See page 31, page 39 and page 40 for input.

HYDROGEN STORAGE SIMULATION – PART 1

Marco Talice

Revision 02

Date: 22/06/2025

Author: Marco Talice

Contents

1 Introduction

The aim of the project is investigating the use of metal hydride tanks for hydrogen storage and the development of a dedicated numerical solver to simulate the physical processes occurring within the metal matrix during hydrogen absorption and desorption (Rabienataj Darzi AA 2016). The system under investigation consists of an 80 cm-long cylindrical vessel with an inner radius of 4 cm, filled with an intermetallic compound composed of lanthanum and nickel ($LaNi_5$). Hydrogen is injected or extracted through a coaxial inlet with a radius of 1 cm, under specified thermodynamic conditions (pressure and temperature).

The general form of the absorption/desorption reaction between the metal and hydrogen is:

$$M + \left({}^{x}/_{2}\right)H_2 \rightleftharpoons MH_x \pm \Delta H$$

This reaction is exothermic during absorption and endothermic during desorption. Since temperature significantly influences process efficiency, accurate control of thermal energy exchange is essential. Various methods may be employed for this purpose, including the use of heat exchangers and phase change materials (PCMs).

The research project divides into two phases:

- **Phase 1** focuses on developing and validating the numerical tool by testing it against a series of fundamental cases to ensure accurate resolution of the physical processes. This preliminary stage is limited to:
    1. Laminar flows
    2. Non-reactive flows in porous media
- **Phase 2** will involve the detailed modeling of hydrogen absorption/desorption processes, thermal exchange with the external environment, and temperature regulation.

This report covers the work conducted during Phase 1 of the research project and is structured as follows:

Section 1 outlines the working plan, establishing the foundation for the content presented in the subsequent sections.

Section 2 describes the development of a zero-dimensional model used to predict the equilibrium pressure inside a tank when hydrogen, at a given total pressure and temperature, is pumped into an empty tank (i.e., one not filled with metallic material).

Section 3 presents a comparison between results obtained using simpleFoam on a plane Poiseuille test case and the analytical velocity profile derived from the Hagen–Poiseuille equation.

Section 4 briefly discusses the implementation of the Darcy model for porosity in the OpenFOAM solvers. It also compares the results from simpleFoam simulations of laminar flow through a plane channel filled with a porous medium (of assigned permeability) to predictions from the Darcy model, highlight the shortcomings of the solver, and suggest a possible way-out.

Section 5 considers a more realistic scenario in which hydrogen, at a specified total pressure and temperature, is pumped into an empty tank. The simulation is time-accurate and aims to predict the moment at which the internal tank pressure equals the imposed inlet pressure. The results are compared to those from the zero-dimensional model introduced in Section 2.

Section 6 presents results obtained in a similar scenario, this time with the tank filled with a metallic, non-reactive matrix possessing the physical characteristics used in the actual device.

Finally, Section 7 summarizes the main conclusions and outlines the proposed research plan for Phase 2 of the project.

2 Zero-Dimensional (0D) Reactor Model for Pressure

To model a gas tank being filled, we consider a 0-D (lumped parameter) approach where pressure $(P(t))$, temperature $(T(t))$, and inlet velocity $(v_{\text{in}}(t))$ evolve over time.

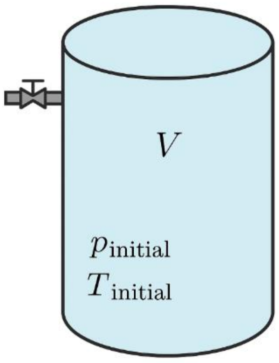


Figure 1: The tank of volume V is initially filled with H_2 at pressure and temperature $p_{initial}$ and $T_{initial}$. A hydrogen flow with total pressure and temperature $p_{T,in}$ and $T_{T,in}$ is fed to the tank through the inlet pipe

A zero-dimensional (0-D) tank-filling model enhanced with a hybrid acoustic “round-trip” delay and damping to better emulate the transient behavior seen in compressible CFD (e.g., OpenFOAM `rhoPimpleFoam`) simulations is developed here. The model computes hydrogen filling into a capped cylindrical tank via a constant-stagnation-pressure inlet, tracking tank pressure, temperature, mass, and inlet mass-flow rate.

2.1 Thermodynamics and gas properties

Hydrogen can be treated as an ideal diatomic gas, for which the following relations hold:

$p = \rho RT, \qquad c = \sqrt{\gamma RT}, \qquad U = m\,c_v\,T = \frac{mRT}{\gamma - 1}$[1]

Where: p [Pa] is the static pressure, $\rho = m/V$ [kg/m³] is density, R [J/(kg·K)] is the specific gas constant, T [K] is the static temperature, γ the ratio of specific heats, c [m/s] the local speed of sound, and, finally, U [J] is the internal energy of the gas in the tank.

2.1.1 Compressible-nozzle (inlet) mass-flow rate

We can model the inlet as an isentropic nozzle with assigned stagnation ("total") conditions (P_t, T_t).

$$M = \frac{V_\infty}{\sqrt{\gamma RT}}, \quad T_t = T\left(1 + \frac{\gamma-1}{2}M^2\right), \quad P_t = p\left(1 + \frac{\gamma-1}{2}M^2\right)^{\gamma/(\gamma-1)}$$

The Mach number can be computed from the specified Reynolds number through V_∞:

$$V_\infty = \frac{\mu\, Re}{\rho\, D_i}$$

Where D_i is the inlet diameter. This is useful to ensure the "nature" of the flow, laminar or turbulent.

Then the mass-flow through the inlet area A can be:

1. **Subcritical** $(p/P_t > p^*/P_t)$:

Equation 1

$$m = A\,P_t \sqrt{\frac{\gamma}{R\,T_t}} \left(\frac{p}{P_t}\right)^{1/\gamma} \sqrt{\frac{2}{\gamma - 1}\left[1 - \left(\frac{p}{P_t}\right)^{\frac{\gamma-1}{\gamma}}\right]}$$

This follows directly from the isentropic relations and continuity.

2. **Choked** $(p/P_t \leq \left(2/(\gamma+1)\right)^{\gamma/(\gamma-1)})$:

Equation 2

$$m = C_d\,A\,P_t \sqrt{\frac{\gamma}{R\,T_t}} \left(\frac{2}{\gamma + 1}\right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

[1] Under the ideal gas assumption, the internal energy per unit mass (specific) is $u = c_v T = \frac{RT}{\gamma-1}$, so the total tank internal energy $U = m\,c_v T$.

At this critical pressure ratio, the throat Mach number is unity and the mass-flow no longer increases with further back-pressure decrease. C_d is a discharge coefficient (often ≈0.8–1). We can compute a critical pressure, as:

$$p_{\text{crit}} = P_t \left(\frac{2}{\gamma + 1}\right)^{\gamma/(\gamma-1)}$$

We also require no reverse flow **(**$\dot{m} = 0$) if P_t is actually lower than the static pressure p.

2.1.2 Modifications needed for porous media

When considering the presence of a porous material that fills the tank, the value of pressure p used in the Equation 1 and Equation 2 needs to be modified by the pressure drop due to the presence of the porous media. This pressure drop can be computed via the Darcy law, as:

$$\Delta p = \frac{\mu L_t}{\mathrm{K}} V$$

Where µ and ρ are the fluid's dynamic viscosity and density, L_t the tank's length, and K is the porous media permeability.

So, in the Equation 1 and Equation 2 a modified pressure value $p - \Delta p$ is used instead of p.

2.2 Tank energy and mass balance

Assuming spatially uniform p and T inside the tank, we can write conservation laws of mass and energy between time t and t + dt:

Mass conservation:

$$m(t + dt) = m(t) + \dot{m}_{\text{eff}}\, dt$$

Energy conservation:

Equation 3

$$U(t + dt) = U(t) + \dot{m}_{\text{eff}}\, h_t\, dt, \quad h_t = c_p\, T_t = \frac{\gamma R T_t}{\gamma - 1}$$

Since $U = m\, c_v\, T$, T can be updated via:

$$T = \frac{(\gamma - 1)\, U}{m\, R}, \qquad p = \frac{m\, R\, T}{V}$$

This is a standard "lumped-parameter" model for a pressurizing vessel.

As a side note, we must observe that when we write the tank's energy-balance as:

$$\underbrace{U(t+dt)}_{\text{new internal}} = \underbrace{U(t)}_{\text{old internal}} + \underbrace{\dot{m}_{\text{eff}} h_t dt}_{\text{enthalpy injection}}$$

we are explicitly ignoring the following three other possible reservoirs or fluxes of energy:

1. Heat-transfer to or from the tank shell
2. Kinetic-energy changes of the bulk gas in the tank
3. Potential-energy changes (elevation work)

Below is a deeper look at each of these terms and why, in typical "rapid fill" scenarios, they can safely be neglected.

2.2.1 Heat-transfer (Q)

In more general terms, if the tank walls are at temperature $T_{\text{wall}} \neq T(t)$, there will be a conductive/convective heat-flow to/from the gas and the tank's wall:

$$\dot{Q} = h_{\text{conv}}\, A_{\text{wall}}\, [T_{\text{wall}} - T(t)]$$

that would appear in the energy-balance as:

Equation 4

$$U(t+dt) = U(t) + \dot{m}\, h_t\, dt + \dot{Q}\, dt$$

There are some valid justifications for neglecting this contribution to the energy equation. The first one is the typically short filling time (fractions of a second to a few seconds). To clarify this, we better refer to the thermal time constant and its meaning. When we model the "tank wall + gas" as a single lumped mass with heat capacitance C and overall conductance K to the surroundings, its temperature response to a step change in ambient is:

$$T(t) = T_\infty + [T(0) - T_\infty]\, e^{-t/\tau_{\text{th}}}$$

Where $\tau_{\text{th}} = \frac{C}{K}$ is the thermal time constant. As an example, a τ_{th} of "tens to hundreds of seconds" means that in one time-constant the tank walls and gas would change by about 63% of the way toward the ambient temperature. We can then compare the fill time (typically less than 1 s up to a few seconds) with the thermal time constant (e.g. 50–200 s).

Because the fill happens in $t_{\text{fill}} \ll \tau_{\text{th}}$, the wall-to-gas heat transfer has barely "gotten started" by the time the fill is over. Quantitatively, the fraction of a temperature change in time t_{fill} is:

$$1 - e^{-\,t_{\text{fill}}/\tau_{\text{th}}} \;\approx\; \frac{t_{\text{fill}}}{\tau_{\text{th}}} \quad (\text{for } t_{\text{fill}} \ll \tau_{\text{th}})$$

So, if $t_{\text{fill}} = 2$ s and $\tau_{\text{th}} = 100$ s, the response would be approximately 2%.

The thermal time constant of a metal tank is typically tens to hundreds of seconds, so little heat crosses the wall during the fill.

A second reason to neglect the heat transfer is the presence of good insulation (or low Biot number). If the tank is well-insulated or thick-walled, the interior gas hardly “sees” the external temperature. This could not be the case if one wants to model the system presented in (Rabienataj Darzi AA 2016), as thermal energy is actually subtracted or given to the system during the phases of adsorption and desorption. In this case the energy flux must be included in the energy equation and Equation 4 should be used in Equation 3.

A third, and final reason, to neglect heat transfer is the dominance of enthalpy inflow. In fast fill processes, $\dot{m}\, h_t$ is orders of magnitude larger than $\dot{Q}$. As an example, for a 0.8 m^3 tank being filled in $\sim 1\ s$, $\dot{m}h_t$ might be on the order of 10^5 W, whereas the heat loss is a few hundred watts (< 1%).

2.2.2 Kinetic energy of the tank gas

If the bulk velocity inside the tank were significant, we would have:

$$\Delta E_{\text{kin}} = \frac{1}{2}\, m\, \overline{v}^2$$

and any change in $\frac{1}{2} m\, v^2$ would augment the energy-balance.

In practice this term can be neglected for the following two reasons:

Well-mixed, low-velocity: once inside the tank the flow decays quickly, typical bulk velocities are < 1 m/s in a few-liter tank, resulting in $\frac{1}{2} \rho V v^2 \ll U$.

Small relative magnitude: for example, even $v = 5$ m/s in a 0.01 m^3 tank yields $E_{\text{kin}} \sim 0.5 \cdot \rho \cdot 0.01 \cdot 25$ $\sim$ a few joules, versus internal energy on the order of 10^5 J.

2.2.3 Potential-energy changes

If the tank or the gas column is lifted against gravity g, the associated energy would be:

$$\Delta E_{\text{pot}} = m\, g\, \Delta z$$

This term is usually negligible as tanks are subjected to small elevation differences (unless we are pumping from a deep well or a height difference of many meters, Δz is negligible). For example, if $m = 1$ kg, $\Delta z = 1\ \ m\ \Rightarrow\ m\, g\, \Delta z\ \approx\ 10$ J, again tiny compared to the internal-energy changes (10^4–10^5 J).

2.3 Hybrid-lag (transport delay + first-order filter)

Real piping causes both a time-of-flight delay and a distributed inertance/compliance. In the general context of compressible flows in pipes or ducts, the terms inertance and compliance refer to the two key physical effects that govern how pressure disturbances propagate and attenuate. When “lumping” these effects uniformly along the line, we get what is called a distributed inertance/compliance model, which underpins the hybrid delay + filter approach used in the model.

To better understand the physics at play, we can think of the pipe as an acoustic transmission line. A compressible fluid in a long, straight pipe can be described by the one-dimensional linearized continuity and momentum equations:

Equation 5

$$\frac{\partial \rho'}{\partial t} + \rho_0 \frac{\partial u'}{\partial x} = 0$$

Equation 6

$$\rho_0 \frac{\partial u'}{\partial t} + \frac{\partial p'}{\partial x} = 0$$

Where $p'(x,t)$, $\rho'(x,t)$, and $u'(x,t)$ are small deviations in pressure, density, and velocity, and ρ_0 is the mean density.

Equation 5 and Equation 6 form the classic wave-equation for acoustic disturbances, propagating at the speed of sound $c = \sqrt{\gamma R T}$.

If we instead view the line as a cascade of infinitesimal lumped elements each of length dx, we can associate to each element dx:

- its inertance L' [Pa·s^2/m^3] per unit length, that represents the fluid's inertia, (i.e. the mass that must be accelerated for a change in flow). From momentum:

  $$L' = \frac{\rho_0}{A}$$

  where A is the cross-sectional area of the pipe.

- Its compliance C' [m^3/(Pa·m)] per unit length, that represents the fluid's compressibility, (i.e. how much the volume changes for a given pressure change). From continuity and the ideal-gas law:

  $$C' = \frac{A}{\rho_0 \, c^2}$$

So, over a segment dx, we have an inertance $L'\,dx$ in series with a compliance $C'\,dx$ to ground (in an electrical-analogy sense). Stacking these along the pipe gives the full transmission-line model of the duct.

Solving the full transmission-line (i.e. a PDE in x and t) can be expensive in a system simulation. A common simplification is:

1. Pure time delay $\tau_0 = \frac{L_{\text{total}}}{c}$: it captures the finite propagation time of pressure waves down the pipe.
2. Single lumped compliance + inertance equivalent to the upstream volume (in our case, the tank volume plus a bit of line volume), which behaves like a first-order low-pass filter on the mass-flow signal.

Concretely, if we treat the entire upstream "line" as one inertance L_{lump} feeding into one compliance C_{lump}, the linearized relationship between the pressure-drop Δp and flow $\dot{m}$ is:

Equation 7

$$L_{\text{lump}} \frac{d\dot{m}}{dt} + \frac{\dot{m}}{C_{\text{lump}}} = \Delta p$$

Rearranging, changes in $\dot{m}$ have a characteristic time-constant

$$\tau = L_{\text{lump}}\, C_{\text{lump}}$$

Once discretized, Equation 7 provides:

$$\dot{m}_{\text{eff}}(t+dt) = \dot{m}_{\text{eff}}(t) + \frac{dt}{\tau}[\dot{m}_{\text{del}}(t) - \dot{m}_{\text{eff}}(t)]$$

Which is a first-order filter on the delayed flow $\dot{m}_{\text{del}}$, with time-constant τ. Physically, it models the fact that the fluid cannot instantaneously accelerate (inertance) or being compressed (compliance) to produce a new flow. There is a "blending" or smoothing of any abrupt changes.

Summarizing: the delay captures the finite speed-of-sound propagation. A change of the inlet mass-flow only affects the tank after a time τ_0; the filter captures the inertia + elasticity. Once the disturbance arrives, it doesn't produce an instant change in flow into the tank but is smoothed over the acoustic

2.4 Octave implementation

The model has been implemented in Octave. Figure 2 lists the implemented Octave script. The model covers the case of adiabatic or isothermal flow, with or without porosity.

```
% Zero-D Fill Model with Hybrid Delay + Stored Stop Time on Delayed mdot
clear; clc;

%—— Physical constants & geometry ——
R     = 4124;        % J/(kg·K)
gamma = 1.40;
Lt    = 0.80;       % m – tank length
Li    = 0.10;       % m – pipe length
L     = Lt + Li;    % total length
Rt    = 0.04;       % m – tank radius
Ri    = 0.01;       % m – inlet pipe radius
s     = 5;          % deg sector
A     = (s/360)*pi*Ri^2;     % inlet area [m^2]
At    = (s/360)*pi*Rt^2;     % Tank cross-section area [m^2]
Vt    = (s/360)*pi*Rt^2 * Lt; % tank volume [m^3]
Vi    = (s/360)*pi*Ri^2 * Li; % pipe volume [m^3]
V     = Vt + Vi;             % total volume [m^3]
IsDarcy = 0;                 % switch to activate/deactivate Darcy model
IsAdiabatic = 1;
K     = 1e-09;               % permeability [m^2]
Cd    = 1;

%—— Time discretization ——
t_end = 100;     % total sim time [s] (made a bit longer)
nt    = 10000;    % number of steps
dt    = 1e-06; %t_end/nt;
time  = (0:nt)*dt;

%—— Inlet & initial conditions ——
p     = 101325;   % Pa
T     = 300;      % K

Re    = 2000; mu = 8.4e-6;
rho_s = p/(R*T);
Vinl  = Re*mu/(rho_s*2*Ri);
Mach  = Vinl/sqrt(gamma*R*T);
Pt_in = p*(1+0.2*Mach^2)^3.5;
Tt_in = T*(1+0.2*Mach^2);
Pr    = 0.7;

%—— Preallocate storage ——
P          = zeros(nt+1,1);
Tvec       = zeros(nt+1,1);
M          = zeros(nt+1,1);
rho        = zeros(nt+1,1);
mdot_raw   = zeros(nt+1,1);
mdot_del   = zeros(nt+1,1);
mdot_filt = zeros(nt+1,1);
dp_Darcy  = zeros(nt+1,1);
m_inflow = zeros(nt+1, 1);
P(1)=p; Tvec(1)=T; M(1)=p*V/(R*T); rho(1) = p(1)/(R*Tvec(1));

%—— Hybrid-lag parameters ——
c0         = sqrt(gamma*R*T);
tau0       = 2*L / c0;
Ndelay     = ceil((40*IsDarcy +2*(1-IsDarcy))*tau0 / dt); %we want to slower flow decay in case of porous media
buffer     = zeros(Ndelay,1);
bufIdx     = 1;
```

```
alpha_fac = 1; % filter τ_factor
mdot_eff  = 0;

%—— Stop-criterion setup ——
stopTime = NaN;
reached  = false;
relTol   = 1e-9;
thresh   = 0;     % kg/s threshold for zero

%—— Main loop ——
for i = 1:nt
  t = (i-1)*dt;

  % 1) raw mdot
  % Critical pressure for choked flow
  p_crit = Pt_in * (2/(gamma+1))^(gamma/(gamma-1));

  if p < Pt_in
    ratio = p/Pt_in;
    mdot = A * Pt_in * sqrt(gamma/(R*Tt_in)) ...
         * ratio^(1/gamma) ...
         * sqrt((2/(gamma-1))*(1 - ratio^((gamma-1)/gamma)));
  elseif p < p_crit
    mdot = Cd * A * Pt_in * sqrt(gamma/(R*Tt_in)) ...
         * (2/(gamma+1))^((gamma+1)/(2*(gamma-1)));
  else
    mdot = 0;
  end

  mdot_raw(i) = mdot;

  % 2) delay
  mdot_del(i)      = buffer(bufIdx);
  buffer(bufIdx)   = mdot;
  bufIdx           = bufIdx + 1;
  if bufIdx > Ndelay, bufIdx = 1; end

  % 3) filter (for dynamics only)
  c     = sqrt(gamma * R * T);
  tau   = 2 * Lt / c;
  alpha = dt / (alpha_fac * tau);
  mdot_filt(i) = mdot_eff;
  mdot_eff = mdot_eff + alpha * (mdot_del(i) - mdot_eff);

  if P(i) >= Pt_in
    mdot = 0;
    buffer(:) = 0;          % flush buffer
    mdot_del(i) = 0;
    mdot_eff = 0;
  end

  if i > 1
    m_inflow(i+1) = m_inflow(i) + mdot_eff * dt;
  else
    m_inflow(i) = mdot_filt(i) * dt;
  end
  % 4) check zero-flow on the delayed signal
  if ~reached && i > Ndelay && mdot_del(i) <= thresh % it was mdot_del
    stopTime = t;
    reached  = true;
       % Final update before break
    m_inflow(i+1)  = m_inflow(i) + mdot_filt(i) * dt;
    M(i+1)         = m;
    P(i+1)         = p;
    Tvec(i+1)      = T;
    rho(i+1)       = rho_val;
    dp_Darcy(i+1)  = dp_Darcy_val;
    break;
  end

  % 5) update tank state using mdot_eff
  if (i == 1 )
    InitialMdot = p(i)*V/(R*Tvec(i));
    thresh = relTol * InitialMdot;
  end

  m = M(i);
  if(IsDarcy == 1) Ut_val = m/(rho(i)*At); dp_Darcy_val = mu*Lt/K * Ut_val; else dp_Darcy_val = 0; end
  if (IsAdiabatic)
    U = m * R * T / (gamma - 1);
    U = U + mdot_filt(i) * dt * gamma * R * Tt_in / (gamma - 1);
    m = m + mdot_eff * dt;
    T = (gamma - 1) * U / (m * R);
  else
    T = 300;  % Fix temperature to wall temperature
    m = M(i) + mdot_eff * dt;
  end
  p_static = m * R * T / V;  % without Darcy loss
  if p_static >= Pt_in
    mdot = 0;
  end
  p = p_static - dp_Darcy_val;
  rho_val = p/(R*T);
  dp_Darcy(i) = dp_Darcy_val;
  pfill = p - dp_Darcy_val;

  P(i+1)    = p;
  Tvec(i+1) = T;
  M(i+1)    = m;
  rho(i+1)  = rho_val;
  dp_Darcy(i+1) = dp_Darcy(i);
  m_inflow(i+1) = m_inflow(i) + mdot_filt(i) * dt;
  pfill = p ;

end

%—— Report stop time ——
if reached
  net_mass_added = M(i+1) - M(1);
  fprintf('Initial mdot = %.9e kg/s \n', InitialMdot);
  fprintf('Convergence control = %.9e \n', thresh);
  fprintf('Delayed mdot ≤ %.1e kg/s at t = %.6f s\n', thresh, stopTime);
  fprintf('Inlet initial conditions, Re = %.0f rho = %7.6f kg/m^3 V = %6.4f m/s\n', Re, rho_s, Vinl);
  fprintf('Inlet initial conditions, Mach = %7.6f Pt = %14.8f Pa Tt = %14.8f K\n', Mach, Pt_in, Tt_in);
  fprintf('Initial Lag Time = %7.6f s Final Lag Time = %7.6f s\n', tau, tau0);
  fprintf('Net Filling Time (w/o delay) = %7.6f s\n', stopTime-tau0);
  fprintf('Total Filling Time (w/ delay) = %7.6f s\n', stopTime);
  fprintf('alpha_factor = %7.6f \n', alpha_fac);
  fprintf('final tank pressure = %.6f Pa\n', pfill);
  fprintf('\n--- Mass Balance Report ---\n');
  fprintf('Initial tank mass = %.9e kg\n', M(1));
  fprintf('Final tank mass   = %.9e kg\n', M(i+1));
  fprintf('Net mass added    = %.9e kg\n', M(i+1) - M(1));
  fprintf('Cumulative inflow = %.9e kg\n', m_inflow(i+1));
  fprintf('Mass balance error = %.3e %%\n', 100 * (M(i+1) - M(1) - m_inflow(i+1)) / (M(i+1) - M(1)));
else
  fprintf('Delayed mdot never ≤ %.1e kg/s in %.6f s\n', thresh, t_end);
end

%—— Plot ——
fig = figure('Name', 'Tank Simulation Results');
% Set figure size: [left bottom width height] in pixels
set(fig, 'Position', [0, 0, 800, 700]);  % adjust values as needed
subplot(3,1,1);
plot(time, P/1); xlabel('Time [s]'); ylabel('p [Pa]'); grid on; title('Tank Pressure'); xlim([0, 1.1*stopTime]);

subplot(3,1,2);
plot(time, mdot_raw,'--', time, mdot_del,':', time, mdot_filt,'-');
xlabel('Time [s]'); ylabel('mdot [kg/s]');
legend('raw','delayed','delayed+filt', "location" , "northwest"); grid on; title('Inlet Mass Flow'); xlim([0, 1.1*stopTime]);

subplot(3,1,3);
plot(time, sqrt(gamma*R*Tvec));
xlabel('Time [s]'); ylabel('c [m/s]'); grid on; title('Sound Speed'); xlim([0, 1.1*stopTime]);
```

Figure 2: the Octave script that implements the 0-D model

3 Laminar Channel Flow (Poiseuille)

Laminar Poiseuille flow describes the steady, incompressible flow of a Newtonian fluid driven by a constant pressure gradient between two infinite, parallel plates separated by distance H. Because viscous forces dominate (low Reynolds number), the velocity profile is fully developed into a parabola in the wall-normal direction, and there is no cross-stream (secondary) flow.

3.1 Velocity Profile

For plates located at y=0 and y=H, driven by $\frac{dp}{dx} = -G(constant)$, the analytical solution is

Equation 8

$$u(y) = \frac{G}{2\mu} y (H - y) = \frac{3}{2} U_{\text{avg}} \left[1 - \left(\frac{2y}{H} - 1\right)^2\right]$$

Where:

- μ is the dynamic viscosity,

- $U_{\text{avg}} = \frac{1}{H}\int_0^H u(y)\ dy = \frac{GH^2}{12\mu}$

the centerline (peak) velocity is: $U_{max} = \frac{GH^2}{8\mu} = \frac{3}{2}\ U_{\text{avg}}$

3.2 Entrance-Length for Fully-Developed Flow

A “fully developed” flow occurs when the two wall-layers merge at the channel’s mid-height. Empirically for laminar channel flow one finds

Equation 9

$$L_{f.d.}\ \approx\ 0.05\ Re\ H \quad \text{(to within 5–10\% accuracy)}.$$

3.3 Mesh Resolution Advice

To capture the parabolic profile and the developing boundary layer accurately in a CFD mesh:

- **Wall-normal (y) direction**:
  - At least 20–30 cells should be used across the full channel height H.
  - Cells clustering near the walls should be used so that the boundary-layer region (where velocity gradients are steepest) has 10–15 cells within the boundary layer thickness.
- **First-cell height**:
  - Choose the first cell so that $y^+ \lesssim 1$ (for laminar, ensure it’s small enough to resolve the viscous sublayer; this typically means $\Delta y_1 \approx 0.001\,H$ or finer at moderate Re).
- **Streamwise (x) direction**:
  - Use a refinement in the entrance length region $x\ <\ 0.1\,Re\,H$ to capture boundary-layer growth: a streamwise aspect ratio near unity (or up to 2:1) is recommended in that region.

3.4 simpleFoam Test Case

This section describes how to set up the Poiseuille test case using a geometry similar to that employed in (Rabienataj Darzi AA 2016), with a few modifications. In our case, the tank is modeled as planar, open at both ends, and the inlet—through which hydrogen enters the tank—extends over the entire height H of the channel.

The geometry dimensions in the x, y, and z directions are:

- Length L=0.8 m
- Height H=0.04 m
- Width W=0.01 m

All formulas presented in Section 3.3 above depend on the Reynolds number, Re, of the flow. In the work of Darzi et al. (Rabienataj Darzi AA 2016) the Reynolds number is not specified, as the boundary condition for the hydrogen flow is given in terms of total pressure, rather than a specified mass flow rate. However, based on the general context of their study, it is reasonable to assume that the hydrogen flow inside the tank is laminar[2]. Since the exact Reynolds number used in their simulations is unknown, we are free to choose a suitable value that ensures the hydrodynamic entrance length remains well below the total channel length L.

Assuming that the flow becomes fully developed within the first 10% of the channel length, we use the empirical relation for the entrance length of laminar flow:

$$0.1 * \mathrm{L} \ = \ 0.05\, Re \ * \ H \Longrightarrow 0.1 * 0.8 = 0.05\mathrm{Re} * 0.04$$

Solving for Re:

$$Re = \frac{0.8}{0.04 * 0.05} = 40$$

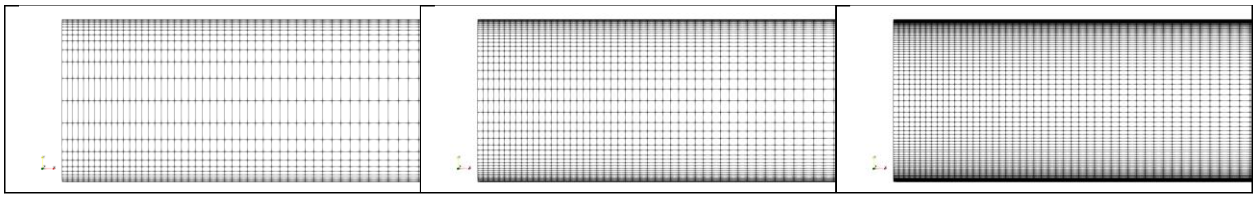
For meshing purposes, we also require that the height of the first cell near the wall is approximately: $\Delta y_1 \approx 0.001\, H = 4\mathrm{x}10^{-5}\ \mathrm{m}$

Furthermore, we apply cell clustering in both the streamwise (x) and wall-normal (y) directions to accurately resolve the growth of the boundary layer.

3.5 Mesh Generation

The computational mesh was generated using the blockMesh tool available in OpenFOAM. The corresponding blockMeshDict is provided in APPENDIX I.

To assess grid independence, three meshes were generated. While the number of cells in the streamwise direction (x) was kept constant, the number of cells in the wall-normal direction (y) was varied: 24, 48, and 96, corresponding to coarse, medium, and fine meshes, respectively. The center-to-wall distances of the first cell near the wall for the three meshes are approximately $1.73\mathrm{x}10^{-4}\ \mathrm{m}$, $8.98\mathrm{x}10^{-5}\ \mathrm{m}$, and $4.58\mathrm{x}10^{-5}\ \mathrm{m}$, with the latter value being close to the target Δy_1 $(4\mathrm{x}10^{-5}\ \mathrm{m})$. Each mesh contains 160 cells in the x-direction. Only one cell is used in the z-direction, as this is a 2D simulation. The first cell near the inlet section has a size of approximately $6.38\mathrm{x}10^{-4}\ \mathrm{m}$. The total number of cells in the three meshes are 3840, 7680, and 15360 cells. Figure 3 shows an enlarged view of the three mesh levels.



[2] In the actual device, which is filled with a porous metallic matrix where hydrogen is adsorbed, it is reasonable to assume that the hydrogen flow is laminar.

Figure 3: grid independence study: view of the three computational mesh levels near the channel's inlet; coarse (left), medium (center), fine (right).

3.6 Boundary Conditions

We assume that the hydrogen flow inside the channel is incompressible, and its thermodynamic properties are constant. The molecular weight of hydrogen is 2.016 kg/kmol. The universal gas constant R is 8.3143 kJ/kmol.K. From these values one can compute the hydrogen specific gas constant, as:

$$R_{H_2} = \frac{R}{MW_{H_2}} = 4.124156746 \quad \frac{kJ}{K.kg} \quad \equiv \quad 4124.156746 \quad \frac{J}{K.kg}$$

The hydrogen dynamic viscosity value is $\mu = 8.4\text{x}10^{-6}\ \frac{kg}{m.s}$. Let's assume that the hydrogen static pressure and temperature are 101325 Pa, and 300 K, respectively. From the equation of state for ideal gas:

$$\rho = \frac{p}{R_{H_2} \cdot T} = 0.081895529 \quad \frac{kg}{m^3}$$

From the definition of the Reynolds number:

$$Re_H = \frac{\rho U_{avg} H}{\mu}$$

With $Re_H\ =\ 40$, the inlet average velocity U_{avg} can be computed as:

$$U_{avg} = \frac{\mu Re_H}{\rho H} = 0.102569702 \quad \frac{m}{s}$$

Table 1 summarizes the boundary conditions used throughout the simulation:

Table 1: summary of the boundary conditions used in the Poiseuille flow simulation

Boundary side	Boundary type	Variable	Value
Inlet	Fixed value	Velocity	0.102569702 m/s
Outlet	Fixed value	Pressure[3]	0
Walls	No-slip	None	None

3.7 Numerical settings

Numerical settings are given in APPENDIX II and APPENDIX I.

3.8 Results

Figure 5 shows the convergence history of the residuals for the three mesh levels. Figure 5 and Figure 6 present a comparison between the computed maximum velocity (at the channel centerline) and the

[3] In *simpleFoam*, the pressure used by the solver is actually $\frac{p}{\rho}$, and does not have a direct physical meaning, as it is not treated as a true variable in the incompressible, isothermal formulation. Only its gradient appears in the momentum equation, which is the quantity that influences the flow.

velocity profile at x=0.75 m along the channel, and their corresponding analytical solutions. Table 2 compares the computed maximum centerline velocity values with the analytical solution given by the Poiseuille equation. Table 3 reports the computed entrance length on the three meshes, and compares it with the prediction given by the empirical relation (Equation 9). The results demonstrate convergence toward the analytical solution as the computational mesh is refined.

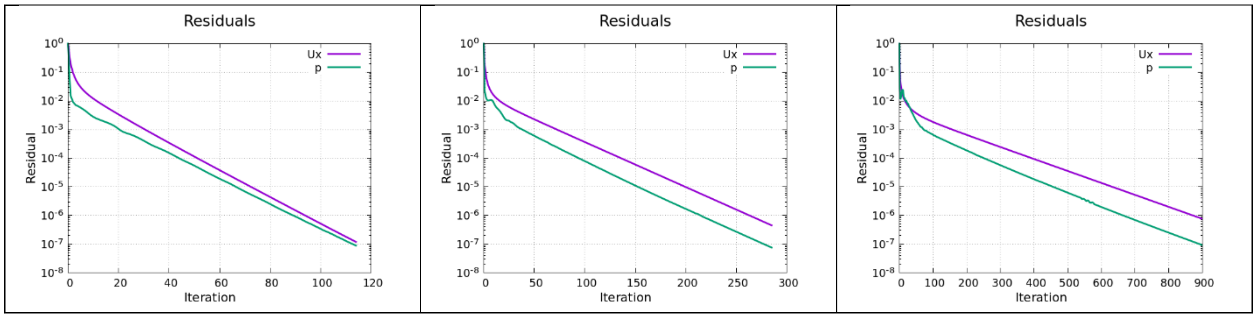


Figure 4: grid independence study: numerical residuals of streamline velocity and pressure; coarse grid (left), medium (center), fine (right)

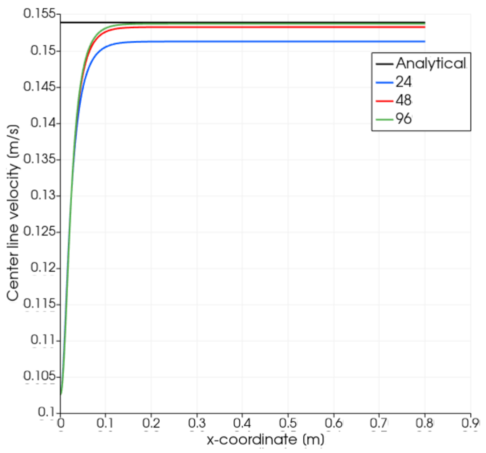


Figure 5: grid independence study: comparison of the computed centerline velocity with the analytical value given by the Poiseuille equation

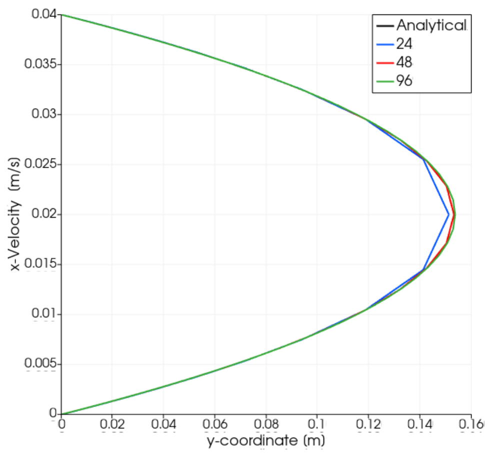


Figure 6: grid independence study: velocity profile at x = 0.75 m and comparison with the analytical solution of the Poiseuille equation

Table 2: grid independency study: comparison of the computed maximum centerline velocity values with the analytical solution given by the Poiseuille equation

Analytical	Coarse	Medium	Fine
0.153855	0.151238	0.153226	0.153685
Error [%]	-1.7010	-0.4088	-0.1105

Table 3: grid independency study: comparison of the predicted entrance length[4] with computed results; values are in cm.

Predicted[5]	Coarse	Medium	Fine
8	8.29	8.04	8.0

3.9 Order of Accuracy and Richardson Extrapolation

The Richardson Extrapolation (RE) is a numerical technique used to estimate the exact solution of a discretized equation by combining results from simulations on systematically refined grids. It is based on the idea that the discretization error (E) scales with the grid spacing (h) as:

$$[E = f_h - f_{\text{exact}} = g_1 h^p + g_2 h^{p+q} + \cdots]$$

where (f_h) is the solution on grid spacing (h), (f_{exact}) is the exact solution (or a reference solution), (p) is the observed order of convergence, and (g_i) are constants. RE estimates (f_{exact}) by eliminating leading error terms using solutions from multiple grid resolutions.

[4] Computed from results as the length over which velocity is 99% of U$_{\text{max}}$.
[5] Equation 9.

The technique has many useful applications, as it helps in discretization error quantification (e.g. provides an error band for solutions), in verification studies (e.g. validates the code order of convergence), in assessing grid Independence (e.g. identifies resolutions where further refinement yields negligible gains), and as an aid in solution improvement (e.g. the extrapolated results serve as high-fidelity benchmarks for coarse-grid simulations).

The following two sections explain how to apply the technique in the case one have results obtained on three (3.9.1) or more than three (3.9.2) grid levels.

3.9.1 RE with three grid levels

Let's start from the three grid levels case, and consider three grids with spacings $(h_1 > h_2 > h_3)$ (e.g., coarse, medium, fine) and their respective solutions $(f_1), (f_2), (f_3)$. Define the refinement ratio $(r = h_1/h_2 = h_2/h_3)$ (typically $(r = 2)$[6]).

To proceed we need to compute the observed order of convergence (p):

Equation 10

$$p = \frac{\ln\left(\frac{f_2 - f_3}{f_1 - f_2}\right)}{\ln(r)}$$

Equation 10 assumes that errors reduce consistently across grids. As a side result, computing p allows to verify the actual order of accuracy of the numerical scheme used to compute results.

Once p is known, one can estimate the exact solution (f_{exact}), extrapolating the fine-grid solution by adding an error correction term, as:

Equation 11

$$f_{\text{exact}} \approx f_3 + \frac{f_3 - f_2}{r^p - 1}$$

With the values of Table 2, Equation 10 and Equation 11 provides $p = 2.115$ (which confirm the expected 2nd order accuracy of the used numerical scheme), and $f_{\text{exact}} = 0.153812$

3.9.2 RE with more than three grid levels

For $(n > 3)$ grids $(h_1 > h_2 > \cdots > h_n)$, the Richardson extrapolation uses higher-order error expansions or regression to minimize uncertainty. To this aim, we can start from modelling the error expansion:

Equation 12

$$f_h = f_{\text{exact}} + g_1 h^p + g_2 h^{p+q} + \cdots$$

where (q) is the order increment (e.g., $(q = 1)$ for structured grids).

Then we solve for the parameters in Equation 12 using one of the following methods:

[6] This is true for the grid used in our simulation along the y-direction, where the spacing has been halved when moving from one grid level to the next one. In the x-direction that ratio is one.

- Nonlinear regression: Fit (f_{exact}), (p), (g_i) to all data points.
- Least-squares minimization: Optimize parameters to match solutions across grids.
- Generalized RE: Extend the three-grid formula iteratively for higher accuracy.

4 Porosity model in OpenFoam

The classical Darcy law for slow, viscous flow in a homogeneous porous medium:

Equation 13

$$q = -\frac{K}{\mu}\nabla p$$

where:

- q is the Darcy flux (volumetric flow per unit area)
- K is the permeability $[\mathrm{m}^2]$
- μ is the dynamic viscosity [Pa·s]
- ∇p is the pressure gradient

is implemented in OpenFOAM in a more general form, the Darcy + Forchheimer model[7], as a sink term in the momentum equation[8] as:

Equation 14

$$\rho\left(\frac{\partial U}{\partial t} + U\cdot\nabla U\right) = -\nabla p + \mu\nabla^2\boldsymbol{U} - \underbrace{\left(\mu\,\alpha\,\boldsymbol{U} + \frac{1}{2}\rho\,\beta\,|\boldsymbol{U}|\,\boldsymbol{U}\right)}_{\text{Darcy + Forchheimer drag}}$$

Where:

- $\alpha = d = \frac{1}{k}$ is the **Darcy coefficient**
- β is the **Forchheimer (inertial) coefficient**

So that Equation 14 can be rewritten as:

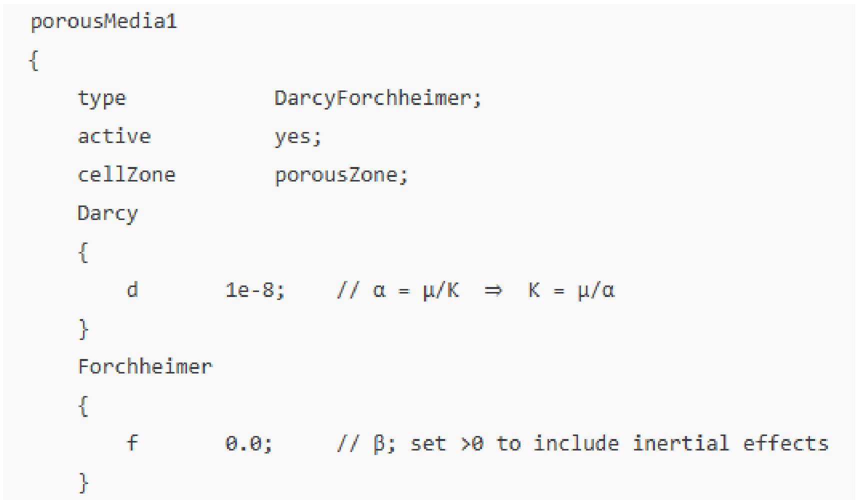
Equation 15

$$\rho\left(\frac{\partial U}{\partial t} + U\cdot\nabla U\right) = -\nabla p + \mu\nabla^2\boldsymbol{U} - \underbrace{\left(\mu\cdot d\cdot\boldsymbol{U} + \frac{1}{2}\rho\,\beta\,|\boldsymbol{U}|\,\boldsymbol{U}\right)}_{\text{Darcy + Forchheimer drag}}$$

[7]https://www.openfoam.com/documentation/guides/v2112/api/classFoam_1_1porosityModels_1_1DarcyForchheimer.html
[8]In incompressible form

Note that d is in general a (1x3) vector, as the porous material can be anisotropic and d assume different values along the three coordinated directions. In the present case, as in (Rabienataj Darzi AA 2016), the porous material is assumed isotropic, and d is a constant scalar.

The model is activated through a dedicated entry in the constant/fvOptions file:



```
porousMedia1
{
    type            DarcyForchheimer;
    active          yes;
    cellZone        porousZone;
    Darcy
    {
        d       1e-8;    // α = μ/K  ⇒  K = μ/α
    }
    Forchheimer
    {
        f       0.0;     // β; set >0 to include inertial effects
    }
```

Figure 7: example of porosity model function in constant/fvOptions file

4.1 simpleFoam Solver

To test the porous model available in OpenFOAM, a bounding box is placed inside the channel used for the Poiseuille test case. The bounding box spans the entire height of the channel and extends from *x* = 0.5 m to *x* = 0.7 m, which is well beyond the entrance length, so that the flow velocity profile inside the channel can be assumed constant.

The porous medium permeability is chosen equal to the value used in (Rabienataj Darzi AA 2016): $K = 10^{-9}\ m^2$, so that, as $d\ =\ 1/K$, we have: d $= \frac{1}{10^{-9}} = 10^9\ \left(\frac{1}{m^2}\right)$. This is the parameter used in OpenFoam porous model (Equation 15). Because, given the nature of the flow, non linear effects can be neglected, the value of β in Equation 15 is set to zero.

As the flow is incompressible (constant density), and the channel's cross section area is uniform, we can assume that the average value of the velocity profile at x = 0.5 m (where the porous media starts) is equal to $U_{avg} = 0.102569702\ \ \left(\frac{m}{s}\right)$, which is the boundary condition value used at the channel's inlet.

The expected pressure drop through the porous media is then, from Equation 13:

$$-\nabla p = \frac{\Delta p}{L} = \frac{\mu}{K} \cdot U_{avg} \quad \Longrightarrow \quad \Delta p\ =\ \mu \cdot d \cdot U_{avg}\, L$$

Where L is the porous media length. With our values, we get:

$$-\Delta p = \mu \cdot d \cdot U_{avg} \cdot L = 10^9 \cdot 8.4 \cdot 10^{-6} \cdot 0.102569702 \cdot 0.2 = 172.317\quad Pa$$

4.1.1 Results

The test case has revealed a shortcoming of the *simpleFoam* solver: the code does not explicitly use the fluid density, nor does it provide a way for the user to specify its value at runtime. Instead, the fluid's transport properties are defined in terms of kinematic viscosity ($\nu = \frac{\mu}{\rho}$), rather than dynamic viscosity μ. While this is generally not an issue (as the results presented in Section 3.8 demonstrate), it becomes problematic when using a model that requires explicit use of dynamic viscosity, such as the Darcy model.

In a first round of simulations, the kinetic viscosity is assigned in the transport properties file, as it is required by simpleFoam. Results are presented here below.

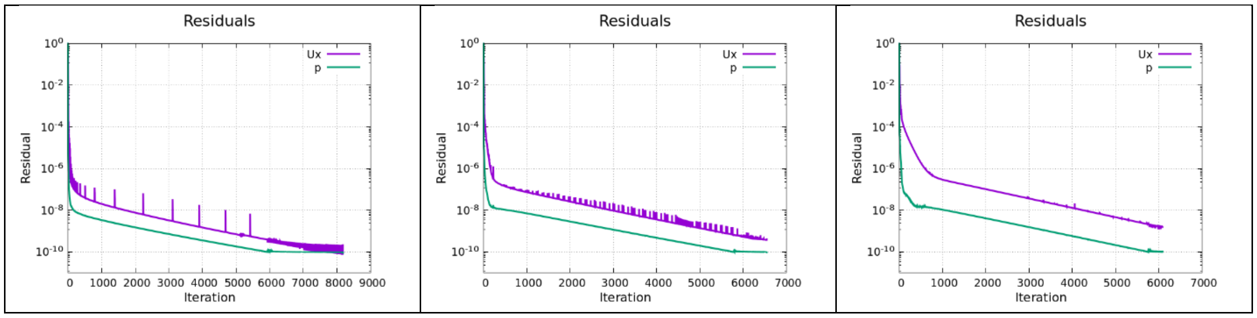


Figure 8: grid independence study: kinematic viscosity is assigned in the transport properties file; numerical residuals of streamline velocity and pressure; coarse grid (left), medium (center), fine (right)

Table 4: grid independence study: kinematic viscosity is assigned in the transport properties file; comparison of computed pressure drops through the porous medium with the analytical value of the Darcy law

Analytic	Coarse	Medium	Fine
172.317	1987.490	1988.414	1989.805

Table 4 shows that the computed pressure drop is overestimated by an order of magnitude. In a second round of simulations, we used the dynamic viscosity instead of the kinematic viscosity in the transport properties file, aiming to force the Darcy model to use the correct value. The results, in terms of residual convergence and pressure drop, are presented in Figure 9 and Table 5 below. The computed pressure drop underestimates the theoretical value by approximately 5.4%. We also note that due to the smaller viscosity value, the solution on the fine mesh is no longer capable to reach full numerical convergence.

Residual stagnates
earlier of fine meshes.
Not sure whether smaller relaxation
factors are beneficial to have here.

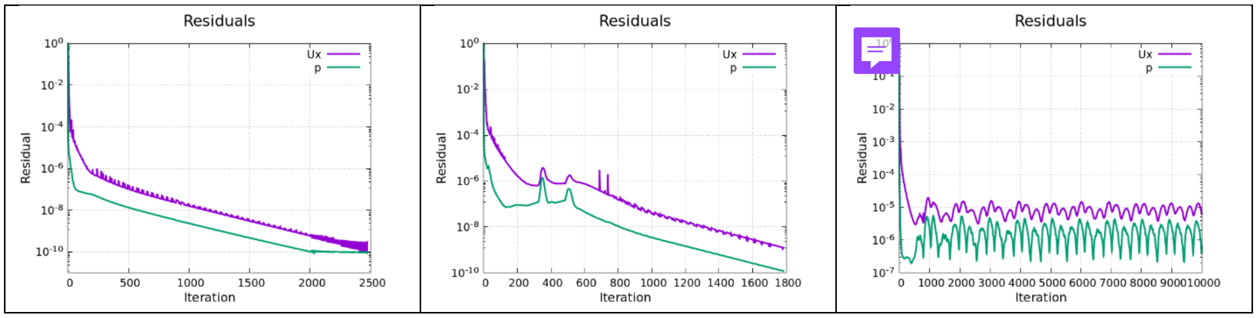


Figure 9: grid independence study: dynamic viscosity is assigned in the transport properties file; numerical residuals of streamline velocity and pressure; coarse grid (left), medium (center), fine (right)

Table 5: grid independence study: dynamic viscosity is assigned in the transport properties file; comparison of computed pressure drops through the porous medium with the analytical value of the Darcy law

Analytic	Coarse	Medium	Fine
172.317	162.773	162.850	162.966
Error [%]	-5.539	-5.494	-5.427

As can be seen, specifying the dynamic viscosity instead of the kinematic viscosity causes the Darcy model to yield the correct pressure drop through the porous zone. We now aim to verify how this affects the velocity profile. To this end, we repeat the simulations using both viscosity values, with the porosity model switched off. This experiment is performed only on the fine grid, and the results are compared with the theoretical solution given by Poiseuille's equation.

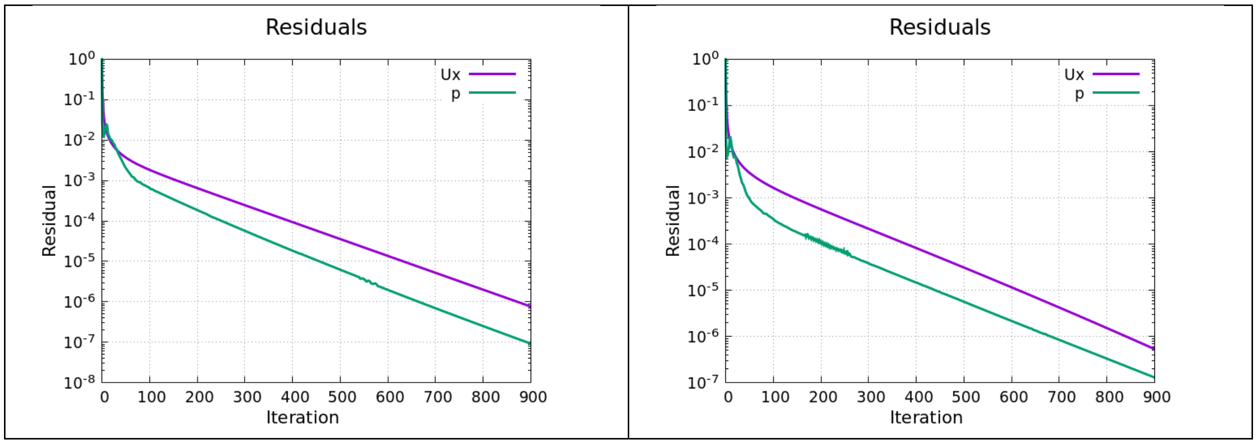


Figure 10: convergence of numerical residuals; kinematic viscosity (left), dynamic viscosity (tight)

Figure 10 shows similar residual convergence for both viscosity values. Figure 11, however, clearly illustrates the effect of using dynamic viscosity instead of its kinematic counterpart in the transport properties file. While the red line (computed with the assigned ν) and the black line overlap, the blue line exhibits a velocity deficit. This is not surprising when considering that the value of U_{avg} used as the inlet boundary condition was computed based on the definition of the Reynolds number. Using a different value for the viscosity brings to a different flow behavior. This finding highlights a critical issue that must be considered when using *simpleFoam* to simulate flows in porous media.

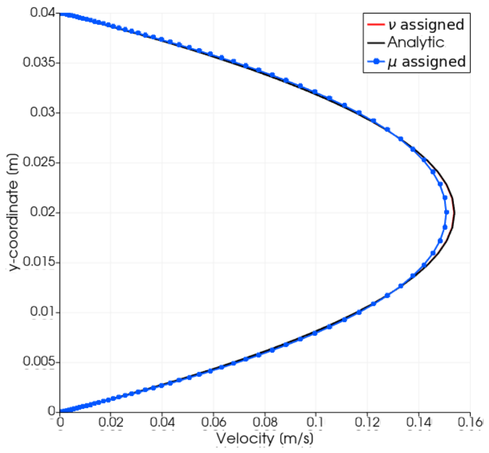


Figure 11: comparison of computed velocity profiles at x = 0.7 m with the analytical solution of the Poiseuille equation; viscosity model is set OFF

4.2 rhoSimpleFoam Solver

To overcome this limitation, we switched to *rhoSimpleFoam*, a solver that uses the ideal gas equation of state, thereby accounting for the actual fluid density. Accordingly, it requires the user to specify the dynamic viscosity in the *thermoPhysicalProperties* file (the equivalent of the *transportProperties* file used in *simpleFoam*). The *thermoPhysicalProperties*, *fvSchemes*, and *fvSolution* files are provided in APPENDIX IV, APPENDIX V, and APPENDIX VI, respectively.

In an initial round of simulations, the solver was tested against Poiseuille flow, and the results were compared with the analytical velocity profile (Equation 8). The boundary conditions were kept the same as in the previous cases, except for the pressure, which was now set to 101325 Pa instead of 0. This change was necessary because pressure now has physical meaning, unlike in the incompressible solver, where it serves merely as a reference value due to the assumption of constant density.

4.2.1 Results

Figure 12 shows the convergence history of the numerical residuals. Table 6 compares the maximum centerline velocity with the analytical solution. The agreement is slightly better than that obtained previously with simpleFoam.

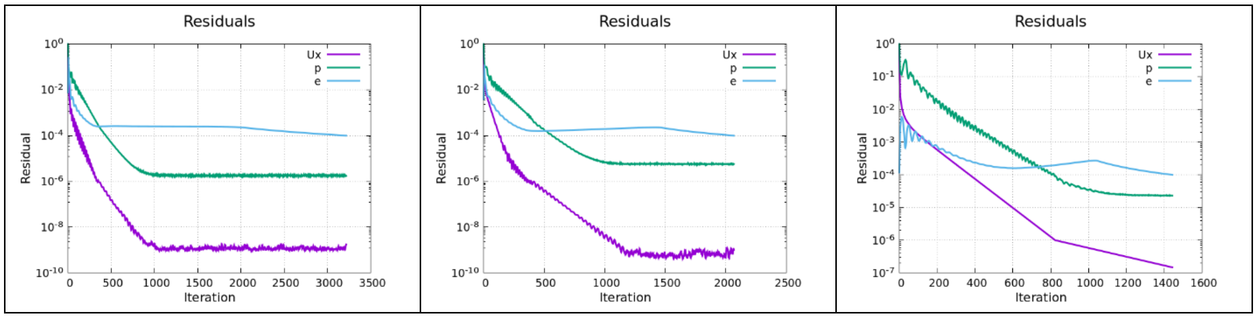


Figure 12: grid independence study: rhoSimpleFoam, numerical residuals of streamline velocity, pressure, and internal energy; coarse grid (left), medium (center), fine (right)

Table 6: grid independency study: rhoSimpleFoam, comparison of the computed maximum centerline velocity values with the analytical solution given by the Poiseuille equation

Analytical	Coarse	Medium	Fine
0.153855	0.151238	0.153229	0.153700
Error [%]	-1.7010	-0.4069	-0.1007

Simulations have been then repeated with the porous model ON. Figure 13 shows the convergence history of the numerical residuals. Table 7 compares the analytical pressure drop value with those computed on the three mesh levels.

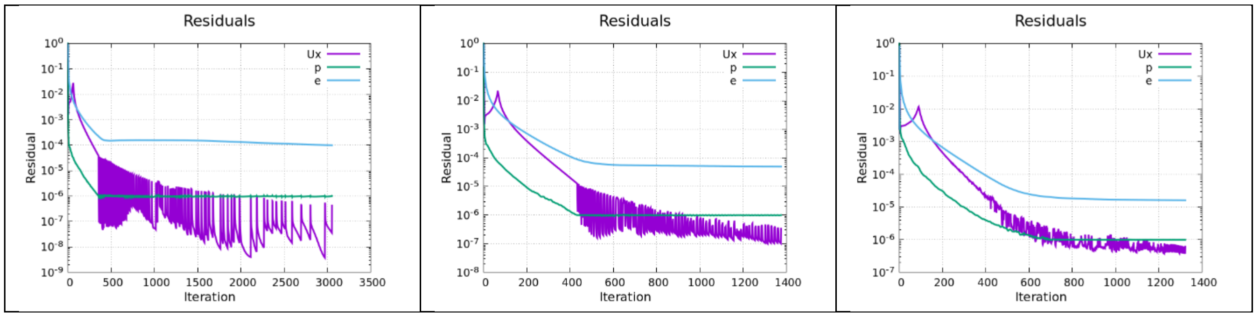


Figure 13: grid independence study: rhoSimpleFoam, Darcy model, numerical residuals of streamline velocity, pressure, and internal energy; coarse grid (left), medium (center), fine (right)

Table 7: grid independence study: rhoSimpleFoam, Darcy model*; comparison of computed pressure drops through the porous medium with the analytical value of the Darcy law*

Analytic	Coarse	Medium	Fine
172.317	162.881	162.967	163.081
Error [%]	-5.476	-5.426	-5.360

4.3 Takeaways

It has been found that the *simpleFoam* solver, due to the fact that it does not account for the actual fluid density and relies on kinematic viscosity, is not well suited for problems involving porous media modeled using the Darcy model implemented in OpenFOAM. This limitation arises because the Darcy model requires the dynamic viscosity, which cannot be derived from the kinematic viscosity specified in the *transportProperties* file. Switching to *rhoSimpleFoam*, a solver that accounts for both the fluid's density and dynamic viscosity, has enabled the recovery of the desired accuracy, both in the computed velocity field for Poiseuille flow and in the pressure drop for Darcy-type flow simulations.

5 Tank filling simulation – empty tank

In the actual hydrogen storage device presented by (Rabienataj Darzi AA 2016), the tank being filled with hydrogen contains a metallic matrix in which absorption and desorption processes occur. The ultimate goal of the present research project is to develop a mathematical tool capable of simulating these processes. This section represents a further step in that direction.

The tank filling process is analyzed here under two scenarios:

i) the tank from (Rabienataj Darzi AA 2016) is assumed to be empty, and

ii) it is filled with a non-reactive metallic matrix possessing the same porosity characteristics.

Unlike the real case, where, during absorption, hydrogen binds to the metallic matrix as it is introduced into the tank, here, as soon as the gas enters the tank at a given total pressure and temperature, the tank pressure begins to rise until it reaches the value of the total pressure at which hydrogen is injected. At that point, recalling for sake of simplicity, the relation valid for incompressible flow:

$$p_t = p_s + \frac{1}{2}\rho U^2$$

$$p_t = p_s \Longrightarrow \frac{1}{2}\rho U^2 = 0$$

the hydrogen flow into the tank effectively drops to zero.

Building on the experience gained in the previous sections, the code used for the simulations presented here is the unsteady counterpart of **rhoSimpleFoam**, namely **rhoPimpleFoam**. Results of simulation i) will be compared with the 0-D model.

We aim to maintain the hydrogen flow within the laminar regime throughout the present numerical tests. Reynolds (Reynolds 1883), and Nikuradze (Nikuradze, Gesetzmäßigkeiten der turbulenten Strömung in glatten Rohren (Laws of Turbulent Flow in Smooth Pipes) 1932), (Nikuradze, Strömungsgesetze in rauhen Rohren (Laws of Flow in Rough Pipes) 1933), (K. Holz 1953), experimentally established the transitional Reynolds number in smooth pipes to be approximately 2300. Therefore, we select a Reynolds number of 2000 to ensure the flow remains within the laminar regime. From the Reynolds number definition, we can compute the value of the hydrogen velocity at the feeding pipe inlet section:

$$\mathrm{Re} = \frac{\rho \mathrm{U} \mathrm{D_i}}{\mu} \Longrightarrow U = \frac{Re\,\mu}{\rho D_i}$$

Which, with the chosen value of Re and the physical properties of H_2, yields U = 10.2566 m/s.

We can then compute the corresponding value of the Mach number, M = 0.007793, and, recalling that in the more general case of an isentropic, adiabatic, compressible flow is:

$$\frac{p_t}{p_s} = \left(1 + \frac{\gamma - 1}{\gamma} M^2\right)^{\frac{\gamma}{\gamma - 1}}$$

$$\frac{T_t}{T_s} = \left(1 + \frac{\gamma - 1}{\gamma} M^2\right)$$

Where γ is the specific heat ratio.

It results:

p_t = 101329.307829 Pa

Tt = 300.003644K

These are the values that are used as total conditions at the inlet boundary.

To reduce computational cost, the axial symmetry of the flow is exploited by modeling only a 5-degree wedge of the actual device, as shown in Figure 14. Only a single cell is used in the tangential direction. Figure 15 provides an overall view of the computational mesh in the x–y plane, while Figure 16 shows a close-up of the inlet region.

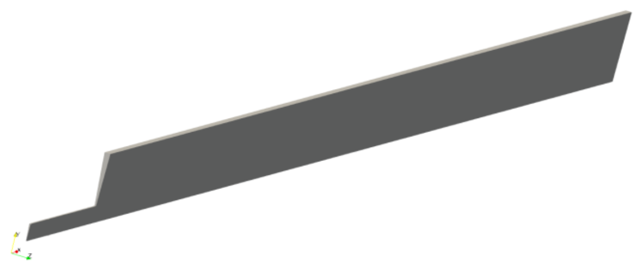


Figure 14: The five-degree wedge of the device used in the axis symmetric simulations

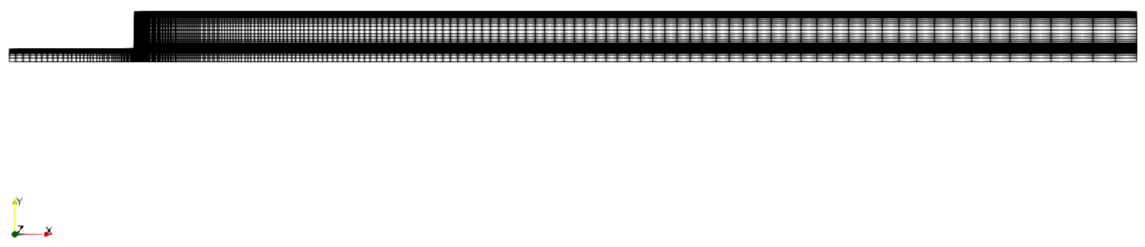


Figure 15: the overall view of the computational medium mesh on the x-y section plane.

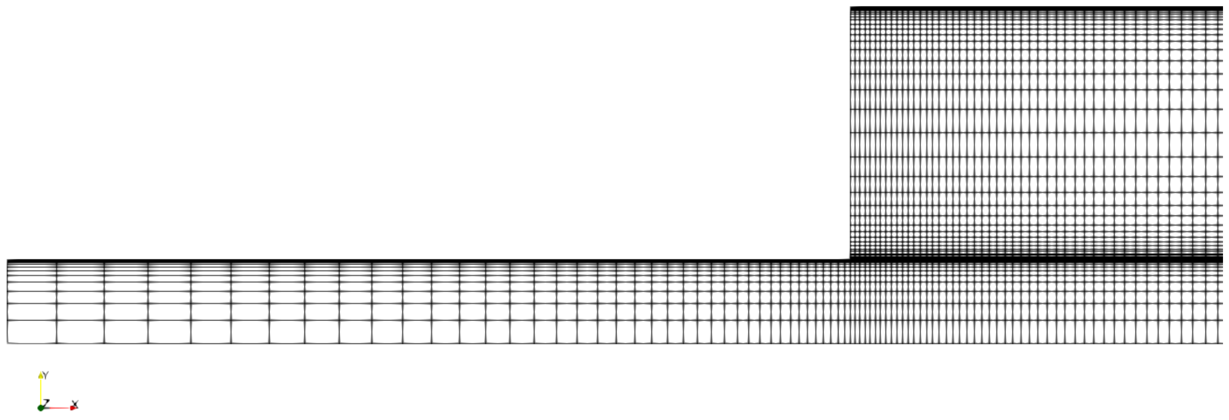
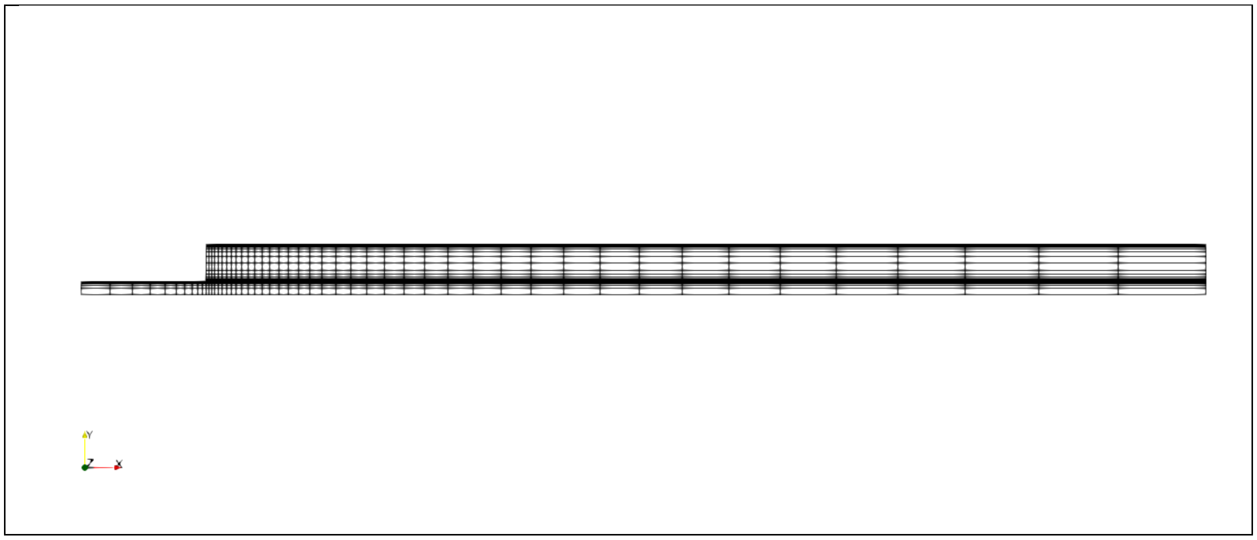


Figure 16: an enlarged view of the computational medium mesh near the inlet region.

The blockMeshDict file used to generate the mesh is provided in APPENDIX VII. We have conducted a mesh sensitivity analysis on three grids, obtained by recursively doubling the number of cells in the x and y directions of the previous refinement level. The computational domain is divided in three blocks, covering the feeding pipe, the tank's region that extends from the axis to the feeding pipe's radius, and the remaining tank's volume, respectively. Table 8 summarizes each block's discretization used in the axial and radial directions, and the respective total number of computational cells. Only one cell is used in the circumferential direction.

Table 8: Computational mesh size

Block	Coarse	Medium	Fine
1	10 x 6	20 x 12	40 x 24
2	40 x 6	80 x 12	160 x 24
3	40 x 18	80 x 36	160 x 72
Total number of cells	1020	4080	16320



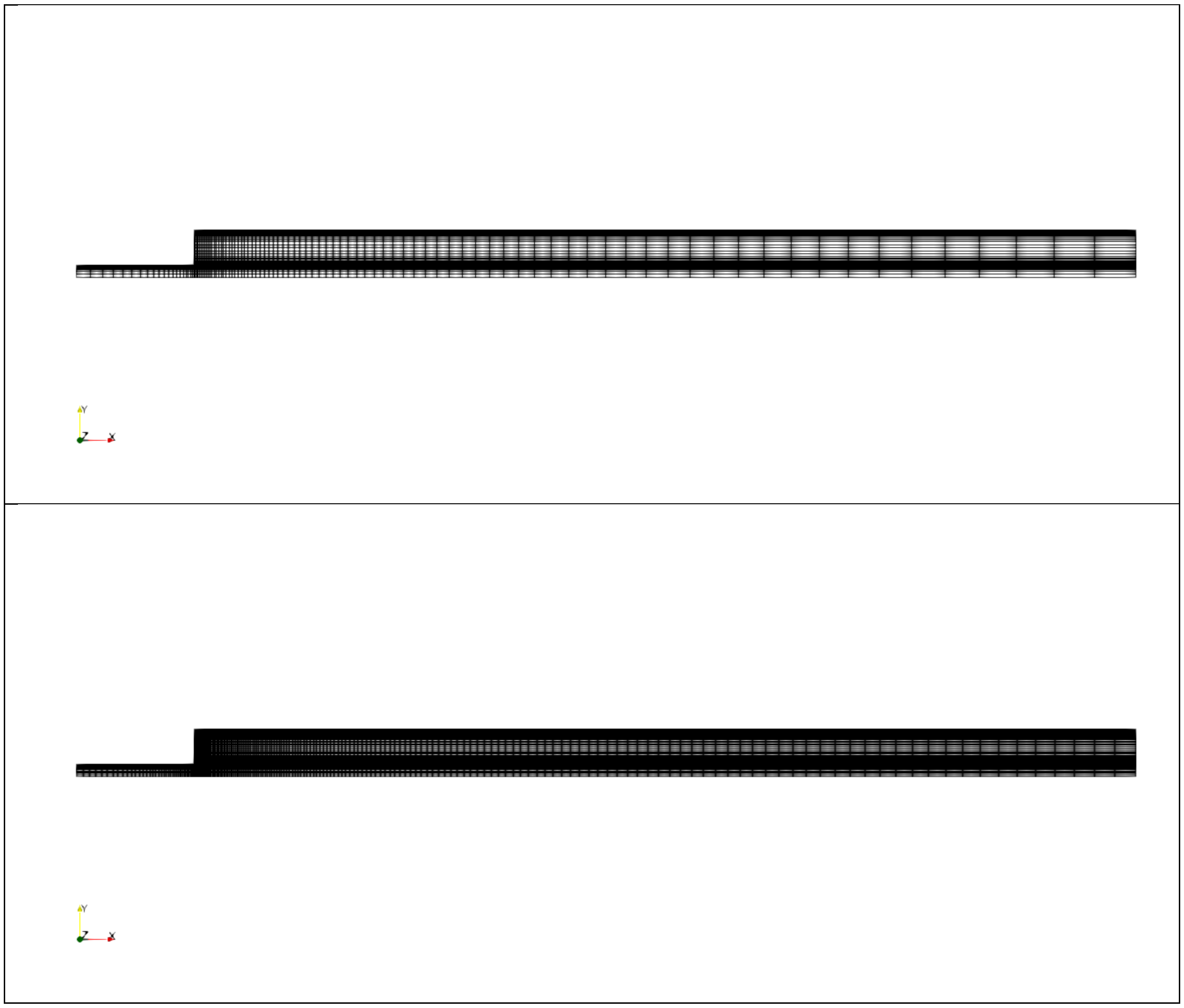


Figure 17: Wedge (5 degrees) computational mesh; from top to bottom: coarse, medium, and fine grid

5.1 Zero-D model's results

With a tentative value of $\alpha_{fac} = 1$, the zero-dimensional model predicts a filling time of 0.003054 s. This is the filling time that will be used in the next sections to compare results. Figure 18 shows the graphical output of the model, while Figure 19 presents the Octave print-out. Note how the zero-D model correctly predicts the final pressure value inside the tank to be approximately equal to the inlet total pressure value.

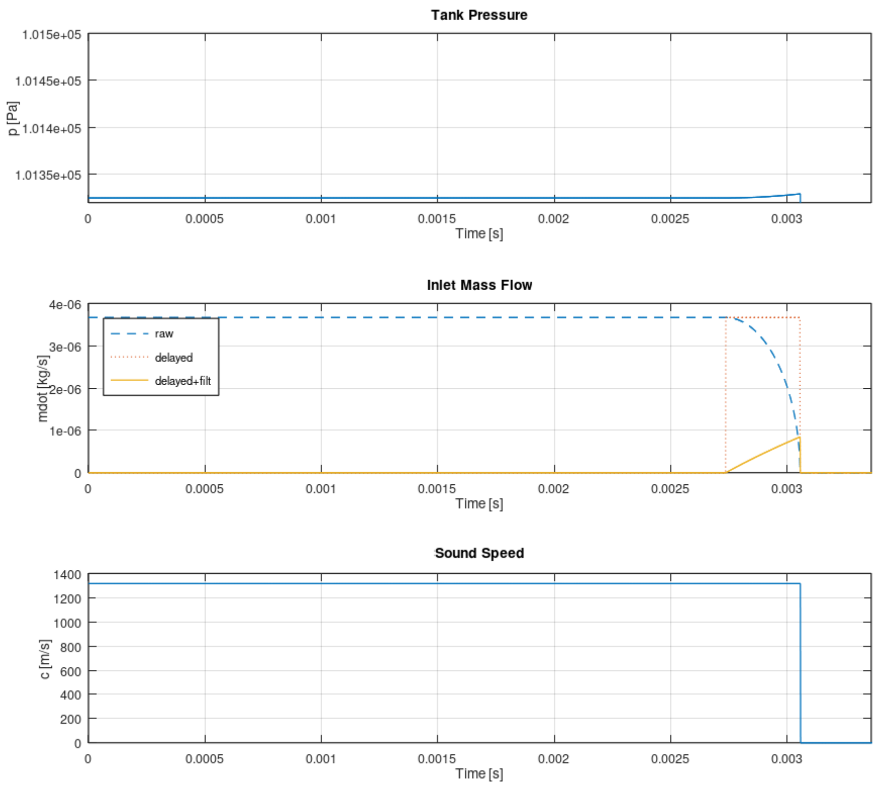
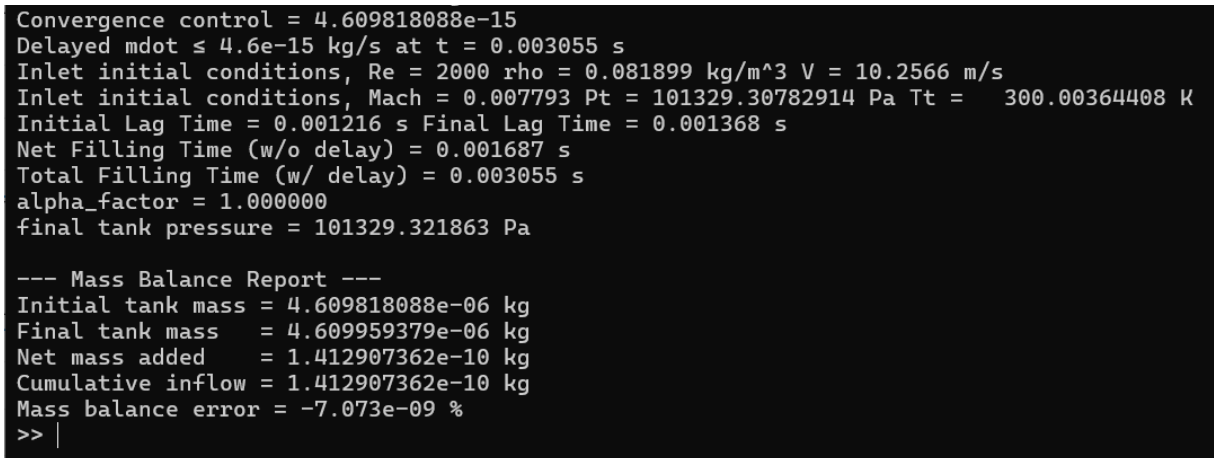


Figure 18: zero-dimensional model simulation of the empty tank filling process, w/o porous media



```
Convergence control = 4.609818088e-15
Delayed mdot ≤ 4.6e-15 kg/s at t = 0.003055 s
Inlet initial conditions, Re = 2000 rho = 0.081899 kg/m^3 V = 10.2566 m/s
Inlet initial conditions, Mach = 0.007793 Pt = 101329.30782914 Pa Tt =   300.00364408 K
Initial Lag Time = 0.001216 s Final Lag Time = 0.001368 s
Net Filling Time (w/o delay) = 0.001687 s
Total Filling Time (w/ delay) = 0.003055 s
alpha_factor = 1.000000
final tank pressure = 101329.321863 Pa

--- Mass Balance Report ---
Initial tank mass = 4.609818088e-06 kg
Final tank mass   = 4.609959379e-06 kg
Net mass added    = 1.412907362e-10 kg
Cumulative inflow = 1.412907362e-10 kg
Mass balance error = -7.073e-09 %
>> |
```

Figure 19: results of the 0-D model in Octave, w/o porous media

5.2 rhoPimpleFoam simulation, adiabatic vs isothermal walls

In an initial round of simulations, a grid-independence study is performed to assess the influence of:

i) spatial discretization, and

ii) the temperature boundary condition at the walls.

The motivation for point (ii) is that, as hydrogen is pumped into the tank, its temperature increases. Two aspects are of interest:

- To quantify the resulting internal temperature difference when the tank is either thermally insulated (adiabatic) or has walls maintained at a constant temperature.
- To evaluate the impact of this thermal effect on the overall filling time.

5.2.1 Tank filling time and internal temperature – Isothermal vs Adiabatic walls' boundary conditions

As soon as the simulation starts, a pressure wave moves from the inlet section into the tank and starts reflecting back and forth, damped by the fluid's viscosity (physical damping) and by the "numerical" viscosity due to the grid and the numerical scheme's diffusivity. Figure 20 shows the time evolution on 0.1 s of the hydrogen mass flow rate through the inlet and the average pressure inside the tank. The time interval of 0.1 s is approximately one hundred time the length of the acoustic time $\tau_0 = \frac{2L}{c}$, where c is the speed of sound, thus the time needed to the pressure wave, travelling at the speed of sound, to travel back and forth from the inlet to the closed end of the tank.

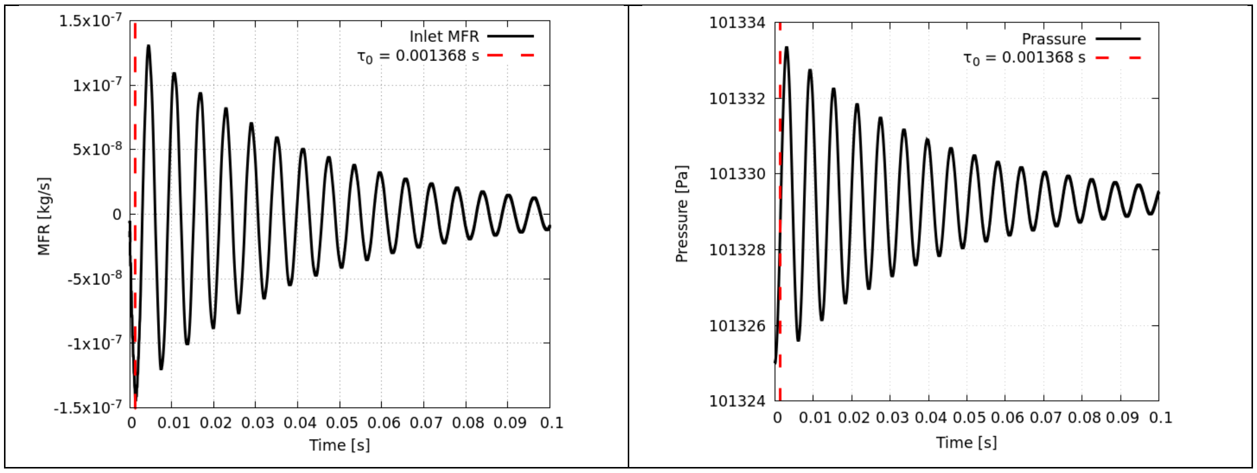


Figure 20: inlet mass flow rate (left) and pressure evolution with time (right) when the simulation is run over approximately 100 times the acoustic time τ_0

We have thus the need to define what it should be intended for the tank's filling time. A possible definition could be the time at which the inlet mass flow rate goes to zero, but it is easy to see from Figure 20 that this happens many times. Another one would be the time at which the inlet mass flow rate goes to and remain zero, but in reality, it would probably be unacceptable to have a backflow of hydrogen from the tank. We have arbitrarily decided to define the filling time as the first-time occurrence after the acoustic time, at which the time derivative of the total mass contained inside the tank vanishes.

OpenFOAM does not include a built-in feature to automatically record results and stop the simulation when a specific condition is met. However, it does offer a time control option (writeNow) that forces the solver to write the solution and terminate.

To automate this process, we developed a customized function in system/controlDict that changes the stopAt setting in system/controlDict to writeNow, when the derivative with respect to time of the mass contained inside the tank drops below a given tolerance (ideally goes to zero, so no new mass enters the tank), prompting the solver to output the final solution and terminate the simulation. The controlDict file is provided in APPENDIX VIII. APPENDIX IX and APPENDIX X provide the fvSchemes and fvSolution files used throughout the simulations.

We aim to explore the influence of the wall thermal treatment (isothermal vs. adiabatic) on the tank filling time.

Figure 21 displays the numerical residuals of the momentum, pressure, and energy equations for both the isothermal and adiabatic cases.

Figure 22 shows the time history of the hydrogen mass contained inside the tank, while Figure 23 shows its time derivative. Figure 24 presents the time evolution of the tank's average pressure.

Table 9 compares the tank filling times obtained from simulations to the result of the zero-dimensional model, taken as reference. Both thermal boundary condition cases closely match the 0-D reference, but the adiabatic case results in a slightly shorter filling time.

The number of iterations is grid independent due to deltaT = maxDeltaT = 1e-6 s, with the CFL condition ON. In some parts of the domain the CFL condition would allow higher deltaT values. Here, however, the aim was to have a constant (spatially independent) time step size.

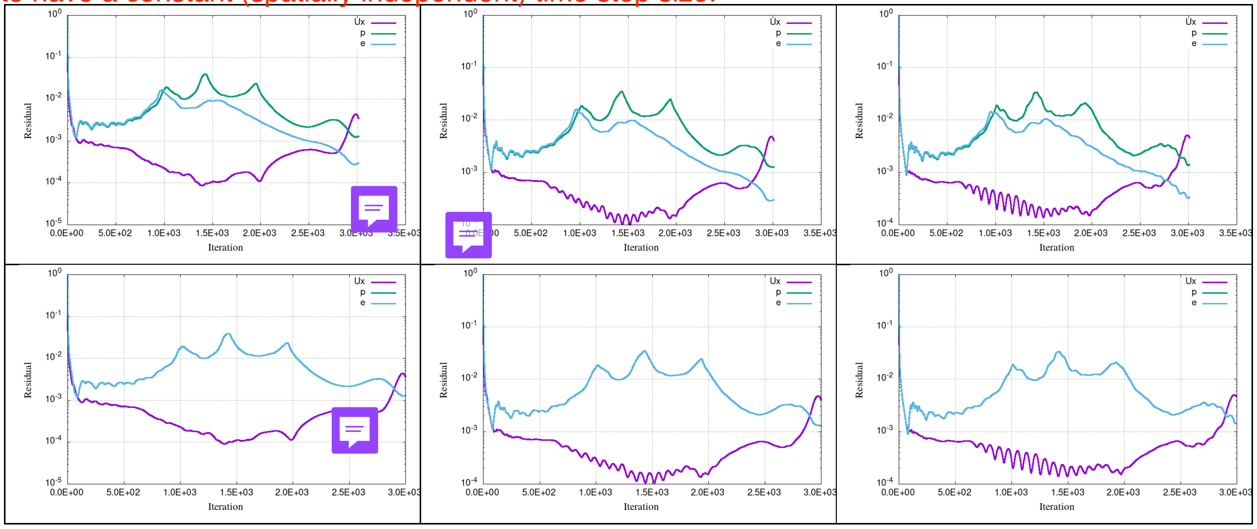
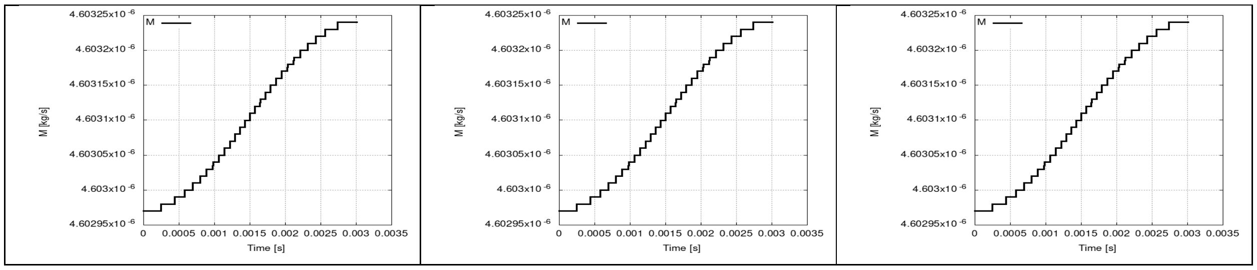


Figure 21: grid independence study: tank filling, axial symmetric case, isothermal (top) vs adiabatic (bottom) walls: numerical residuals; from left to right: coarse, medium, and fine grid level

residual(p) and residual(Ux) nearly coincide.



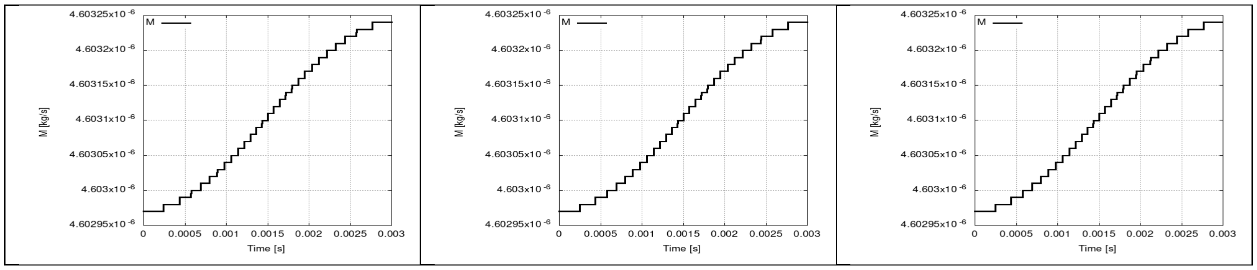


Figure 22: grid independence study: tank filling, axial symmetric case, isothermal (top) vs adiabatic (bottom) walls: time evolution of the mass inside the tank; from left to right: coarse, medium, and fine grid level

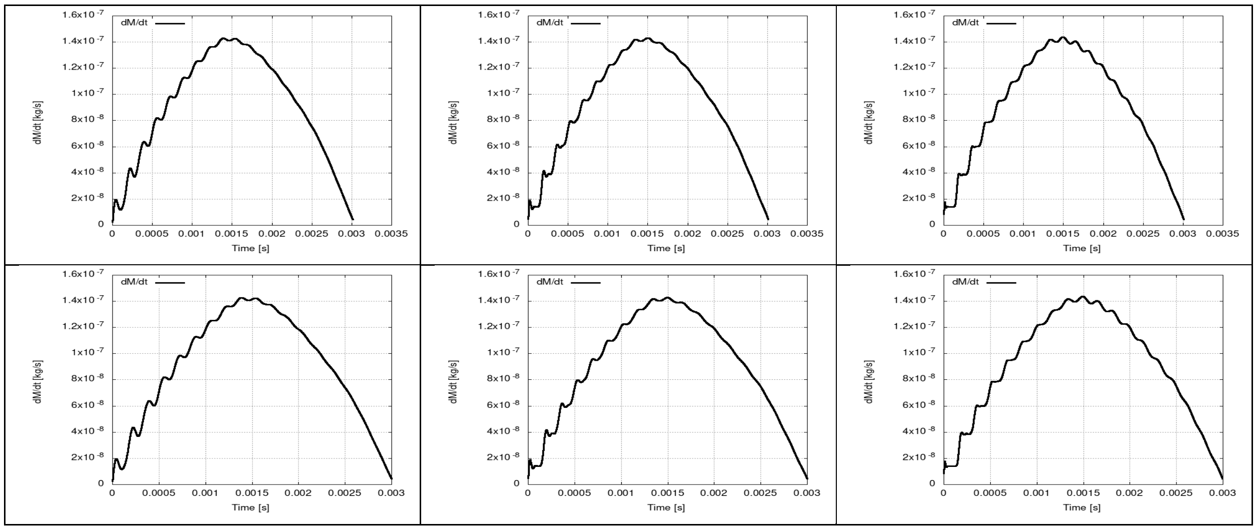


Figure 23: grid independence study: tank filling, axial symmetric case, isothermal (top) vs adiabatic (bottom) walls: time evolution of the derivative of the mass inside the tank; from left to right: coarse, medium, and fine grid level

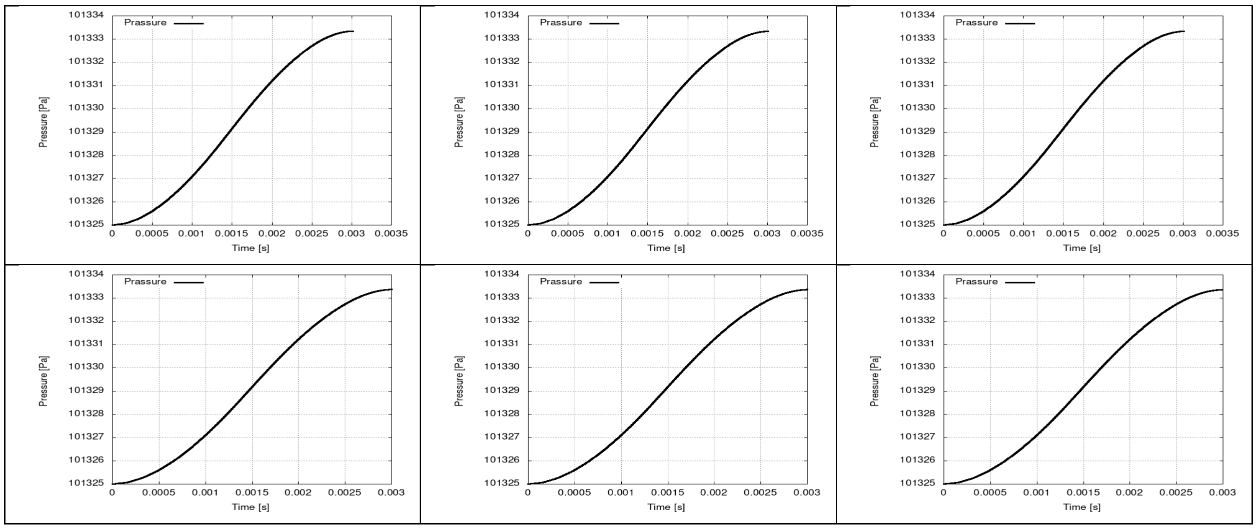


Figure 24: grid independence study: tank filling, axial symmetric case, isothermal (top) vs adiabatic (bottom) walls: time evolution of the tank's average pressure; from left to right: coarse, medium, and fine grid level

Table 9: Tank filling time [s], isothermal vs adiabatic walls, comparison with the zero-dimensional code

	Isothermal			Adiabatic		
Zero-D solver (reference)	Coarse	Medium	Fine	Coarse	Medium	Fine
0.003055	0.003013	0.003012	0.003009	0.002999	0.002999	0.002996
|Error %|	1.374795	1.407528	1.770867	1.833060	1.833060	1.833060

Table 10: grid independence study, isothermal vs adiabatic walls, tank average temperature [K] when the stop condition is reached

Isothermal			Adiabatic		
Coarse	Medium	Fine	Coarse	Medium	Fine
300.0069	300.0069	300.0069	300.0071	300.0071	300.0071

Table 11: grid independence study, isothermal vs adiabatic walls, tank average pressure [Pa] when the stop condition is reached

Isothermal			Adiabatic		
Coarse	Medium	Fine	Coarse	Medium	Fine
101333.344	101333.339	101333.336	101333.372	101333.367	101333.365

Table 9 demonstrates an increasing percentage difference between the CFD results and the reference lumped model with grid refinement. The reason is that finer grids lower numerical dissipation, enabling the CFD simulation to resolve finer-scale flow physics. Since the lumped model represents a highly simplified (inherently dissipative) approximation, the more physically accurate CFD solution naturally becomes less similar to it as the grid resolution improves.

Using the values from Table 9 (isothermal case), the application of Equation 10 and Equation 11 allows us to compute the order of accuracy of the numerical model, as well as the "exact" solution obtained via Richardson extrapolation. The results are: p = -1.585, and f_{exact} = 0.0030136 s. The finding of a negative order of accuracy in general suggests a non-convergent nature of the scheme. We can analyze the error taking as reference the Richardson extrapolation value. Table 12 shows the error sequence for the three grids.

Table 12: empty tank case, error sequence on the three grids

Grid	Value	Absolute Error $\phi - \phi_{\text{exact}}$
Coarse	0.003013	6e-07
Medium	0.003012	16e-07
Fine	0.003009	46e-07

Table 12 shows that the absolute error increases with the grid refinement, showing a divergent behavior. But, is this the true reality? We must remember that the **Richardson extrapolation and order-of-accuracy analysis assumes the solution is in the asymptotic regime**, either steady-state or sufficiently smooth and well-resolved in time and space.

This is not the case here, because:

- The solution is dominated by **transient oscillatory dynamics**, likely reflecting a natural mode of the system (e.g., compressible pressure waves or inertial oscillations).
- Discretization error includes contributions from **time integration**, **spatial resolution**, and **numerical dispersion/dissipation** of the oscillations.
- Therefore, any estimate of accuracy order is **polluted by transient dynamics**.

To determine the actual order of accuracy of the scheme, we have to repeat the analysis an continue the simulation until the oscillations shown in Figure 20 are substantially damped-out.

Figure 25 shows the evolution of the tan's average pressure over the first 0.2 s of simulation. Table 13 summarizes the averaged values of the pressure signal in the last 0.03 s of simulation, as well as its average slope in the same time interval.

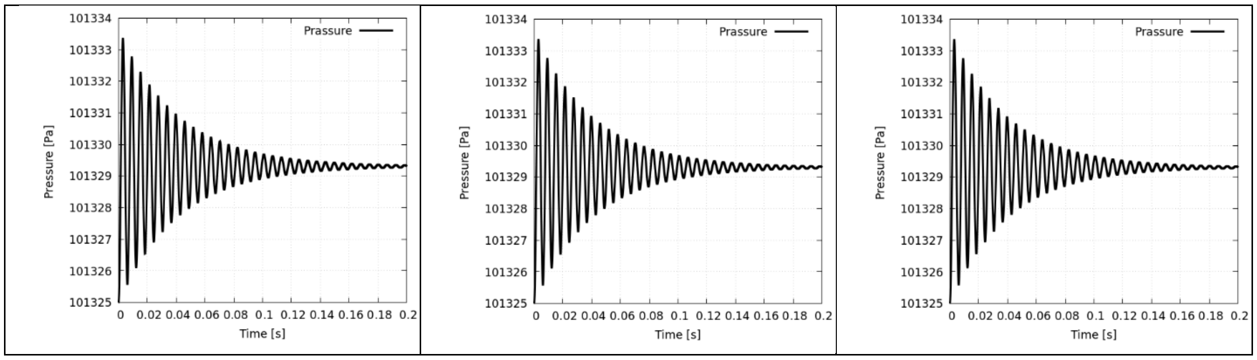


Figure 25: empty tank case, evolution of the tank's average pressure over time until oscillations are damped-out (0.2 s)

Table 13: grid independence study, empty tank, isothermal walls, tank average pressure over the last 0.03 s, and its average slope

Average Pressure [Pa]			Average Slope [Pa/s]		
Coarse	Medium	Fine	Coarse	Medium	Fine
101329.306732	101329.306838	101329.306861	−1.179427	−1.089788	−1.039011

If now we repeat the Richardson extrapolation analysis on the calculated pressure results, we find: p = 2.182 and f_{exact} = 101329.306868 Pa. From which we can compute the error sequence (Table 16), which shows the method's convergence.

Table 14: empty tank case, error sequence on the three grids at 0.2 s

Grid	Value	Absolute Error f−f_{exact}
Coarse	101329.306732	1.36e-04
Medium	101329.306838	0.30e-04
Fine	101329.306861	0.07e-04

5.3 Takeaways

The axisymmetric rhoPimpleFoam simulations of the empty tank agree very closely with the zero-dimensional (0-D) hybrid-lag model, accurately capturing both the rapid pressure rise and the filling time predicted analytically. With an inlet Reynolds number of 2000, grid refinement from coarse to fine, influences the computed filling time by less than 1 %, and the mesh-converged filling time of ∼0.00306 s matches the 0-D prediction within ±2 %

Thermal boundary conditions have a negligible effect under rapid-fill conditions: whether the walls are treated as adiabatic or held at constant temperature, the difference in filling time is below 0.2 % and the peak gas temperature remains essentially at inlet conditions.

Overall, rhoPimpleFoam proves capable of resolving the fast transient gas dynamics and pressure oscillations (damped over a few acoustic “round trips”) in the empty tank, providing a reliable CFD benchmark for validating the much faster 0-D model.

6 Tank filling simulation – porous media filled tank

In the final step of Phase 1 of the study, we present results obtained for the case in which the tank is no longer empty but filled with a porous medium. The permeability used is the same as that reported by (Rabienataj Darzi AA 2016), namely $K = 1 \cdot 10^{-9}\ m^2$. To this end, the cells inside the tank, excluding those within the feeding pipe, were selected using the topoSet tool available in OpenFOAM. The corresponding dictionary, topoSetDict, is provided in APPENDIX IX. The porosity model was then applied to the selected cells, as defined by the porosity function specified in the fvOptions file, which is reported in APPENDIX XI. Simulations were carried out on the same three mesh resolutions used previously, and isothermal conditions are applied on solid boundaries.

Figure 26 and Figure 27 show the output of the 0-D model when accounting for porosity. The filling time increases from 0.003055 s (without porous media) to 0.05784 s. Notably, even with the porous-filled tank, the model correctly predicts the final pressure inside the tank, which approximates the inlet total pressure.

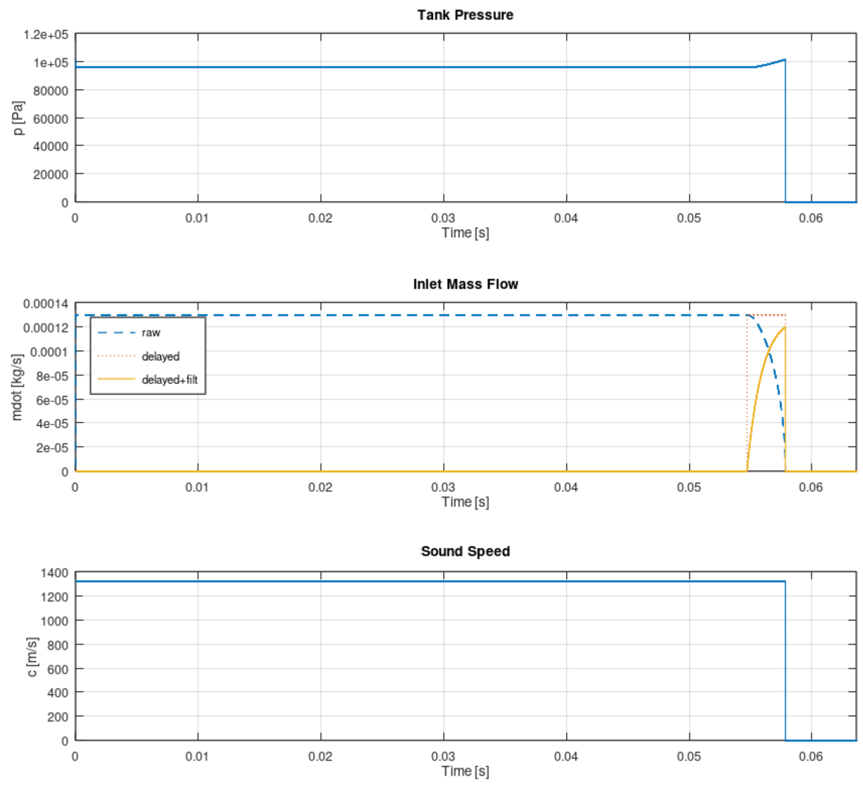
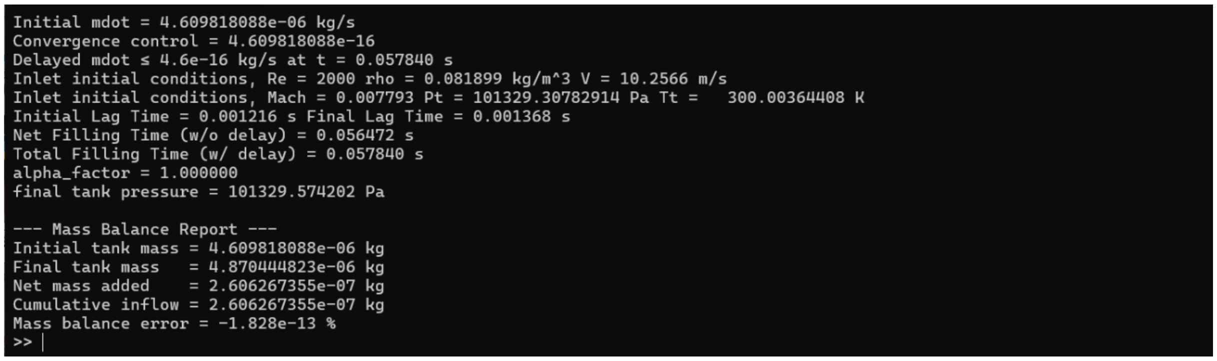


Figure 26: zero-dimensional model simulation of the empty tank filling process, w/ porous media



```
Initial mdot = 4.609818088e-06 kg/s
Convergence control = 4.609818088e-16
Delayed mdot ≤ 4.6e-16 kg/s at t = 0.057840 s
Inlet initial conditions, Re = 2000 rho = 0.081899 kg/m^3 V = 10.2566 m/s
Inlet initial conditions, Mach = 0.007793 Pt = 101329.30782914 Pa Tt =   300.00364408 K
Initial Lag Time = 0.001216 s Final Lag Time = 0.001368 s
Net Filling Time (w/o delay) = 0.056472 s
Total Filling Time (w/ delay) = 0.057840 s
alpha_factor = 1.000000
final tank pressure = 101329.574202 Pa

--- Mass Balance Report ---
Initial tank mass = 4.609818088e-06 kg
Final tank mass   = 4.870444823e-06 kg
Net mass added    = 2.606267355e-07 kg
Cumulative inflow = 2.606267355e-07 kg
Mass balance error = -1.828e-13 %
>> 
```

Figure 27: results of the 0-D model in Octave, empty tank, w/ porous media

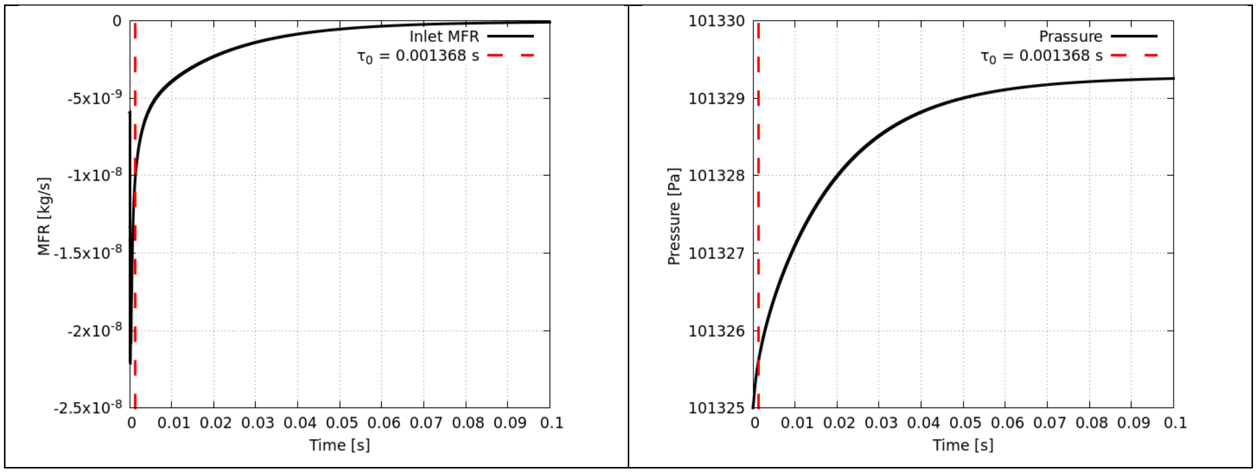


Figure 28: porous-filled tank, inlet mass flow rate (left) and pressure evolution with time (right) when the simulation is run over approximately 100 times the acoustic time τ_0

Figure 29 displays the numerical residuals of the momentum, pressure, and energy equations for both the isothermal and adiabatic cases. Compared to Figure 21, a notable difference in the behavior of the residuals can be observed. The porous medium inside the tank dampens the pressure reflections, resulting in smoother residual curves. Figure 30, Figure 31, and Figure 32 display the time evolution of the mass inside the tank, its time derivative, and of the average tank's pressure.

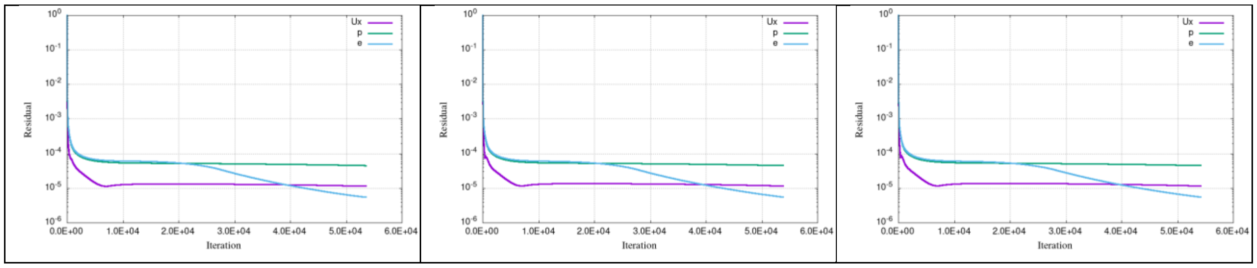


Figure 29: grid independence study: tank filling, axial symmetric case, isothermal walls, porous medium: numerical residuals; from left to right: coarse, medium, and fine grid level

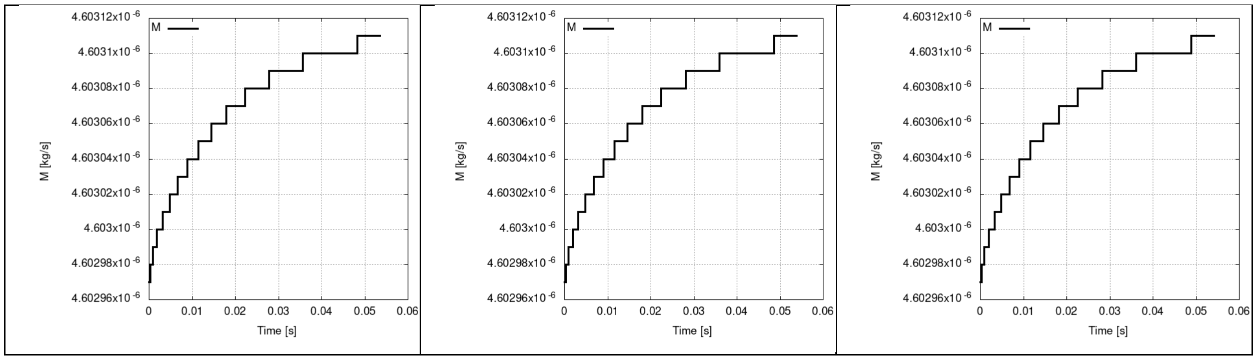


Figure 30: grid independence study: porous-filled tank, axial symmetric case, isothermal walls: time evolution of the mass inside the tank; from left to right: coarse, medium, and fine grid level

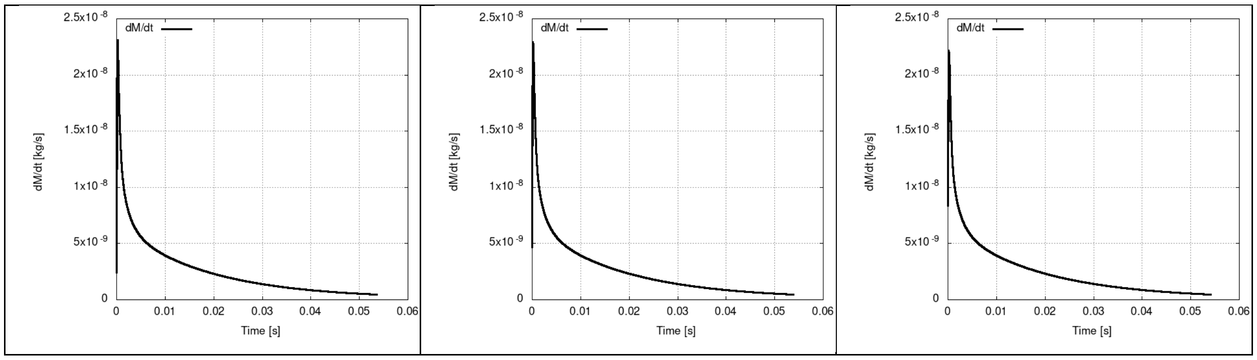


Figure 31: : grid independence study: porous-filled tank, axial symmetric case, isothermal walls: time evolution of the derivative of the mass inside the tank; from left to right: coarse, medium, and fine grid level

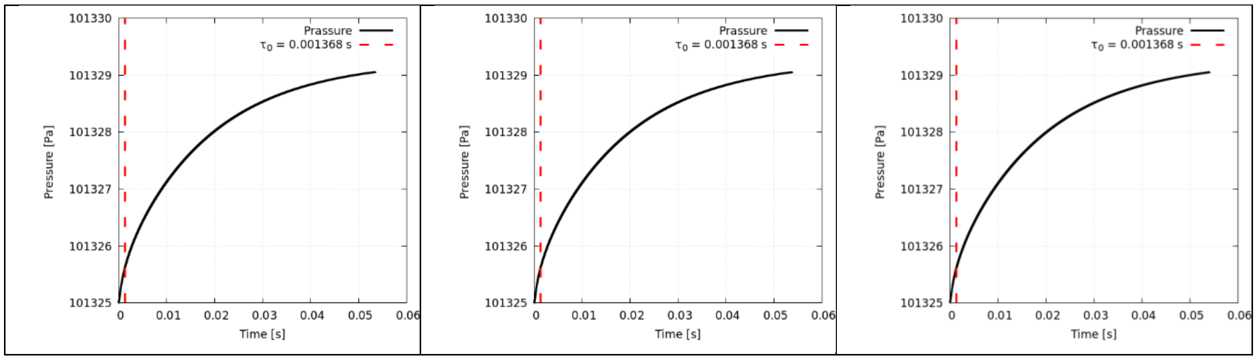


Figure 32: grid independence study: porous-filled tank, axial symmetric case, isothermal walls: time evolution of the tank average pressure; from left to right: coarse, medium, and fine grid level

Table 15: Tank filling time [s], porous-filled-tank, isothermal vs adiabatic walls, comparison with the zero-dimensional code

Zero-D solver (reference)	Coarse	Medium	Fine
0.05784	0.053523	0.053857	0.054125
|Error %|	7.463693	6.886238	6.422891

Table 16: grid independence study, porous-filled-tank, isothermal walls, tank average temperature [K] and average pressure [Pa] when the stop condition is reached

Temperature [K]			Pressure [Pa]		
Coarse	Medium	Fine	Coarse	Medium	Fine
300.0015	300.0030	300.0030	101329.055	101329.053	101329.051

Using the values from Table 15, the application of Equation 10 and Equation 11 allows us to compute the order of accuracy of the numerical model, as well as the "exact" solution obtained via Richardson extrapolation. The results are: p = 0.318, and f_{exact} = 0.055212 s.

Given the smooth and non-oscillatory behavior of the solution shown in Figure 28, we can analyze the error taking as reference the Richardson extrapolation value. Table 17 shows the error sequence for the three grids.

Table 17: porous-filled tank case, error sequence on the three grids

Grid	Value	Absolute Error $\phi - \phi_{\text{exact}}$
Coarse	0.053523	1.674e-03
Medium	0.053857	1.340e-07
Fine	0.054125	1.072e-07

Table 17 shows that the error decreases with grid refinement, but very slowly, confirming a low order of accuracy (p ≈ 0.32). The solution changes very little with refinement, a clear sign of excessive numerical damping, likely from the Darcy source term.

6.1 Takeaways

Introducing a non-reactive porous matrix (permeability K = 1×10^{-9} m^2) increases the filling time by nearly an order of magnitude compared to the empty-tank case. The 0-D model, modified to include the Darcy pressure drop, predicts a filling time of 0.05784 s; the medium and fine CFD meshes converge to 0.0539–0.0541 s, representing a 6–7 % underprediction relative to the extrapolated 0-D value

The porous medium also smooths out pressure reflections, yielding markedly less oscillatory residual histories. Tank-average temperature remains effectively at inlet conditions (∼300 K), and final static pressure reaches the inlet total pressure in all cases.

Grid-refinement analysis indicates a low observed order of accuracy (p≈0.32) for the porous-filled simulations, suggesting that the discretized Darcy penalty term introduces significant numerical damping. Despite this, rhoPimpleFoam successfully captures the key effect of the porous matrix, namely, the slower fill rate, while maintaining good agreement with the augmented 0-D model.

7 Conclusions

In this Phase 1 study, we have developed and benchmarked a hierarchy of models for hydrogen-tank filling, from a zero-dimensional (0-D) lumped parameter approach up to axisymmetric CFD simulations with porous media. Key findings include:

- **0-D Model Accuracy**: The 0-D "hybrid-lag" model captures transient pressure and temperature evolutions with filling-time predictions within ±1.5 % of CFD results for an empty tank. Its implementation in Octave provides rapid, robust estimates for design iterations.
- **Laminar Flow Validation**: simpleFoam accurately reproduces analytical Poiseuille profiles (error <0.5 % on fine grids). However, its reliance on kinematic viscosity limits its use in explicit–dynamic-viscosity–based porous formulations. rhoSimpleFoam resolves both incompressible and porous

Is it feasible to extend the 0D Model with a porous medium?
Is it feasible / worthwhile to extend the 0D Model with H2-gas absorption?

the term “compressible” confusing here?

flows correctly (velocity errors <0.2 %, Darcy pressure-drop errors <5 %), validating its suitability for coupled flow–porosity simulations.

- **Porous Media Modeling**: The implemented Darcy penalty term yields pressure-drop predictions within 5 % of theory across grid levels. The solver’s sensitivity to viscosity specification underscores the importance of using a compressible density-aware formulation.
- **Tank Filling with Heat Effects**: Axisymmetric rhoPimpleFoam simulations show that wall thermal treatment (adiabatic vs. isothermal) alters filling time by <0.2 % under rapid-fill conditions, justifying the adiabatic assumption for preliminary design.

In Phase 2 the study will be extended to include the actual adsorption/desorption reaction between the metallic matrix and hydrogen, and the model will be developed in the OpenFoam framework.

References

K. Holz, K. Boie, H. Christoph. 1953. *Laws of Turbulent Pipe Flow (Beginning at Small Reynolds Numbers).* NACA Technical Memorandum No. 1346, NACA. NACA TM-1346 (PDF).

Nikuradze, J. 1932. *Gesetzmäßigkeiten der turbulenten Strömung in glatten Rohren (Laws of Turbulent Flow in Smooth Pipes).* Heft 356 (Issue 356), Forschungsarbeiten auf dem Gebiete des Ingenieurwesens (Research Works in the Field of Engineering), Berlin: VDI-Verlag.

Nikuradze, J. 1933. *Strömungsgesetze in rauhen Rohren (Laws of Flow in Rough Pipes).* Heft 361 (Issue 361), Forschungsarbeiten auf dem Gebiete des Ingenieurwesens (Research Works in the Field of Engineering), Berlin: VDI-Verlag.

Rabienataj Darzi AA, et al. 2016. «Absorption and desorption of hydrogen in long metal hydride tank equipped with phase change material jacket.» *International Journal of Hydrogen Energy* 1-16. doi:http://dx.doi.org/10.1016/j.ijhydene.2016.04.051.

Reynolds, O. 1883. «An Experimental Investigation of the Circumstances Which Determine Whether the Motion of Water Shall Be Direct or Sinuous, and of the Law of Resistance in Parallel Channels.» *Philosophical Transactions of the Royal Society of London* 174: 935-982. doi:10.1098/rstl.1883.0029 .

APPENDIX I

blockMeshDict

Here is a brief explanation of how the grading works in blockMesh. We want to cluster the grid cells over the bottom and upper walls of a channel, with the same, but oppose, grading ratio. In this example[9] the block is divided uniformly in the x and z -directions and split into three grading sections in the y-direction described by three triples: ((0.2 0.3 4) (0.6 0.4 1) (0.2 0.3 0.25)). Each of the grading sections is described by a triple consisting of the fraction of the block, the fraction of the divisions and the grading ratio (size of first division/size of last division). Both the fraction of the block and the fraction of the divisions are normalized automatically so they can be specified scaled in anyway, e.g. as percentages, and they need not sum to 1 or 100.

```
blocks
  (
     hex (0 1 2 3 4 5 6 7) (20 60 20)
     simpleGrading
     (
        1
        ((2 3 4) (6 4 1) (2 3 0.25))
        1
     )
);
```

In the example above the total size "H" along y-direction is split in three parts. The first part starts from the bottom wall and covers 20% of H, contains 30% of the total number of cells along y, Ny, and has a grading ratio of 4. The second part covers 60% of H, contains 40% of the total number of cells along y, Ny, uniformly spaced (grading ratio of 1). Finally, the third and last part covers 20% of H, contains 30% of the total number of cells along y, Ny, and has a grading ratio of 0.25, (which is equal to ¼, the reciprocal of the grading ratio used on the bottom wall).

In our case, we do not intend to have an equally spaced distribution in the mid-height of the channel, so we can use the following two triples to impose the desired grading in the y-direction: ((0.5 0.5 32) (0.5 0.5 0.03125)). This statement says that the two divisions cover 50% of H each, and 50% of the total number of cells in the y-direction, with grading ratio of 32 on the bottom wall and 1/32 = 0.03125 on the top wall.

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////

FoamFile
{
   version     2.0;
   format      ascii;
   class      dictionary;
   object      blockMeshDict;
}
```

[9] See https://openfoamwiki.net/index.php/BlockMesh

```
scale 0.01; // dimensions in centimeters

vertices
(
   (0 0 0)
   (80 0 0)
   (80 4 0)
   (0 4 0)
   (0 0 1)
   (80 0 1)
   (80 4 1)
   (0 4 1)
);

blocks
(
   hex (0 1 2 3 4 5 6 7)
   (160 96 1) //24, 48, 96 along y for grid independence study
   simpleGrading
   (
      10                                      // x-direction (stretching toward inlet)
      ((0.5 0.5 32) (0.5 0.5 0.03125)) // y-direction: stretch from center to walls
      1                                       // z-direction

  edges
(
);

boundary
(
   inlet
   {
      type patch;
      faces
      (
         (0 3 7 4)
      );
   }

   outlet
   {
      type patch;
      faces
      (
         (1 2 6 5)
      );
   }
```

```
    walls
    {
        type wall;
        faces
        (
            (3 2 6 7) // top wall
            (0 1 5 4) // bottom wall
        );
    }

    empty
    {
        type empty;
        faces
        (
            (0 1 2 3)
            (4 5 6 7)
        );
    }
);

mergePatchPairs
(
);       // z-direction
    )
);
```

APPENDIX II

fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2306                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;

    div(phi,U)      bounded Gauss LUST grad(U);

    turbulence      bounded Gauss limitedLinear 1;
    div(phi,k)      $turbulence;
    div(phi,epsilon) $turbulence;
    div(phi,omega)  $turbulence;

    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
```

```
interpolationSchemes
{
   default        linear;
}

snGradSchemes
{
   default        corrected;
}

wallDist
{
   method         meshWave;
}
```

APPENDIX III

fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2306                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-10;
        relTol          0.1;
        smoother        DICGaussSeidel;
    }

    "(U|k|epsilon|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-10;
        relTol          0.1;
    }SIMPLE
{
    nNonOrthogonalCorrectors 0;
    consistent      yes;

    residualControl
    {
        p               1e-4;
        U               1e-6;
    }
}
relaxationFactors
{
    fields
    {
        p              0.3;
    }
    equations
    {
        U              0.7;
    }
}
}
```

APPENDIX IV

thermoPhysicalProperties file.

```
\*---------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class      dictionary;
   object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
thermoType
{
   type           hePsiThermo;
   mixture        pureMixture;
   transport      const;
   thermo         hConst;
   equationOfState perfectGas;
   specie         specie;
   energy         sensibleInternalEnergy;
}

mixture // hydrogen at temperature = 300 K
{
   specie
   {
      molWeight   2.01568;
   }
   thermodynamics
   {
      Cp         14310;
      Hf         0;
   }
   transport
   {
      mu         8.40e-06;
      Pr         0.71;
   }
   equationOfState
   {
      rho        1;
   }
}
// ************************************************************************* //
```

APPENDIX V

fvSchemes file.

```
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
   default        steadyState;
}

gradSchemes
{
   default        Gauss linear;

   limited        cellLimited Gauss linear 1;
   grad(U)        $limited;
   grad(k)        $limited;
   grad(omega)    $limited;
   grad(subsetU)   $limited;
}

divSchemes
{
   default        none;

   div(phi,U)      bounded Gauss linearUpwind limited;

   energy         bounded Gauss linearUpwind limited;
   div(phi,e)      $energy;
   div(phi,K)      $energy;
   div(phi,Ekp)    $energy;

   turbulence      bounded Gauss upwind;
   div(phi,k)      $turbulence;
   div(phi,omega)  $turbulence;

   div(phid,p)     Gauss upwind;
   div((phi|interpolate(rho)),p)  bounded Gauss upwind;

   div(((rho*nuEff)*dev2(T(grad(U)))))    Gauss linear;
}
laplacianSchemes
{
   default        Gauss linear corrected;
}

interpolationSchemes
{
   default        linear;
}

snGradSchemes
{
   default        corrected;
}

wallDist
{
   method         meshWave;
}
// ************************************************************************* //
```

APPENDIX VI

fvSolution file.

```
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
   p
   {
      solver          FPCG;//GAMG;(FPCG GAMG PBiCGStab PCG PPCG PPCR smoothSolver)
      preconditioner
      {
         preconditioner  GAMG;
         tolerance       1e-05;
         relTol          0;
         smoother           DICGaussSeidel;//(DIC DICGaussSeidel FDIC GaussSeidel nonBlockingGaussSeidel
symGaussSeidel)
      }
      tolerance       1e-6;
      relTol          0.01;
   }

   "(U|k|omega|e)"
   {
      solver          PBiCGStab;//(GAMG PBiCG PBiCGStab smoothSolver)
      preconditioner  DILU;//(DILU GAMG diagonal distributedDILU none)
      tolerance       1e-6;
      relTol          0.01;
   }
}
SIMPLE
{
   residualControl
   {
      p             1e-4;
      U             1e-4;
      "(k|omega|e)"   1e-4;
   }

   nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
   fields
   {
      p             0.01;
      rho           0.01;
   }
   equations
   {
      U             0.7;
      e             0.01;
      "(k|omega)"    0.7;
   }
}
// ************************************************************************* //
```

APPENDIX VII

The block mesh file used to generate the coarser mesh level for the axis symmetric simulation of the filling tank. To obtain finer grids double the number of cells used in the block section in the "y" direction (ny), while leaving nx and nz unchanged.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2306                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

scale 0.01;  // Convert cm to m

// Geometry parameters
xmin  -10;  // Feeding pipe start (cm)
R1    1;   // Feeding pipe radius (cm)
R2    4;   // Cylinder radius (cm)
L     80;   // Cylinder length (cm)

// Precomputed values for 2.5° wedge angle
cs    0.9990482216;  // cos(2.5°)
sn    0.04361938736; // sin(2.5°)

vertices
(
    // Front vertices (wedge angle = -2.5°)
    (-10 0 0)                              // 0
    (-10 0.9990482216 -0.04361938736)      // 1
    (0    0.9990482216 -0.04361938736)     // 2
    (0    3.9961928864 -0.17447754944)     // 3
    (80   3.9961928864 -0.17447754944)     // 4
    (80   0 0)                             // 5
    (0    0 0)                             // 6
    (80   0.9990482216 -0.04361938736)     // 7

    // Back vertices (wedge angle = +2.5°)
```

```
   (-10 0 0)                              // 8
   (-10 0.9990482216 0.04361938736)          // 9
   (0    0.9990482216 0.04361938736)         // 10
   (0    3.9961928864 0.17447754944)         // 11
   (80   3.9961928864 0.17447754944)         // 12
   (80   0 0)                             // 13
   (0    0 0)                             // 14
   (80   0.9990482216 0.04361938736)         // 15
);

blocks
(
   // Feeding pipe (40 axial, 6, 12, 24 radial for the coarse, medium, fine mesh levels)
   hex (0 6 2 1 8 14 10 9)
   (40 6 1)
   simpleGrading
   (
      0.125   // Axial clustering toward inlet
      0.03125 // Radial clustering toward wall
      1
   )

   // Cylinder inner (160 axial, 12 radial) 6, 12, 24 radial for the coarse, medium, fine mesh levels)
   hex (6 5 7 2 14 13 15 10)
   (160 6 1)
   simpleGrading
   (
      32     // Axial uniform
      0.03125 // Radial clustering toward inner wall
      1
   )

   // Cylinder outer (160 axial, 18, 36, 72 radial for the coarse, medium, fine mesh levels)
   hex (2 7 4 3 10 15 12 11)
   (160 18 1)
   simpleGrading
   (
      32                              // Axial uniform
      ((0.25 0.5 32) (0.25 0.5 0.03125) ) // Radial clustering
      1
   )
);

edges ();

boundary
(
   wedgeFront
```

```
{
   type wedge;
   faces
   (
      (0 1 2 6)
      (6 2 7 5)
      (2 3 4 7)
   );
}

wedgeBack
{
   type wedge;
   faces
   (
      (8 9 10 14)
      (14 10 15 13)
      (10 11 12 15)
   );
}

inlet
{
   type patch;
   faces
   (
      (0 8 9 1)   // Feeding pipe inlet
   );
}

walls
{
   type wall;
   faces
   (
      (1 9 10 2)   // Feeding pipe wall
      (3 11 12 4)  // Cylinder outer wall
      (5 13 15 7)  // Outlet inner (x = L)
      (7 15 12 4)  // Outlet outer (x = L)
      (2 3 11 10)  // Left face (x = 0)
   );
}

axis
{
   type empty;
   faces
   (
            (0 6 14 8)  // Axis face at feeding pipe start
            (6 5 13 14)
        );
    }
);

mergePatchPairs ();
```

APPENDIX VIII

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2306                                  |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
FoamFile
{
    version     2;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
application     rhoPimpleFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         0.1;
deltaT          1e-06;
maxDeltaT       1e-06;
maxCo           0.5;
adjustTimeStep  on;
writeControl    adjustable;
writeInterval   1;
purgeWrite      1;
writeFormat     binary;
writePrecision  8;
writeCompression off;
timeFormat      general;
timePrecision   9;
runTimeModifiable true;
functions
{
    massConvergenceTol 0.0001;
    volFieldValueP
    {
        type            volFieldValue;
        libs            ( "fieldFunctionObjects" );
        enabled         true;
        log             true;
        fields          ( p );
        operation       volAverage;
        zone            Porous;
        zoneType        cellZone;
        writeFields     false;
        writeToFile     true;
        writePrecision  16;
        writeFormat     scientific;
        executeControl  timeStep;
        executeInterval 1;
        writeControl    timeStep;
        writeInterval   1;
    }
    volFieldValueT
    {
        type            volFieldValue;
        libs            ( "fieldFunctionObjects" );
        enabled         true;
        log             true;
        fields          ( T );
        operation       volAverage;
        zone            Porous;
        zoneType        cellZone;
        writeFields     false;
        writeToFile     true;
        writePrecision  6;
        writeFormat     scientific;
        executeControl  timeStep;
        executeInterval 1;
        writeControl    timeStep;
        writeInterval   1;
    }
    H2InletMassFlow
    {
        type            surfaceFieldValue;
        libs            ( libfieldFunctionObjects );
        enabled         true;
        log             true;
        writeFields     false;
        regionType      patch;
        name            inlet;
        operation       sum;
        fields          ( phi );
        writeToFile     true;
        writePrecision  6;
        writeFormat     scientific;
        executeControl  timeStep;
        executeInterval 1;
        writeControl    timeStep;
        writeInterval   1;
    }
```

```
pAverageInlet
{
   type          surfaceFieldValue;
   libs          ( libfieldFunctionObjects );
   enabled       true;
   log           true;
   writeFields    false;
   regionType     patch;
   name          inlet;
   operation      average;
   fields         ( p );
   writeToFile    true;
   writePrecision  20;
   writeFormat    scientific;
   executeControl  timeStep;
   executeInterval 1;
   writeControl   timeStep;
   writeInterval   1;
}
massConvergence
{
   type          coded;
   libs          ( "libutilityFunctionObjects.so" );
   name          massConvergence;
   enabled       false;
   log           true;
   writeControl   timeStep;
   writeInterval   1;
   writePrecision  3;
   codeExecute    #{
      static Foam::scalar Mprev = -1.0;
      static Foam::scalar timePrev = -1.0;
      static Foam::scalar initialRate = -1.0;  // initial dM/dt
      static Foam::scalar it = 0;
      const fvMesh& mesh = this->mesh();
      const Time& runTime = mesh.time();
      const volScalarField& rho = mesh.lookupObject<volScalarField>("rho");
      scalar M               = gSum(rho.internalField() * mesh.V().field());
      scalar t               = runTime.timeOutputValue();
      scalar deltaT          = runTime.deltaTValue();
      scalar lagTime         = 2*0.9/sqrt(1.4*4124*300);
      scalar relTol          = 1e-9;   // relative tolerance
      // Create output file on first call
      static std::ofstream outFile;
      if (!outFile.is_open())
      {
         mkDir("postProcessing/dMassTankdt");
         outFile.open("postProcessing/dMassTankdt/dMassTankdt.dat");
```

```
            outFile << "# Time(s)    dM/dt (kg/s)    M (kg)    dM (kg)     dt (s)" << std::endl;
        }
        if (Mprev > 0 && timePrev >= 0)
        {
            scalar dM   = (M - Mprev);
            scalar dt   = (t - timePrev);
            scalar dMdt = dM / dt;
            outFile << t << "    " << dMdt << "    " << M << "    " << dM << "    " << dt << std::endl;
    // Save first nonzero rate as reference
            if(it == 0) //if (initialRate < 0 && Foam::mag(dMdt) > SMALL)
            {
                initialRate = M/dt; //Foam::mag(dMdt);
                Foam::Info << "M: " << M << " kg/s\n";
                Foam::Info << "Mprevious: " << Mprev << " kg/s\n";
                Foam::Info << "Initial dM/dt set to: " << initialRate << " kg/s\n";
                Foam::Info << "Convergence control set to: " << relTol*initialRate << " kg/s\n";
                it = 1;
            }


            if (initialRate > 0)
            {
                scalar convTol = relTol * initialRate;
                if (Foam::mag(dMdt) < convTol && t > lagTime)
                {
                    Foam::Info << "\n*** Convergence: |dM/dt| = "
                               << Foam::mag(dMdt)
                               << " kg/s < " << convTol << " kg/s ("
                               << relTol << " × initial). Writing and stopping. ***\n";
                    const_cast<Time&>(runTime).writeNow();
                    const_cast<Time&>(runTime).stopAt(Time::saWriteNow);
                }
            }
        }
        // Store current values
        Mprev = M;
        timePrev = t;
    #};
  }
}
// ************************************************************************* //
```

APPENDIX IX

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2306                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;

    limited         cellLimited Gauss linear 1;
    grad(U)         $limited;
    grad(k)         $limited;
    grad(omega)     $limited;
}
divSchemes
{
    default         none;

    div(phi,U)      Gauss linearUpwind limited;

    energy          Gauss linearUpwind limited;
    div(phi,e)      $energy;
    div(phi,K)      $energy;
    div(phi,Ekp)    $energy;

    turbulence      Gauss linearUpwind limited;
    div(phi,k)      $turbulence;
    div(phi,omega)  $turbulence;

    div(phiv,p)     Gauss upwind;
    div((phi|interpolate(rho)),p) Gauss upwind;

    div(((rho*nuEff)*dev2(T(grad(U)))))    Gauss linear;
}

laplacianSchemes
{
    default        Gauss linear corrected;
}

interpolationSchemes
{
    default        linear;
}

snGradSchemes
{
    default        corrected;
}

wallDist
{
    method         meshWave;
}
// ************************************************************************* //
```

APPENDIX X

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2306                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          FPCG;//GAMG;(FPCG GAMG PBiCGStab PCG PPCG PPCR smoothSolver)
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-05;
            relTol          0;
            smoother        DICGaussSeidel;//(DIC DICGaussSeidel FDIC GaussSeidel nonBlockingGaussSeidel
symGaussSeidel)
        }
        tolerance       1e-6;
        relTol          0.01;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    "(rho|U|k|omega|e)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-6;
        relTol          0.1;
    }
    "(rho|U|k|omega|e)Final"
    {
        $U;
        relTol          0;
    }
}

PIMPLE
{
    nCorrectors              1;
    nNonOrthogonalCorrectors 1;
    nOuterCorrectors         1;
}

relaxationFactors
{
    fields
    {
        p     0.3;
        rho   0.3;
    }
    equations
    {
        U     0.7;
        e     0.7;
    }
}


// ************************************************************************* //
```

APPENDIX XI

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

actions
(
    {
        name    PorousCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box (0 -5 -5) (1 5 5);
    }
    {
        name    Porous;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        sourceInfo
        {
            set PorousCellSet;
        }
    }
);


// ************************************************************************* //
```

APPENDIX XII

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2012                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

porosity
{
    type            explicitPorositySource;
    active          no;
    selectionMode  cellZone;
    cellSet         Porous;
    explicitPorositySourceCoeffs
    {
        selectionMode   cellZone;
        cellZone        Porous;
        type            DarcyForchheimer;
        DarcyForchheimerCoeffs
        {
            d   (1.e9 0 0);
            f   (0 0 0);
            coordinateSystem
            {
            type    cartesian;
            origin  (0 0 0);
            rotation none;
            }
        }
     }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```