Going beyond one dimension

Stephen Millmore, Louisa Michael and Nikos Nikforakis

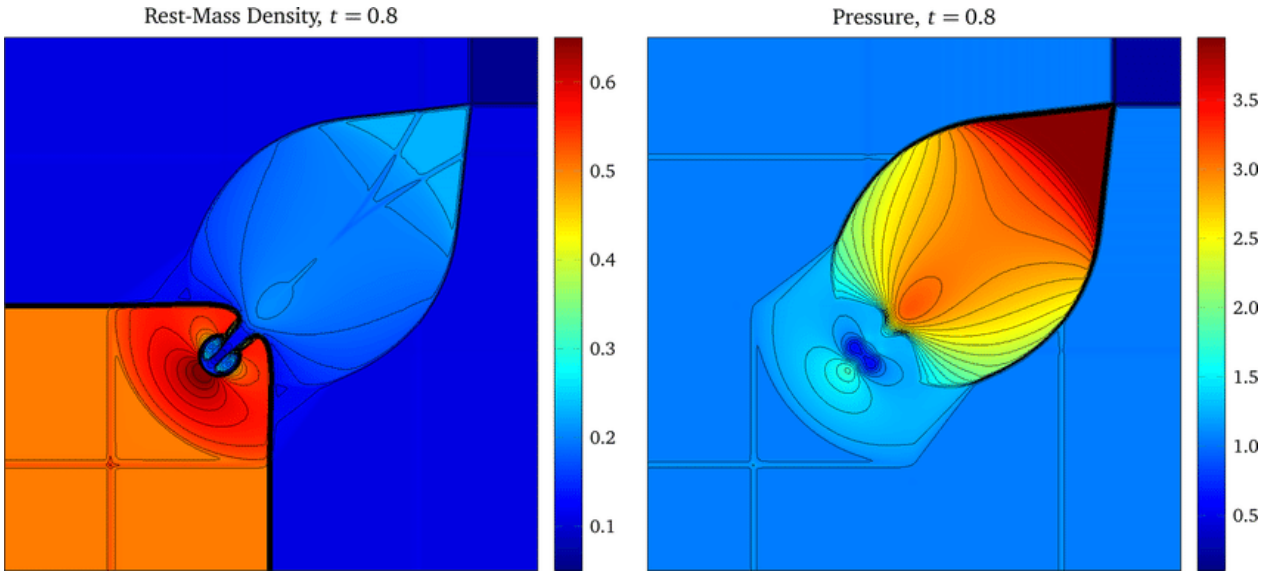
Laboratory for Scientific Computing, University of Cambridge

Outline

Multi-dimensional simulations

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0$$

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ E \end{pmatrix}$$



Outline

Two dimensional Euler equations

- We shall start by considering just two dimensions
- The first obvious change is that velocity and momentum are now **vector quantities**
- Based on our discretisation decision, way back in the first lecture, under a Cartesian representation, we now consider x-velocity and y-velocity (v_x and v_y)
- We are going to need evolution equations for both velocity components, and we are also going to need to consider evolution in the y-direction
- Recall, we first introduced the Euler equations as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + \mathbf{I}p) = 0$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p)\,\mathbf{v}] = 0$$

Conservation of mass - revisited

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}\left(\rho v_x\right) + \frac{\partial}{\partial y}\left(\rho v_y\right) = 0$$

- If we consider a two-dimensional square (or rectangular) control volume, then in the x-direction, the flow of mass is given only by ρv_x
- Similarly, in the y-direction, we have mass flow given by ρv_y

Conservation of momentum - revisited

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho v_x\\ \rho v_y\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho v_x^2+p\\ \rho v_x v_y\end{pmatrix}+\frac{\partial}{\partial y}\begin{pmatrix}\rho v_x v_y\\ \rho v_y^2+p\end{pmatrix}=0$$

- We now have two equations for the conservation of momentum, one for each component
- Each term has a component which is the momentum flow through the boundaries of the control volume
- For example, x-momentum through the x-boundary is $(\rho v_x)\, v_x$ and through the y-boundary is $(\rho v_x)\, v_y$
- The pressure term (the 'push' at the boundary) acts only in the **normal direction**, hence is only present in the x-component of ρv_x and the y-component of ρv_y

Conservation of energy - revisited

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x}\left[\left(E + p\right) v_x\right] + \frac{\partial}{\partial y}\left[\left(E + p\right) v_y\right] = 0$$

- Energy conservation is affected only by flow across the boundary - the extension of the equation to two dimensions is as expected
- However, the total energy is a combination of internal energy and **kinetic energy**
- Recall the definition of kinetic energy

$$\mathrm{KE} = \frac{1}{2}\rho v^2 = \frac{1}{2}\rho \mathbf{v} \cdot \mathbf{v}$$

- The total energy contains a contribution from both x- and y-velocity

Vector form of the 2D equations

- There are now two different flux vectors, one for each direction
- Notation for the vector form of the equations is as might be expected:

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial}{\partial x}\mathbf{f}\left(\mathbf{u}\right) + \frac{\partial}{\partial y}\mathbf{g}\left(\mathbf{u}\right) = 0$$

- Note that it is not uncommon to see a vector quantity within the conserved variable vector

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ E \end{pmatrix} = \begin{pmatrix} \rho \\ \rho \mathbf{v} \\ E \end{pmatrix}$$

- This second form is also valid, no matter how you've discretised your domain

Waves in the 2D Euler equations

- How many waves do we expect in a solution to the 2D Euler equations?
- There are two commonly stated means to determine the number of waves in a system of equations:
  1. One wave for each equation
  2. One wave for each independent eigenvalue
- For the one-dimensional Euler equations, we have three equations, three independent eigenvalues and we saw three waves in our Riemann problem solutions
- In 2D do we expect four waves?
- Waves might be harder to distinguish in 2D - for example, how do we represent an x-t diagram, or an x-y-t diagram?
- To achieve some understanding, it is common to consider the behaviour along a **one-dimensional slice** through the domain

Waves in the 2D Euler equations

- Without loss of generality, we shall consider a slice in the x-direction, reducing our equations to

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho v_x\\ \rho v_y\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho v_x\\ \rho v_x^2+p\\ \rho v_x v_y\\ (E+p)\,v_x\end{pmatrix}=0$$

- In order to compute the eigenvalues, it is again easiest to use the primitive variable form
- Fortunately, we did most of the work for this for the 1D Euler equations, we only need to alter the ρv_y equation
- This time we have

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ v_x\\ v_y\\ p\end{pmatrix}+\begin{pmatrix}v_x & \rho & 0 & 0\\ 0 & v_x & 0 & \frac{1}{\rho}\\ 0 & 0 & v_x & 0\\ 0 & \rho c_s^2 & 0 & v_x\end{pmatrix}\frac{\partial}{\partial x}\begin{pmatrix}\rho\\ v_x\\ v_y\\ p\end{pmatrix}=0$$

Waves in the 2D Euler equations

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ v_x\\ v_y\\ p\end{pmatrix}+\begin{pmatrix}v_x & \rho & 0 & 0\\ 0 & v_x & 0 & \frac{1}{\rho}\\ 0 & 0 & v_x & 0\\ 0 & \rho c_s^2 & 0 & v_x\end{pmatrix}\frac{\partial}{\partial x}\begin{pmatrix}\rho\\ v_x\\ v_y\\ p\end{pmatrix}=0$$

- The eigenvalues of this matrix still straightforward

$$\lambda_1 = v_x - c_s, \quad \lambda_2 = \lambda_3 = v_x, \quad \lambda_4 = v_x + c_s$$

- We still have four **real** eigenvalues, but we do not have four **distinct** eigenvalues - the system is hyperbolic, but not strictly hyperbolic
- This can have mathematical implications on the uniqueness of the solution - the matrix term may not be diagonalisable; however, in this case it still is
- However, what it does mean is that there are not four **distinct** waves in the solution to the 2D Euler equations

Degenerate waves

- If we have multiple waves moving at the same speed, they are indistinguishable in the solution itself, they are **degenerate waves**
- Although indistinguishable in the solution, they have mathematical properties which requires that they are treated separately
- One example, for the Euler equations, is finding the characteristic behaviour, and as a result, the solution to the Riemann problem for the 2D system
- If we repeat the derivation of the characteristic variables shown for the one-dimensional equations, including our extra variable, we get:

$$\mathrm{d}\mathbf{v} = \begin{pmatrix} \mathrm{d}v_0 \\ \mathrm{d}v_+ \\ \mathrm{d}v_- \\ \mathrm{d}v_{\mathrm{sh}} \end{pmatrix} = \begin{pmatrix} \mathrm{d}\rho - \frac{\mathrm{d}p}{c_s^2} \\ \mathrm{d}v_x + \frac{\mathrm{d}p}{\rho c_s} \\ \mathrm{d}v_x - \frac{\mathrm{d}p}{\rho c_s} \\ \mathrm{d}v_y \end{pmatrix} \quad \text{along} \quad \begin{aligned} \mathrm{d}x &= v_x \mathrm{d}t \\ \mathrm{d}x &= (v_x + c_s)\,\mathrm{d}t \\ \mathrm{d}x &= (v_x - c_s)\,\mathrm{d}t \\ \mathrm{d}x &= v_x \mathrm{d}t \end{aligned}$$

Shear waves

- We labelled the fourth characteristic variable $\mathrm{d}v_{\mathrm{sh}}$ since this is the characteristic associated with a **shear wave**
- Conceptually, this is just a wave in which a velocity moving perpendicular to the wave, a **transverse velocity**, jumps, but the other variables do not
- Care is taken to separate the description of contact discontinuities and shear waves because the wave degeneracy is a special feature of the inviscid Euler equations
- If there was any viscosity between materials, the tangential flows would interact (though this would be a diffusive behaviour)
- Magnetic fields (and plasma) also results in a separation of these waves, as we shall see in "Simulation of Matter under Extreme Conditions"

Shear waves

- We now also have an additional variable to consider in Riemann problem solutions
- However, Because the transverse velocity (v_y in our example) effectively decouples from the other quantities in the characteristic variables, then in the Riemann problem, across the shear wave is the **only** place the transverse velocity jumps
- This means that the Riemann problem solution is no more challenging for the 2D Euler equations than it is for the 1D equations

$$\mathbf{w}_L^* = \begin{pmatrix} \rho_L^* \\ v_x^* \\ v_{y,L} \\ p^* \end{pmatrix} \qquad \mathbf{w}_R^* = \begin{pmatrix} \rho_R^* \\ v_x^* \\ v_{y,R} \\ p^* \end{pmatrix}$$

- The quantities ρ_K^*, v_x^* and p^* are calculated exactly as seen previously

A general note on degenerate waves

- The shear wave and the contact discontinuity always move at the same speed, there is always degeneracy in this wave
- When discussing multi-dimensional Euler equations, it is common for laziness to take over, and describe a jump in transverse velocity at a contact discontinuity
- With the correct initial data, other waves in a Riemann problem solution can **also be degenerate**
- For example, if you set up a Riemann problem, with states $\mathbf{u}_L$ and $\mathbf{u}_R$, you can compute $\mathbf{u}_L^*$ and $\mathbf{u}_R^*$
- What would happen if you set up a **new** Riemann problem, with states $\mathbf{u}_L$ and $\mathbf{u}_L^*$?
- Your results would show a single wave; other waves would exist mathematically, with a zero-height jump across the waves

Is this 1D slice a general approach?

- Why have we carried out our analysis of the 2D Euler equations using only the x-derivative?
- In general, 2D solutions to the Euler equations exhibit complex behaviour, not because there are more waves present than we have considered so far, but because of the **wave interactions**
- Effectively we are considering a system reduced to the extent that we can identify individual waves
- The choice to align the wave direction with the x-direction is a convenient coordinate transformation, the Euler equations are rotationally invariant (see Toro for more details)
- What's more, when solving the equations on a Cartesian grid, our numerical method will **always** be considering flow decomposed into components aligned with the coordinate axes

Outline

Numerical methods for the 2D Euler equations

- It appears to be reasonably obvious how we can apply a numerical method to the Euler equations

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = 0$$

$$\mathbf{u}_{i,j}^{n+1} = \mathbf{u}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}_{i+1/2,j}^{n} - \mathbf{f}_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathbf{g}_{i,j+1/2}^{n} - \mathbf{g}_{i,j-1/2}^{n}\right)$$

- The computational grid is now divided up into rectangular cells of size $\Delta x \times \Delta y$
- Convention is that each cell is labelled $\mathbf{u}_{i,j} = \mathbf{u}\left(x_i, y_j\right)$
- The x-fluxes are those that pass through $x_{i\pm 1/2}$, and the y-fluxes are those that pass through $y_{j\pm 1/2}$, hence the notation $\mathbf{f}_{i\pm 1/2,j}$ and $\mathbf{g}_{i,j\pm 1/2}$
- Note that the x-fluxes are **effectively one dimensional** (constant j-index), as are the y-fluxes (constant i-index)

Fluxes for the 2D Euler equations

$$\mathbf{u}_{i,j}^{n+1} = \mathbf{u}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}_{i+1/2,j}^{n} - \mathbf{f}_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathbf{g}_{i,j+1/2}^{n} - \mathbf{g}_{i,j-1/2}^{n}\right)$$

- Any numerical method seen so far can be used for the 2D Euler equations
- In the x-direction, the flux is computed from cells in the i-direction, e.g. $\mathbf{f}_{i+1/2,j}^{\mathrm{FORCE}} = \mathbf{f}_{i+1/2,j}^{\mathrm{FORCE}}(\mathbf{u}_{i,j}^{n}, \mathbf{u}_{i+1,j}^{n})$ and

$$\mathbf{f}\left(\mathbf{u}\right) = \left(\rho v_x, \rho v_x^2 + p, \rho v_x v_y, (E+p)v_x\right)^T$$

- In the y-direction, the flux is computed from cells in the j-direction, e.g. $\mathbf{g}_{i,j+1/2}^{\mathrm{FORCE}} = \mathbf{g}_{i,j+1/2}^{\mathrm{FORCE}}(\mathbf{u}_{i,j}^{n}, \mathbf{u}_{i,j+1}^{n})$ and

$$\mathbf{g}\left(\mathbf{u}\right) = \left(\rho v_y, \rho v_x v_y, \rho v_y^2 + p, (E+p)v_y\right)^T$$

Fluxes for the 2D Euler equations

$$\mathbf{u}_{i,j}^{n+1} = \mathbf{u}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}_{i+1/2,j}^{n} - \mathbf{f}_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathbf{g}_{i,j+1/2}^{n} - \mathbf{g}_{i,j-1/2}^{n}\right)$$

- For both x- and y-fluxes, the calculation is treated as entirely one-dimensional
- For $\mathbf{f}_{i\pm1/2,j}^{n}$, only Δx needs to be used for computing slopes for limiters, and for the half time step update for slope limited schemes
- Similarly, $\mathbf{g}_{i,j\pm1/2}^{n}$, only Δy needs to be used for these grid-dependent quantities
- Only one-dimensional Riemann problem solvers are needed too
- There is no requirement that $\Delta x = \Delta y$, though this is often desirable. Why?
- Is this all that is needed to run 2D simulations?

Time step constraints

$$\mathbf{u}_{i,j}^{n+1} = \mathbf{u}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}_{i+1/2,j}^{n} - \mathbf{f}_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathbf{g}_{i,j+1/2}^{n} - \mathbf{g}_{i,j-1/2}^{n}\right)$$

- Clearly, we need the same time step for both x- and y-updates
- We also need the system to be stable for both of these updates, i.e.

$$\Delta t = C\frac{\min\left(\Delta x, \Delta y\right)}{a_{\max}}$$

- The maximum wave speed needs to be calculated correctly, there is no guarantee that this is aligned with any of the coordinate axes
- Instead we need to consider the magnitude of the velocity vector

$$a_{\max} = \max_{i,j}\left(|\mathbf{v}_{i,j}| + c_{s,i,j}\right)$$

- What about the CFL number, if our 1D method has $C_{1\mathrm{D}} \leq 1$, does our 2D method work the same way?

Dealing with diagonal movement

$$\mathbf{u}_{i,j}^{n+1} = \mathbf{u}_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}_{i+1/2,j}^{n} - \mathbf{f}_{i-1/2,j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathbf{g}_{i,j+1/2}^{n} - \mathbf{g}_{i,j-1/2}^{n}\right)$$

- If $v_x \neq 0$ and $v_y \neq 0$, then movement of material is not aligned to the grid, and therefore, over a single time step, material from cell (x_i, y_j) will end up in cells $(x_{i\pm1}, y_{j\pm1})$
- Does the scheme at the top of this slide allow this?
- It will take two time steps for material to move into cells $(x_{i\pm1}, y_{j\pm1})$
- Practically, we find that (and can show that) the stable time step must be halved to ensure that material can reach these offset cells, whilst not being able to cross more than a single cell in areas where velocity is aligned to the grid
- In other words, the CFL number for this scheme is

$$C = \frac{1}{2} C_{\mathrm{1D}}$$

What are our options?

- Having to reduce the CFL number in order to get stable 2D solutions isn't ideal
- Fortunately, it is also not necessary
- One option is to design a numerical scheme which specifically takes account of the waves moving diagonally
- Versions of MUSCL-Hancock, Weighted Average Flux (WAF) and Wave Propagation methods exist to cope with this
- These carefully consider wave movement through all possible cell vertices over a time step (including the case where material crosses both x- and y-vertices)
- However, these methods are not straightforward to implement - keeping track of information moving diagonally is tricky
- We won't go into more details - I had promised you that we would be able to use the methods we studied in Computational Continuum Modelling (and that I wouldn't introduce any more methods)

Dimensional splitting

- The approach for solving the 2D Euler equations so far is referred to as a **dimensionally unsplit** approach
- An alternative is to update the x-direction first, and then update the y-direction using the results of the x-update

$$\bar{\mathbf{u}}_{i,j} = \mathbf{u}^n_{i,j} - \frac{\Delta t}{\Delta x}\left(\mathbf{f}^n_{i+1/2,j}\left(\mathbf{u}\right) - \mathbf{f}^n_{i-1/2,j}\left(\mathbf{u}\right)\right)$$

$$\mathbf{u}^{n+1}_{i,j} = \bar{\mathbf{u}}_{i,j} - \frac{\Delta t}{\Delta y}\left(\mathbf{g}^n_{i,j+1/2}\left(\bar{\mathbf{u}}\right) - \mathbf{g}^n_{i,j-1/2}\left(\bar{\mathbf{u}}\right)\right)$$

- This is a **dimensionally split** approach
- This approach is **consistent** with the underling PDE (considering the limit $\Delta x \to 0$, $\Delta y \to 0$)
- However, this method also allows diagonal movement of information over a single time step
- We can show that the underlying CFL number is that of the method used for the 1D fluxes (i.e. we can get $c = 1$)

Notation and order

- Clearly solving in the x-direction, and then the y-direction is not the only way dimensional splitting could be done
- When denoting a scheme being solved using a splitting technique, you might see the method from the previous slide written as

$$\mathbf{u}_{i,j}^{n+1} = \mathcal{Y}^{(\Delta t)} \mathcal{X}^{(\Delta t)} \left(\mathbf{u}^n\right)$$

- Here, $\mathcal{X}^{(\Delta t)}$ and $\mathcal{Y}^{(\Delta t)}$ are flux update operators which operate over Δt
- They are resolved right-to-left, i.e. $\mathcal{Y}^{(\Delta t)}$ acts on $\mathcal{X}^{(\Delta t)} \left(\mathbf{u}^n\right)$, with

$$\mathcal{X}^{(\Delta t)} \left(\mathbf{u}^n\right) = \bar{\mathbf{u}}$$

- It is just as valid to perform the dimensional splitting through

$$\mathbf{u}_{i,j}^{n+1} = \mathcal{X}^{(\Delta t)} \mathcal{Y}^{(\Delta t)} \left(\mathbf{u}^n\right)$$

- However, both of these techniques result in a method that is first order accurate **in time** - is this a problem?

Choice of splitting technique

- Having a scheme that is first-order in time might sound like a bad idea, but, it is (usually) the errors in the spatial discretisation that dominate
- Therefore, first-order in time, second-order in space methods will show second order convergence, because it is these largest errors which are being reduced at the higher order of accuracy
- However, it is still possible to obtain second-order accuracy in your dimensional splitting

$$\mathbf{u}_{i,j}^{n+1} = \frac{1}{2}\left[\mathcal{Y}^{(\Delta t)}\mathcal{X}^{(\Delta t)} + \mathcal{X}^{(\Delta t)}\mathcal{Y}^{(\Delta t)}\right](\mathbf{u}^n)$$

$$\mathbf{u}_{i,j}^{n+1} = \mathcal{X}^{\left(\frac{1}{2}\Delta t\right)}\mathcal{Y}^{(\Delta t)}\mathcal{X}^{\left(\frac{1}{2}\Delta t\right)}(\mathbf{u}^n)$$

$$\mathbf{u}_{i,j}^{n+2} = \frac{1}{2}\left[\mathcal{Y}^{(\Delta t)}\mathcal{X}^{(\Delta t)}\mathcal{X}^{(\Delta t)}\mathcal{Y}^{(\Delta t)}\right](\mathbf{u}^n)$$

- It is common to see this sort of splitting referred to as Strang splitting, due to the work of Strang on these techniques
- Note that these methods are more computationally expensive than the first-order splitting (first and second) or require the same time step for two consecutive iterations (third)

Split or unsplit

- Because split methods are easier to implement with $C = 1$ stable schemes, it may seem like there is no reason to use unsplit methods
- Whenever solving simulations on a grid, it is possible for some behaviour to become grid-orientated
- On a Cartesian grid, this can be worse for split methods than unsplit methods
- Additionally, techniques exist in which the flux behaviour within a cell may need to be altered - cut cell methods, embedding a complex geometry within a Cartesian mesh, are an example
- Some complex systems of equations can suffer stability issues in a split treatment, unsplit might be useful here too
- But unless you know you need them, if you have a Cartesian mesh, split methods are the best choice to start with

Coding in two dimensions

- We now have a trickier storage situation for storing an $x \times y \times 4$ data structure
- A vector of vectors of arrays is probably the most useful

```
std::vector< <std::vector< std::array<double, 4> > > u;
u.resize(nxPoints, std::vector<std::array<double, 4> > nyPoints);
```

- This notation allows for access through

```
u[i][j][var] // var_{i,j};
```

- Other options are available (e.g. a single $x \times y$ vector) but keeping track of indices could be tricky
- You may wish to introduce specific x- and y-directional functions, or you may want to pass coordinate direction to functions (e.g. for computing fluxes)

Coding in two dimensions - example loop

```
for(int i = 0; i < nxCells+1; i++) {
  for(int j = 0; j < nyCells+1; j++) { //Define the fluxes
    flux[i][j] = getXFlux(u[i][j],u[i+1][j]);
  }
}
for(int i = 1; i < nCells+1; i++) {
  for(int j = 1; j < nyCells+1; j++) { //Update the data
    uBar[i][j] = u[i][j] - (dt/dx) * (flux[i+1][j] -
    flux[i][j]);
  }
}
for(int i = 0; i < nxCells+1; i++) {
  for(int j = 0; j < nyCells+1; j++) { //Define the fluxes
    flux[i][j] = getYFlux(uBar[i][j],uBar[i][j+1]);
  }
} // Can reuse flux here
for(int i = 1; i < nCells+1; i++) {
  for(int j = 1; j < nyCells+1; j++) { //Update the data
    u[i][j] = uBar[i][j] - (dt/dx) * (flux[i][j+1] -
    flux[i][j]);
  }
} // Can overwrite u here without errors
```

Outline

Three-dimensional Euler equations

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho v_x\\ \rho v_y\\ \rho v_z\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho v_x\\ \rho v_x^2+p\\ \rho v_x v_y\\ \rho v_x v_z\\ (E+p)\,v_x\end{pmatrix}+\frac{\partial}{\partial y}\begin{pmatrix}\rho v_y\\ \rho v_y v_x\\ \rho v_y^2+p\\ \rho v_y v_z\\ (E+p)\,v_y\end{pmatrix}+\frac{\partial}{\partial z}\begin{pmatrix}\rho v_z\\ \rho v_z v_x\\ \rho v_z v_y\\ \rho v_z^2+p\\ (E+p)\,v_z\end{pmatrix}=0$$

- Having seen how the two-dimensional Euler equations were derived, hopefully the route to these three-dimensional equations is clear
- We needed to do a reasonable amount of work to go from 1D to 2D - do we need to do this all over again to go to 3D?

Three-dimensional Euler equations

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial}{\partial x}\mathbf{f}(\mathbf{u}) + \frac{\partial}{\partial y}\mathbf{g}(\mathbf{u}) + \frac{\partial}{\partial z}\mathbf{h}(\mathbf{u}) = 0$$

- The jump from two to three dimensions is a lot more straightforward:
  - There is yet another eigenvalue, and again it is a repeated value, very similar to the previous new value

$$\lambda_1 = v_x - c_s, \quad \lambda_2 = \lambda_3 = \lambda_4 = v_x, \quad \lambda_5 = v_x + c_s$$

  - The new wave is a **second shear wave** (parallel to the first and the normal direction)
  - Dimensional splitting works as expected, and second-order in time splits exist, e.g.

$$\mathbf{u}_{i,j}^{n+1} = \mathcal{X}^{\left(\frac{1}{2}\Delta t\right)}\mathcal{Y}^{\left(\frac{1}{2}\Delta t\right)}\mathcal{Z}^{(\Delta t)}\mathcal{Y}^{\left(\frac{1}{2}\Delta t\right)}\mathcal{X}^{\left(\frac{1}{2}\Delta t\right)}\left(\mathbf{u}^n\right)$$

  - The CFL number for a naively implemented unsplit scheme is now reduced by a third (flow into corner vertices)

A practical note on higher dimensions

- When coding the two and three dimensional Euler equations (or any system greater than 1D), code run time, memory usage and data storage become much more important considerations
- The run time scaling is fairly obvious - if we are running a simulation in 1D with 100 cells, and wish to run at the same resolution in 2D, we need $100^2 = 10{,}000$ cells, and $100^3 = 1{,}000{,}000$ in 3D
- In other words, each increase in dimensionality is an order of magnitude increase in run time
- Memory usage and (simple) data storage follow this pattern
- Optimisation is more important for these higher-dimensional codes
- When testing your methods, try to stick to one-dimensional and pseudo-one-dimensional tests for as long as possible, e.g. using one cell in the y-direction, 100 in the x-direction, and then reversing this pattern, to check your x- and y-flux calculations are working as expected