Introduction to Computational Multiphysics
Practical 6: Tabulated equations of state.

This practical involves implementing a tabulated equation of state for plasma in a one-dimensional Euler equation evolution code. Although this equation of state allows for material conductivity, and hence magnetic fields, it is also valid for flow without a magnetic field, and this offers an ideal test case for implementing this equation of state.

There are two key challenges to this implementation; firstly the tabulated equation of state needs to be read in to the code, and secondly, interpolation routines need to be written for conversion between variables.

The plasma 19 file:

The equation of state file itself does not contain a lot of useful identifying information, since this makes it easier to parse data knowing that only numbers exist within the file. However, it does mean you need to know what is in the file! Here, the file contains the following information (in this order):

- Number of densities (ND), number of pressures (NP), number of species (NS)
- A list of specific gas constants of each of the species [J/(kg K)] (NS of them)
- A list of heats of formation of each of the species (NS of them)
- A list of vibrational temperatures of each of the species [K] (NS of them)
- A list of the densities [kg/m^3] (ND of them)
- A list of the pressures [Pa] (NP of them)
- A table of sound speeds [m/s]
- A table of specific internal energies [J/kg]
- A table of temperatures [K]
- A table of electrical conductivities [1/(Ω m)]
- NS tables of mass fractions
- A table of thermal conductivities [W/(m K)]

Each table is of the format ND $\times$ NP, i.e. ND rows, each with NP columns, each row corresponding to a different pressure value, and each column to a different density value.

For this practical, only the densities, pressures, sound speeds and specific internal energies are actually needed. Though other quantities will need to be read in, if only to be discarded.

Data should can be stored for each variable separately, density and pressure (and the other list variables) as an `std::vector<double>` and tables as an `std::vector < std::vector < double > >;`. When reading in tabulated data, read in with an outer loop over pressure indices, and an inner loop over density indices, giving a structure of `variable[pressureIndex][densityIndex]`.

Converting between variables:

Converting between variables is essentially straightforward - all variables exist on a rectangular grid in pressure and density. It is assumed that between any two grid values, there is a linear change in the variable value. We are then simply working out where a given variable sits, between two tabulated values, either on a 1D grid or a 2D grid.

For all variable calculations, we will know two variables, and need to calculate a third. There are three possibilities here:

1. We know pressure and density, the two one-dimensionally stored variables, and need one of the tables of variables. This is most likely used to compute energy:
   (a) For pressure, identify the two values in the vector which are closest to the given value (one lower, one higher) - these are indices np_1 and np_2
   (b) Use a linear interpolation to give a fractional index np_p, i.e. where your given pressure sits between np_1 and np_2
   (c) Repeat for density, to obtain nd_ρ which sits between nd_1 and nd_2
   (d) For the given table of data, make a bilinear interpolation. For this, you will need the equation from slide 30 of lecture 4, and you are computing e.g. $e(p, \rho)$, not $\rho(p, T)$
2. We know density and energy, and are computing pressure. This is the only time we need to compute a one-dimensionally stored variable, as we always know density:
   (a) Compute the density fractional index, nd_ρ, as above

(b) This uniquely identifies the columns in which our energy lies between. That is, the value we have must have come from somewhere between columns nd_1 and nd_2

(c) We now need to find the point at which a bilinear interpolation from the pressure rows, and at nd_ρ, will give the energy we have. Here, we are trying to find np_p. This comes down to guessing values for np_p, such that we get back our energy. This will need an iterative procedure, making guesses, and refining the guesses - a bilinear interpolation is the best option.

(d) Once we have worked out np_p, we use a regular linear interpolation to get this value from the pressure vector values.

3. Finally, we may know density and energy, and want to compute e.g. sound speed. We could do this by first computing pressure (case 2) and then use this to compute sound speed (as in case 1). However, we can do it in one step:

   (a) Follow case 2 until you have obtained nd_ρ and np_p

   (b) With these values, you are now ready to use the bilinear interpolation (part (d) of case 1) to compute the desired variable

When computing these functions, it is easy to end up with bugs and off-by-one errors. Fortunately, it is normally also easy to identify these, for example, the first time you compute nd_ρ, you can output the values of the density array at indices nd_1 and nd_2, and check that your density really does lie between these variables.

Exercises:

The hard work of this practical is getting the data interpolation correct. It will then be used for some familiar looking tests. However, the first step is to make sure that it really is correct. Although this is a plasma equation of state, it is one valid from ambient conditions to partial ionisation; this means it is also capable of modelling regular air. This means you can test this in your code by:

1. Assigning a primitive variable state to be atmospheric conditions $(\rho, \mathbf{v}, p) = (1.225, \mathbf{0}, 101325)$

2. Convert this to conserved variables, and back to primitive variables. Do you recover the initial state?

3. Also, compute sound speed from your conserved state, do you get something around 340 m/s (it won't necessarily be exact, but it should be within about 10 m/s)

Once you are reasonably confident it is working, the tabulated equation of state can be used for Toro's tests (1, 3-5 anyway, test 2 may leave the range of validity as it approaches vacuum). In order to use this equation of state, though, we do need to re-dimensionalise the variables in these equations, as a pressure of 1 Pa is not a physical ambient condition.

Re-dimensionalising density is straightforward (don't do anything), and velocity is fine for the first few tests, since zero doesn't need re-dimensionalising. However, the final test does need work, as we need to re-dimensionalise time. This comes from our choice of normalisation, 1 atm$\rightarrow$1. This divides pressure by 10^5 Pa (or 101,325 Pa if you are being accurate). The units of pressure, in SI base units, are $\mathrm{kg\,m\,s^{-2}}$, and we are not changing the definitions of metres or kilograms. Therefore, time is normalised by dividing through by $\sqrt{10^5}$, and velocity by multiplying by this factor.

With these initial data (and final time) changes, the code can be run, and should the generate results you have seen before. The exercise part of this practical is then to check this; start with test 1, then look at 3,4 and 5. You should find that for test 1, at least, you get plots which are indistinguishable when viewed side-by-side, though may differ slightly when overlaid. Why are there these slight differences?