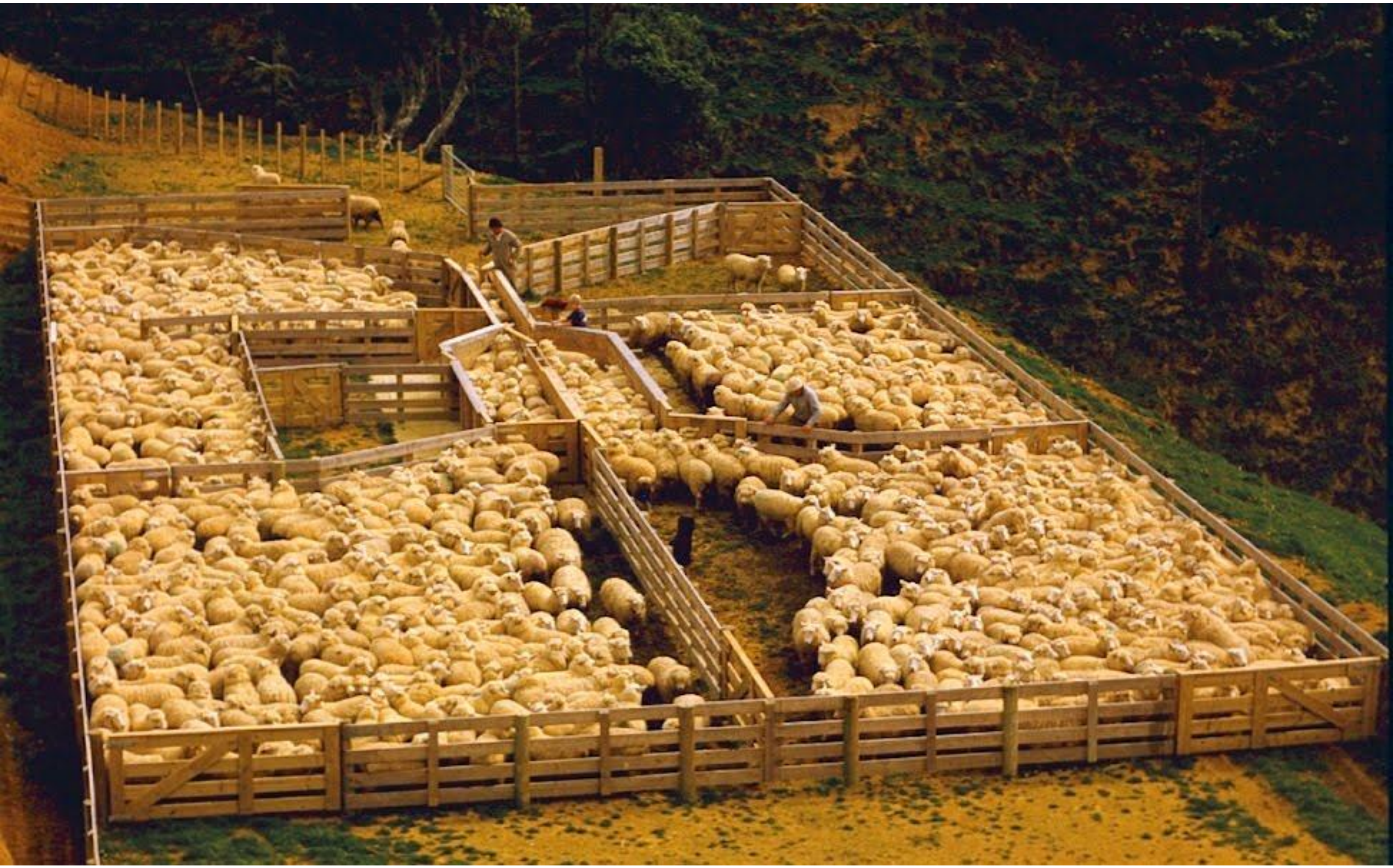
Finite volume schemes

An introduction

PART 2

Bert Vandenbroucke, bv7@st-andrews.ac.uk



http://woolshed1.blogspot.co.uk/2010/02/sheep-yards-design-and-construction.html

Note that the entire derivation until now was *exact*, if we can guarantee that the fluid quantities are continuous inside a cell, and that the cell volume does not change with time.

$$\frac{\partial Q_i}{\partial t} = -\iint \underline{F}(U).\,\mathrm{d}\underline{S_i}$$

To convert the equation above into a useful numerical method, we will need to make some *approximations*

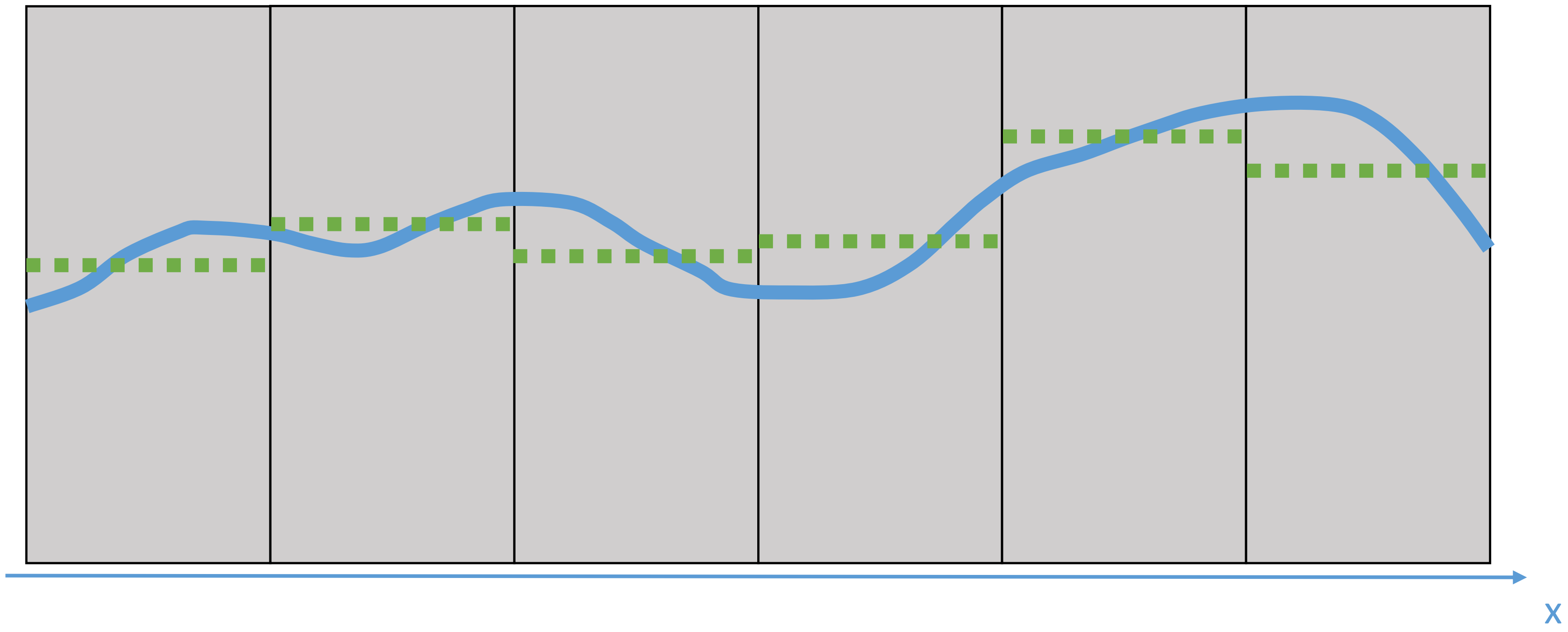
about how to obtain useful fluxes

about how to approximate the surface integral

about how to deal with discontinuities in the fluid quantities (shocks, contact discontinuities)

about how to discretize time

Obtaining useful fluxes



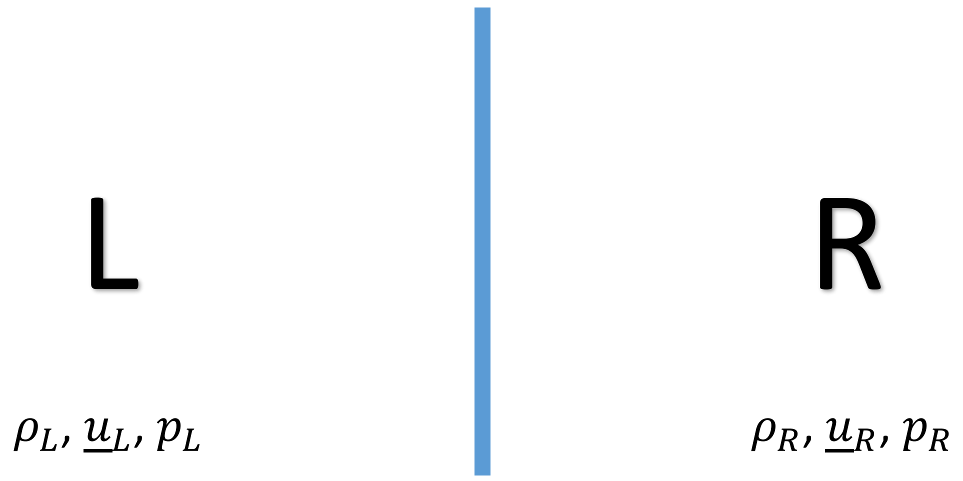
We need to evaluate the fluxes at the boundary between cells.

However, our integration scheme will only be *stable* if we respect the natural flow direction of the fluid, a so called *upwind* scheme.

This means we cannot simply use the average of the fluid quantities on both sides of the boundary.

Obtaining useful fluxes: the Riemann problem

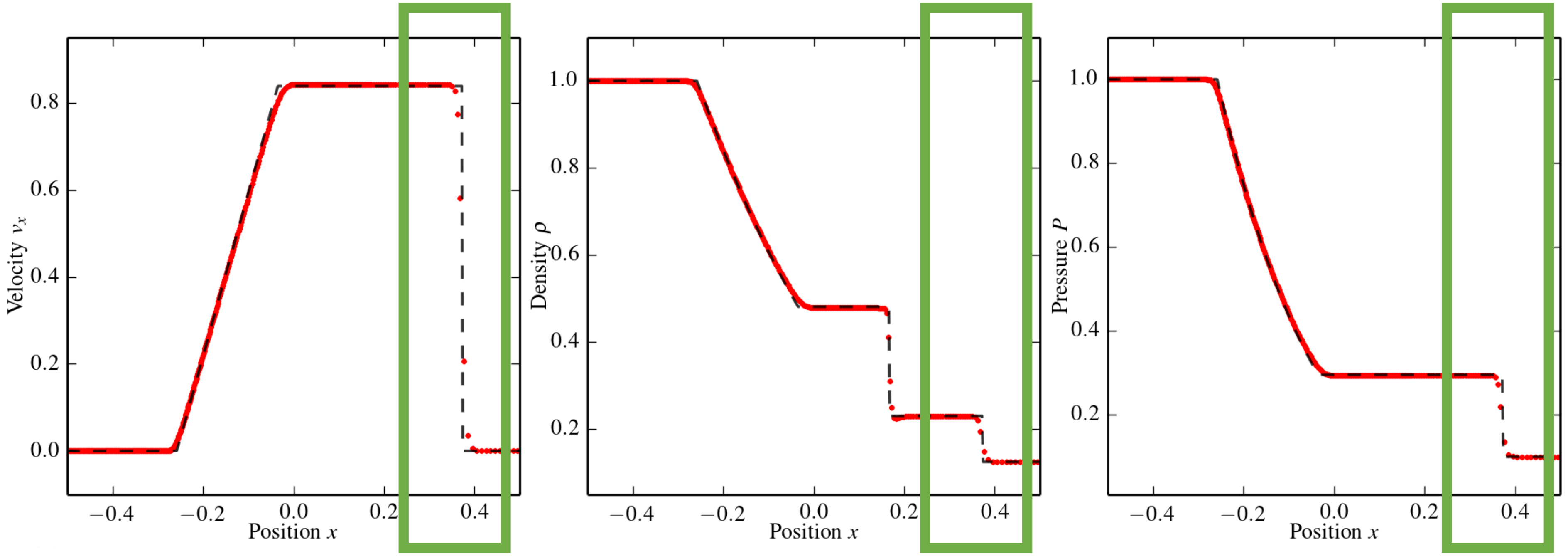
What is the natural flow direction in a two state problem?


L
R
$\rho_L, \underline{u}_L, p_L$
$\rho_R, \underline{u}_R, p_R$

Physical acceptable solutions contain three distinct waves:

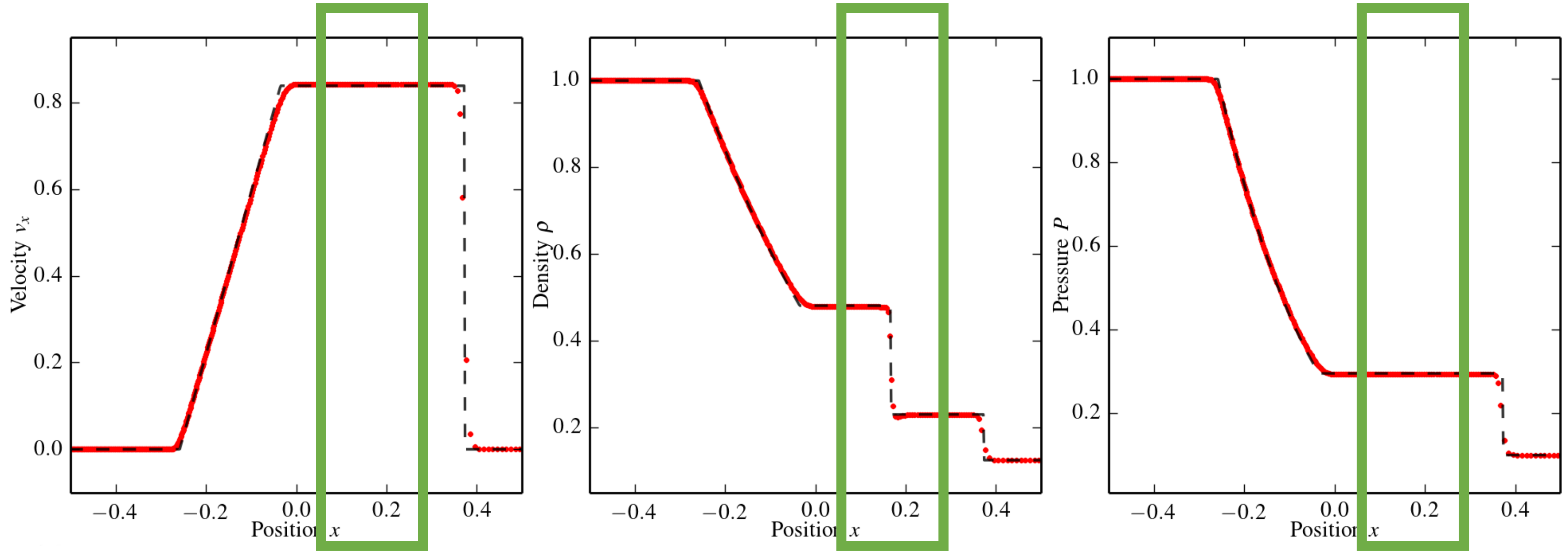
- a left rarefaction or shock wave
- a central contact discontinuity
- a right rarefaction or shock wave

Wave types: shock wave



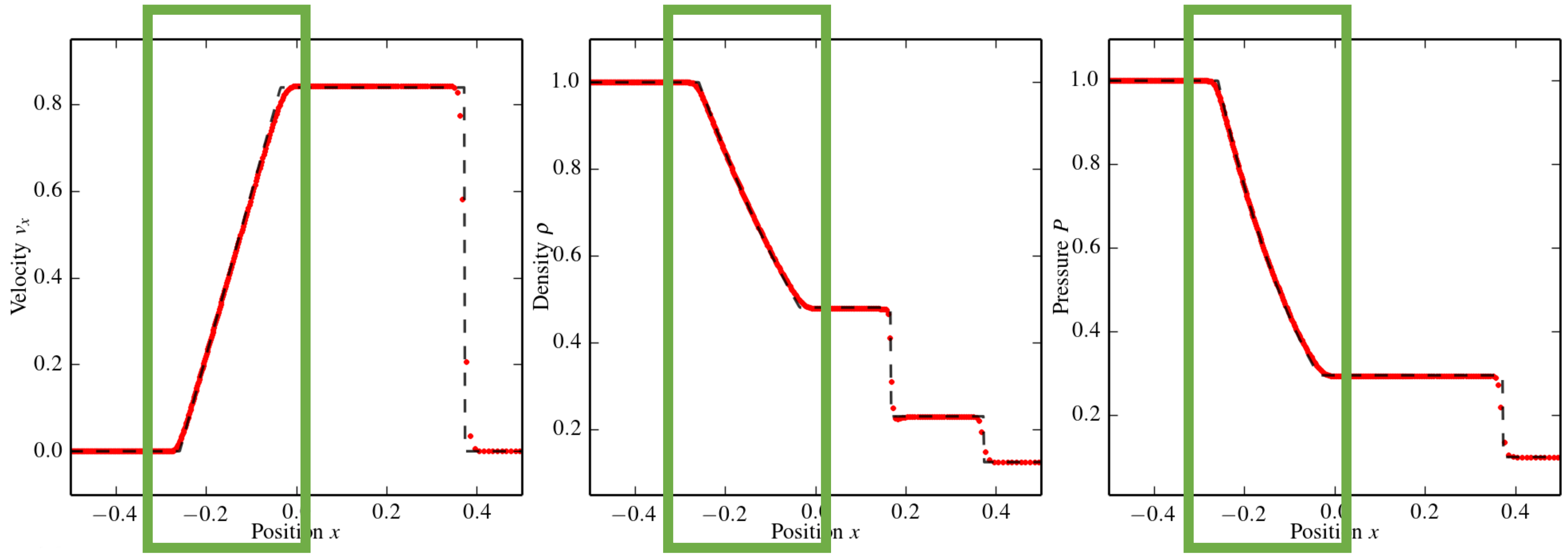
Strong discontinuity in density, pressure and fluid velocity.
Supersonic movement caused by a strong pressure or velocity gradient.

Wave types: contact discontinuity



Discontinuity in density, pressure and fluid velocity are constant.
Moving with the local fluid velocity.

Wave types: rarefaction wave



Continuous change in density, pressure and fluid velocity.
Moving with the sound speed relative w.r.t. the local fluid velocity.
Normal flow, described by the *Euler equations*.

Obtaining useful fluxes: the Riemann problem

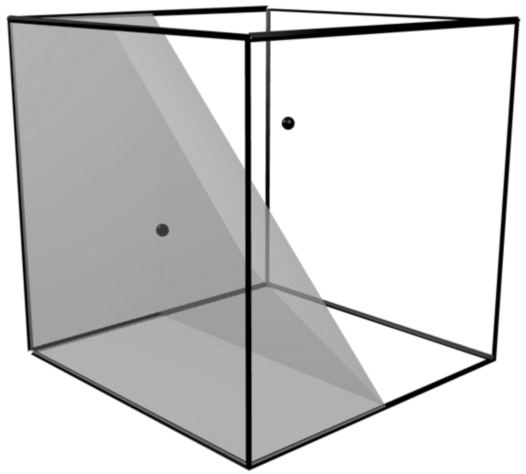
Exact solution to Riemann problem can be found using an iterative procedure

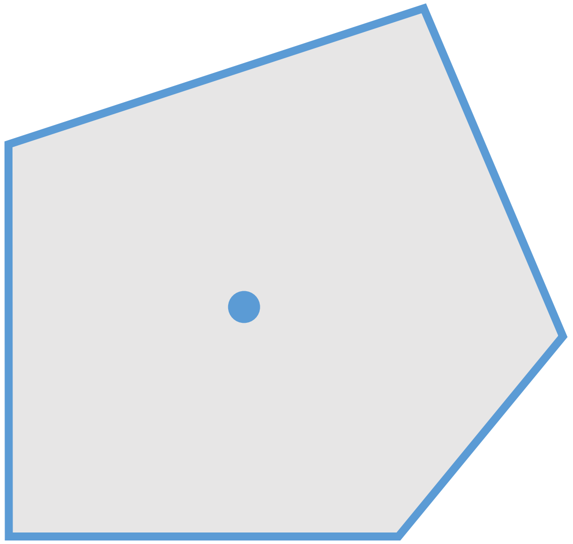
Good approximate solvers exist that do not require an iteration (more efficient)

The solution of the Riemann problem is then used to obtain fluxes that respect the natural direction of the flow.

Since the Riemann problem explicitly deals with discontinuities, our integration scheme can handle discontinuous fluid quantities, as long as they coincide with cell boundaries!

Approximating the surface integral





The integral over the entire cell boundary surface is a sum over boundaries between the cell and its neighbouring cells:

$$\frac{\partial Q_i}{\partial t} = -\iint \underline{F}(U).\,\mathrm{d}\underline{S_i} \approx -\sum_j \underline{F}_{ij}.\underline{A}_{ij}$$

Time integration

A first order accurate integration scheme is a simple Euler scheme:

$$\Delta Q_i = -\left(\sum_j \underline{F}_{ij} . \underline{A}_{ij}\right) \Delta t$$

If we use upwind fluxes, this scheme will be conditionally stable, if

$$\Delta \mathrm{t} < \mathrm{C}_{\mathrm{CFL}} \frac{\Delta x}{v_{signal}}$$

Δx is the size of a single cell

v_{signal} is the velocity of the flow

C_{CFL} is the Courant-Friedrichs-Lewy number, $0 < C_{CFL} < 1$ for a stable solution

Summary: first order finite volume scheme

1

Convert conserved to primitive variables

$$\rho_i = \frac{m_i}{V_i} \qquad p_i = (\gamma - 1)\rho_i \left(e_{tot,i} - \frac{1}{2} u_i^2 \right)$$

2

Use primitive variables across cell boundaries as input for Riemann solver

$$\text{RiemannSolver}(\rho_L, \underline{u}_L, p_L, \rho_R, \underline{u}_R, p_R) \longrightarrow \rho_{sol}, \underline{u}_{sol}, p_{sol}$$

3

Use Riemann problem solution to obtain upwind fluxes

$$\underline{F}_{ij} = \underline{F}(\rho_{sol}, \underline{u}_{sol}, p_{sol})$$

4

Use fluxes to update conserved variables

$$\Delta Q_i = -\left(\sum_j \underline{F}_{ij} . \underline{A}_{ij} \right) \Delta t$$

Example Python code:

```
# the cell class
class Cell:

    def __init__(self):
        self._volume = 0.
        self._mass = 0.
        self._momentum = 0.
        self._energy = 0.
        self._density = 0.
        self._velocity = 0.
        self._pressure = 0.

        self._right_ngb = None
        self._surface_area = 1.
```

```
# the constant adiabatic index
GAMMA = 5./3.
# the constant time step
timestep = 0.001
```

```
# set up the cells
cells = []
for i in range(100):

    cell = Cell()
    cell._volume = 0.01
    if i < 50:
        cell._mass = 0.01
        cell._energy = 0.01 / (GAMMA - 1.)
    else:
        cell._mass = 0.00125
        cell._energy = 0.001 / (GAMMA - 1.)
    cell._momentum = 0.

    # set the neighbour of the previous cell
    cells[-1]._right_ngb = cell

    cells.append( cell )
```

1

Convert conserved to primitive variables

$$\rho_i = \frac{m_i}{V_i} \qquad p_i = (\gamma - 1)\rho_i \left(e_{tot,i} - \frac{1}{2} u_i^2 \right)$$

Example Python code:

```
for cell in cells:
    volume = cell._volume
    mass = cell._mass
    momentum = cell._momentum
    energy = cell._energy

    density = mass / volume
    velocity = momentum / mass
    pressure = (GAMMA – 1) * (energy / volume – 0.5 * density * velocity * velocity)

    cell._density = density
    cell._velocity = velocity
    cell._pressure = pressure
```

2 Use primitive variables across cell boundaries as input for Riemann solver

$$\mathrm{RiemannSolver}(\rho_L, \underline{u}_L, p_L, \rho_R, \underline{u}_R, p_R) \longrightarrow \rho_{sol}, \underline{u}_{sol}, p_{sol}$$

Example Python code:

```
for cell in cells:
   cell_right = cell._right_ngb

   densityL = cell._density
   velocityL = cell._velocity
   pressureL = cell._pressure

   densityR = cell_right._density
   velocityR = cell_right._velocity
   pressureR = cell_right._pressure

   densitysol, velocitysol, pressuresol = RiemannSolver(densityL, velocityL, pressureL,
                                                         densityR, velocityR, pressureR)

   # ... continued in step 3 ...
```

3 Use Riemann problem solution to obtain upwind fluxes

$$\underline{F}_{ij} = \underline{F}(\rho_{sol}, \underline{u}_{sol}, p_{sol})$$

Example Python code:

```
# for cell in cells:
# ...

flux_mass = densitysol * velocitysol
flux_momentum = densitysol * velocitysol * velocitysol + pressuresol
flux_energy = (pressuresol * GAMMA / (GAMMA – 1) + \
                    0.5 * densitysol * velocitysol * velocitysol) * velocitysol

# ... continued in step 4 ...
```

4

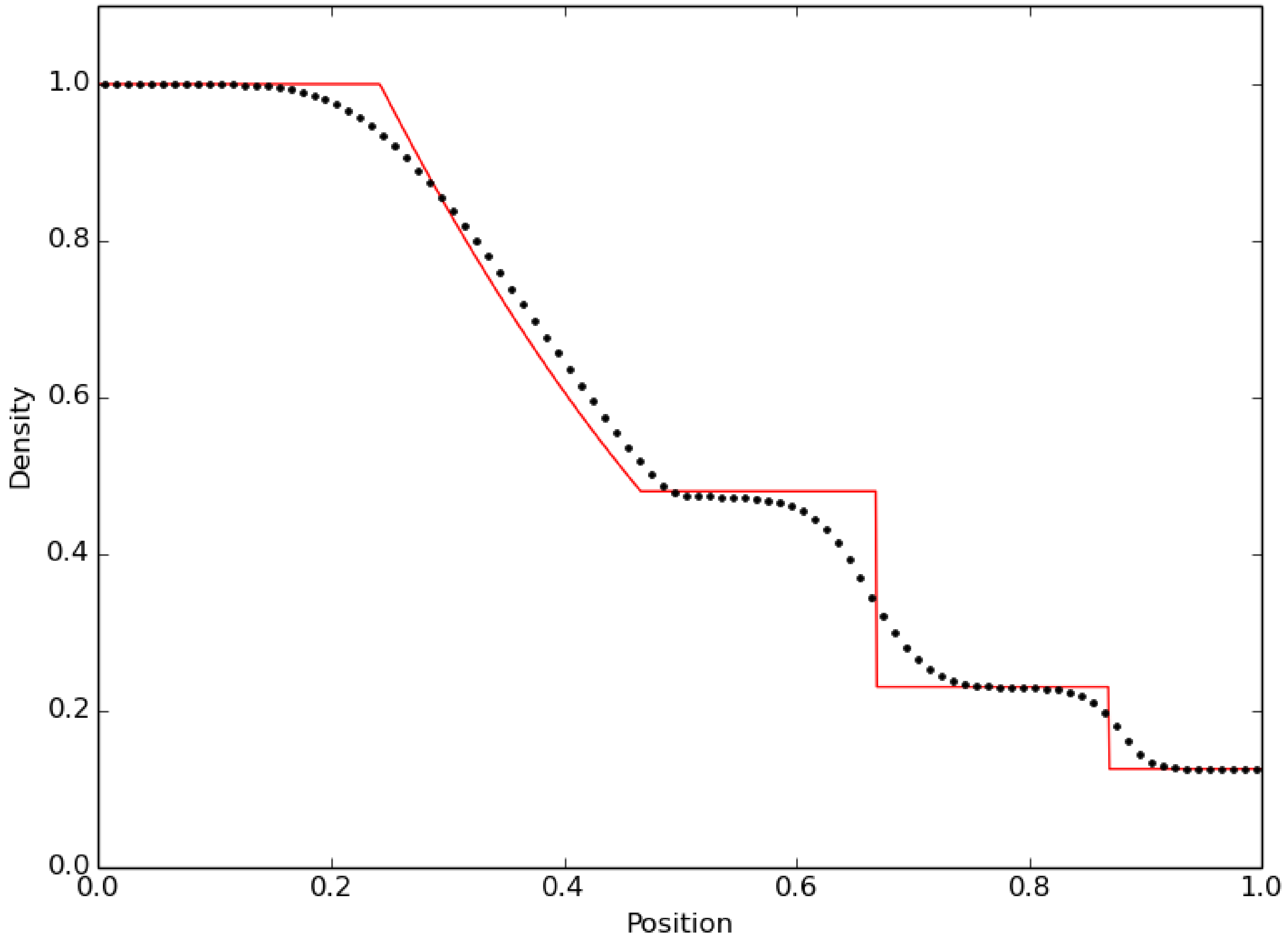
Use fluxes to update conserved variables

$$\Delta Q_i = -\left(\sum_j \underline{F}_{ij} . \underline{A}_{ij}\right) \Delta t$$

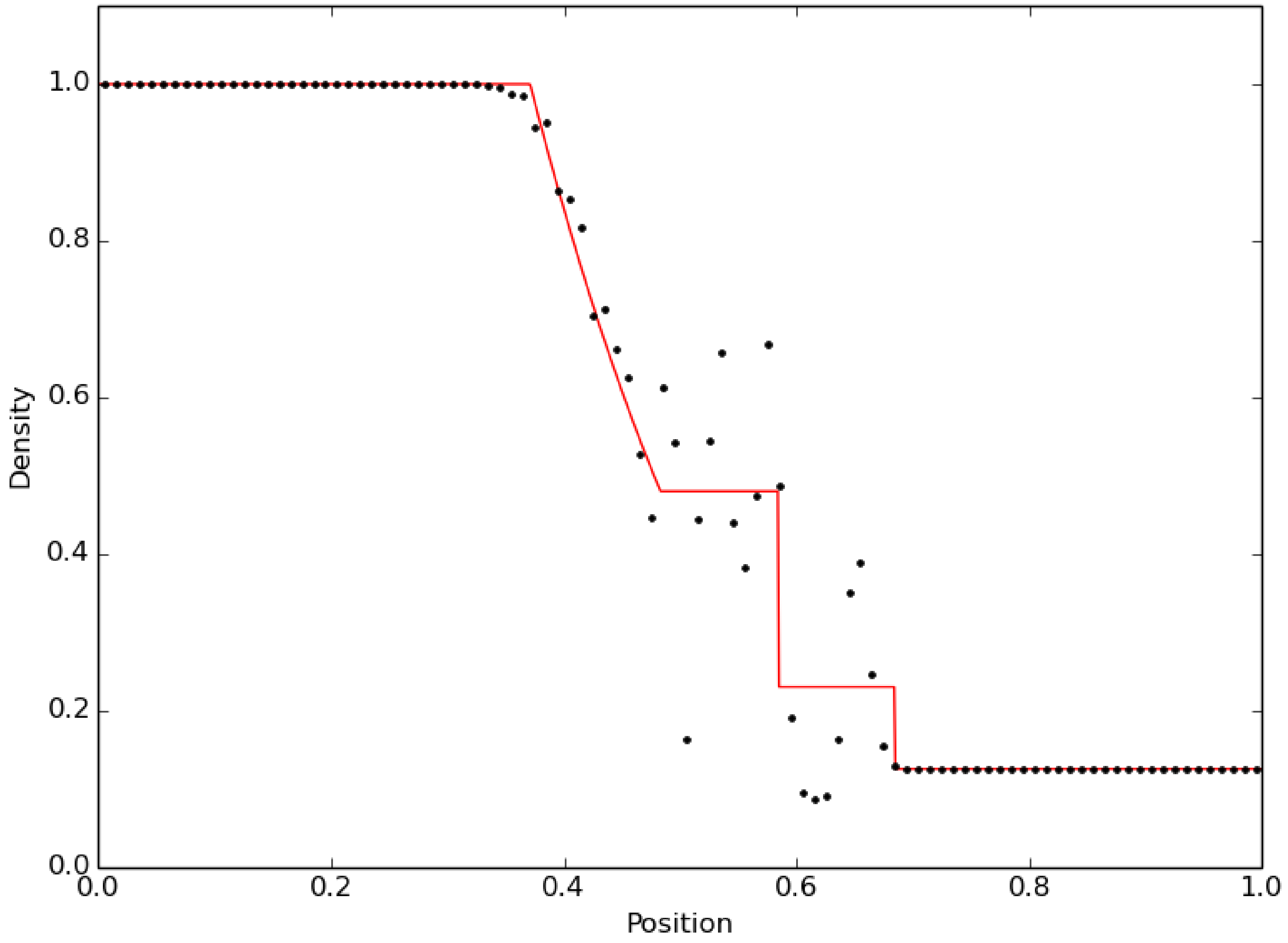
Example Python code:

```
# for cell in cells:
# ...

A = cell._surface_area

cell._mass = cell._mass – flux_mass * A * timestep
cell._momentum = cell._momentum – flux_momentum * A * timestep
cell._energy = cell._energy – flux_energy * A * timestep

# flux exchange: what goes out of cell has to go into cell_right
cell_right._mass = cell_right._mass + flux_mass * A * timestep
cell_right._momentum = cell_right._momentum + flux_momentum * A * timestep
cell_right._energy = cell_right._energy + flux_energy * A * timestep
```

Reference solution at t = 0.2 (after 200 steps)



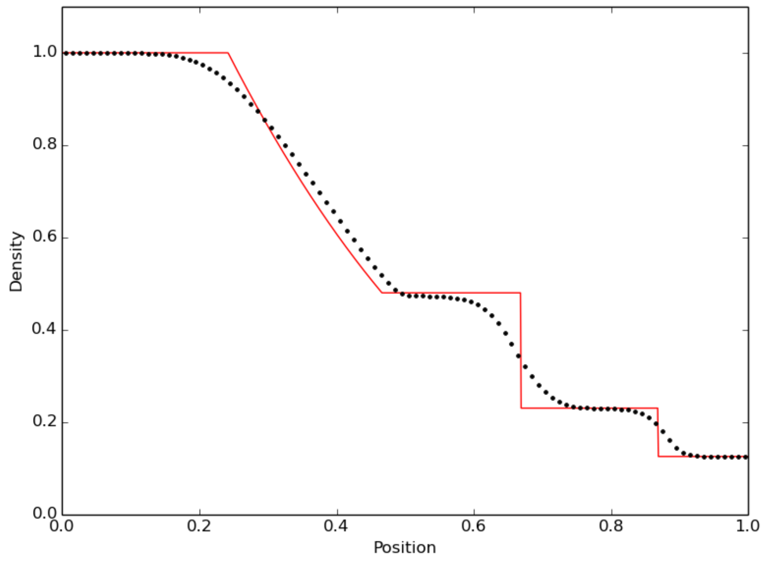
Solution without Riemann solver at t = 0.1



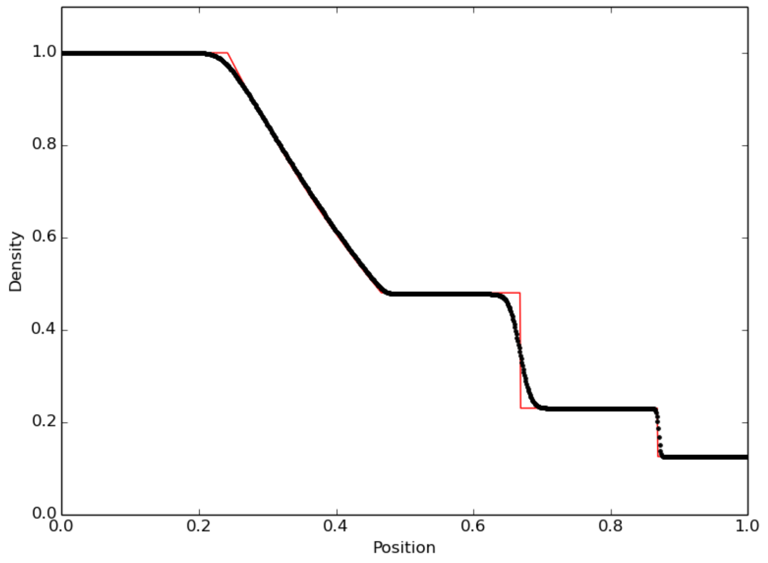
EXTRA: Improving the result

The method we have seen so far is first order in space and time.
This means that the accuracy of the solution improves linearly with the number of cells used to discretize the fluid.

100 cells

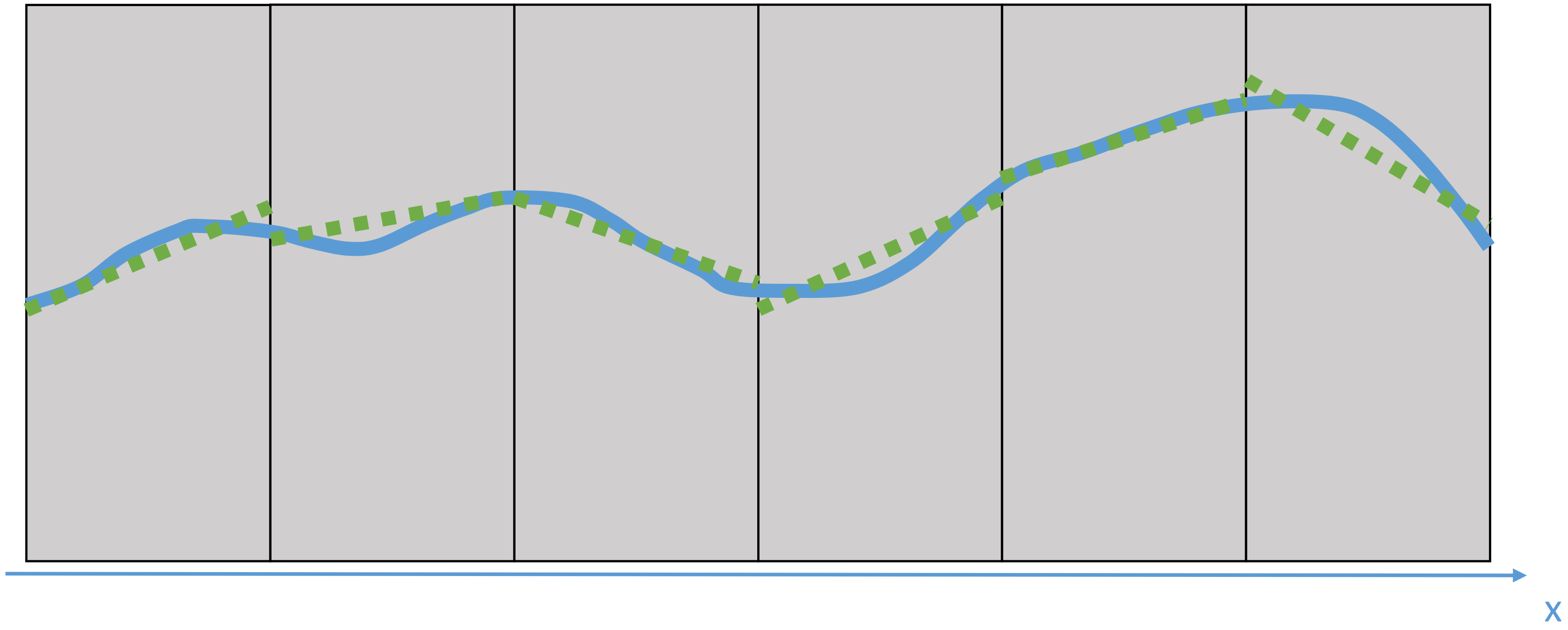


1000 cells



We can do better by using a second order method.

Second order schemes



Use linear interpolation to obtain more accurate values at cell boundaries

Requires the computation of *gradients* across cells

Second order in space

Second order schemes

Gradients can also be used to predict variables forward in time:

$$\rho' = \rho - \frac{1}{2}\Delta t\left(\rho\underline{\nabla}.\underline{u} + \underline{u}.\underline{\nabla}\rho\right)$$

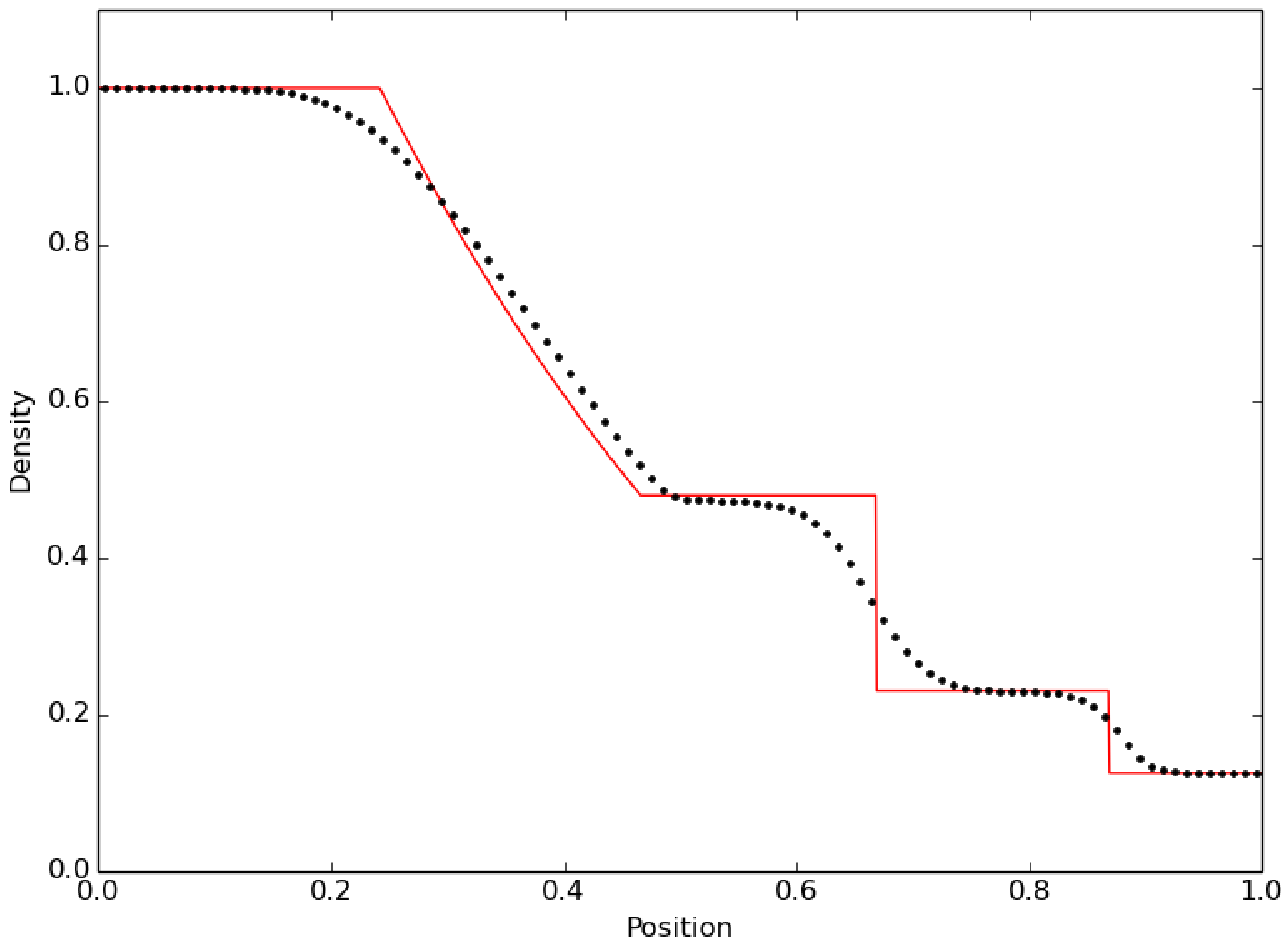
$$\underline{u}' = \underline{u} - \frac{1}{2}\Delta t\left(\underline{u}\,\underline{\nabla}.\underline{u} + \frac{1}{\rho}\underline{\nabla}p\right)$$

(these are just the Euler equations)

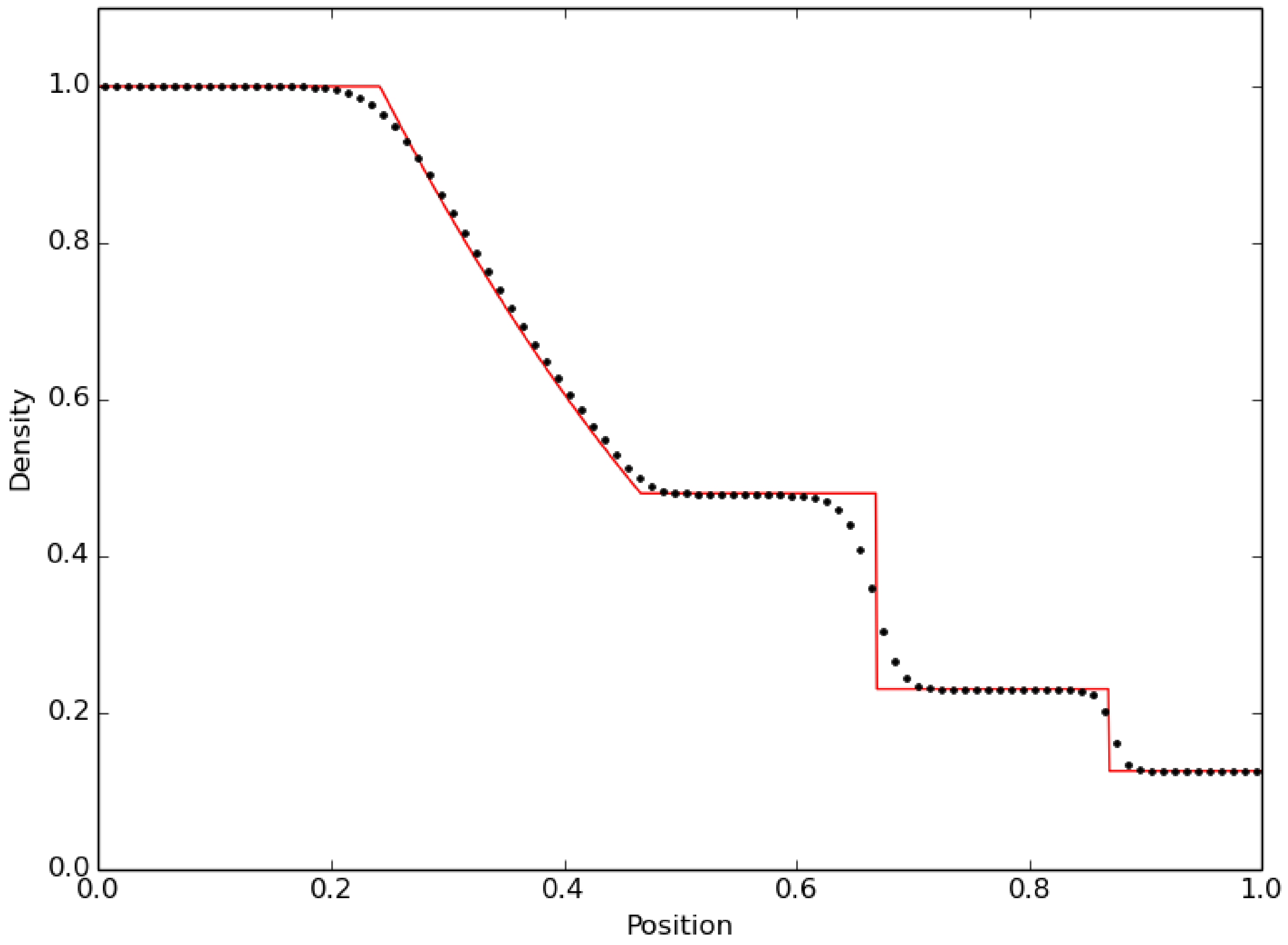
$$p' = p - \frac{1}{2}\Delta t\left(\gamma p\,\underline{\nabla}.\underline{u} + \underline{u}.\underline{\nabla}p\right)$$

Second order in time

Reference solution at t = 0.2 (first order)



Reference solution at t = 0.2 (second order)



Lagrangian schemes

We have imposed no restrictions on the shape of a cell

However, we have required the cell volume to be independent of time

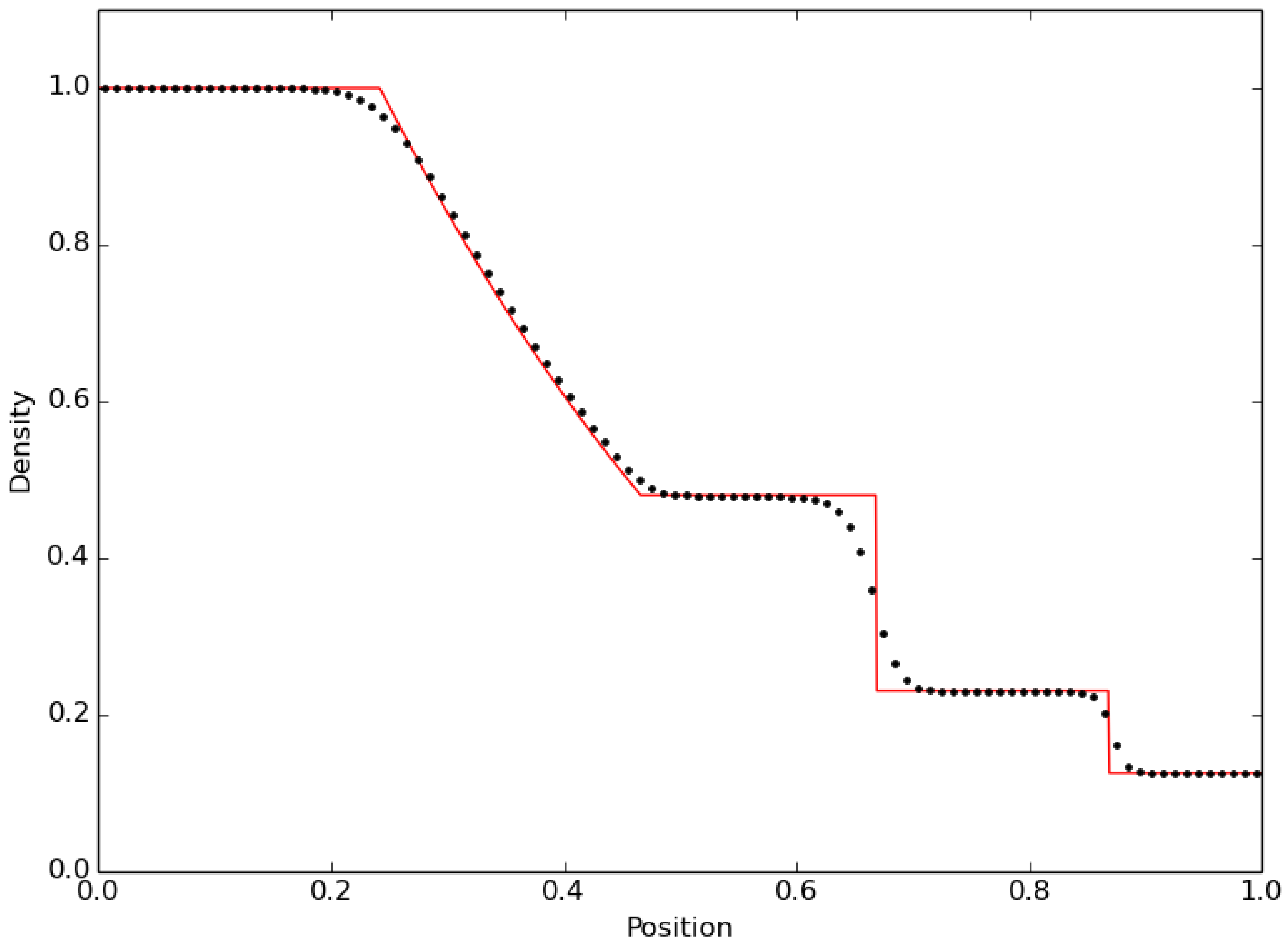
This requirement can be relaxed by adding correction terms to the fluxes:

$$\underline{F}(U) = \begin{pmatrix} \rho(\underline{u} - \underline{w}) \\ \rho\underline{u}(\underline{u} - \underline{w}) + p\underline{\underline{1}} \\ \rho e_{tot}(\underline{u} - \underline{w}) + p\underline{u} \end{pmatrix}$$

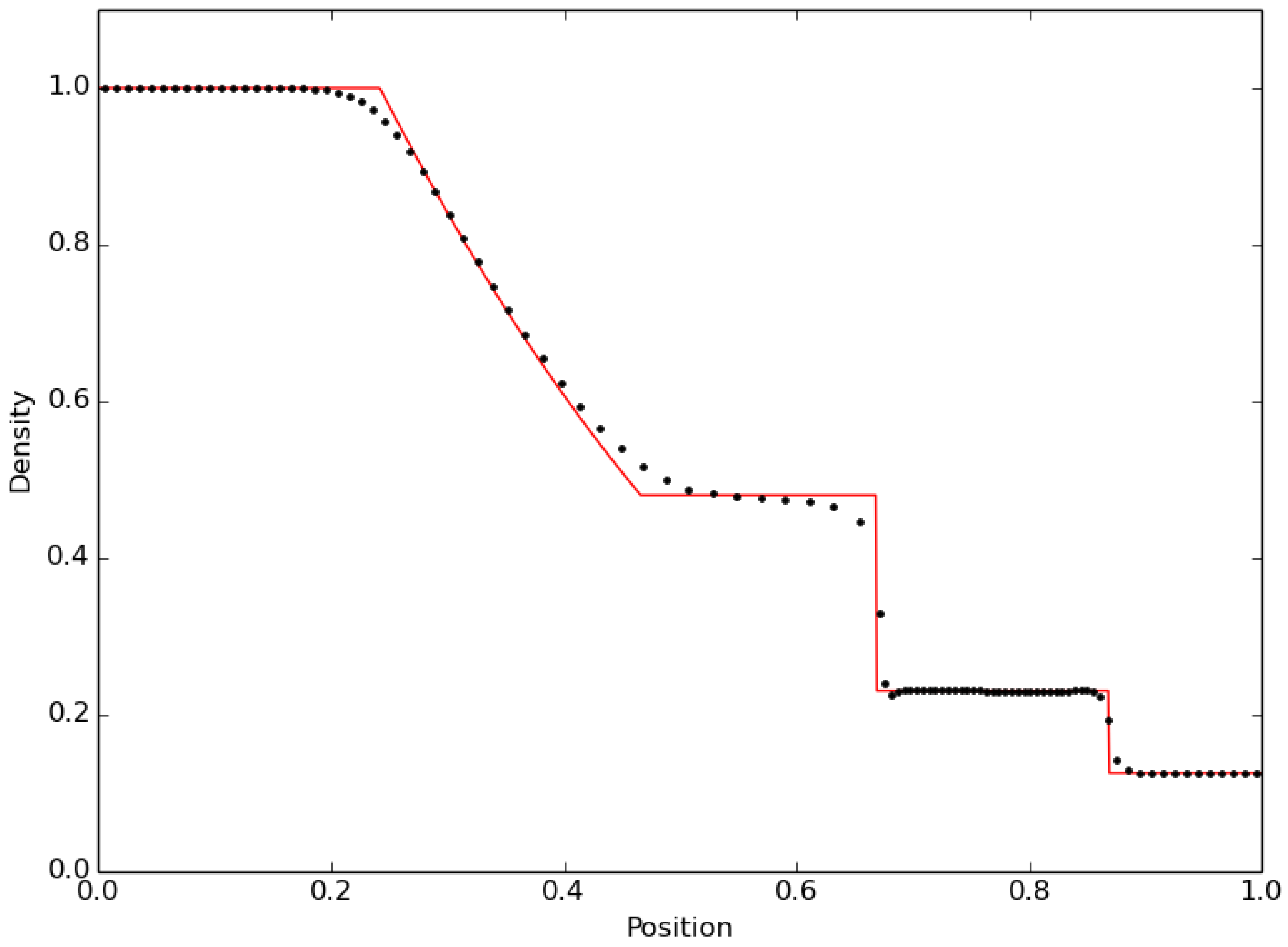
$\underline{w}$ is the velocity of the surface boundary of the cell

This also means we can solve the Riemann problem in the rest frame of the boundary surface, cancelling out large flow velocities that might affect the accuracy of the solver.

Reference solution at t = 0.2 (Eulerian)



Reference solution at t = 0.2 (Lagrangian)


Density
Position