1 Heat Conductivity

1.1 Introduction

Plasmas are generally not in equilibrium with their surroundings. In particular, laboratory plasmas are sustained by a power source, usually electromagnetic, and lose part of this energy through heat conduction. This is why we want to describe this phenomenon. In today's exercises, we will create a simple model of heat conductivity in a plasma.

1.2 The ϕ-equation

In plasma physics, many transportable parameters can be described by a general transport equation:

$$\frac{\partial \rho\phi}{\partial t} + \vec{\nabla} \cdot (\rho\vec{u}\phi) = \vec{\nabla} \cdot (\Gamma_\phi \nabla\phi) + S_\phi \tag{1}$$

Here, ρ is the density, ϕ is the parameter, t is the time, $\vec{u}$ is the velocity, Γ_ϕ is a general diffusion coefficient, and S_ϕ is a source.

We can describe heat conductivity with this equation. For doing this, we start out with the specific (per mass) enthalpy equation:

$$\frac{\partial \rho h}{\partial t} + \vec{\nabla} \cdot (\rho\vec{u}h) = \vec{\nabla} \cdot (\Gamma_h \nabla h) + S_h \tag{2}$$

Here, h is the specific enthalpy. Because it is very uncommon to use the specific enthalpy rather than the enthalpy, we will rewrite this to

$$\frac{\partial H}{\partial t} + \vec{\nabla} \cdot (\vec{u}H) = \vec{\nabla} \cdot (\Gamma_H \nabla H) + S_H \tag{3}$$

with H the enthalpy per unit volume. If our heat capacity is constant, we use

$$H = c_p T \tag{4}$$

with c_p the heat capacity and T the temperature. We thus obtain:

$$\frac{\partial T}{\partial t} + \vec{\nabla} \cdot (\vec{u}T) = \vec{\nabla} \cdot \left(\frac{\lambda}{c_p} \nabla T\right) + \frac{S_H}{c_p} \tag{5}$$

with λ the heat conductivity.[1]

1.3 Heat conduction in a bar

Install the program in a directory of your choice. There should be 2 C++ files: `plasma.h` with the functions you will need, and `example1.cpp`. By including `plasma.h` in a `.cpp`file, you can use its functions. By calling the right functions in the `.cpp` file, using standard programming techniques and appropriate values of physical constants, you can make simulations.

[1] This is tricky, because ϕ-variables are supposed to be extensive, and the temperature is intensive. This derivation is a sketch of how to get around this problem. Pay very keen attention on the units—S_H should be in Km^{-3}, λ is in $\mathrm{Wm}^{-1}\mathrm{K}^{-1}$ and c_p is in JK^{-1}.

First, we will discuss the contents of `plasma.h`

We will now look at the example, which is in fact a working simulation of heat conductivity in a bar. At line 3, `plasma.h` is included. As you know, C++ programs always begin with a `main` function. We start out by creating a grid. This function takes three arguments: the first one is the number of grid cells, the second the physical coordinates of the left coordinate, and the right one the coordinates of the right side. Note that this does not uses units: you should therefore make sure you either make the problem dimensionless, an approach which is often taken in fluid mechanics, or make sure you use a consistent set of units. We will use the SI system — make sure you don't accidentally use eV in describing energies or temperatures.

Next, we define the bulk (convective) flow. In this case, there is none, so we use the `ZeroFlow`. We are going to simulate a rod that is isolated on the left end. "Isolated", or no heat flux out of it, means that the derivative of the temperature is zero. Boundary conditions that specify derivatives are named after Neumann. Because it is isolated, the derivative is zero; this is known as a Homogeneous Neumann boundary condition. We make such a boundary condition by calling `Plasma::NeumannBC temp_left(0,0)` The first number is the value of the derivative, while the second is a linerarization factor, which is not commonly used[2]. We cool the wall at the right boundary. Here, we assume we have a perfect device for keeping the temperature at a fixed value, in this case, 20 K. Specifying an exact value for the boundary point is known as a Dirichlet boundary condition, and we create one by `Plasma::DirichletBC temp_right( 20 );`.

Next, we make a temperature field, defined on this grid. The first argument is the grid, the second the flow field, the third is the left boundary condition, and the last is the right boundary condition. This creates an instance of the class, also known as an object. If you structure your program right, such an object corresponds to a real concept, in our case, the ϕ-variable. It, therefore, contains all the things it needs, like the boundary conditions, the source terms, the value of the field, etc. This grouping of things that belong together in a programming structure is called encapsulation.

Next, we define heat conductivity. This is the `gamma` member. The [i] refers to the number of the grid cell. We next set a constant source `sc`, which represent a constant heating per unit length.

Finally, we compute our result by calling `Update()` to solve the discretized equation. We next write the result to console, by iterating over the grid and writing to standard out.

Exercise 1 *Compile the program, and run it. Comment on the output.*

[2]It becomes useful when the derivative depends linearly on the ϕ, for instance, with wall reactions.

1.4 Heat conduction in a plasma

Our next objective will be to simulate heat conductivity in a system that resembles a plasma. Create a new folder, and copy the C++ files there. To simulate the "plasma", we will have to come up with guesstimates of the heat conductivity, the heat production, the wall temperature, and the size of the plasma. We'll start out with a high-pressure plasma, with parameters resembling a cascaded arc.

For a cascaded arc, the diameter is about 2 mm. We also know that a cascaded arc is made of copper, which has a good heat conductivity, through which cooling water flows. This leads us to think the wall temperature of the plasma will be close to the temperature of the cooling water, or 300 K.

The hardest part will be the heat conductivity. An approximate formula for the heat conductivity of the heavy particles in a plasma is give on the formula sheet. We see the heat conductivity depends on various parameter, so we'll need to make an estimate of those. The atom-atom collision cross section of the gas in the arc, argon, is simply a circle with twice the atom radius of argon, 142 pm (www.webelements.com), which has an area of $6.33{\cdot}10^{-20}$ m^2. This is a typical value for these kind of cross sections — although noble gases have somewhat small atomic radii. Next, we have to find the mass of argon. Our trusty BINAS (or other standard reference of choice) tells us an argon atom weighs $6.623{\cdot}10^{-26}$ kg. For computing the heat capacity of the heavy particles in the arc, we need to know the heavy particle density; you may assume it is equal to $1.6{\cdot}10^{23}$ Next, we spot a problem: The heat conductivity depends on the heavy particle temperature, the very thing we try to solve! We'll discuss a way of handling this in the next question; for now, we notice it only depends on the square root of the temperature, and with a guess of 2000 K for the temperature, we are hopefully not too far off. Finally, we need an estimate of the power density. The cascaded arc has a current of approximately 50 amperes flowing through it, with a voltage drop of about 1000 V/m.

Exercise 2 *Compute the heat dissipation density*

Exercise 3 *Compute the thermal conductivity*

Exercise 4 *Use these parameters in a model. Do you think the results are realistic?*

1.5 Iterative solution strategy

As we noticed in 1.4, the heat conductivity in the plasma is a function of the temperature itself. We can solve this by using an iterative procedure. In an iterative procedure, you start out with a guessed field. You use this guess to compute the new values of the parameters which depend on the field. You then use these parameters to solve the field. This field can be used to obtain new values of the parameters, which are used to solve a new field, etc. This technique may cause the field to come ever closer to the solution of the problem; this is called convergence. It is, however, not guaranteed that the field will converge, or that it will converge to the right solution.

Exercise 5 *Derive an equation for the heat conductivity as a function of temperature.*

Exercise 6 *Create an iterative procedure to solve the temperature in the cascaded arc. First, you should assign a reasonable starting value for the iterative procedure to every point of the temperature field. You should also assign the heat source you computed in Exercise 1.4. Then, create a for-loop that iteratively solves the heat conductivity at each grid point, the temperature at each grid point, and writes the result to screen. 10 iterations should suffice given a reasonable starting value.*

Exercise 7 *Rerun the code, but use a poor value of the starting condition this time (20000K). Can you explain these oscillations?*

In order to get sufficiently accurate results in a minimum of time, we want to terminate the iteration when the result is sufficiently close to the converged value. The `update()` procedure returns the so-called residual, which is an indication for the relative change in the field. When this is a lot less than the maximum error you want to have, you can terminate the calculation.

Exercise 8 *Give an estimate of the residual you think is appropriate for terminating the calculation, and explain why you think this is appropriate.*

Exercise 9 *Change the code so that it terminates at a residual of 10^{-6}. At the end of each iteration, write the iteration number and the residual. How many iterations do you need? We will use this number as a default stop criterium.*

1.6 Underrelaxation

In Exercise 7, we noticed that it is possible for the fields to behave quite wild during the convergence. Because in this case the dependence on temperature is only modest (a square root), the computation still managed to converge. In a real plasma, we are normally not so lucky.

A simple technique to stabilize the computation is underrelaxation. In this case, we compute the new temperature field, and then compute a weighted average of it and the value of the previous iteration.[3] This smoothes out oscillations. The relative weight of the new solution is called the underrelaxation factor α, and it should be between 1 and 0.[4] The relative weight of the old solution is then equal to (1-α).

[3] In practice, the implementation of underrelaxation is different, although the result is the same as what is obtained by taking the weighted average.

[4] It is theoretically possible to have an underrelaxation greater than one; This is called overrelaxation. This amplifies the changes, thus potentially speeding up the convergence. It does have a strong tendency to cause divergence. Underrelaxation must be smaller than 2; if it is larger, the problem can never converge.

In the code, it is quite easy to specify an underrelaxation. By calling `foo.urf=`α, you can set the `urf` of `foo` to α.

Exercise 10 *Redo Exercise 1.5 using a starting temperature of 300 K, with an underrelaxation of 1.0, 0.9, 0.5 and 0.1. Notice the dampening of the oscillations. How many iterations do you need? The final result differs in each case by much more than the residual. Why?*

Exercise 11 *Run the code of Exercise 10, for all four underrelaxation factors, with a starting temperature that is 2000 K. What happens to the number of iterations?*

1.7 The grid

By solving a discretized version of the problem rather than the real problem, we only obtain an approximate solution. With some exercises, we will now attempt to show some points when choosing the number of grid cells.

The first issue we have to deal with is stability.

Exercise 12 *Set the underrelaxation factor to 1.01, and the starting condition to 2000 K. Try changing the amount of grid points, and see how the amount of grid points impacts stability. Note how even a small overrelaxation has a dramatic impact on stability.*

A second issue when choosing the amount of grid cells is computational cost. For a 1 D system, the amount of is proportional to the number of gridpoints. For a 2 D system, the solving of the discretized equations becomes highly non-trivial and computationally expensive. 2 D matrix solvers typically take time proportional to the amount of grid points to the power 3, and even a modest 32 by 32 grid already has 1024 grid points. In practice, the reduced stability can make it necessary to use more underrelaxation, reducing convergence speed, and making convergence even more expensive. Computational cost can put fairly hard constraints on the maximum grid size you can use: if a computation on a 32 by 32 grid costs you a day, going to 64 by 64 will costs you at least 2 months.

There is, however, and obvious advantage to going to a finer grid: the results will be more accurate. It is therefore recommended that you use a grid which is as accurate as possible in real-life applications, given the remarks mentioned above.

1.8 Conculsion

We have seen how a simple model of heat conductivity in a bar can be made using standard numerical methods and the physics typical to the problem. We have used this model to simulate the heat transport in an Ar cascaded arc. We have seen how to deal with tranport parameters that depend on the parameters we try to investigate by using iteration. We have investigated the impact of underrelaxation and grid refinement.

The heat transport problem is a key part of almost any plasma. We will use it in the creation of a more realistic plasma model.