Shock tubes and dust

Andrea Viceré
Università di Urbino "Carlo Bo" and INFN, Sezione di Firenze

VIR-0516A-16, November 28, 2016

1 Introduction

During a venting, an unbalance of pressure is present between the venting tubes and the Virgo tower. The resulting flow of air may carry dust potentially dangerous for fused silica fibers, if sufficiently fast and energetic.

It is the purpose of this note to estimate the speed of the flow and the resulting speed of the dust.

In the following, for the sake of concreteness, we will assume that in the initial phase of the venting the pressure inside the Virgo tower is $p_{low} = 10^{-2}\mathrm{Pa}$, and that this vacuum enclosure is placed in contact with a chamber having a pressure $p_{high} = 10\mathrm{Pa}$, by means of a tube. In the approximation we will make, the size of the tube does not matter provided its diameter is much larger than the mean free path of gas molecules, an assumption we will have to verify.

In this form, turns out that this is a *classical* problem, the so-called Sod's shock tube.

Gary Sod proposed this problem as a standard benchmarks to assess the accuracy of numerical solvers; it as a classic example of one-dimensional compressible flow, and allows an analytical solution of Euler equations.

Since the analytical solution, though direct, is somewhat cumbersome to obtain and a check of the literature is required, we will review its derivation.

Then, we will use the resulting simulated flow to drag dust particles, and see what happens.

This is a Jupyter notebook, and uses `Python` for performing calculations. The sources are available at https://dl.dropboxusercontent.com/u/18548797/ShockTubeAndDust.zip

2 Theory of a shock tube

A shock tube is an idealized device that generates a one-dimensional shock wave in a **compressible** gas.

We have a tube with two regions containing gas at different pressure, separated by an infinitely-thin, rigid diaphragm. The gas is initially at rest, and the left region is at a higher pressure than the region to the right of the diaphragm. At time $t = 0.0s$, the diaphragm is ruptured instantaneously; this simulates the opening of a valve.

What happens? A shock wave is generated: the gas at higher pressure, no longer constrained by the diaphragm, rushes into the lower-pressure area and a one-dimensional unsteady flow is established, consisting of:

- a shock wave traveling to the right
- an expansion wave traveling to the left

- a moving contact discontinuity

The shock-tube problem is an example of a *Riemann problem* and it has an analytical solution, as we said. The situation is illustrated in Fig. 1

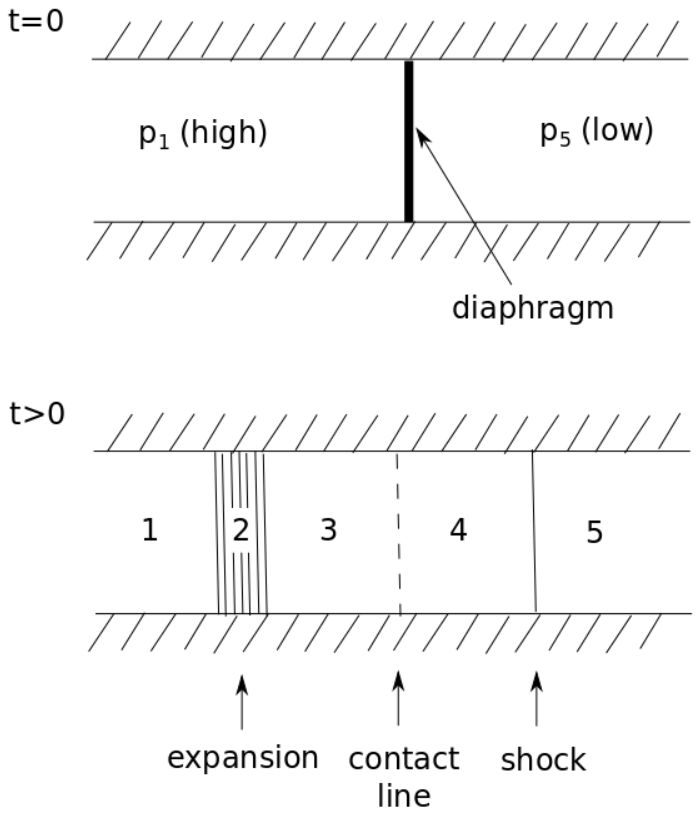


Figure 1: The shock-tube problem

The *contact line* is the line separating the fluid on the left side from the one on the right side: note that it does not stay fixed at the location of the diaphragm (don't be misled by the figure).

2.1 The Euler equations

The Euler equations govern the motion of an inviscid fluid (no viscosity). They consist of the conservation laws of mass and momentum, and often we also need to work with the energy equation.

Let's consider a 1D flow with velocity u in the x-direction. The Euler equations for a fluid with density ρ and pressure p, appropriate for a fluid flowing in a tube with constant section, are:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) = 0 \qquad \text{Mass conservation} \tag{1}$$

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u^2 + p) = 0 \qquad \text{Momentum conservation} \tag{2}$$

plus the Bernoulli equation, which we can write in this form:

$$\frac{\partial}{\partial t}(\rho e_T) + \frac{\partial}{\partial x}(\rho u e_T + pu) = 0 \qquad \text{Energy conservation} \tag{3}$$

where $e_T = e + u^2/2$ is the total energy per unit mass, equal to the internal energy plus the kinetic energy (per unit mass). We will call it also specific energy later.

This is the conservative form of Euler equations, which is more accurate for situations with shock waves: in vector form

$$\frac{\partial}{\partial t}\begin{bmatrix} \rho \\ \rho u \\ \rho e_T \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} \rho u \\ \rho u^2 + p \\ (\rho e_T + p)u \end{bmatrix} = 0 \tag{4}$$

There's one major problem there: we have 3 equations and 4 unknowns: ρ, u, e, p. However, we can use an equation of state to calculate the pressure—in this case, we'll use the ideal gas law.

2.2 Calculating the pressure

For an ideal gas, the equation of state is

$$e = e(\rho, p) = \frac{p}{(\gamma - 1)\rho} \tag{5}$$

where $\gamma = \frac{C_p}{C_V} = 1.4$ is a reasonable value to model air

$$p = (\gamma - 1)\rho e \; . \tag{6}$$

Recall from above the specific total energy

$$e_T = e + \frac{1}{2}u^2 \qquad \Rightarrow e = e_T - \frac{1}{2}u^2 \; ; \tag{7}$$

putting it all together, we arrive at an equation for the pressure

$$p = (\gamma - 1)\left(\rho e_T - \frac{\rho u^2}{2}\right) . \tag{8}$$

2.3 Location of interfaces

The evolution of the interfaces in the gas can be found in analytical form. First we need to define the dynamic of the separation surfaces: referring to the following Fig. 2

we define the following locations

- x_0: location of the initial membrane
- x_1: head of the *rarefaction* wave which moves left
- x_2: other boundary of the *rarefaction* wave
- x_3: contact discontinuity that separates the left and right fluids
- x_4: head of the *compression* wave which moves right

for $t >$ one can state immediately that

- $x_1(t) = x_0 - c_1 t$ where c_1 is the speed of sound in the "left" undisturbed fluid.
- $x_3(t) = x_0 + u_{post} t$ where u_{post} is the speed of the fluid (moving to the right) before the shock front.

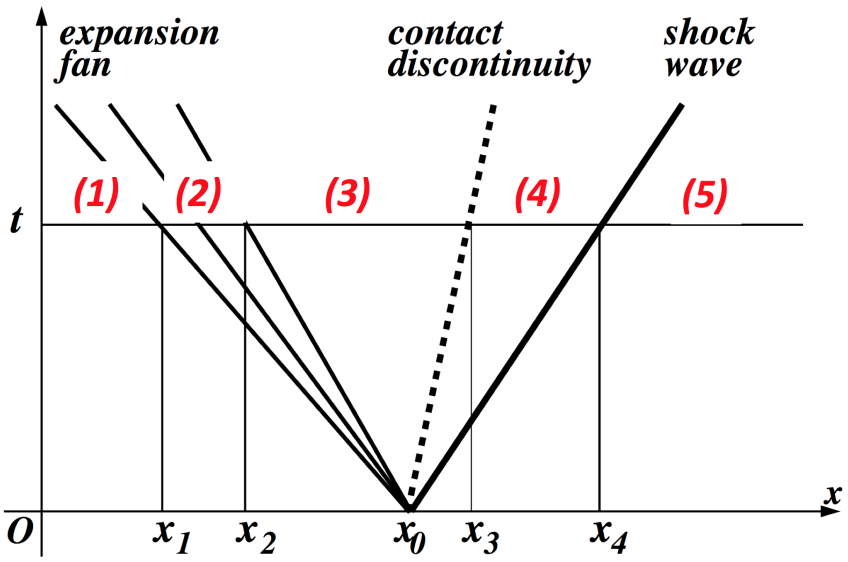


Figure 2: Regions and interfaces at $t > 0$

- $x_4(t) = x_0 + u_{shock}t$ where u_{shock} is the speed of the shock front propagating in the "right" fluid.

given our model of the fluids, we simply have that

$$c_{sound} = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_S} = \sqrt{\frac{\gamma\, p}{\rho}}\,; \tag{9}$$

it will be also useful the *ideal gas* equation of state

$$\begin{aligned} pV &= mR_{specific}T \quad \Rightarrow \quad \rho = \frac{p}{R_{specific}T} \\ R_{specific} &= 287.058\, J\, kg^{-1}\, K^{-1} \end{aligned} \tag{10}$$

where we will call `R_air` the value of $R_{specific}$ appropriate for dry air.

2.4 Qualitative solution

The qualitative behaviour of the solution is displayed in Fig. 3, which shows the behaviour of the quantities p, u, ρ at a given time t, in the different regions separated by the points $x_{1,2,3,4}(t)$ that we have defined earlier.

At location $x_4(t)$, which is the separation between regions 4 and 5, all quantities are discontinuous: we have a *shock*.

Instead, at point $x_3(t)$ *a priori* only the density ρ is discontinuous. In the drawing, $\rho_3 < \rho_4$, but it is not necessarily so (won't be so for the parameters we will use).

In all regions the quantities are constant within the region, except in the expansion (or fan) zone 2 in which they vary.

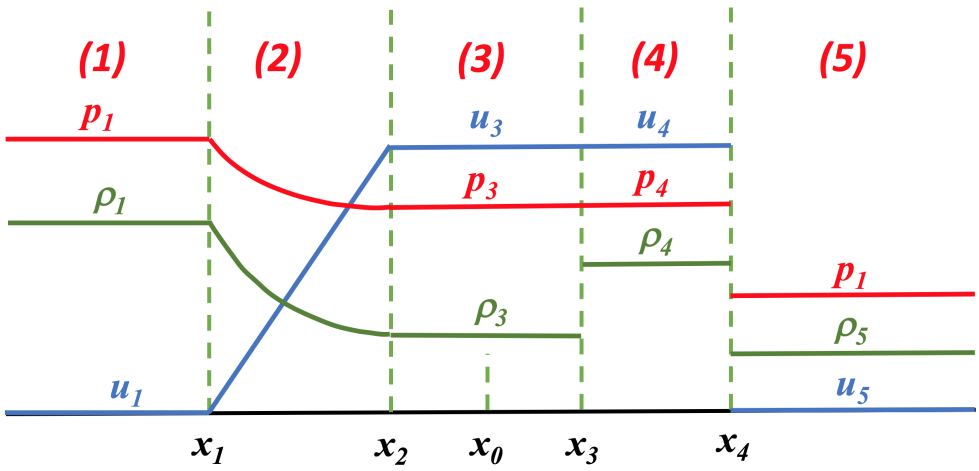


Figure 3: Gas quantities at $t > 0$

2.5 Rankine-Hugoniot conditions

If a conserved quantity q exhibits a jump at a certain position $x_s(t)$, hence a *shock*, which travels from a "left" space to a "right" space with speed $u_s(t) = \frac{dx_s}{dt}$ we can write the conservation equation in integral form as follows

$$\frac{d}{dt}\int_{x_L}^{x_s(t)} q(x,t)dx + \int_{x_s(t)}^{x_L} q(x,t)dx = -\int_{x_L}^{x_R} \frac{df}{dx}dx\,; \tag{11}$$

taking the derivative with respect to time, then setting $x_L = x_s - \epsilon, x_R = x_s + \epsilon$, and sending $\epsilon \to 0$, one obtains

$$u_s\,(q_L - q_R) = f_L - f_R \tag{12}$$

where the suffixes L, R represent the limit values immediately to the left and to the right of the discontinuity.

This allows obtaining the Rankine-Hugoniot conditions for discontinuities obeying Euler continuity equations

$$\begin{aligned} u_s(\rho_L - \rho_R) &= \rho_L u_L - \rho_R u_R \\ u_s(\rho_L u_L - \rho_R u_R) &= (\rho_L u_L^2 + p_L) - (\rho_R u_R^2 + p_R) \\ u_s\,(\rho_L e_{T,L} - \rho_R e_{T,R}) &= u_L(\rho_L e_{T,L} + p_L) - u_R(\rho_R e_{T,R} + p_R) \end{aligned} \tag{13}$$

These equations will allow us obtaining the values of the different quantities defined in Fig. 3

Shock speed u_s Considering the shock between regions 4 and 5, and using the first RH conditions, plus the condition $u_5 = 0$, we obtain

$$u_{shock}(\rho_4 - \rho_5) = \rho_4 u_{post} \tag{14}$$

where $u_3 = u_4 = u_{post}$ is the speed of the fluid immediately to the left of the discontinuity. Hence

$$u_{shock} = u_4 \frac{\rho_4}{\rho_4 - \rho_5} = u_4 \left(\frac{\rho_4}{\rho_5}\right) \left[\left(\frac{\rho_4}{\rho_5}\right) - 1\right]^{-1} \tag{15}$$

we need now the ratio $\frac{\rho_4}{\rho_5}$, and of course u_4.

Ratio ρ_4/ρ_5 We can now exploit the other two conditions (here $u_4 = u_{post}$)

$$\begin{aligned} u_4 \frac{\rho_4}{\rho_4 - \rho_5} \rho_4 u_4 &= \rho_4 u_4^2 + p_4 - p_5 \\ u_4 \frac{\rho_4}{\rho_4 - \rho_5} \left[\rho_4 \left(e_4 + \frac{1}{2}u_4^2\right) - \rho_5 e_5\right] &= u_4 \left[\rho_4 \left(e_4 + \frac{1}{2}u_4^2\right) + p_4\right] \end{aligned} \tag{16}$$

which simplify into

$$\begin{aligned} \frac{\rho_4 \rho_5}{\rho_4 - \rho_5} u_4^2 &= p_4 - p_5 \\ \frac{\rho_4 \rho_5}{\rho_4 - \rho_5} \left(e_4 + \frac{1}{2}u_4^2 - e_5\right) &= p_4 \end{aligned} \tag{17}$$

Eliminating u_4^2 one obtains

$$\frac{\rho_4 \rho_5}{\rho_4 - \rho_5} (e_4 - e_5) = \frac{1}{2} (p_4 + p_5) \tag{18}$$

using the equation of state this becomes

$$\frac{p_4 \rho_5 - p_5 \rho_4}{\rho_4 - \rho_5} = \frac{\gamma - 1}{2} (p_4 + p_5) \tag{19}$$

which allows to write

$$\begin{aligned} \frac{\rho_4}{\rho_5} &= \left[\frac{p_4}{p_5} + \eta^2\right] \left[1 + \eta^2 \frac{p_4}{p_5}\right]^{-1} \\ \eta^2 &\equiv \frac{\gamma - 1}{\gamma + 1} \end{aligned} \tag{20}$$

These will be useful as references, however a different choice of variables allows an easier solution.

2.5.1 Formulas in terms of M_s

It is convenient to express the three RH conditions in terms of the Mach number of the shock wave, defined as

$$M_s = \frac{u_s}{c_5}, \quad \text{where} \quad c_5 = \sqrt{\frac{\gamma p_5}{\rho_5}} \tag{21}$$

u_s: starting from

$$
\begin{aligned}
u_4 &= u_s \left(1 - \frac{\rho_5}{\rho_4}\right) \\
\frac{\rho_4 \rho_5}{\rho_4 - \rho_5} u_4^2 &= p_4 - p_5 \\
\frac{p_4 \rho_5 - p_5 \rho_4}{\rho_4 - \rho_5} &= \frac{\gamma - 1}{2} (p_4 + p_5)
\end{aligned} \tag{22}
$$

one can re-express as

$$
\begin{aligned}
u_4 &= M_s c_5 \left(1 - \frac{\rho_5}{\rho_4}\right) \\
M_s^2 \left(1 - \frac{\rho_5}{\rho_4}\right) &= \frac{1}{\gamma} \left(\frac{p_4}{p_5} - 1\right) \\
\frac{p_4}{p_5}\frac{\rho_5}{\rho_4} - 1 &= \frac{\gamma - 1}{2} \left(1 - \frac{\rho_5}{\rho_4}\right) \left(\frac{p_4}{p_5} + 1\right)
\end{aligned} \tag{23}
$$

solving for u_4 and for the ratios of pressures and densities one obtains

$$
\begin{aligned}
\frac{p_4}{p_5} &= \frac{2\gamma}{\gamma + 1} M_s^2 - \frac{\gamma - 1}{\gamma + 1} \\
\frac{\rho_5}{\rho_4} &= \frac{2}{\gamma + 1} \frac{1}{M_s^2} + \frac{\gamma - 1}{\gamma + 1} \\
u_4 &= c_5 \frac{2}{\gamma + 1} \left(M_s - \frac{1}{M_s}\right)
\end{aligned} \tag{24}
$$

in terms of the Mach number of the shock, which we have now to determine. To that end, we will proceed backwards through the other transitions.

2.6 Contact discontinuity

The boundary between regions 3 and 4 is actually continuous for p and u, hence we can write

$$
u_3 = u_4 \qquad p_3 = p_4 \tag{25}
$$

instead, the density ρ exhibits a jump; with the condition $u_{post} = u_{3,4}$ on the speed of the contact discontinuity, the RH relations are trivially satisfied.

2.7 Relation between regions 3 and 1

Since transformations are isoentropic, except through the shock which causes the gas to reheat, the regions 1 and 3 are related by an adiabatic transformation, hence

$$
\frac{p_1}{\rho_1^\gamma} = \frac{p_3}{\rho_3^\gamma} \qquad \Rightarrow \qquad \frac{\rho_3}{\rho_1} = \left(\frac{p_3}{p_1}\right)^{\frac{1}{\gamma}} . \tag{26}
$$

The other result we need is the conservation of Riemann invariants, which requires developing the solution of hyperbolic equations with the method of characteristics: the interested reader may consult for instance the excellent notes by Susanne Höfner. We have

$$u_1 + \frac{2c_1}{\gamma - 1} = u_3 + \frac{2c_3}{\gamma - 1} \; ; \tag{27}$$

as $u_1 = 0$, one obtains

$$u_3 = \frac{2}{\gamma - 1} \left(c_1 - c_3 \right) \tag{28}$$

We have now sufficient information and we can establish the analytical solution

2.8 Compatibility equation

From the relation between u_4 and M_s obtained with RH conditions between regions 4 and 5, and equality of velocity in regions $3, 4$, we have

$$M_s - \frac{1}{M_s} = \frac{\gamma + 1}{2} \frac{u_4}{c_5} = \frac{\gamma + 1}{2} \frac{u_3}{c_5} \; ; \tag{29}$$

then using the just mentioned condition on u_3 we have

$$\frac{\gamma + 1}{2} \frac{u_3}{c_5} = \frac{\gamma + 1}{\gamma - 1} \frac{c_1 - c_3}{c_5} = \frac{c_1}{c_5} \frac{\gamma + 1}{\gamma - 1} \left(1 - \frac{c_3}{c_1} \right) \; . \tag{30}$$

Then we can use the isoentropic relation to establish that

$$\frac{c_3}{c_1} = \sqrt{\frac{p_3}{p_1} \frac{\rho_1}{\rho_3}} = \sqrt{\frac{p_3}{p_1} \left(\frac{p_1}{p_3} \right)^{\frac{1}{\gamma}}} = \left(\frac{p_3}{p_1} \right)^{\frac{\gamma - 1}{2\gamma}} = \left(\frac{p_3}{p_5} \frac{p_5}{p_1} \right)^{\frac{\gamma - 1}{2\gamma}} \tag{31}$$

and we use one of the RH conditions at interface $4, 5$

$$\frac{p_3}{p_5} = \frac{2\gamma}{\gamma + 1} M_s^2 - \frac{\gamma - 1}{\gamma + 1} \tag{32}$$

to close the circle and come back to M_s. So in summary we have a **compatibility equation**

$$M_s - \frac{1}{M_s} = \frac{c_1}{c_5} \frac{\gamma + 1}{\gamma - 1} \left\{ 1 - \left[\frac{p_5}{p_1} \left(\frac{2\gamma}{\gamma + 1} M_s^2 - \frac{\gamma - 1}{\gamma + 1} \right) \right]^{\frac{\gamma - 1}{2\gamma}} \right\} \tag{33}$$

which can be solved for M_s numerically.

2.9 Fan expansion region 2

We have left out from our discussion the evolution of the "other boundary" of the expansion region 2, which we called x_2; again using the technique of characteristics, it is possible to show that

$$\begin{aligned} x_1(t) &= x_0 - c_1 t \\ x_2(t) &= x_0 + (u_3 - c_3)\, t \end{aligned} \tag{34}$$

now again, in any point x between $x_1(t)$ and $x_2(t)$, the fluid will have speed u, sound speed c, density ρ which depend on time: again using the method of characteristics it is possible to assert that the relation between the position x and the time t is given by

$$x = x_0 + (u - c)\, t \qquad \Leftrightarrow \qquad \frac{x - x_0}{t} = u - c\,; \tag{35}$$

using the characteristic from region 1 one obtains also

$$u_1 + \frac{2c_1}{\gamma - 1} = u + \frac{2c}{\gamma - 1} \qquad \Rightarrow \qquad c = c_1 - \frac{\gamma - 1}{2} u \tag{36}$$

combining one obtains, inside the fan expansion region

$$\begin{aligned} u &= \frac{2}{\gamma + 1}\left[c_1 + \frac{x - x_0}{t}\right] \\ c &= c_1 - \frac{\gamma - 1}{2} u = \frac{2}{\gamma + 1} c_1 - \left(\frac{\gamma - 1}{\gamma + 1}\right)\frac{x - x_0}{t} \\ p &= p_1 \left(\frac{c}{c_1}\right)^{\frac{2\gamma}{\gamma - 1}} \\ \rho &= \gamma \frac{p}{c^2} \end{aligned} \tag{37}$$

2.10 Summary formulas

In summary, we have the following relations which allow us to solve the Shock tube problem

Fundamental definitions

$$\begin{aligned} c &= \sqrt{\frac{\gamma p}{\rho}} \\ e &= \frac{p}{(\gamma - 1)\,\rho} \end{aligned} \tag{38}$$

Compatibility equation

$$M_s - \frac{1}{M_s} = \frac{c_1}{c_5}\frac{\gamma + 1}{\gamma - 1}\left\{1 - \left[\frac{p_5}{p_1}\left(\frac{2\gamma}{\gamma + 1} M_s^2 - \frac{\gamma - 1}{\gamma + 1}\right)\right]^{\frac{\gamma - 1}{2\gamma}}\right\} \tag{39}$$

allows to compute M_s from the unperturbed states $1, 5$.

RH conditions

$$\begin{aligned} p_3 = p_4 &= p_5\left[\frac{2\gamma}{\gamma + 1} M_s^2 - \frac{\gamma - 1}{\gamma + 1}\right] \\ \rho_4 &= \rho_5\left[\frac{2}{\gamma + 1}\frac{1}{M_s^2} + \frac{\gamma - 1}{\gamma + 1}\right]^{-1} \\ u_3 = u_4 &= c_5 \frac{2}{\gamma + 1}\left(M_s - \frac{1}{M_s}\right) \end{aligned} \tag{40}$$

allow computing $p_{3,4}, \rho_4, u_{3,4}$

Isoentropy

$$\begin{aligned} \rho_3 &= \rho_1 \left(\frac{p_3}{p_1}\right)^{\frac{1}{\gamma}} \\ c_3 &= \sqrt{\frac{\gamma p_3}{\rho_3}} \end{aligned} \tag{41}$$

allow computing ρ_3, c_3

Lines of separation

$$\begin{aligned} x_1(t) &= x_0 - c_1 t \\ x_2(t) &= x_0 + (u_3 - c_3)\, t \\ x_3(t) &= x_0 + u_3 t \qquad \text{where} \qquad u_3 = u_4 \\ x_4(t) &= x_0 + u_s t = x_0 + M_s c_5 t \end{aligned} \tag{42}$$

describe the position of the separation lines for $t > 0$.

Expansion region (2) At a given time t, for $x \in [x_1(t),\, x_2(t)]$, within the region 2 one has

$$\begin{aligned} u_2(x,t) &= \frac{2}{\gamma+1}\left[c_1 + \frac{x - x_0}{t}\right] \\ c_2(x,t) &= c_1 - \frac{\gamma-1}{2}u = \frac{2}{\gamma+1}c_1 - \left(\frac{\gamma-1}{\gamma+1}\right)\frac{x-x_0}{t} \\ p_2(x,t) &= p_1\left(\frac{c_2(x,t)}{c_1}\right)^{\frac{2\gamma}{\gamma-1}} \\ \rho_2(x,t) &= \gamma\frac{p_2(x,t)}{[c_2(x,t)]^2} = \rho_1\left(\frac{c_2(x,t)}{c_1}\right)^{\frac{2}{\gamma-1}} \end{aligned} \tag{43}$$

which is the only region in which quantities are not constant.

2.11 Code for simulating the state of a shock tube

We will simulate everything with `Python` with the help of a few libraries.

```
In [4]: import numpy
        import math
        from scipy.optimize import fsolve

        def c_sound(gamma,p,rho):
            '''Computes the sound speed

            Parameters:
            -----------
            gamma, p, rho
            '''
            return math.sqrt(gamma*p/rho)

        R_air = 287.058

        def rhoIdealGas(p,R,T):
```

```
    '''Computes the density

    Parameters:
    -----------
    p, R, T
    '''
    return p/(R*T)

def TIdealGas(p,R,rho):
    '''Computes the temperature

    Parameters:
    -----------
    p, R, rho
    '''
    return p/(R*rho)

class ShockTube:
    def __init__(s, **kwargs):
        """Sod's shock tube

        Optional parameters
        -------------------
        p1  : float
             pressure Left of the shock (default 1.)
        rho1 : float
             density Left of the shock (default 1.)
        p5  : float
            pressure Right of the shock (default 0.1)
        rho5 : float
            density Right of the shock (default 0.125)
        x0   : float
            initial position of diaphragm (default 0.)
        gamma : float
            Cv/Cp ratio (default 1.4)
        """
        s.u1 = 0.
        s.p1 = float(kwargs.get('p1',1.))
        s.rho1 = float(kwargs.get('rho1',1.))
        s.u5 = 0.
        s.p5 = float(kwargs.get('p5', 0.1))
        s.rho5 = float(kwargs.get('rho5', 0.125))
        s.gamma = float(kwargs.get('gamma', 1.4))
        s.x0 = float(kwargs.get('x0',0.))
        s.T  = float(kwargs.get('T',293.15))
        s.Rs = float(kwargs.get('Rs',287.058))

        # compute state quantities in constant zones
        s.c1 = math.sqrt(gamma*s.p1/s.rho1)
        s.c5 = math.sqrt(gamma*s.p5/s.rho5)
        s.Ms_func = (lambda Ms : Ms - 1./Ms - s.c1/s.c5 * \
                        (s.gamma + 1.)/(s.gamma - 1.) * \
                        (1. - ( s.p5/s.p1 * ((2. * s.gamma)/(s.gamma + 1.)*Ms**2 -\
                        (s.gamma - 1.)/(s.gamma + 1.)) )**((s.gamma - 1.)/(2.*s.gamma))))
        s.Ms = fsolve(s.Ms_func, s.c5)[0]
        s.u4 = s.c5*2./(s.gamma+1.)*(s.Ms - 1./s.Ms)
        s.u3 = s.u4
        s.p4 = s.p5*((2.*s.gamma)/(s.gamma+1.)*(s.Ms)**2 - (s.gamma-1.)/(s.gamma+1.))
        s.p3 = s.p4
        s.rho4 = s.rho5/(2./((s.gamma+1.)*(s.Ms**2)) + (s.gamma-1.)/(s.gamma+1.))
        s.rho3 = s.rho1*(s.p3/s.p1)**(1./s.gamma)
        s.c3 = math.sqrt(gamma*s.p3/s.rho3)
        s.c4 = math.sqrt(gamma*s.p4/s.rho4)
        s.uShock = s.Ms*s.c5

        # rarefaction zones; calls valid only for t > 0
        s.c2 = (lambda x,t: 2./(s.gamma+1.)*s.c1 - \
                (s.gamma-1.)/(s.gamma+1.)*(x-s.x0)/t if t > 0. else s.c1)
```

```
        s.u2 = (lambda x,t: 2./(s.gamma+1.)*(s.c1 + (x - s.x0)/t) if t > 0. else 0.)
        s.rho2 = (lambda x,t: s.rho1*(s.c2(x,t)/s.c1)**(2./(s.gamma-1.)))
        s.p2   = (lambda x,t: s.p1*(s.c2(x,t)/s.c1)**((2.*s.gamma)/(s.gamma-1.)))

        # limits of zones: calls valid only for t > 0
        s.x1 = (lambda t : s.x0 - s.c1 * t)
        s.x2 = (lambda t : s.x0 + (s.u3 - s.c3) * t)
        s.x3 = (lambda t : s.x0 + s.u3 * t)
        s.x4 = (lambda t : s.x0 + s.uShock * t)

    def __str__(s):
        s0 = r'Shock tube with $\gamma=${}:'.format(s.gamma) + '\n'
        s1 = r'$c_1=${}, $\rho_1=${}, $u_1=${}, $p_1=${}'.format(s.c1,s.rho1,s.u1,s.p1) + '\n'
        s3 = r'$c_3=${}, $\rho_3=${}, $u_3=${}, $p_3=${}'.format(s.c3,s.rho3,s.u3,s.p3) + '\n'
        s4 = r'$c_4=${}, $\rho_4=${}, $u_4=${}, $p_4=${}'.format(s.c4,s.rho4,s.u4,s.p4) + '\n'
        s5 = r'$c_5=${}, $\rho_5=${}, $u_5=${}, $p_5=${}'.format(s.c5,s.rho5,s.u5,s.p5)
        #return r'a\nb'
        return s0 + s1 + s3 + s4 + s5

    def State(s, xVec, t,**kwargs):
        """Tube state

        Parameters
        -------------------
        xVec  : numpy array
             positions at which state should be computed
        t     : time
             time at which the state should be computed
        """
        verbose = bool(kwargs.get('verbose',False))
        pVec = numpy.zeros_like( xVec )
        rhoVec = numpy.zeros_like( xVec )
        uVec = numpy.zeros_like( xVec )
        x1 = s.x1(t)
        x2 = s.x2(t)
        x3 = s.x3(t)
        x4 = s.x4(t)
        for (i,x) in enumerate(xVec):
            if x > x4:
                uVec[i] = s.u5
                pVec[i] = s.p5
                rhoVec[i] = s.rho5
            elif x > x3:
                uVec[i] = s.u4
                pVec[i] = s.p4
                rhoVec[i] = s.rho4
            elif x > x2:
                uVec[i] = s.u3
                pVec[i] = s.p3
                rhoVec[i] = s.rho3
            elif x > x1:
                uVec[i] = s.u2(x,t)
                pVec[i] = s.p2(x,t)
                rhoVec[i] = s.rho2(x,t)
            else:
                uVec[i] = s.u1
                pVec[i] = s.p1
                rhoVec[i] = s.rho1
        if verbose:
            print "x1 = ", x1, " x2 = ", x2, " x3 = ", x3, " x4 = ", x4
        return (uVec, pVec, rhoVec)
```

2.11.1 Check: Sod's test conditions

The first test proposed by Sod in his 1978 paper is as follows, rescaled to atmospheric pressure.

In a tube spanning from $x = -10$m to $x = 10$m with the rigid membrane at $x = 0$m, we have

the following initial gas states:

$$\underline{IC}_L = \begin{bmatrix} \rho_L \\ u_L \\ p_L \end{bmatrix} = \begin{bmatrix} 1\ kg/m^3 \\ 0\ m/s \\ 10^5\ Pa \end{bmatrix}$$

$$\underline{IC}_R = \begin{bmatrix} \rho_R \\ u_R \\ p_R \end{bmatrix} = \begin{bmatrix} 0.125\ kg/m^3 \\ 0\ m/s \\ 10^4\ Pa \end{bmatrix}$$

where $\underline{IC}_L$ are the initial density, velocity and pressure on the left side of the tube membrane and $\underline{IC}_R$ are the initial density, velocity and pressure on the right side of the tube membrane.

The analytical solution to this test for the velocity, pressure and density, should look like the plots in Fig. 4

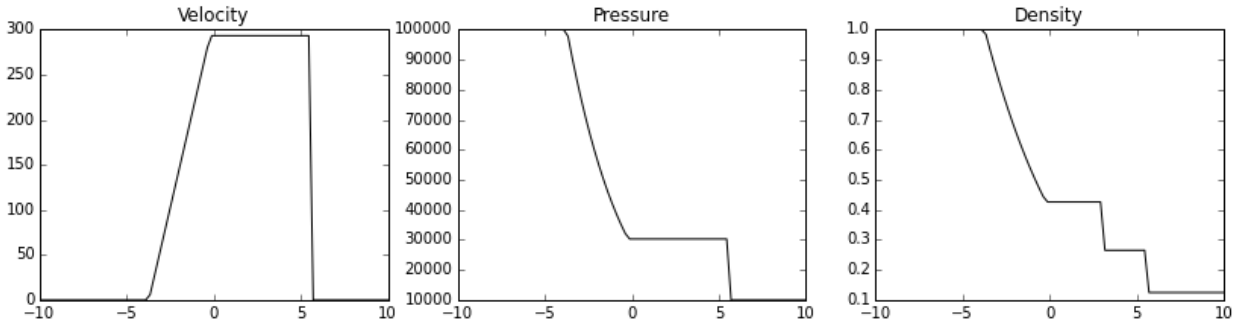


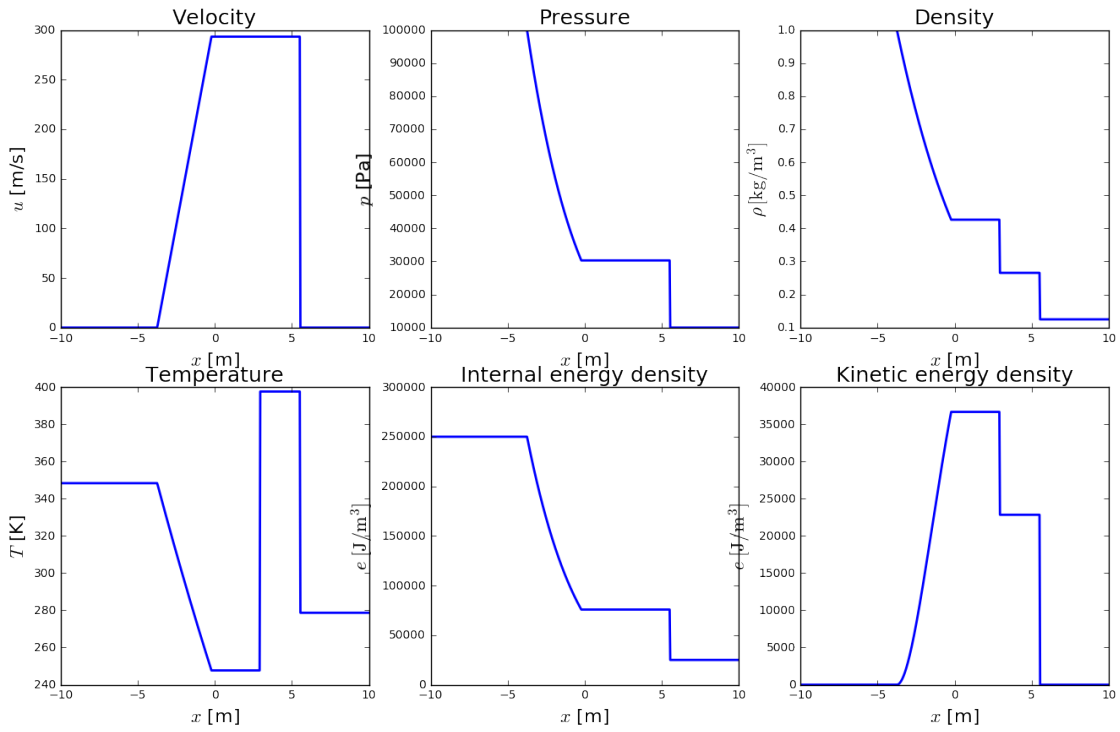
Figure 4: Analytical solution, with Sod's parameters

```
In [22]: # A function to display the solutions
         %matplotlib inline

         import matplotlib.pyplot as plt

         def displayState(xVec, uVec, pVec, rhoVec, gamma, R, title):

             plt.figure("Sod",figsize=(16,10))
             plt.subplot(231)
             plt.plot(xVec,uVec,lw=2)
             plt.xlabel(r"$x$ [m]",fontsize=16)
             plt.ylabel(r"$u$ [m/s]",fontsize=16)
             plt.title("Velocity",fontsize=18)

             plt.subplot(232)
             plt.plot(xVec,pVec,lw=2)
             plt.xlabel(r"$x$ [m]",fontsize=16)
             plt.ylabel(r"$p$ [Pa]",fontsize=16)
             plt.title("Pressure",fontsize=18)

             plt.subplot(233)
             plt.plot(xVec,rhoVec,lw=2)
             plt.xlabel(r"$x$ [m]",fontsize=16)
             plt.ylabel(r'$\rho\,\mathrm{[kg/m^3]}$',fontsize=16)
             plt.title("Density",fontsize=18)

         # Compute also temperature and energy density
             plt.subplot(234)
             plt.plot(xVec,TIdealGas(pVec,R,rhoVec),lw=2)
             plt.xlabel(r"$x$ [m]",fontsize=16)
             plt.ylabel(r'$T$ [K]',fontsize=16)
```

```
        plt.title("Temperature",fontsize=18)

        plt.subplot(235)
        plt.plot(xVec,pVec/((gamma-1.)),lw=2)
        plt.xlabel(r"$x$ [m]",fontsize=16)
        plt.ylabel(r'$e\ \mathrm{[J/m^3]}$',fontsize=16)
        plt.title("Internal energy density",fontsize=18)

        plt.subplot(236)
        plt.plot(xVec,rhoVec * (uVec**2),lw=2)
        plt.xlabel(r"$x$ [m]",fontsize=16)
        plt.ylabel(r'$e\ \mathrm{[J/m^3]}$',fontsize=16)
        plt.title("Kinetic energy density",fontsize=18)

        return
```

In [23]:
```
# Check: with Sod's test parameters

sodTube = ShockTube(gamma=1.4, p1=1.E+5, rho1=1., p5=1.E+4,rho5=0.125)

xVec = numpy.linspace(-10,10,500)
uVec,pVec,rhoVec = sodTube.State(xVec,0.01,verbose=True)

displayState(xVec, uVec, pVec, rhoVec, gamma, R_air, "Sod Test 1")
```

```
x1 =  -3.74165738677  x2 =  -0.222222145279  x3 =  2.93286270125  x4 =  5.540802928
```



It is worth noticing that in Sod's test the two initial gas states are not at the same temperature, if they are air.

2.11.2 Check: Höfner test parameters

Since the coding is tricky, it is worth performing another check: Höfner provides other test parameters, that we try in the following. We should obtain a result similar to the one in the following Figure

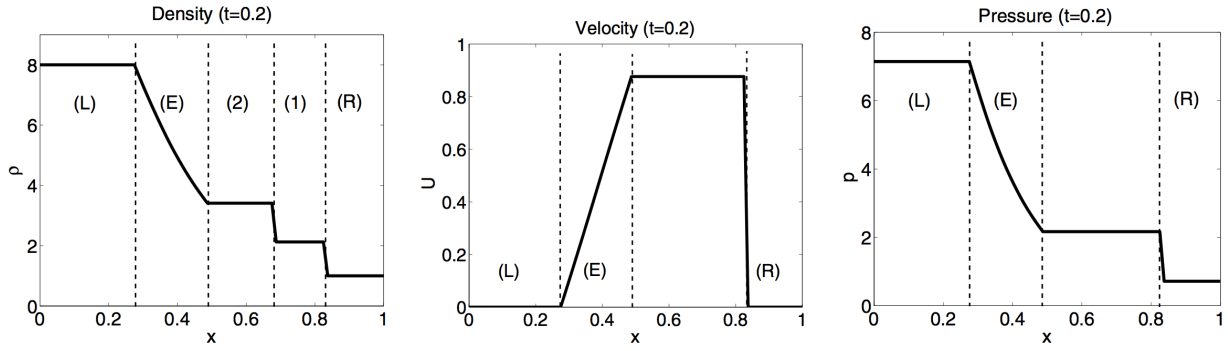
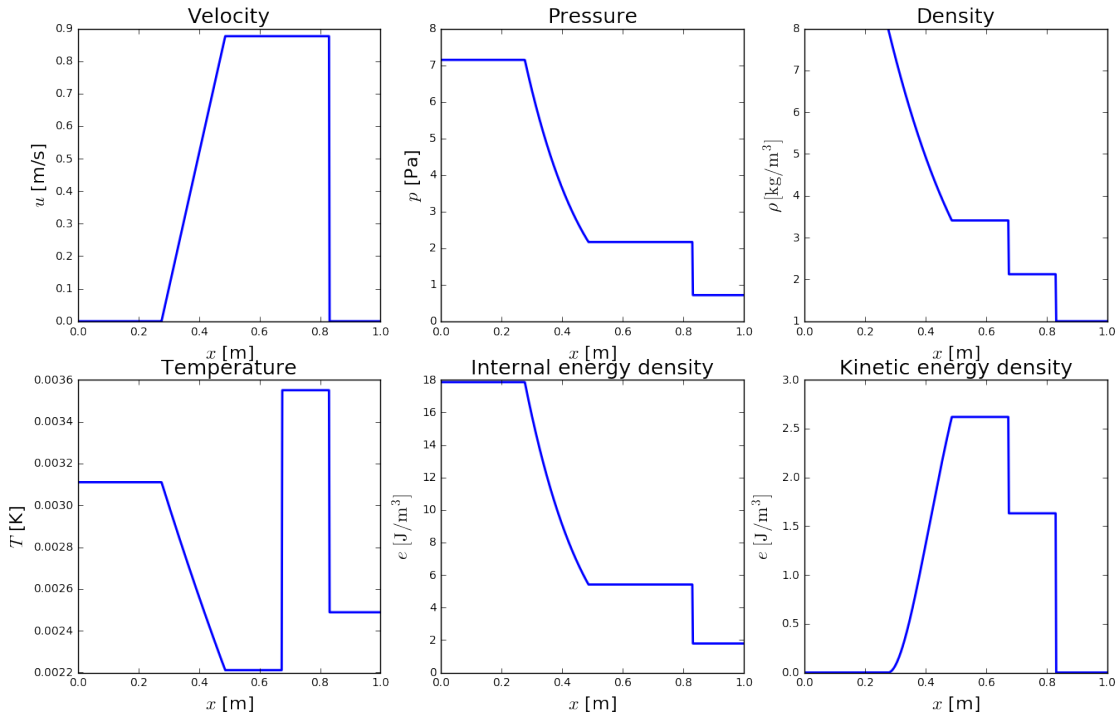


Figure 5: Another analytical solution, using Höfner's test parameters

```
In [27]: # Check: with Höfner's test parameters
         gamma = 1.4
         p1 = 10./gamma
         rho1 = 8.
         p5 = 1./gamma
         rho5 = 1.

         hoefnerTube = ShockTube(x0=0.5,p1=10./gamma,rho1=8., p5=1./gamma, rho5=1.)
         print hoefnerTube

         xVec = numpy.linspace(0.,1.,500)
         uVec,pVec,rhoVec = hoefnerTube.State(xVec,0.2,verbose=True)

         displayState(xVec, uVec, pVec, rhoVec, gamma, R_air, "Sod Test 1")
```

```
Shock tube with $\gamma=$1.4:
$c_1=$1.11803398875, $\rho_1=$8.0, $u_1=$0.0, $p_1=$7.14285714286
$c_3=$0.942761918361, $\rho_3=$3.41055542543, $u_3=$0.876360351945, $p_3=$2.1652155
$c_4=$1.19447496583, $\rho_4=$2.12458969364, $u_4=$0.876360351945, $p_4=$2.16521555
$c_5=$1.0, $\rho_5=$1.0, $u_5=$0.0, $p_5=$0.714285714286
x1 =  0.27639320225  x2 =  0.486719686717  x3 =  0.675272070389  x4 =  0.8311263089
```



We are therefore confident that the code we've written is correct.

2.12 Virgo 1500W conditions

In this case we make the following assumptions

- $p_1 = 10\,Pa$
- $p_5 = 10^{-2}\,Pa$

as for the densities, we assume that the ideal gas law holds, hence

$$\rho = \frac{p}{R_{specific} T}$$
$$R_{specific} = 287.058\,J\,Kg^{-1}\,K^{-1} \tag{44}$$

where $R_{specific}$ is appropriate for dry air. Further, we assume the two gases are both at room temperature, that is $T = 20C$

```
In [28]: # With Virgo 1500W conditions
         def rhoIdealGas(p,R,T):
             return p/(R*T)

         T_room = 20 + 273.15
         gamma = 1.4
```

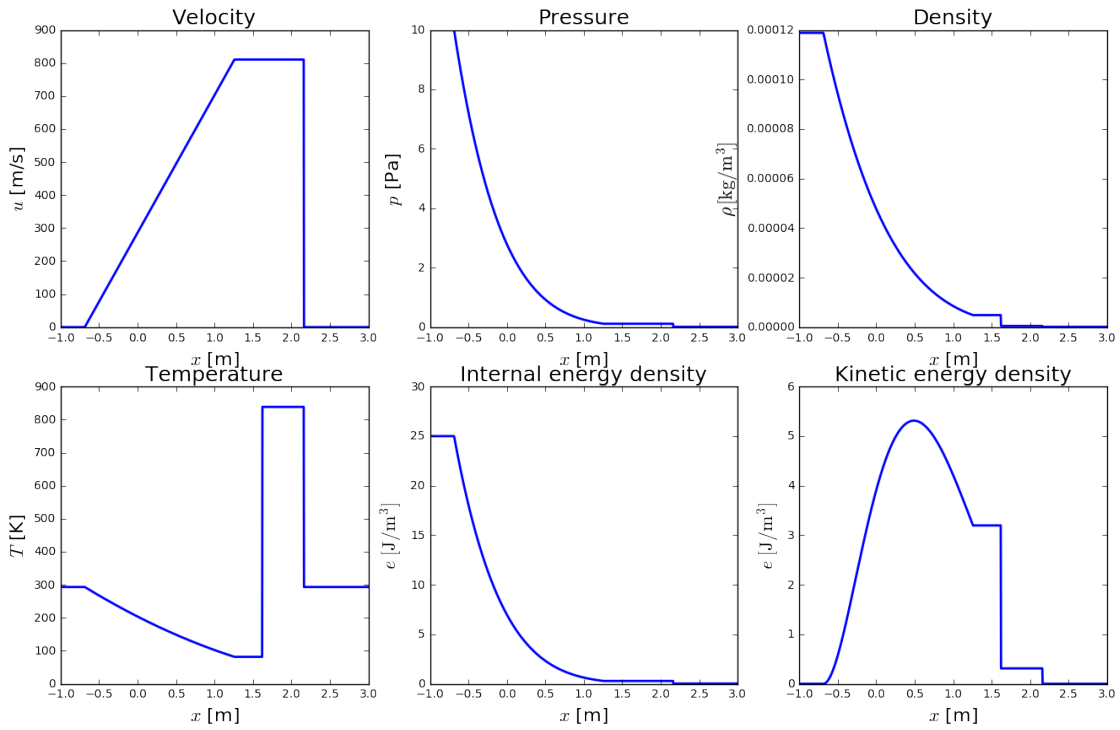
```
        p1 = 10.
        rho1 = rhoIdealGas(p1,R_air,T_room)
        p5 = 1.E-2
        rho5 = rhoIdealGas(p5,R_air,T_room)

        virgoTube = ShockTube(p1=p1,rho1=rho1,p5=p5,rho5=rho5)
        print virgoTube
```

```
Shock tube with $\gamma=$1.4:
$c_1=$343.236760531, $\rho_1=$0.000118833926364, $u_1=$0.0, $p_1=$10.0
$c_3=$181.167519286, $\rho_3=$4.86825325129e-06, $u_3=$810.346206228, $p_3=$0.11413
$c_4=$580.508861062, $\rho_4=$4.74150410755e-07, $u_4=$810.346206228, $p_4=$0.11413
$c_5=$343.236760531, $\rho_5=$1.18833926364e-07, $u_5=$0.0, $p_5=$0.01
```

Shock velocities are higher than in Sod's example, hence the shock propagates over much shorter time scales.

```
In [29]: # Let's check with Virgo 1500W
         %matplotlib inline

         import matplotlib.pyplot as plt

         xVec = numpy.linspace(-1,3,1000)
         uVec,pVec,rhoVec = virgoTube.State(xVec,0.002,verbose=True)
         displayState(xVec, uVec, pVec, rhoVec, gamma, R_air, "Virgo 1500 W")
```

```
x1 =  -0.686473521063  x2 =  1.25835737388  x3 =  1.62069241246  x4 =  2.162725364
```



In this case, the speed of the flow reaches **very** high velocities, order $800\,ms^{-1}$! But densities are low, can they drag dust particles?

The other interesting aspect is that the air at the shock front is heated considerably, up to $800K$. Again density is low, so this overheated air is probably not an issue.

3 Particle transport

We will assume a simple Lagrangian model for the transport of dust [Stover,2014]: neglecting gravity effects since we are in 1D, we will simply write that

$$\frac{du_p}{dt} = F_D\,(u - u_p) \tag{45}$$

where u_p is the speed of the dust particle in the air flux u and F_D is the Stokes drag coefficient, which can be expressed as

$$F_D = \frac{18\,\mu}{\rho_p d_p^2 C_c} \tag{46}$$

in which ρ_p, d_p are respectively the particle's mass density and diameter. We are assuming spherical dust, which is a *conservative* assumption: dust flakes will likely undergo larger accelerations.

The parameter μ is the **dynamic viscosity** of the fluid; at room temperature the air viscosity is assumed to be

$$\mu = 18.27 \times 10^{-6} Pa\,s \tag{47}$$

roughly independent on pressure.

The parameter C_c is the **slip correction factor** (Cunningham, 1910):

$$\begin{aligned} C_c &= 1 + \frac{2\lambda}{d_p}\left(A_1 + A_2\, e^{-\frac{A_3\, d_p}{\lambda}}\right) \\ A_1 &= 1.257, \qquad A_2 = 0.4, \qquad A_3 = 0.55 \qquad (\text{air}) \end{aligned} \tag{48}$$

in the formula, $\frac{\lambda}{d_p}$ is the ratio of the mean free path in the gas to the particle size, and corrects for gas rarefaction, driving F_D to zero as λ increases.

In turn, the mean free path of molecules having diameter d_m can be estimated using kinetic theory and a cross-section for collisions πd_m^2 (Pfeiffer Vacuum, mean-free-path)

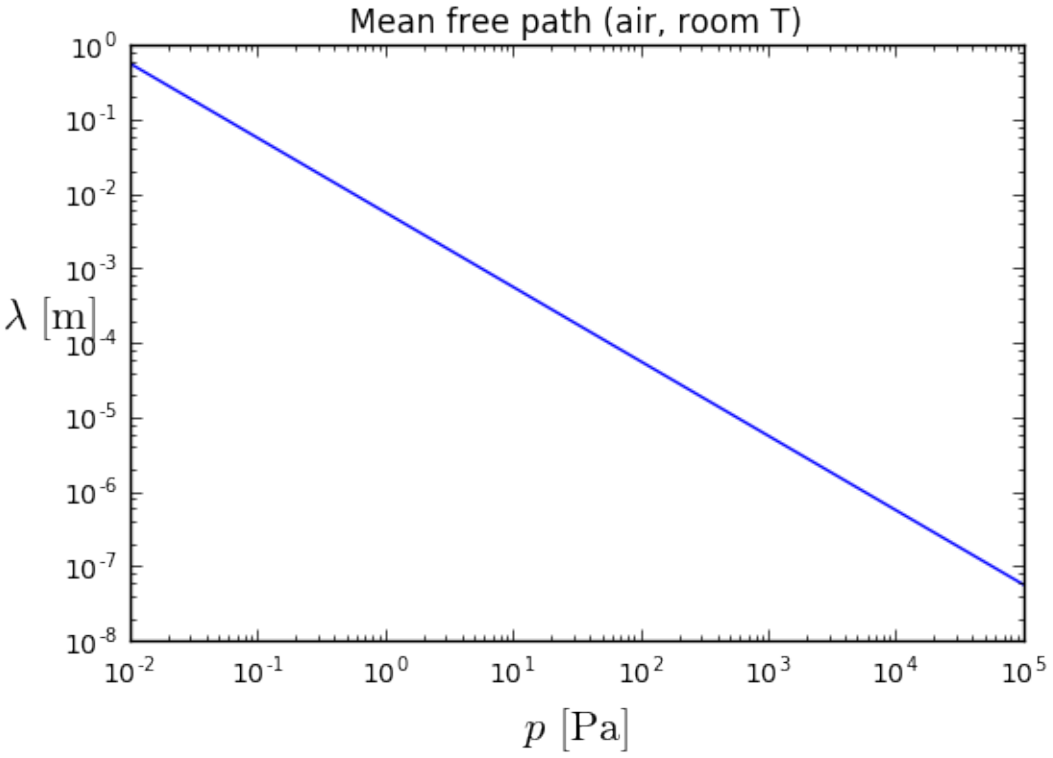
$$\lambda = \frac{k_B\,T}{\sqrt{2}\pi\,p\,d_m^2}\;; \tag{49}$$

the state equation for ideal gases allows to rewrite $T/p = (\rho\,R_{specific})^{-1}$ in terms of the gas density ρ and the gas constant per unit mass $R_{specific}$

$$\lambda = \frac{k_B}{\sqrt{2}\pi\,\rho\,R_{specific}\,d_m^2}\;; \tag{50}$$

we remind that $R_{specific}(\text{dry air}) = 287.058\;[Jkg^{-1}K^{-1}]$, and we will assume $d_m = 4 \times 10^{-10}$ as an average for air.

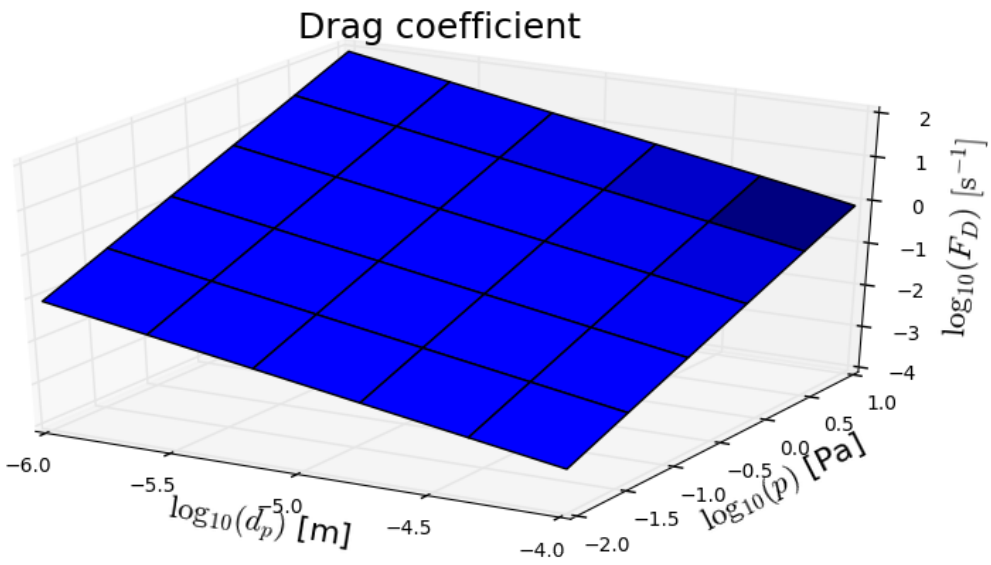
In [30]:
```
# We check here the mean free path

mu_air = 18.27E-6
R_specific = 287.058
d_m_air = 4.E-10

from scipy.constants import Boltzmann as kB

def f_lambda(rho,d_m):
    return kB/(math.sqrt(2.)*math.pi*rho*R_specific*d_m**2)

def f_C_c(lambda_mfp, d_p):
    return 1.0 + (2.*lambda_mfp)/d_p*(1.257 + 0.4*numpy.exp(-(0.55*lambda_mfp)/d_p))

def f_F_D(lambda_mfp, d_p, rho_p):
    return 18.*mu_air/(rho_p*(d_p**2)*f_C_c(lambda_mfp,d_p))

#f_lambda(1.,4.E-10) should be ~ 6.7e-8 m

p_vec = numpy.logspace(-2,5)
rho_vec = p_vec / (R_specific * T_room)

plt.plot(p_vec, f_lambda(rho_vec,d_m_air))
plt.title(r'Mean free path (air, room T)')
plt.yscale('log')
plt.xscale('log')
plt.xlabel(r'$p\ [\mathrm{Pa}]$',fontsize=16)
plt.ylabel(r'$\lambda\ [\mathrm{m}]$',rotation=0,fontsize=16)
```



The mean free path, obviously, varies significantly as a function of pressure and temperature, or equivalently of density. In the following we will have to take this into account, since gas density is not constant.

Note that the Cunningham correction factor is therefore **very significant** and cannot be neglected, in the regimes we consider. In the following 3D plot we show its dependency on d_p and the pressure p.

In [31]:
```
from mpl_toolkits.mplot3d import Axes3D

# assume rho_p appropriate for Al
rho_Al = 2700

d_Vec = numpy.logspace(-6,-4)
p_Vec = numpy.logspace(-2,1)
rho_Vec = p_Vec / (R_specific * 293.15)
lambda_Vec = f_lambda(rho_Vec,d_m_air)

d_Vec3D, p_Vec3D = numpy.meshgrid(d_Vec, p_Vec)
d_Vec3D, lambda_Vec3D = numpy.meshgrid(d_Vec, lambda_Vec)

F_D_Vec = numpy.log10(f_F_D(lambda_Vec3D, d_Vec3D, rho_Al))
d_Vec3D, p_Vec3D = numpy.meshgrid(numpy.log10(d_Vec), numpy.log10(p_Vec))

fig = plt.figure("Cunningham", figsize=(10,5))
axes = fig.gca(projection='3d')
axes.plot_surface(d_Vec3D, p_Vec3D, F_D_Vec)
axes.set_xlabel(r'$\log_{10}(d_p)$ [m]',fontsize=16)
axes.set_ylabel(r'$\log_{10}(p)$ [Pa]',fontsize=16)
axes.set_zlabel(r'$\log_{10}(F_D)\ [\mathrm{s}^{-1}]$',fontsize=16)
axes.set_title(r'Drag coefficient',fontsize=18)
```



If not for the C_c factor, the drag factor F_D would behave as d_p^{-2}, but for large λ/d_p, which is our case, C_c behaves as d_p^{-1} and the drag goes as d_p^{-1} as well. Anyway, smaller particles are dragged better, as expected. Also higher pressures help.

3.1 Simulating dust transport

We will assume that dust is transported by the air flow, without causing a back action on the gas: hence we can use the solutions obtained for the velocity u of the gas, the density ρ and the pressure p in order to deduce the specific force acting on a particle, according to the equations

$$\begin{aligned}\frac{dx_p}{dt} &= u_p(t) \\ \frac{du_p}{dt} &= F_D(\lambda(x_p,t), d_p, \rho_p)\left[u(x_p,t) - u_p(t)\right] \ . \end{aligned} \tag{51}$$

note that the F_D factor depends on quantities $\lambda(x_p,t), u(x_p,t)$ that in turn vary both with position and time, that's why we have to evolve also particle's position $x_p(t)$.

We will integrate in time with a predictor-corrector scheme, which is second-order accurate, with two steps

- predictor:

$$\begin{aligned} x_p^*(t+\Delta t) &= x_p(t) + \Delta t u_p(t) \\ u_p^*(t+\Delta t) &= u_p(t) + \Delta t F_D(\lambda(x_p,t), d_p, \rho_p)\left[u(x_p,t) - u_p(t)\right] \end{aligned} \tag{52}$$

- corrector:

$$\begin{aligned} x_p(t+\Delta t) &= x_p(t) + \frac{\Delta t}{2}\left[u_p(t) + u_p^*(t)\right] \\ u_p(t+\Delta t) &= u_p(t) + \frac{\Delta t}{2}\left\{F_D(\lambda(x_p,t), d_p, \rho_p)\left[u(x_p,t) - u_p(t)\right] + F_D(\lambda(x_p^*,t), d_p, \rho_p)\left[u(x_p^*,t) - u_p^*(t)\right]\right\} \end{aligned} \tag{53}$$

We need now a time grid and the evolution of the corresponding gas quantities. We have seen that eveything happens relatively quickly, so we want a sufficiently fine time grid.

```
In [32]: deltaT = 1.E-3
         T_sim = 0.2

         # At t~0 some formulas fail. Instead of correcting, we start shortly after
         t_Vec = numpy.linspace(10.*deltaT,T_sim,T_sim/deltaT)
         x_Vec = numpy.linspace(-40.,40.,1000)

         # build the status at different times
         # Rows address time, columns address position
         p_Mat   = numpy.zeros( (len(t_Vec), len(x_Vec)) )
         rho_Mat = numpy.array( p_Mat )
         u_Mat   = numpy.array( p_Mat )
         virgoTube = ShockTube(p1=p1,rho1=rho1,p5=p5,rho5=rho5)
         for (i,t) in zip(range(0,len(t_Vec)), t_Vec):
             u_Mat[i,:],p_Mat[i,:],rho_Mat[i,:] = virgoTube.State(x_Vec,t)
```

We can visualize the evolution by means of an animation (won't display in LaTeX)

```
In [33]: # Load libraries to animate and to convert into a movie
         from matplotlib import animation
         from IPython.display import HTML

         def animate(i,x,lines, matrices):
             for (line,mat) in zip(lines,matrices):
                 line.set_data(x,mat[i,:])
             return lines

         fig, axes = plt.subplots(1,3)
         fig.set_size_inches(15,4)
         line_u, = axes[0].plot([],[], lw=2)
         line_p, = axes[1].plot([],[], lw=2)
```

```
        line_rho, = axes[2].plot([],[], lw=2)
        for ax in axes:
            ax.set_xlim(x_Vec[0],x_Vec[-1])
        axes[0].set_ylim(0.,1000.)
        axes[0].set_title('Velocity')
        axes[1].set_ylim(1.E-3,10.)
        axes[1].set_yscale('log')
        axes[1].set_title('Pressure')
        axes[2].set_ylim(1.E-7,1.E-4)
        axes[2].set_yscale('log')
        axes[2].set_title('Density')
```

```
In [34]: # Try animating: we animate only up to 0.05 s
         anim = animation.FuncAnimation(fig,animate, frames=len(t_Vec)/4, fargs=(x_

         HTML(anim.to_html5_video())
```

```
Out[34]: <IPython.core.display.HTML object>
```

We have now air velocities, pressures and densities at specified grid positions: to compute values at arbitrary positions we need interpolating functions, that we will organize in dictionaries

```
In [35]: from scipy.interpolate import interp1d
         dict_u = {}
         dict_p = {}
         dict_rho = {}
         for i in range(0,len(t_Vec)):
             dict_u[i] = interp1d(x_Vec, u_Mat[i,:])
             dict_p[i] = interp1d(x_Vec, p_Mat[i,:])
             dict_rho[i] = interp1d(x_Vec, rho_Mat[i,:])
```

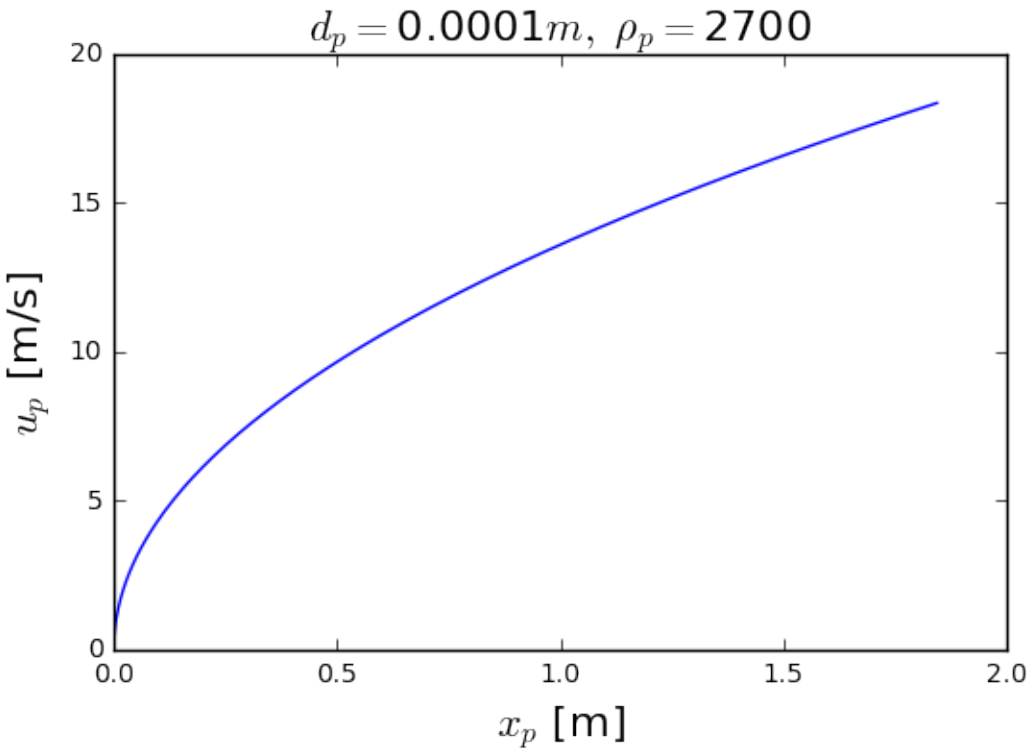
We can now easily generate the solution: we need a function which takes lots of inputs

```
In [36]: def evolveParticle(deltaT, t_Vec, xp_Vec, up_Vec, d_p, rho_p):
             for i in range(0,len(t_Vec)-1):
                 # Predictor step
                 xp_ast = xp_Vec[i] + deltaT*up_Vec[i]
                 F_D_pre = f_F_D(f_lambda(dict_rho[i](xp_Vec[i]),d_m_air), d_p, rho_p)*(dict_u[i](xp_Vec[i]) - up
                 up_ast = up_Vec[i] + deltaT*F_D_pre
                 # Corrector step
                 xp_Vec[i + 1] = xp_Vec[i] + deltaT/2*(up_ast + up_Vec[i])
                 F_D_cor = f_F_D(f_lambda(dict_rho[i](xp_ast),d_m_air), d_p, rho_p)*(dict_u[i](xp_ast) - up_ast)
                 up_Vec[i + 1] = up_Vec[i] + deltaT/2*(F_D_pre + F_D_cor)
```

Equipped with our integrator for the equations of motion, we can proceed.

3.2 Dust particle, spherical, diameter $100\mu m$

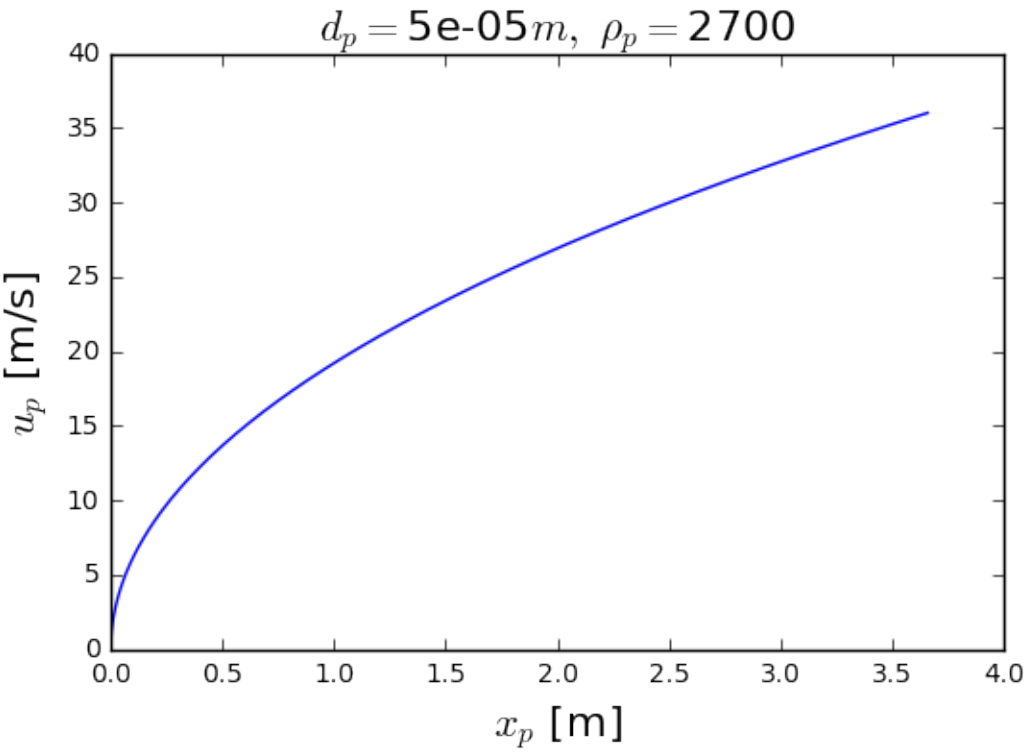
```
In [37]: xp_Vec = numpy.zeros( len(t_Vec) )
         up_Vec = numpy.zeros( len(t_Vec) )
         xp_Vec[0] = 0. # initial position of the particle

         dParticle = 100.E-6

         evolveParticle(deltaT, t_Vec, xp_Vec, up_Vec, dParticle, rho_Al)
         plt.plot(xp_Vec, up_Vec)
         plt.xlabel(r'$x_p$ [m]',fontsize=16)
         plt.ylabel(r'$u_p$ [m/s]',fontsize=16)
         plt.title(r'$d_p = ${}$ m,\ \rho_p = ${}'.format(dParticle, rho_Al),fontsize=16)
```



In this case, speeds of the order of tens of m/s can be reached, and within a few m.

3.3 Dust particle, spherical, diameter $50\mu m$

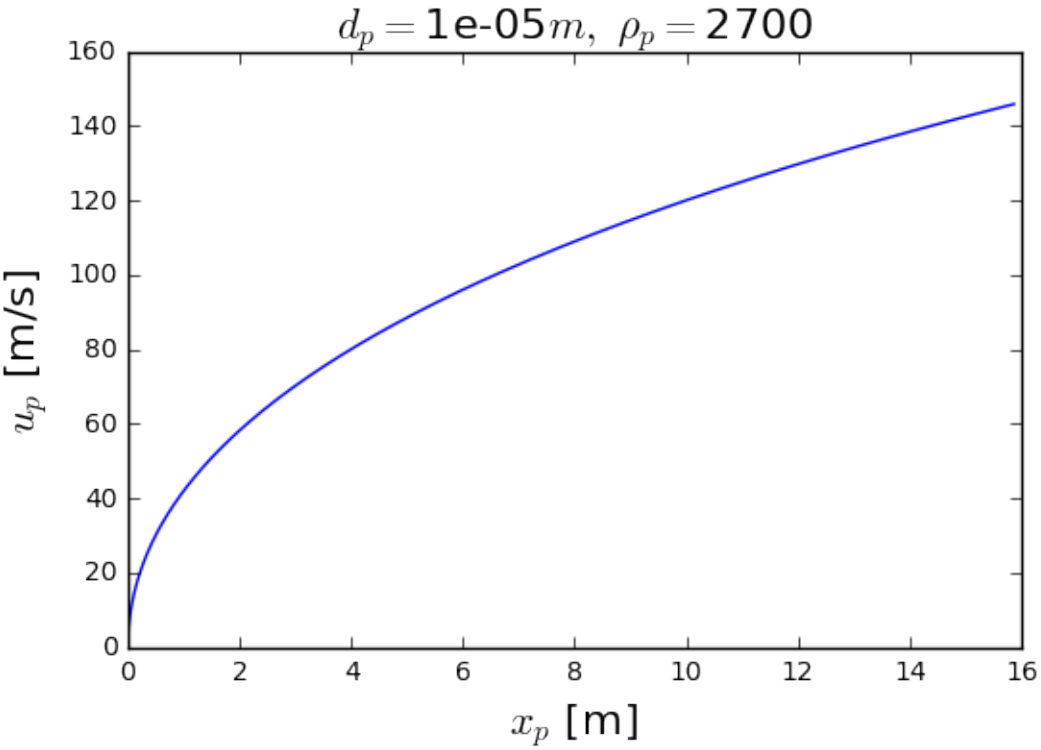
```
In [38]: xp_Vec = numpy.zeros( len(t_Vec) )
         up_Vec = numpy.zeros( len(t_Vec) )
         xp_Vec[0] = 0. # initial position of the particle

         dParticle = 50.E-6

         evolveParticle(deltaT, t_Vec, xp_Vec, up_Vec, dParticle, rho_Al)
         plt.plot(xp_Vec, up_Vec)
         plt.xlabel(r'$x_p$ [m]',fontsize=16)
         plt.ylabel(r'$u_p$ [m/s]',fontsize=16)
         plt.title(r'$d_p = ${}$ m,\ \rho_p = ${}'.format(dParticle, rho_Al),fontsize=16)
```



In this case, speeds can be much higher: after $1m$ of tube the particle is already travelling at $20ms^{-1}$.

3.4 Dust particle, spherical, diameter $10\mu m$

```
In [39]: xp_Vec = numpy.zeros( len(t_Vec) )
         up_Vec = numpy.zeros( len(t_Vec) )
         xp_Vec[0] = 0. # initial position of the particle

         dParticle = 10.E-6

         evolveParticle(deltaT, t_Vec, xp_Vec, up_Vec, dParticle, rho_Al)
         plt.plot(xp_Vec, up_Vec)
         plt.xlabel(r'$x_p$ [m]',fontsize=16)
         plt.ylabel(r'$u_p$ [m/s]',fontsize=16)
         plt.title(r'$d_p = ${}$ m,\ \rho_p = ${}'.format(dParticle, rho_Al),fontsize=16)
```

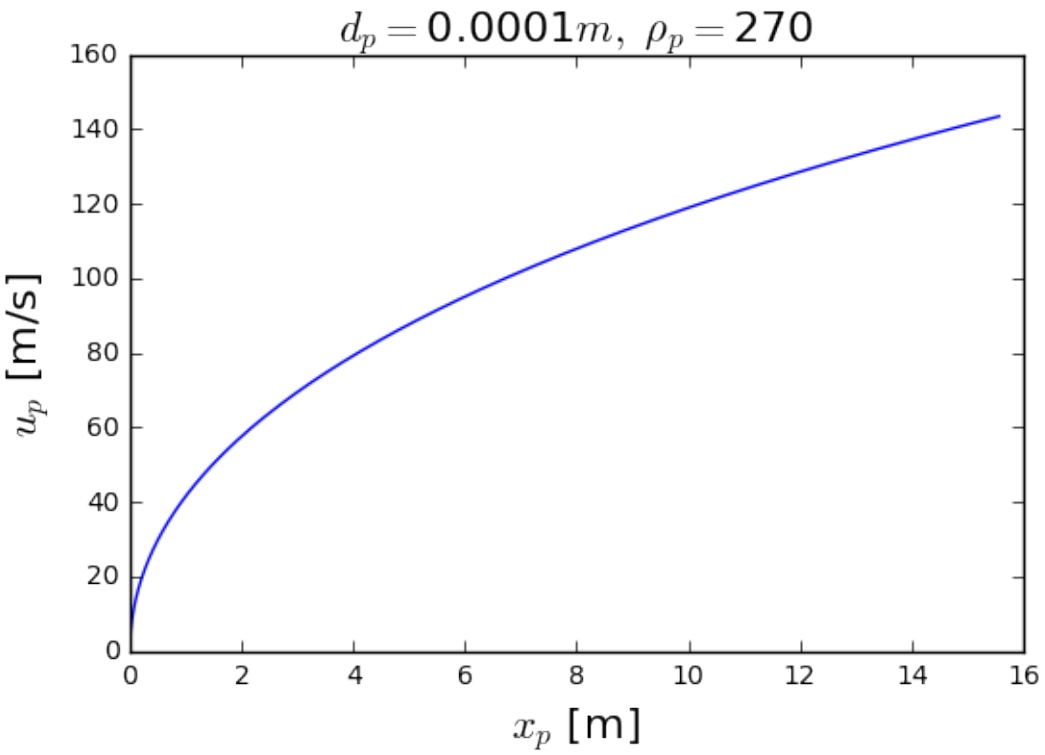


In this case, speeds can be much higher: after $1m$ of tube the particle is already travelling at $40ms^{-1}$. clearly, much higher speeds can be achieved if the geometry allows.

3.5 Dust particle, disk (or flake): disk diameter $100\mu m$, thickness $10\mu m$

This case is easily simulated, it is sufficient to lower the density by a factor 10, which takes into account the reduction of one of the dimensions.

```
In [40]: xp_Vec = numpy.zeros( len(t_Vec) )
         up_Vec = numpy.zeros( len(t_Vec) )
         xp_Vec[0] = 0. # initial position of the particle

         dParticle = 100.E-6

         evolveParticle(deltaT, t_Vec, xp_Vec, up_Vec, dParticle, rho_Al/10)
         plt.plot(xp_Vec, up_Vec)
         plt.xlabel(r'$x_p$ [m]',fontsize=16)
         plt.ylabel(r'$u_p$ [m/s]',fontsize=16)
         plt.title(r'$d_p = ${}$ m,\ \rho_p = ${}'.format(dParticle, rho_Al/10),fontsize=16)
```



Not surpringly, the speed achieved is the same as for the previous case: the reason is that, as we have discussed, in the regime considered the drag coefficient scales as d_p^{-1}, hence lowering d_p by a factor of 10, or lowering the particle's density by the same factor, achieve the same result.

Again, already after 1 m of tube the dust particle is traveling at about $40ms^{-1}$.

4 Conclusions

The pressure imbalance between two sections of tube respectively at $p_1 = 10\text{Pa}$, $p_5 = 10^{-2}\text{Pa}$ is sufficient to generate an air flow with speeds of the order of $800ms^{-1}$.

Even though densities are low, the flow is sufficient to accelerate dust particles to speeds wll above $10ms^{-1}$, and even higher depending on the shape and density.

A limitation of this study is that the cross-section of the tube is assumed constant. Work is in progress to simulate the case of an arbitrary variable cross section, which requires a numerical integrator.

References

- Sod, Gary A. (1978) "A survey of several finite difference methods for systems of non-linear hyperbolic conservation laws," *J. Comput. Phys.*, 27 (1978) 1–31 DOI: 10.1016/0021-9991(78)90023-2 // PDF from Uni-Tuebingen, checked Nov. 3 , 2016.
- Comparison of integration methods:

  http://www.thevisualroom.com/sods_test_problem.html#maccormack
- Stover, M. et al. (2014) "Simulation of windblown dust transport from a mine tailings impounding using a computational fluid dynamics model", Aeolian Res. 14 (2014) 75-83, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4303573/
- Cunningham, E. (1910 "On the velocity of steady fall of spherical particles through fluid medium," Proc. Roy. Soc. A 83 (1910) 357. DOI: 10.1098/rspa.1910.0024
- Pfeiffer Vacuum, https://www.pfeiffer-vacuum.com/en/know-how/introduction-to-vacuum-technology/fundamentals/mean-free-path/
- Bale, D.S. (2002) "Wave propagation algorithms on curved manifolds with applications to relativistic hydrodynamics", Ph.D. Thesis, Washington University http://faculty.washington.edu/rjl/students/dbale/thesis.ps.gz
- Hudson, J (1998) "Numerical techniques for conservation laws with source terms", MSc Dissertation, http://www.reading.ac.uk/web/files/maths/J_Hudson.pdf
- Höfner, S "Gas Dynamics: The Riemann Problem and Discontinuous Solutions: Application to the Shock Tube Problem", http://www.astro.uu.se/~hoefner/astro/teach/ch10.pdf