1 Diffusion and reactions

1.1 Introduction

Plasmas are quite reactive. In particular the hot electrons can easily create many different excited species and ions. Often, it are these excited species or ions that are the reason the plasma is created. A description of these reactions is of paramount importance for a good description of the plasma.

In many plasmas, especially low-pressure or smaller plasmas, diffusion plays an important role. Ambipolar diffusion of electrons and ions to the wall in particular is an important process in many plasmas. Because diffusion and reactions are often closely linked, we will opt for a joint description.

1.2 A simple model

We will start our treatment of the diffusion/reaction problem with an extreme case. There is only one reaction important, which is ionization from the ground state to the ion state. The formed electron/ion pairs are subsequently destroyed by diffusion to the wall, where they recombine. We will try to model an argon plasma that resembles a cascaded arc. For this, we need three things: the size of the plasma, 0.002 m, the ionization source term and the diffusion coefficient.

Exercise 1 *We will assume the electron density is equal to the ion density. When is this realistic?*

Exercise 2 *The power density in the cascaded arc is $4 \cdot 10^9$ Wm^{-3}. Assuming that a quarter of it is used to produce ions, how many ions are produced per second? (The ionization energy of argon is 15.759 eV [www.webelements.com])*

Exercise 3 *Program a function that calculates the ambipolar diffusion coefficient as a function of heavy particle density, electron temperature and heavy particle temperature. Given that the heavy particle density is $1.6 \cdot 10^{23} m^{-3}$, the heavy particle temperature is 10000 K, the electron density is 12000 K, and the atom-ion collision cross section is 7e-19 m^2, compute the diffusion coefficient and have your program compute it as well. HINT: The function should have the following header:*
`double ambipolar_diffusion_coefficient(double n0, double Te, double Th)` *Programming tip: It's bad practice to have numbers reperesenting physical parameters, such as the mass of argon, the charge of an electron, the grid size, etc. floating around in your code. For starters, it's not nice to see a number and not know what it means. Also, it makes it difficult to later change the number. It is therefore better to declare these numbers as constants and use the constant in the functions.*

Exercise 4 *Create two `Fields`: one for the electron temperature T_e, and one for the heavy particle temperature. For the heavy particles, you should set the boundary temperature at the wall to 300 K. The lighter electrons, on the other hand, do not effectively cool themselves to the wall. You may assume the temperature gradient for the electrons is 0. Initialize the heavy particle density with* *a parabola-like temperature profile, with a top of 10000 K in the center and a temperature at the wall of 300 K. You may initialize the electron temperature with a linear profile, with a value of 12000 at the center and 11000 at the wall. HINT: you can access the physical position of a point with the Compute the value of the diffusion coefficient on each point. Write to screen the local value of the electron and heavy particle temperature and of the diffusion coefficient for each point.*

Exercise 5 *Write a model that describes the reactions and diffusion in the plasma. Use the reaction rate `sc` computed in Exercise 1.2 and the ambipolar diffusion coefficient function of Exercise 1.2. Create two new fields, `neutr_dens` for the neutral density and `ion_dens` for the ion density. You can represent the reactive boundary by setting a low density of ions there, for instance $1 \cdot 10^{12}\ m^{-3}$. These numbers are derived from a real plasma, a cascaded arc source; The central density should, therefore, resemble the density in the centare of the arc, which is in the order of $1 \cdot 10^{22}\ m^{-3}$. You do not need an iterative procedure to solve this, as none of the parameters depend on the parameter you are trying to solve, but you can use one if you wish. For now, you may use the given value of the heavy particle density, $1.6 \cdot 10^{23} m^{-3}$.*

1.3 The background gas

When a particle diffuses, the local pressure drops. This will be compensated. A full treatment of this can be extremely complicated. However, when there is a single, dominant background gas, we can considerably simplify matters by assuming it is this background gas which diffuses back and restores the pressure. In this case, we assume the ground state of argon is so dominant it can act as this dominant background gas.

Furthermore, the ionization reaction also destroys ground state argon. This should also be included in the model.

We will take into account both effects by assuming the density of ground state argon is determined by a constant bulk pressure, minus the partial pressures of all other plasma constituents.

Exercise 6 *Write a function which computes the heavy particle density, as a function of the pressure, the heavy particle density, the heavy particle temperature and the electron temperature. You can do so by first computing the ion and electron partial pressure, then use this to compute the ground state pressure, and then use it to compute the ground state density. Add the function to your program, using a bulk pressure of 30000 Pa. You will need to make the solution procedure iterative. Don't forget to change the value of the diffusion coefficient with the background particle density, which you will no longer have to compute. You will also need decent starting values.*

Exercise 7 *We are now going to use a procedure which computes the heavy particle density based on another definition of the composition. In this case, we are going to assume the pressure is constant, but undefined, and that the total average particle density is some given number. This is very appropriate for a* *closed system, where the amount of particles is indeed conserved. Because this is a fairly complicated problem, a function which can do this is provided on the website, called `calculate_buffer_density`*[1] *(If you have a lot of time, and like a challenge, you can try to program it yourself). You should have six fields for using this function:*

- *-An average particle density`ave_dens`.*
- *-The grid `grid`.*
- *-A heavy particle density field called `neutr_dens`.*
- *-An electron or ion density field called `ion_dens`.*
- *-A heavy particle temperature field called `Th`.*
- *-An electron temperature field called `Te`.*

Use this procedure in your program, and calculate the densities of electrons and ions at all points, and also the pressure.

1.4 Three-particle recombination

Apart from diffusion losses, there is another loss mechanism in a plasma: three-particle recombination. In this process, an ion recombines with an electron, while a third spectator particle is present.

Exercise 8 *Why is this third spectator particle necessary?*

Exercise 9 *Write a function that computes the three-particle loss rate coefficient. For argon, assume G is equal to 6, and a k_{rate} of $1{\cdot}10^{-15}$ $m^3 s^{-1}$. Use this rate coefficient to compute the three-particle destruction rate. Subtract this from s_c.Note how many iterations the computation now takes.*
HINT: `std::pow(mantissa,exponent)` computes a power.

1.5 Linearization

In Exercise 1.4, we have a source term which is in fact linear in its field. Our discretization can take this linearity into account. This may result in a more stable and faster convergence. See Table 1.5 for a list of possible linearizations. For those interested, Patankar's book is quite thorough on the subject.

[1]For those interested in C++ (If you are not one of them, by all means skip this paragraph), a few remarks on the programming of the `calculate_buffer_density` procedure: You will notice the `&` in the function declaration. This means that the field is passed by reference: the procedure is manipulating the actual objects, rather than a local copy of the object (Try removing the `&` if you are not convinced this is important). Also note the use of the `const const` tells the compiler that the contents of the object which is declared `const` should not be modified. This is useful for protecting the contents of these objects. For instance, the `calculate_buffer_density` has no business messing with the contents of the electron temperature. (Try modifying the contents of the electron temperature in the function to have the compiler correct you. An immediate correction by the compiler is much better than only figuring out your mistake after months.)

Table 1: Possible linearizations of a field ϕ with a linear source term $-S$. If the problem converges, all yield the same solution

Description	s_c	s_p
Naive	$-S\phi$	0
Natural	0	$-S$
Extra steep	$S\phi$	$-2S$
Unstable	$-2S\phi$	S

Exercise 10 *First, we will try the most obvious linearization: just divide the source term by the ion density, and use it in the linear source term s_p (Line 2 in Table 1.5). We will call this $-S$. What happens with the convergence speed? And with the answer?*

Exercise 11 *Next, we will go for a steeper linearization. We double the linear term, and compensating by adding the ion density times S to s_c.(Line 3 in Table 1.5) Verify for yourself this will result in the same converged solution! Notice the speedy convergence.*

Exercise 12 *Finally, we will try something which Patankar says is unstable. We will use minus two times the ion density times S for s_c, and S for s_p. What happens with the convergence? How can you fix this problem?*

1.6 Conclusion

The final system we have is already quite close to an actual plasma. By adding heat transport, discussed in the first lesson, heat transfer between electrons and ions and a more realistic formation rate, we can in fact obtain a reasonable self-consistent plasma model. As we we will see in the next lesson, this is where the real problems start.