/a))
    x=a-x
    functC=1*sin(x*3.1415927/a)**2
    functC=1
END

```

3

For space variations of the concentration or, finally:

```

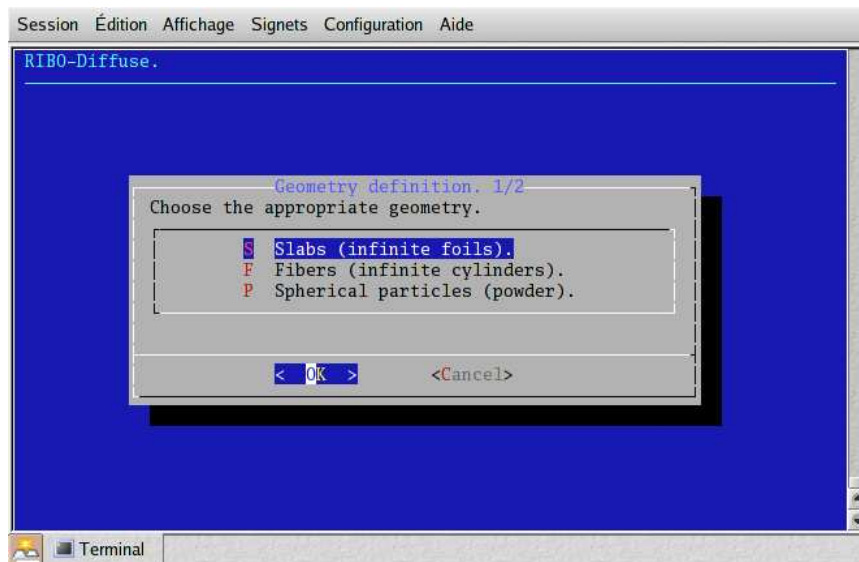
FUNCTION D(i,m,a,t,Dc)
    real*8 x,a,t,Dc,r
    integer*4 i,m
    x=da*(i-0.5)
*   D=Dc*sin(x*3.1415927/a)*exp(-t/100)
    D=Dc
*   r=0.999
*   D=((1+r)-(2*r)*(x/a))*Dc
    IF (t.le.1) THEN
*       D=7.7778*Dc
    END IF
END

```

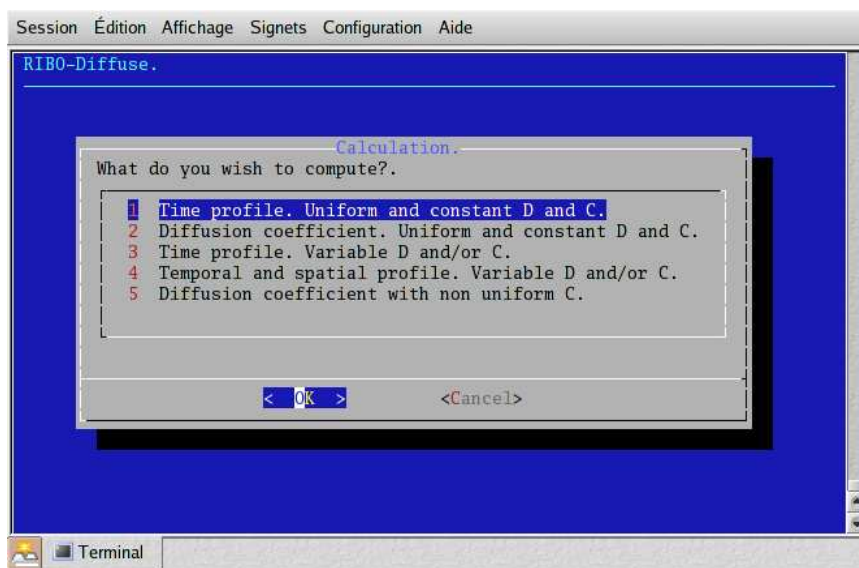
4

5

For space and time variations in D.



(a) First choice of DIFFUSE.



(b) Options for 1-D calculations.

Figure 4.1: Screen captures of the interactive DIFFUSE program.

Reference List

- [1] L. Maunoury, O. Bajeat, R. Lichtenthäler, and A. Villari. Temperature simulations for the SPIRAL ISOL target. *Nucl. Inst. and Meth. B*, 2001. submitted.
- [2] R. Lichtenthäler et al. *A simulation of the temperature distribution in the SPIRAL target*. Report, GANIL, Caen, August 1997.
- [3] J.S. Hendricks et al. MCNPX, version 2.5.e. Manual, Los Alamos National Laboratory. New Mexico., 2004. mcnpx.lanl.gov/.
- [4] N.V. Mokhov. *The MARS Code System User's Guide, Version 13(95) Reference Manual*. Fermilab, 1995.
- [5] N.V. Mokhov. *Status of MARS Code*. Fermilab, fermilab-conf-03/053 edition, 2002. www-ap.fnal.gov/MARS.
- [6] A. Fassò, A. Ferrari, and P.R. Sala. Electron-photon transport in FLUKA: status. In F. Barao, M. Nakagawa, L. Tavora, and P. Vaz, editors, *Proceedings of the MonteCarlo 2000 Conference, Lisbon, October 23–26 2000*, pages 159–164. Springer-Verlag Berlin, October 2001.
- [7] A. Fassò, A. Ferrari, J. Ranft, and P.R. Sala. FLUKA: Status and prospective for hadronic applications. In A. Kling, F. Barao, M. Nakagawa, L. Tavora, and P. Vaz, editors, *Proceedings of the MonteCarlo 2000 Conference, Lisbon, October 23–26 2000*, pages 955–960. Springer-Verlag Berlin, October 2001.
- [8] CERN. *Physics Analysis Workstation*. CERN, paw.support@cern.ch. wwwasd.web.cern.ch/wwwasd/paw.

- [9] Persistence Of Vision Pty. Ltd. *Persistence Of Vision*. Pty. Ltd, Williamstown, Victoria, Australia, 2004. Software.
- [10] B.T. Phong. Illumination for computer generated images. *ACM*, 18(6):311–317, 1975.
- [11] M. Santana-Leitner. *A Monte Carlo code to optimize the production of Radioactive Ion Beams by the ISOL technique*. PhD thesis, Technical University of Catalonia, 2005.
- [12] S. Agostinelli et al. Geant4, a simulation toolkit. *Nucl. Inst. and Meth. A*, (506):250–303, 2003. wwwasd.web.cern.ch/wwwasd/geant4/geant4.html.
- [13] H.M. Kalos and A.P. Whitlock. *Monte Carlo methods*. New York, Wiley & Sons, 1986.
- [14] M. Fujioka and Y. Arai. Diffusion of radioisotopes from solids in the form of foils, fibers and particles. *Nucl. Inst. and Meth.*, 186:409–412, 1981.
- [15] A.E. Fick. *Ann. Phys.*, 94:59, 1855.