Advanced Continuum Modelling
Practical 1: Two-dimensional Euler equations part 1

This practical begins to apply the methods from Computational Continuum Modelling for two-dimensional numerical methods to the Euler equations. A second-order numerical scheme (e.g. SLIC) should be used, and a dimensionally split numerical method should be implemented.

Step 1: Two dimensional equations in a 1D solver

The first step is to add an additional variable, the y-momentum, into your system of conservation equations. At this stage, we will keep the solver one-dimensional; physically, we allow for a non-zero y-velocity, but this must be constant in the y-direction.

This initial step ensures that the modifications to data storage, flux functions and numerical flux updates deal suitably with the additional variable.

Apart from the changes to allow for an additional variable, the numerical method itself should remain unchanged.

Exercises:

Using Toro's tests 1 and 3:

1. Check that your code still produces the same results with $v_y = 0$
2. Give v_y a constant value, does anything change in the test results?

Step 2: Two dimensional grid but without y-update

The next step is to start solving on a two-dimensional domain, but we will continue to ignore x-derivatives. This tests whether the various loops across dimensions continue to work.

A suitable data structure for 2D data is:

```
std::vector < std::vector < std::array<double, 4> > > u;
```

It is up to you how order the x- and y- vectors, for readability, I would choose the first vector object to be x-components, so you can refer to $\mathbf{u}_{i,j}$ =`u[i][j][var]`.

When running this code, choosing the number of cells you use is more important now, $\mathcal{O}(1000)$ cells is fine in 1D, but $\mathcal{O}(1000) \times \mathcal{O}(1000)$ will take a very long time to run. In 2D simulations, 800×800 cells is considered a **high** resolution (hence the need for second-order accuracy!). Make sure

you compile with optimisation (compiler flag `-O3` - that's a capital letter 'O' there, not a zero), unless you plan to run the code through a debugger.

Exercises:

Repeat the tests described above, but with the two-dimensional version. Results should be unchanged.