Simulations, Post-Processing and Visualisations of Detector Cooling Systems

Viren Bhanot, Yann Herpin, Youri Penders (EP-DT-DC) | 2025-10-27

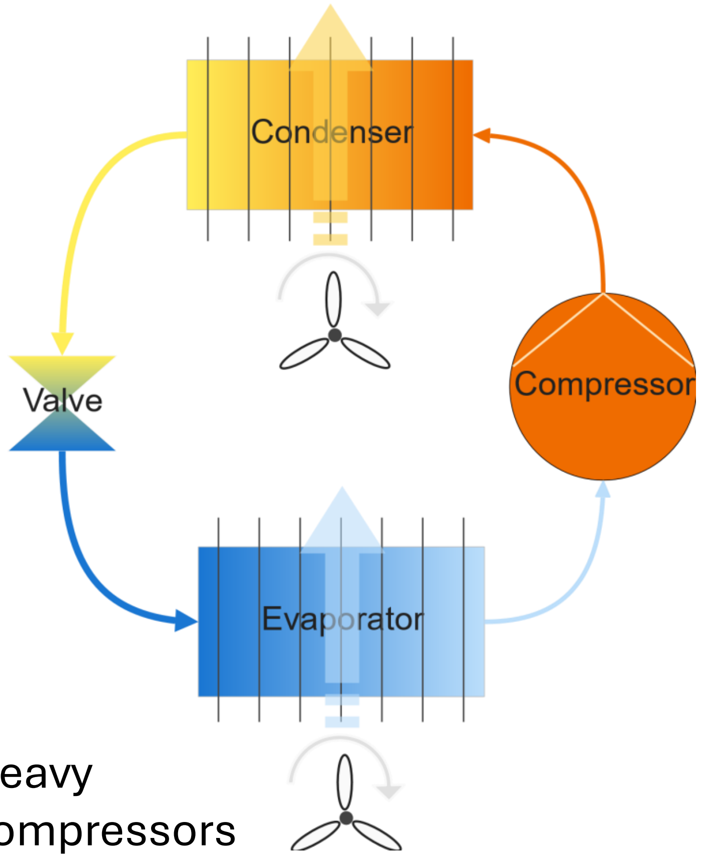
PyHEP 2025 | https://indico.cern.ch/event/1566263

Disclaimer

- This is NOT a ‘coding’ talk.
- I don’t show much code.
- This is a ‘**How we *use* code**’ talk.
- We are NOT software engineers, not even Physicists.

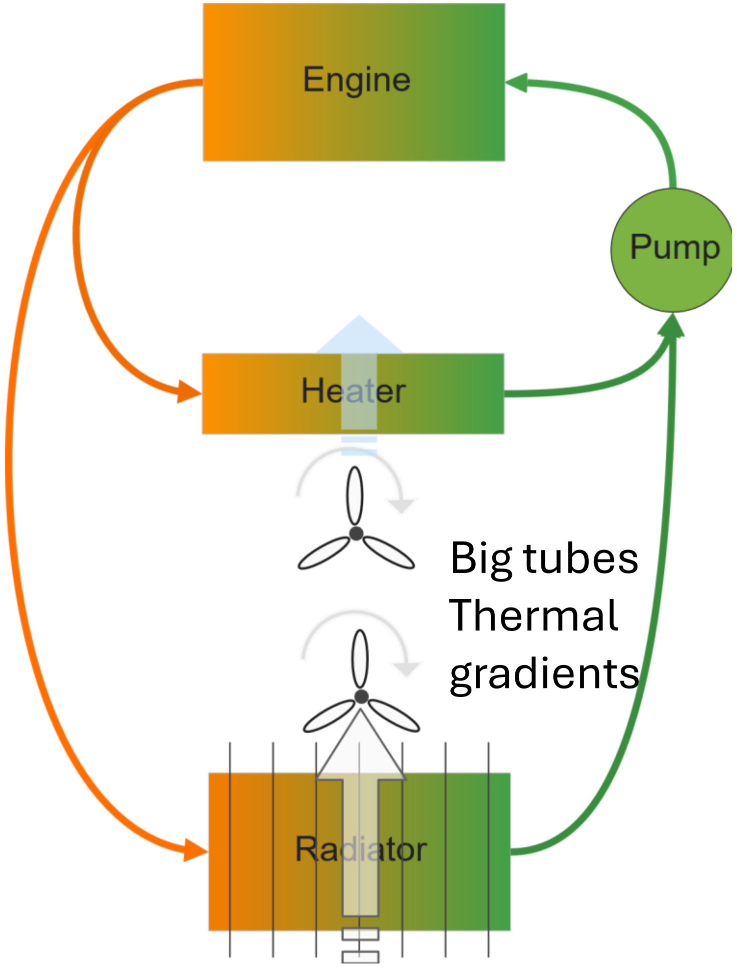
Any and all advice is welcome!

Cooling systems: An overview



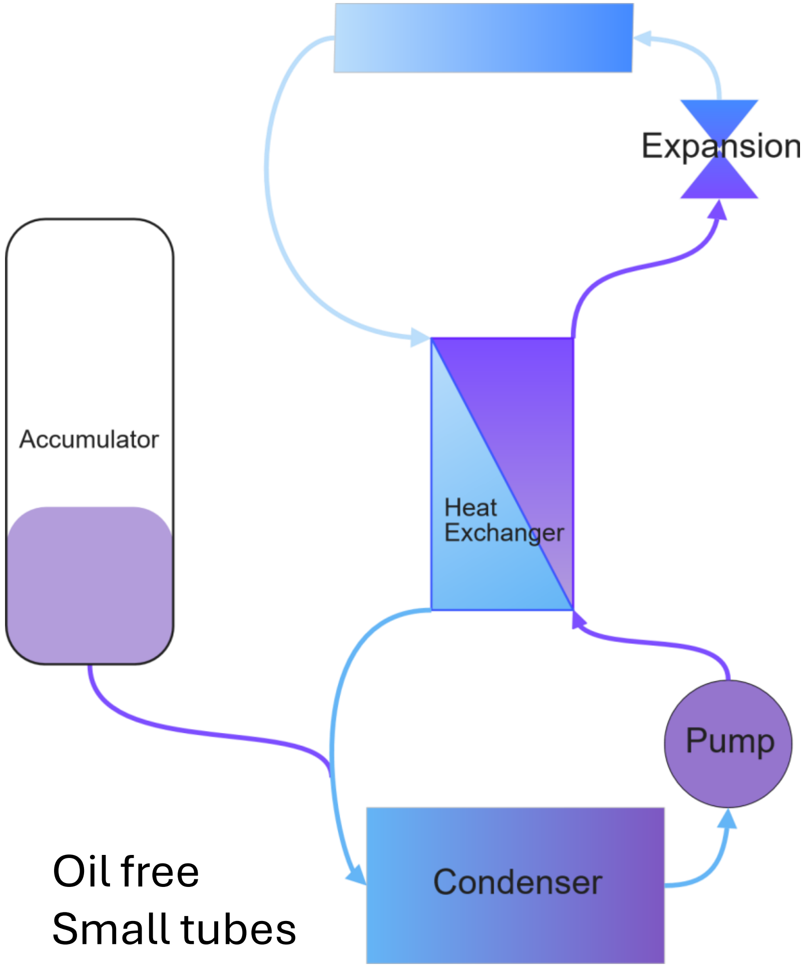
Heavy
compressors
Usually not oil-free

Vapour Compression System



Big tubes
Thermal
gradients

Liquid Cooling



Oil free
Small tubes
Low power

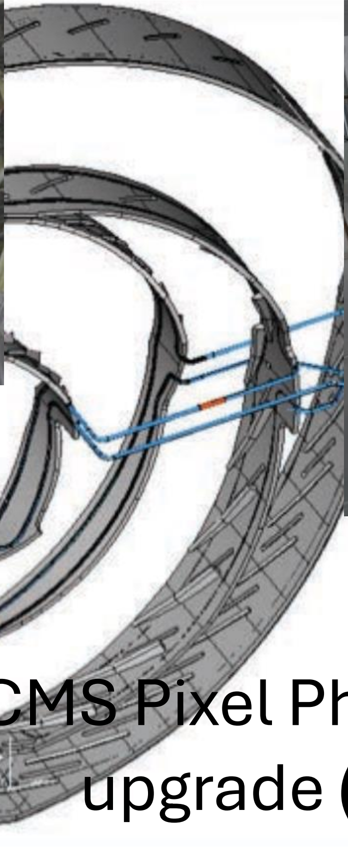
Two-Phase Pumped Loops (CO2-based)

History of CO2 pumped loops at CERN

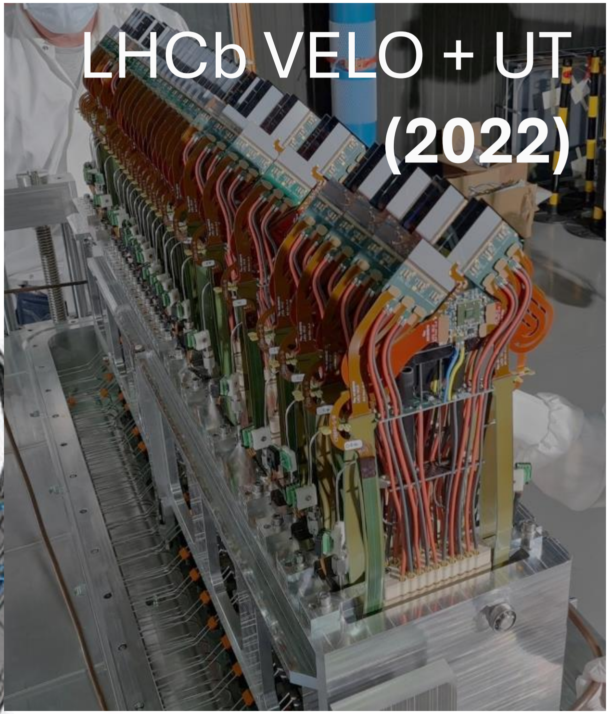
AMS-02 **(2011)**

LHCb VELO **(2008)**

ATLAS IBL upgrade **(2014)**

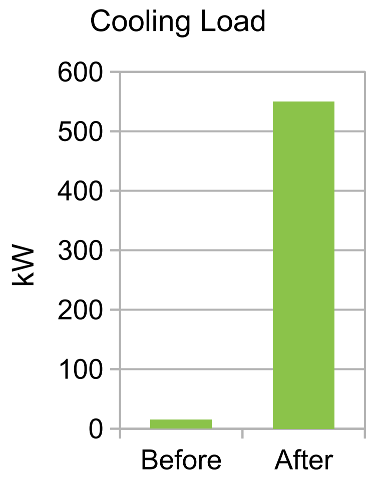


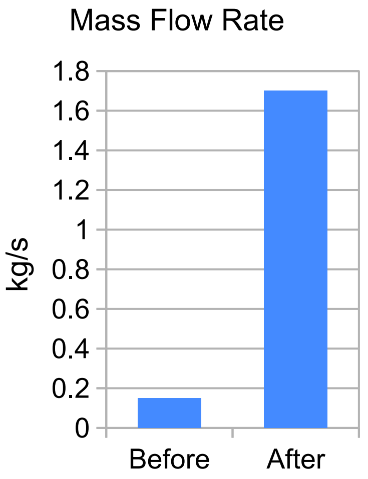
CMS Pixel Phase-1 upgrade **(2016)**

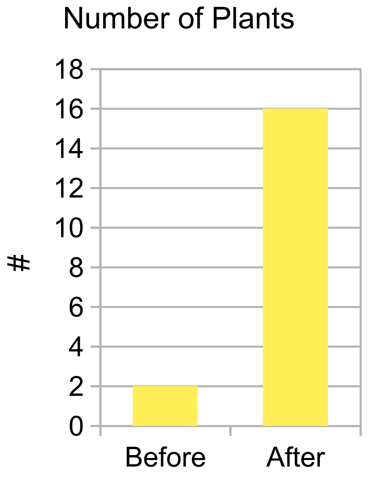


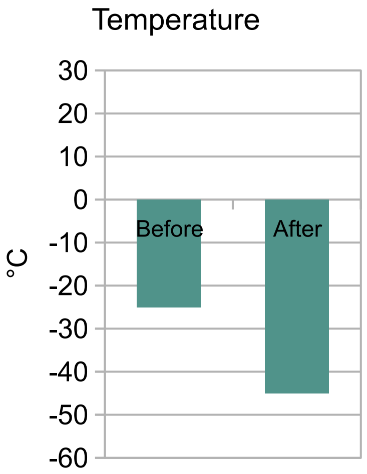
LHCb VELO + UT **(2022)**

Phase-2 Hi-Lumi upgrade









Matlab to Python

- **Matlab for everything**
  - Rich history of helper functions / toolboxes / code
  - Nice ODE solvers
  - Amazing plotting library
  - ‘Compact’ language
- Not open-source/free
- Timber→Matlab = Convoluted
  - CSVs, intermediate steps…
- Chance to learn something new

- **Slowly migrating to Python**
  - Pressure drop/Heat transfer correlations
  - Valve equations
  - Finite volume 1D modelling
  - Plotting data
  - Post-processing and data analysis
  - Visualisation

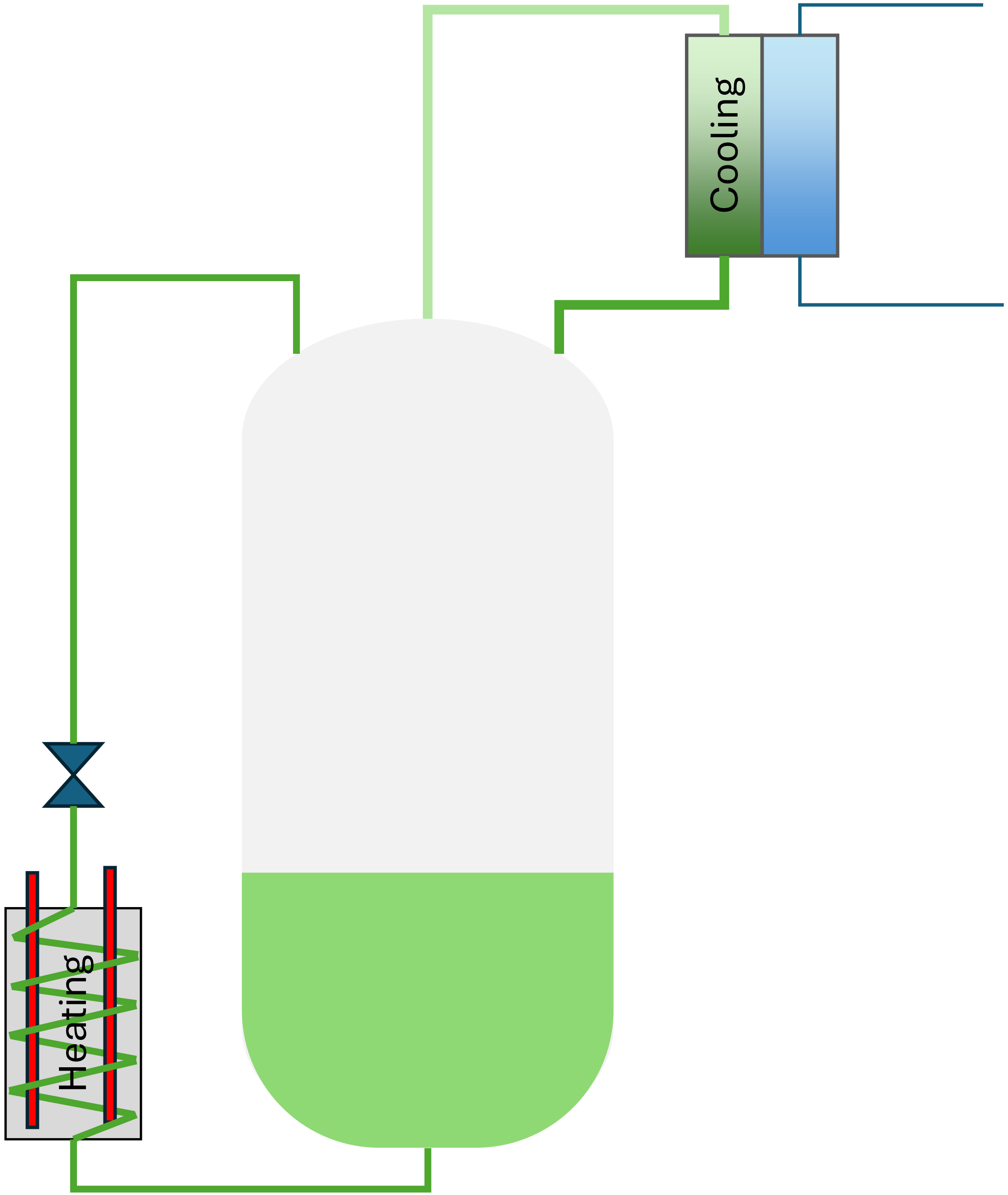
- Long road ahead…

Simulations

0D and 1D Finite Volume Modelling of control volumes.

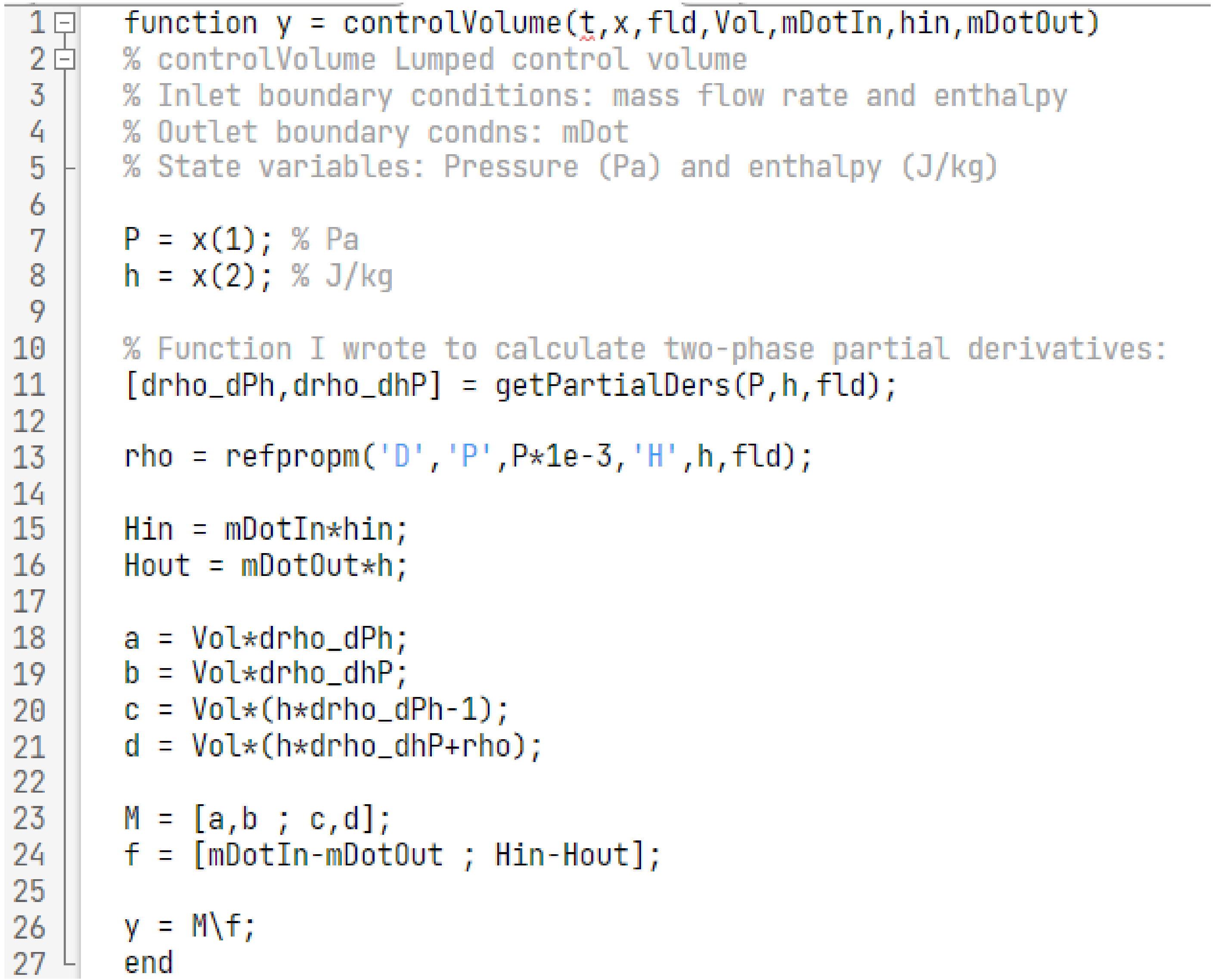
Accumulator

- Reservoir with 2-phase CO2
- Can be heated or cooled
  - Heat → Evaporate → raise pressure
  - Cool → Condense → lower pressure
- Nuances: Safety valves, spray cooling, liquid in and out
- Before: Homogeneous modelling
- Now: Two-fluid model (vapour and liquid phases separate)
  - Python with Scipy DAE solvers



Homogeneous Model in Matlab

- Very basic model:
  - Mass balance
  - Energy balance
  - Pressure/enthalpy as state variables
- `y = A\B`
  - Backslash operator <3
- Matlab
  - ODE15s
  - ODE15i (Implicit form)
  - Progress plot while solving

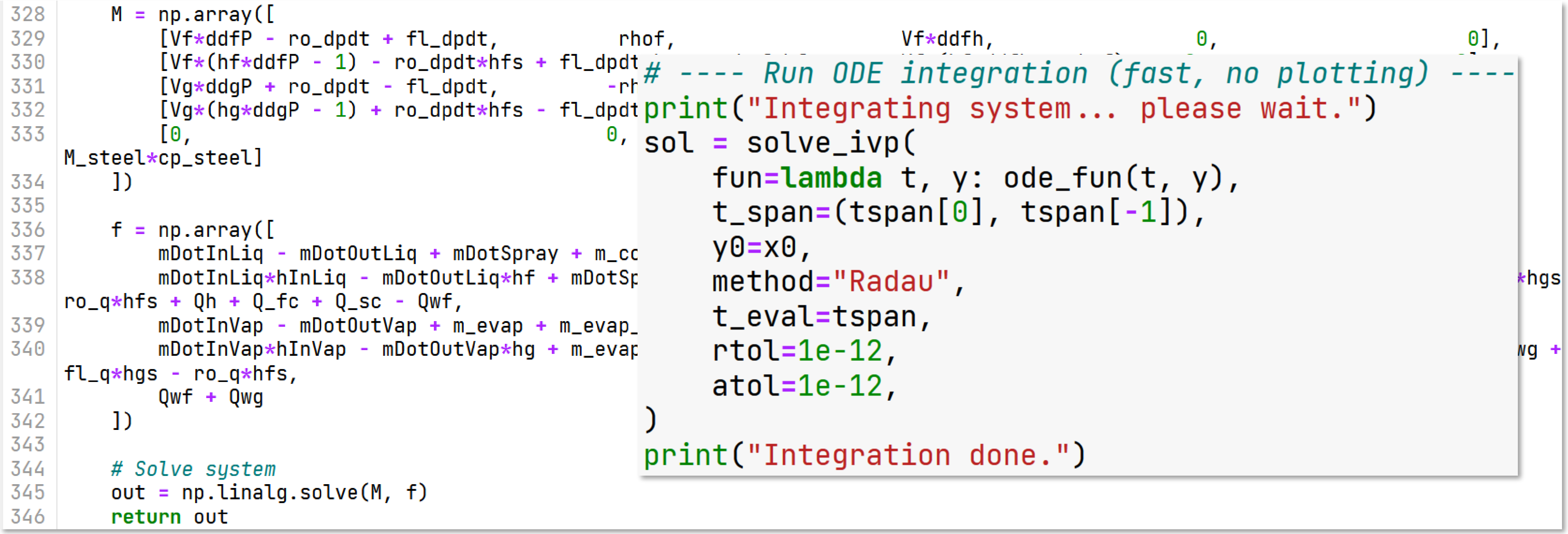


```
function y = controlVolume(t,x,fld,Vol,mDotIn,hin,mDotOut)
% controlVolume Lumped control volume
% Inlet boundary conditions: mass flow rate and enthalpy
% Outlet boundary condns: mDot
% State variables: Pressure (Pa) and enthalpy (J/kg)

P = x(1); % Pa
h = x(2); % J/kg

% Function I wrote to calculate two-phase partial derivatives:
[drho_dPh,drho_dhP] = getPartialDers(P,h,fld);

rho = refpropm('D','P',P*1e-3,'H',h,fld);

Hin = mDotIn*hin;
Hout = mDotOut*h;

a = Vol*drho_dPh;
b = Vol*drho_dhP;
c = Vol*(h*drho_dPh-1);
d = Vol*(h*drho_dhP+rho);

M = [a,b ; c,d];
f = [mDotIn-mDotOut ; Hin-Hout];

y = M\f;
end
```

Two-Fluid Model

- Model the liquid and gas phases separately
- Flash evaporation and rain-out condensation
- Heat transfer between phases

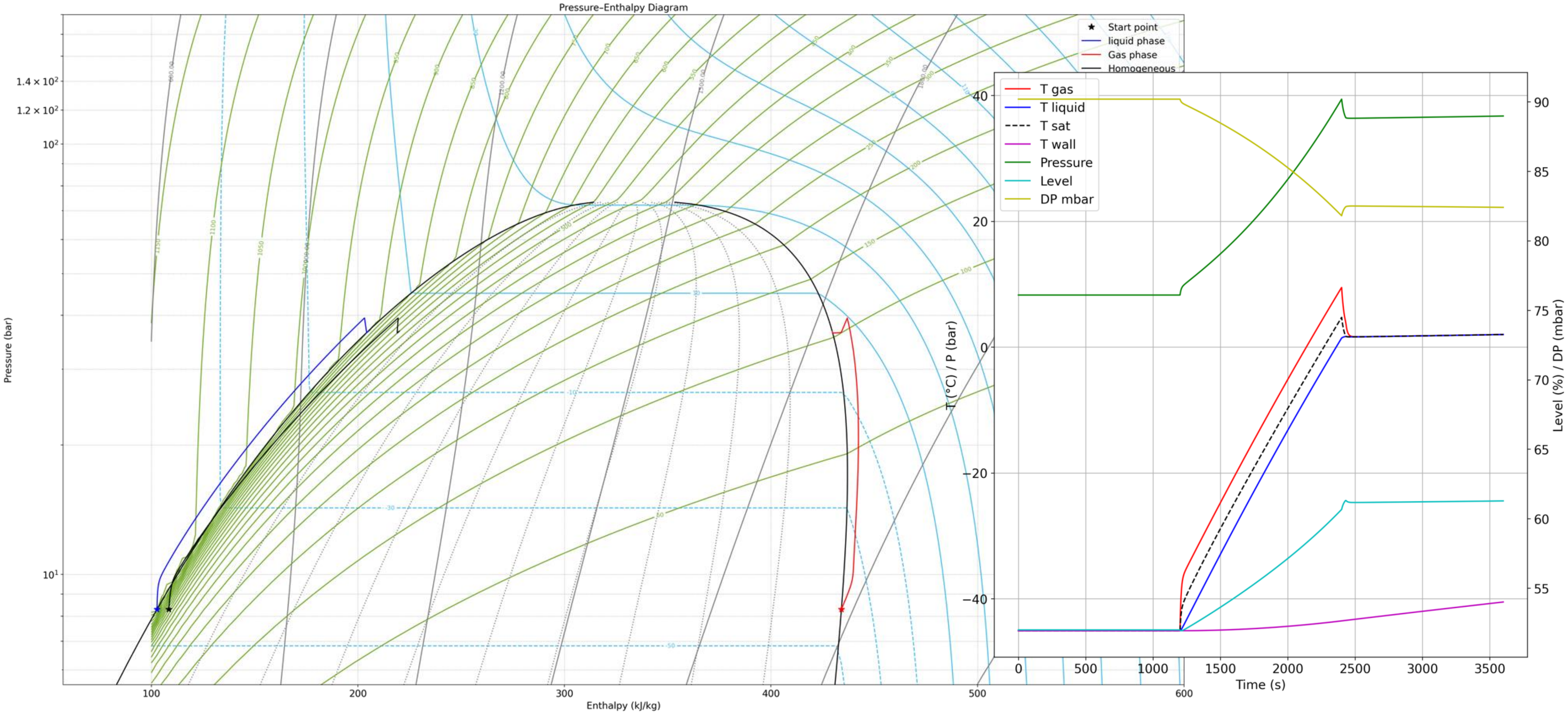
- Scipy ivp problem
- Nice DAE solvers (Radau etc.)



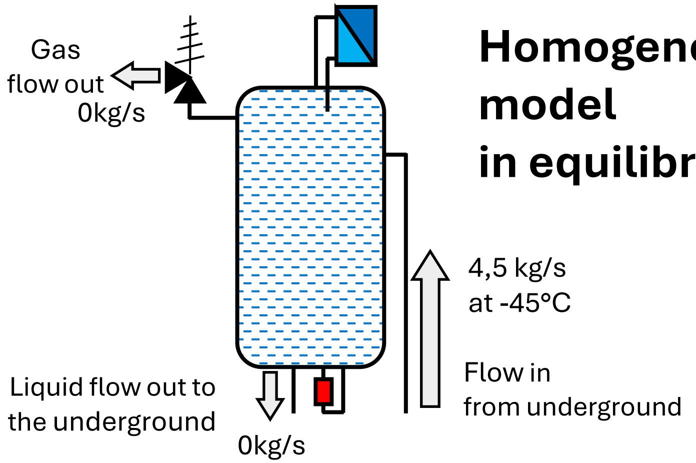
```
    M = np.array([
        [Vf*ddfP - ro_dpdt + fl_dpdt,          rhof,                    Vf*ddfh,              0,                    0],
        [Vf*(hf*ddfP - 1) - ro_dpdt*hfs + fl_dpdt
        [Vg*ddgP + ro_dpdt - fl_dpdt,          -rh
        [Vg*(hg*ddgP - 1) + ro_dpdt*hfs - fl_dpdt
        [0,                                    0,
M_steel*cp_steel]
    ])

    f = np.array([
        mDotInLiq - mDotOutLiq + mDotSpray + m_cc
        mDotInLiq*hInLiq - mDotOutLiq*hf + mDotSp                                                                  *hgs
ro_q*hfs + Qh + Q_fc + Q_sc - Qwf,
        mDotInVap - mDotOutVap + m_evap + m_evap_
        mDotInVap*hInVap - mDotOutVap*hg + m_evap                                                                 wg +
fl_q*hgs - ro_q*hfs,
        Qwf + Qwg
    ])

    # Solve system
    out = np.linalg.solve(M, f)
    return out
```

```
# ---- Run ODE integration (fast, no plotting) ----
print("Integrating system... please wait.")
sol = solve_ivp(
    fun=lambda t, y: ode_fun(t, y),
    t_span=(tspan[0], tspan[-1]),
    y0=x0,
    method="Radau",
    t_eval=tspan,
    rtol=1e-12,
    atol=1e-12,
)
print("Integration done.")
```

Rapid-pressurisation example



Safety valve flow



Gas
flow out
0kg/s

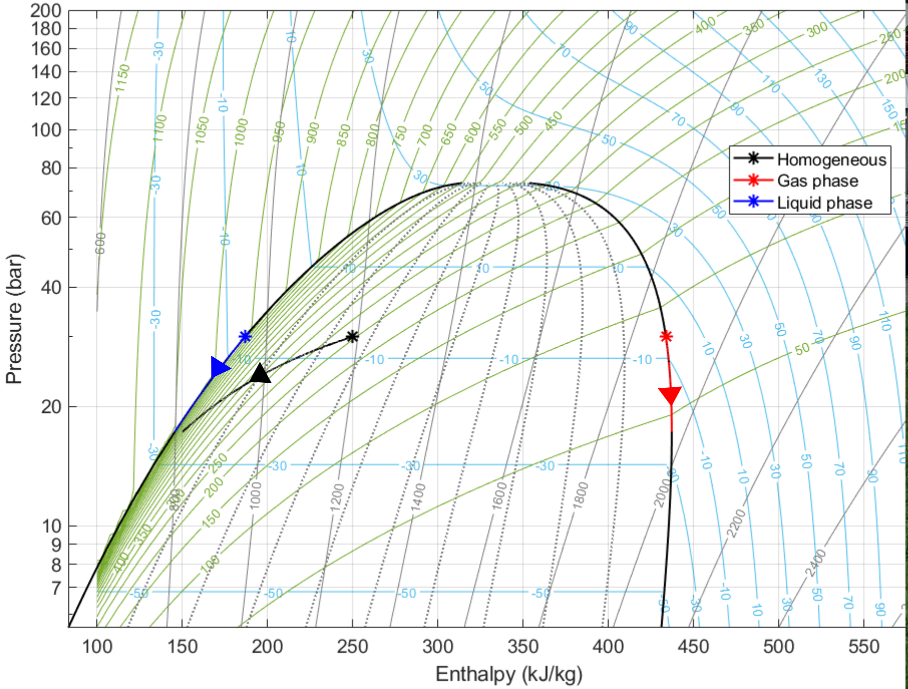
Liquid flow out to
the underground

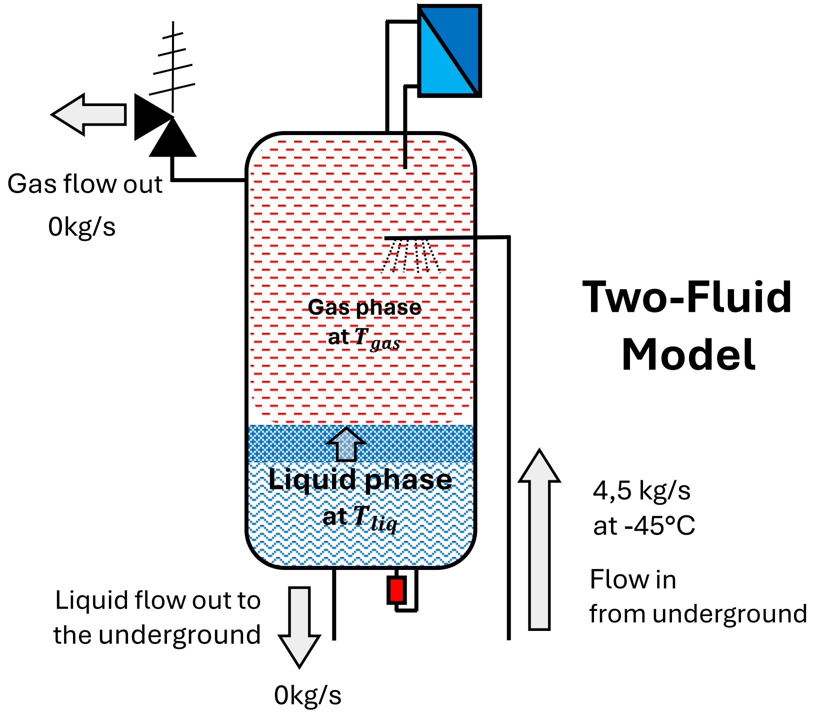
0kg/s

4,5 kg/s
at -45°C

Flow in
from underground

Homogeneous model
in equilibrium





Gas flow out
0kg/s

Gas phase
at T_{gas}

Liquid phase
at T_{liq}

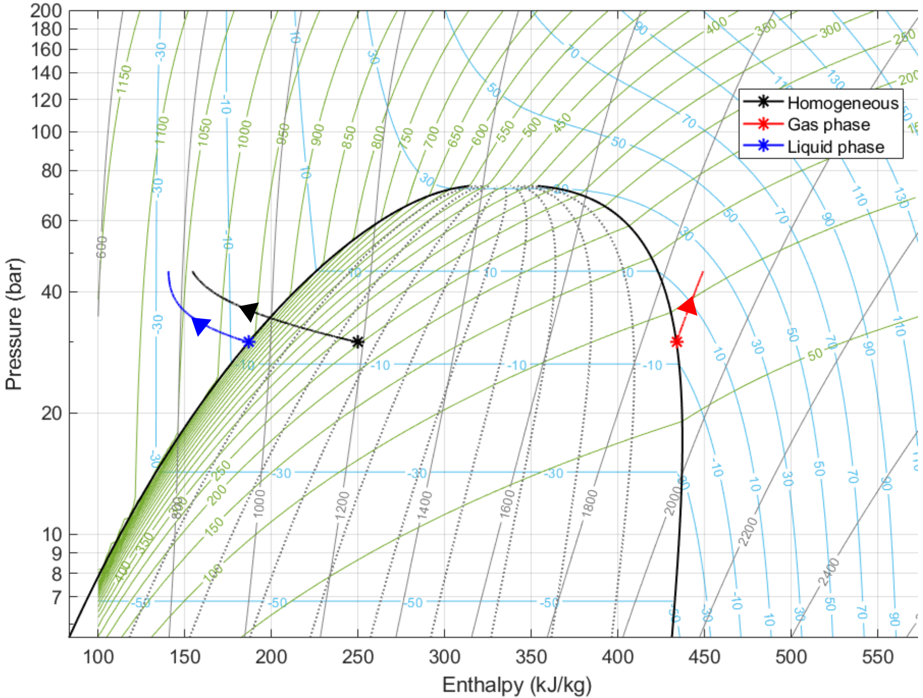
Liquid flow out to
the underground

0kg/s

4,5 kg/s
at -45°C

Flow in
from underground

Two-Fluid
Model

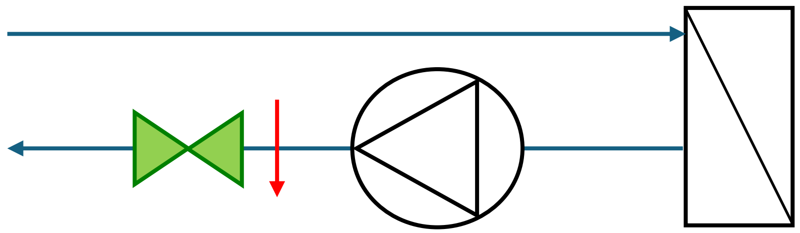


Data analysis and post-processing

Importing from NXCals, analysing test results, developing scans for systems…

Scanning the pump

- Pump curve data → slowly closing valve at pump outlet → pressure increase → measure mass flow
- Pump not 100% efficient ⇒ mass flow not constant over full range

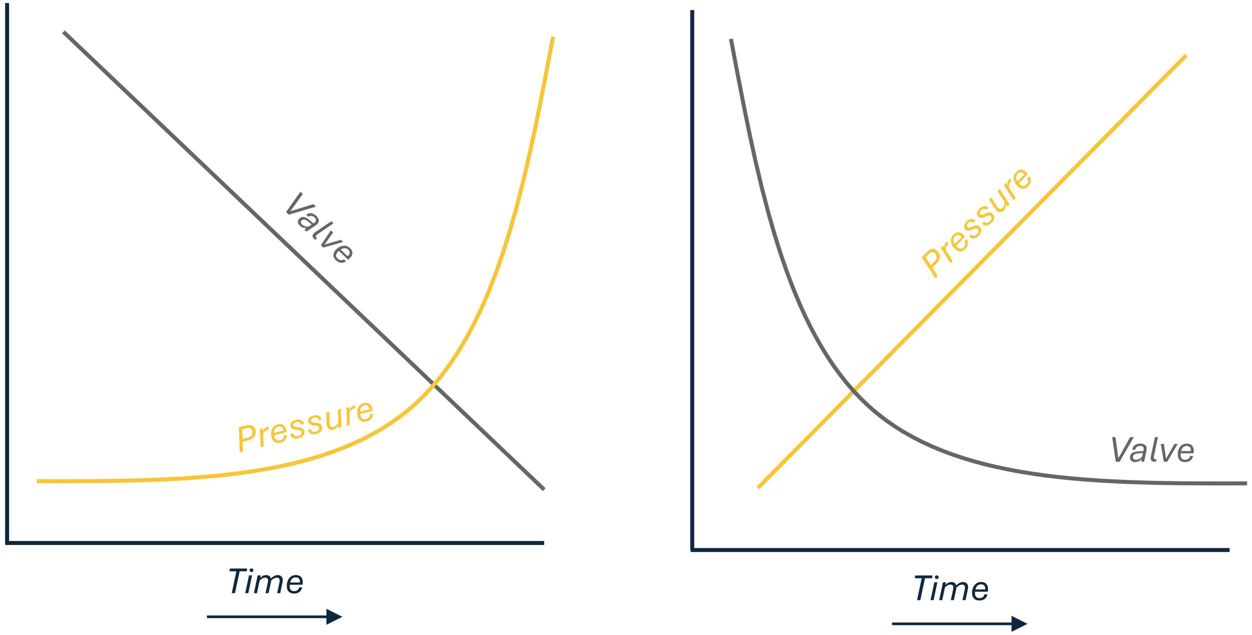


$$Q = C_v\sqrt{\Delta P}$$

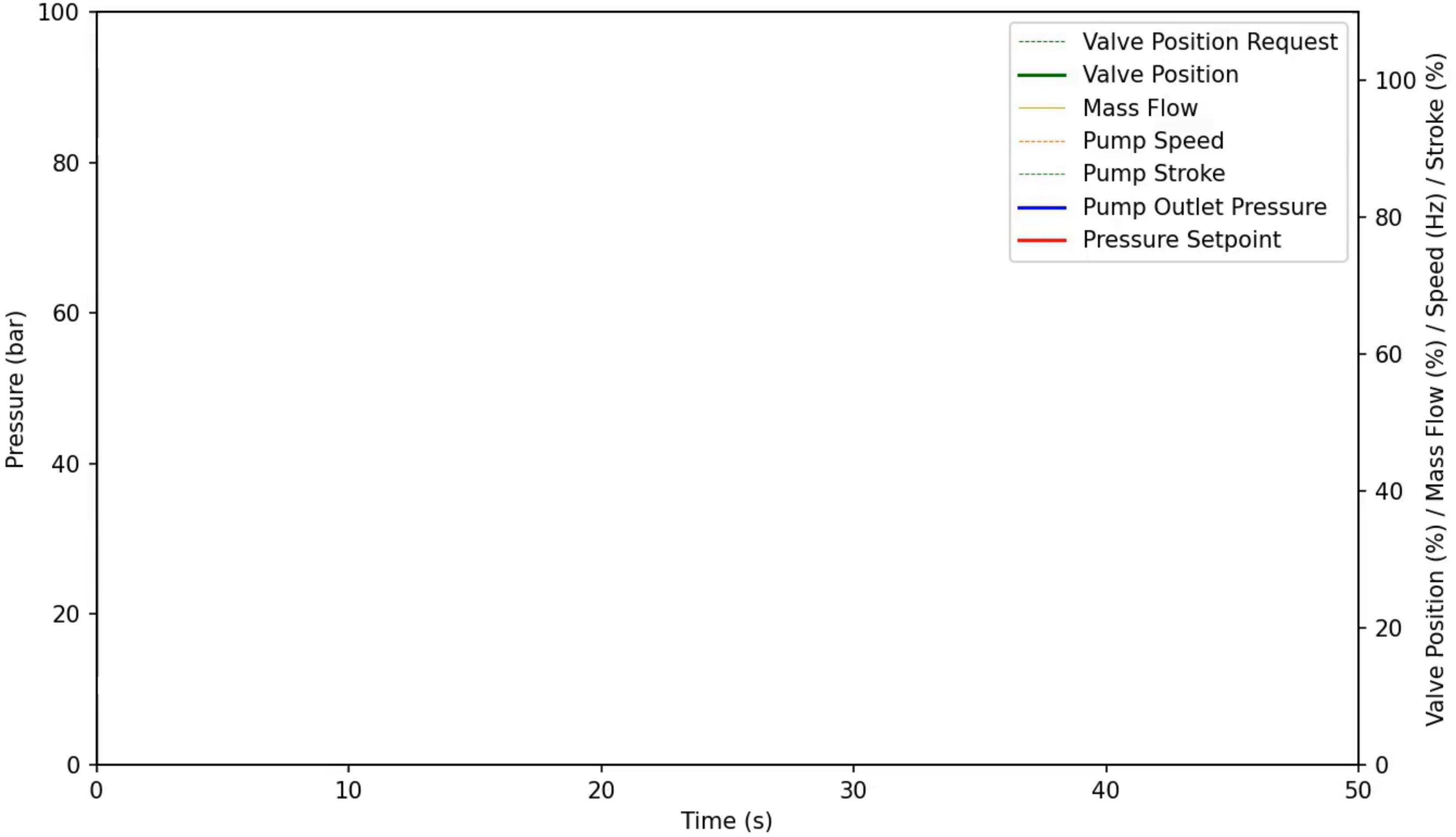
$$\frac{d(\Delta P)}{dt} = constant$$

$$C_v \propto x$$

$$\frac{d}{dt}\left(\frac{1}{x^2}\right) = constant \longrightarrow \boxed{x = \frac{1}{\sqrt{at+b}}}$$

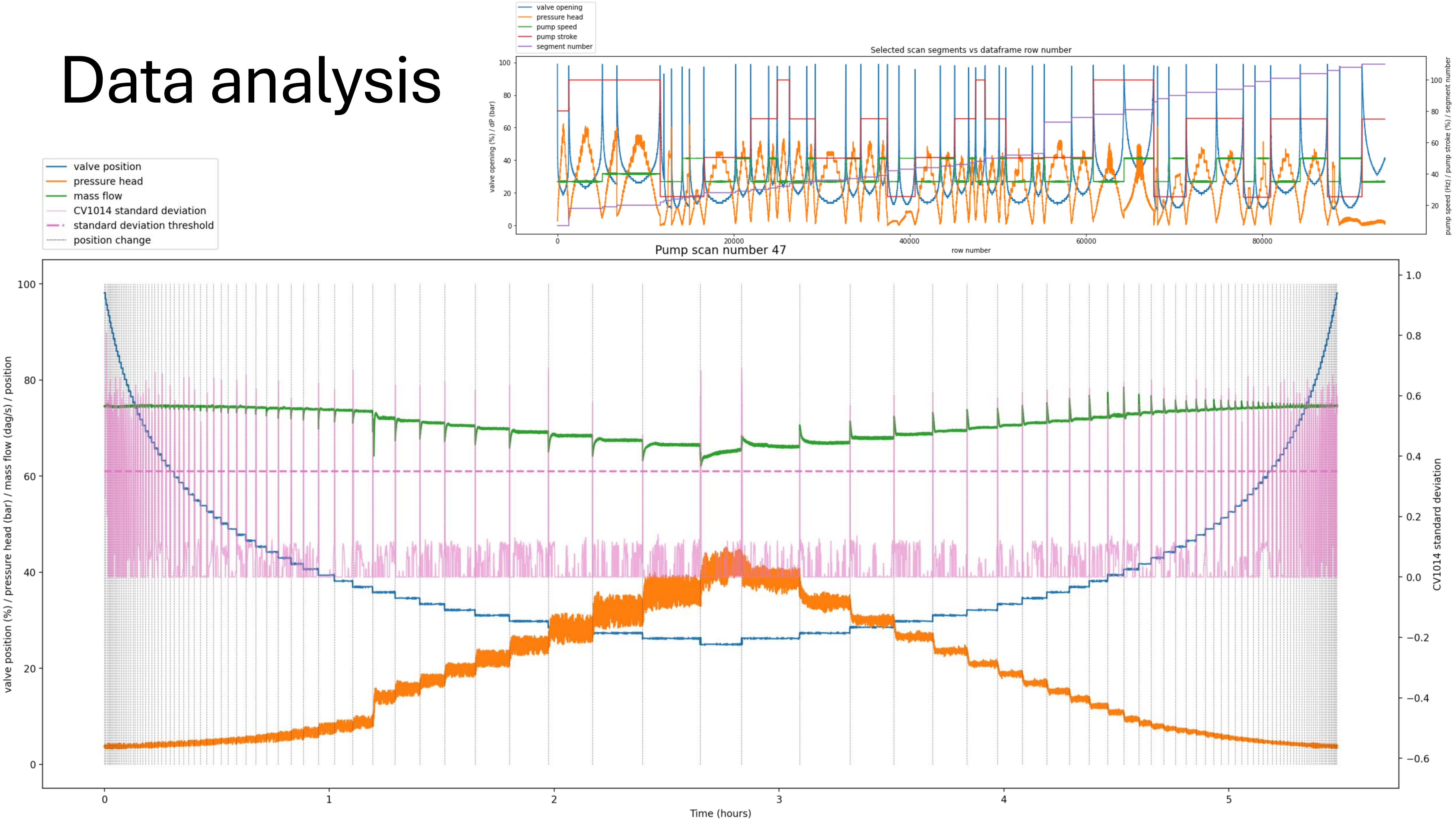


Developing auto-scan

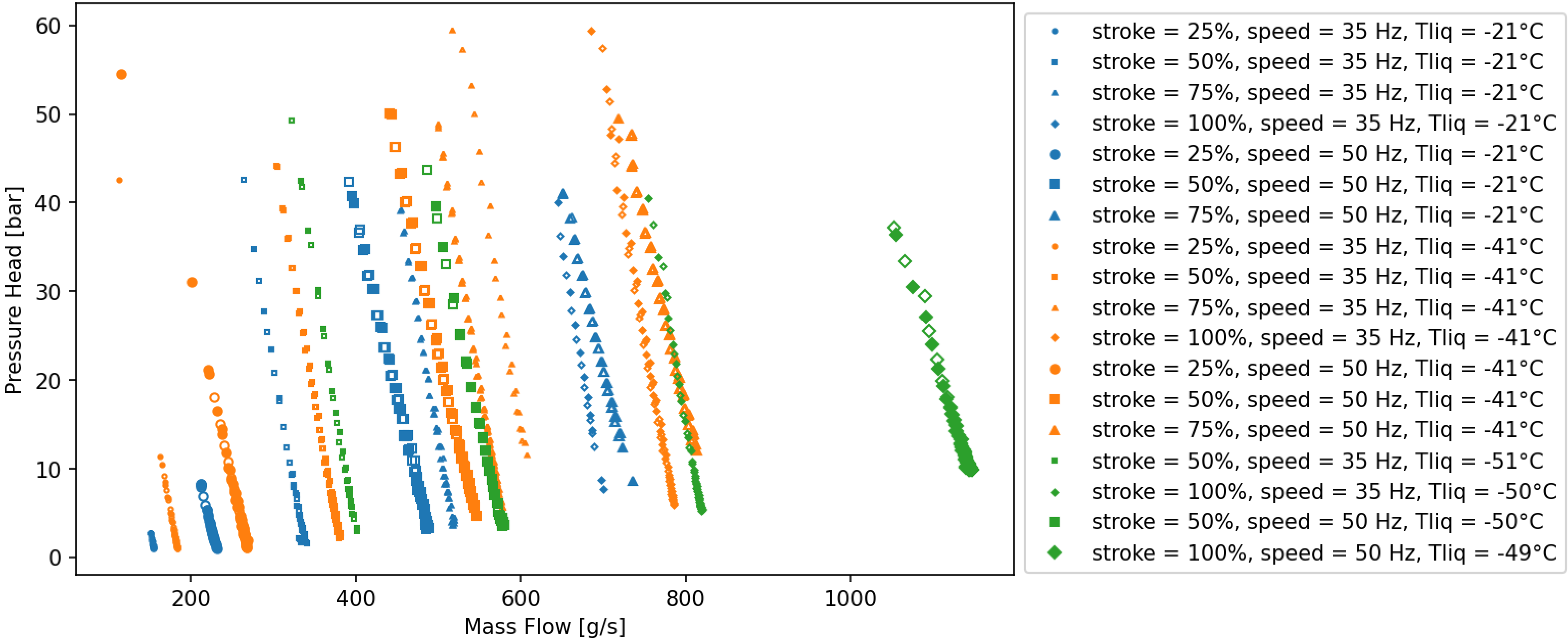


- Let’s simulate it before we do it.
- Play around with coefficients/parameters
- Estimate scan times/speeds
- Get feedback from PLC logic/SCADA engineers

Data analysis

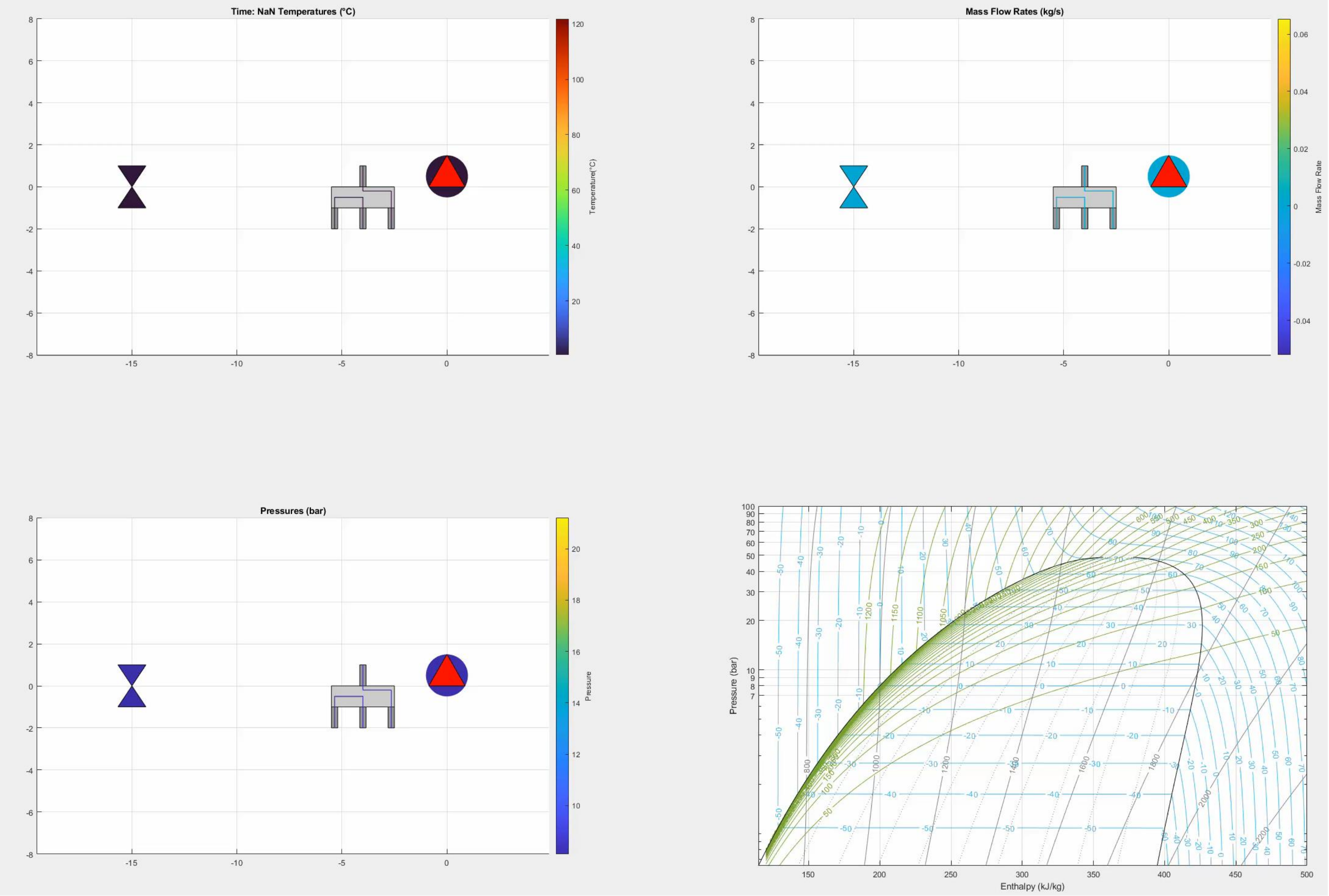


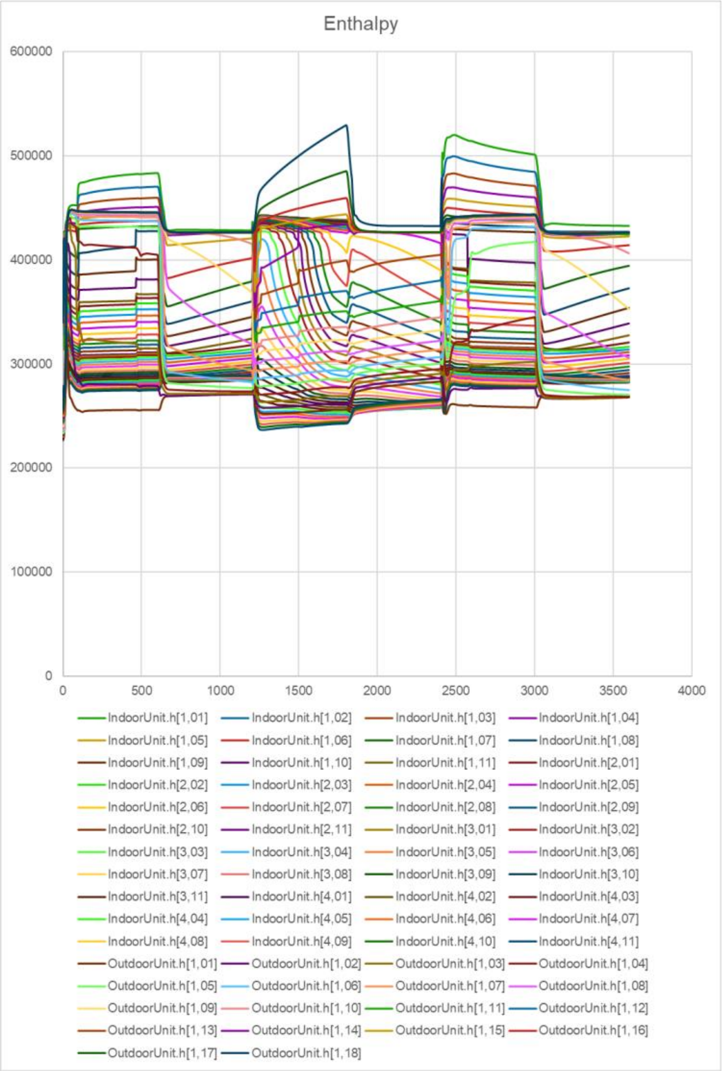
Overall pump curves



Visualisations

Glean: visualising thermofluid systems





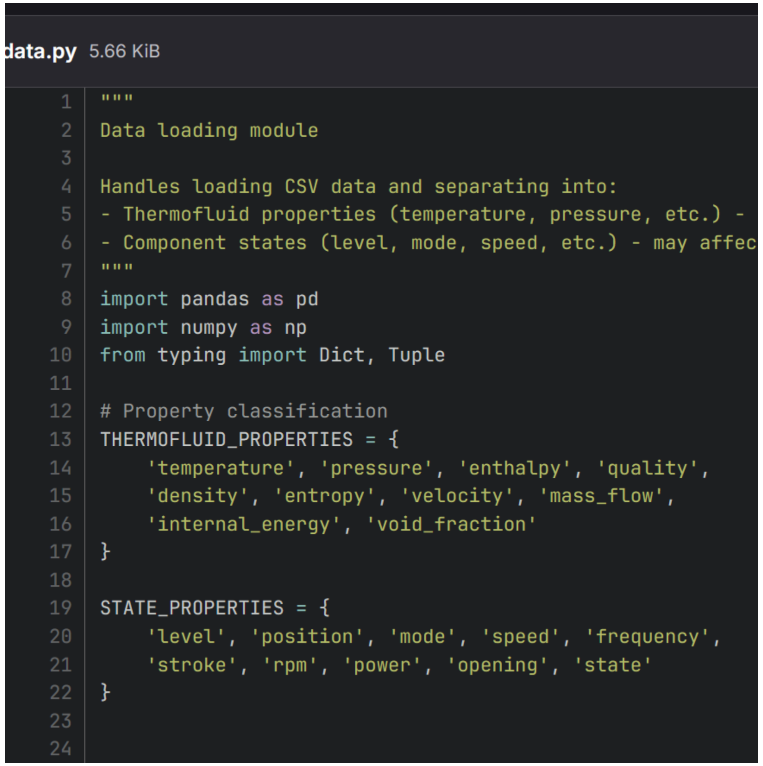
Python vs Matlab implementation

Matlab

- Ad-hoc development of components.
- No overall structure to workflow
- Patch update = slow

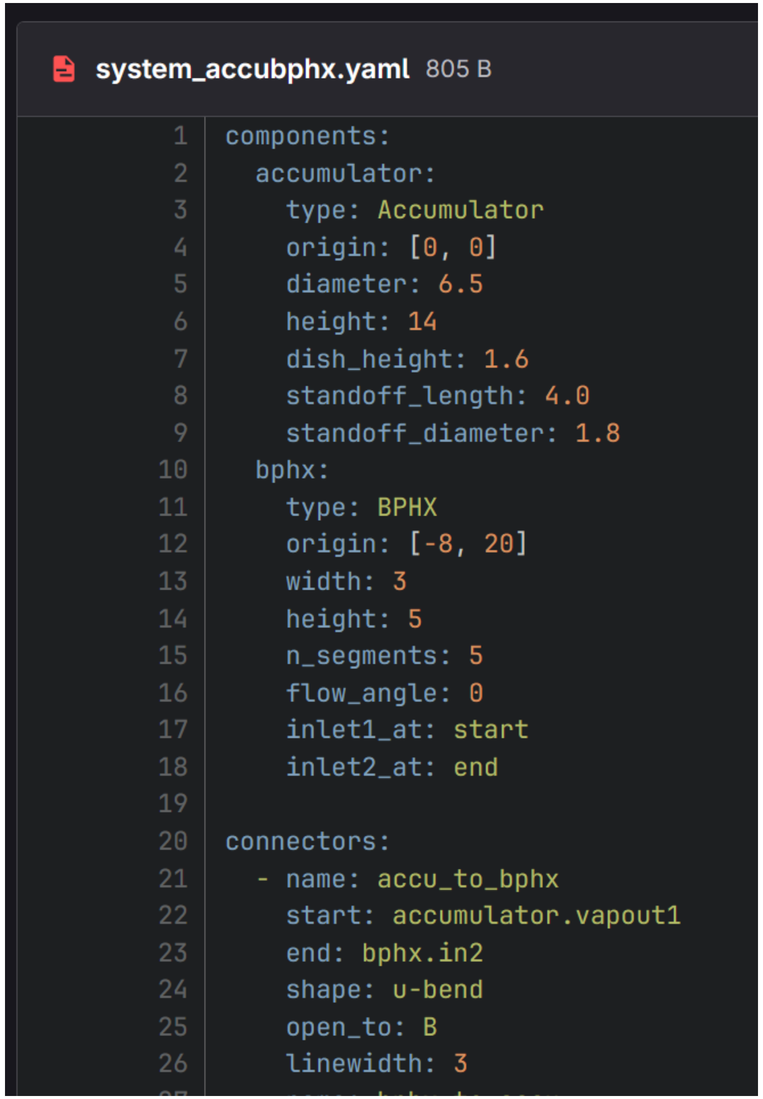
Python

- Modular development: data.py, geometry.py etc.
- YAML to specify system schematic
    - Components, positions, locations
    - Data file column mapping
- Each component is a class
- Two types of properties
    - Thermodynamic (Pressure)
    - State (% opening, frequency etc.)



data.py 5.66 KiB

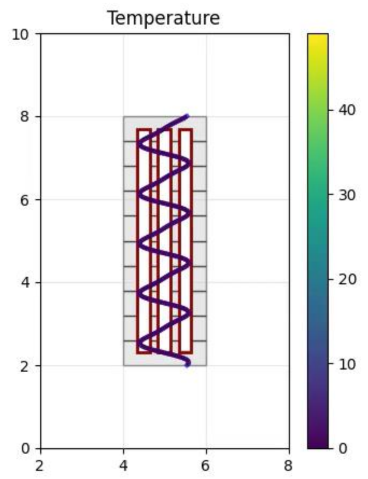
```python
"""
Data loading module

Handles loading CSV data and separating into:
- Thermofluid properties (temperature, pressure, etc.) -
- Component states (level, mode, speed, etc.) - may affec
"""
import pandas as pd
import numpy as np
from typing import Dict, Tuple

# Property classification
THERMOFLUID_PROPERTIES = {
    'temperature', 'pressure', 'enthalpy', 'quality',
    'density', 'entropy', 'velocity', 'mass_flow',
    'internal_energy', 'void_fraction'
}

STATE_PROPERTIES = {
    'level', 'position', 'mode', 'speed', 'frequency',
    'stroke', 'rpm', 'power', 'opening', 'state'
}


```

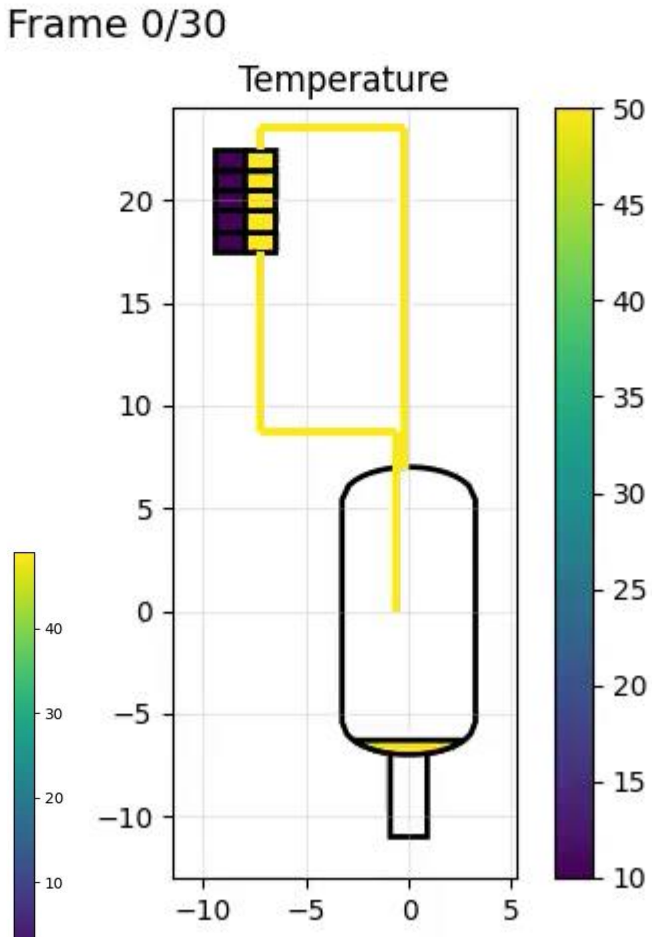


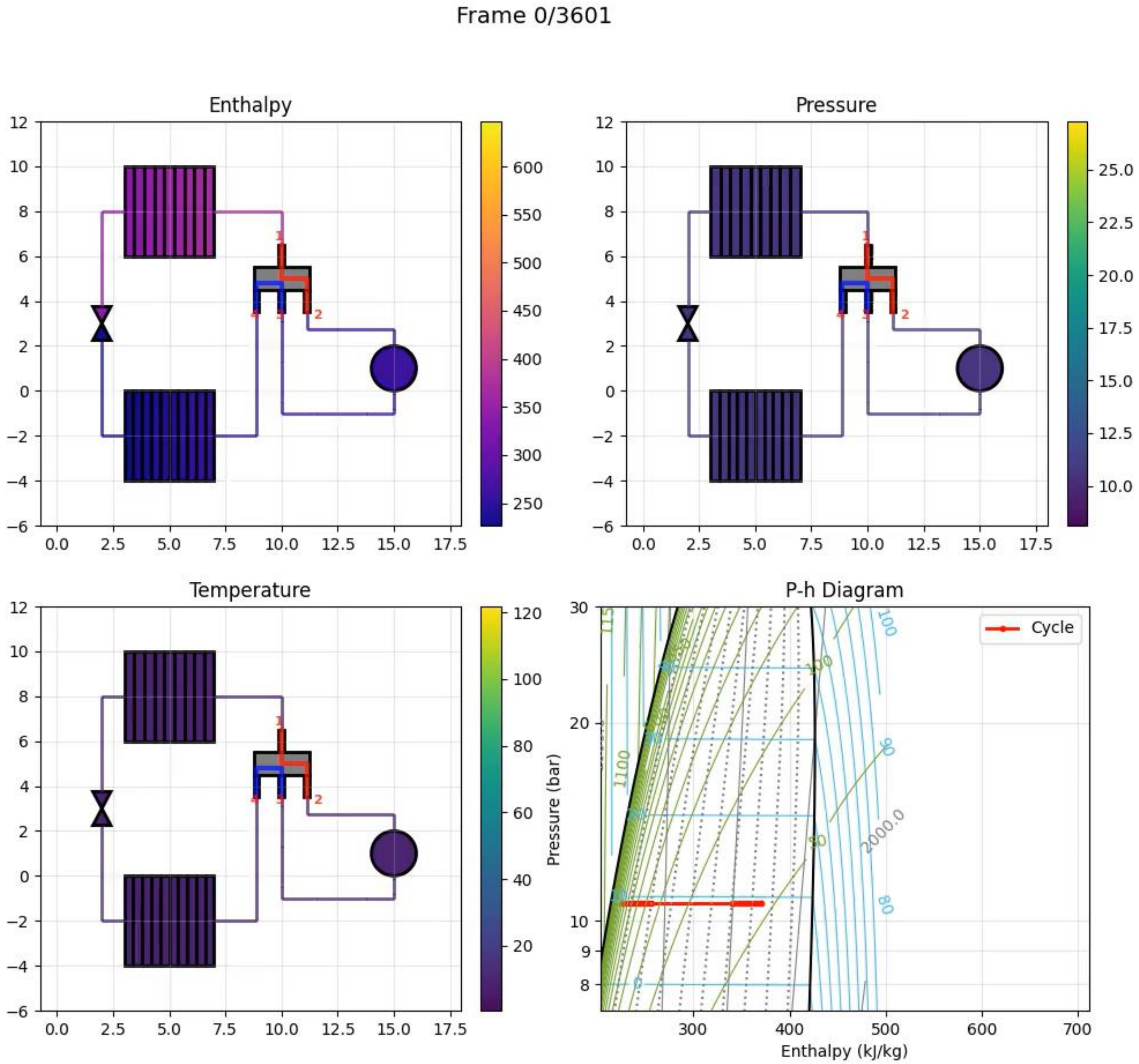
system_accubphx.yaml 805 B

```yaml
components:
  accumulator:
    type: Accumulator
    origin: [0, 0]
    diameter: 6.5
    height: 14
    dish_height: 1.6
    standoff_length: 4.0
    standoff_diameter: 1.8
  bphx:
    type: BPHX
    origin: [-8, 20]
    width: 3
    height: 5
    n_segments: 5
    flow_angle: 0
    inlet1_at: start
    inlet2_at: end

connectors:
  - name: accu_to_bphx
    start: accumulator.vapout1
    end: bphx.in2
    shape: u-bend
    open_to: B
    linewidth: 3
```

Python implementation







Repositories

- Fizz: useful helper functions for CO2 analysis (P-h diagram, valve flow rates, plotting etc.
  - https://gitlab.cern.ch/vbhanot/fizz
- Glean
  - Python: https://gitlab.cern.ch/vbhanot/glean
  - Matlab: https://gitlab.cern.ch/CO2/glean

- Auto-scan/Data-analysis: Only on Swan (CERN’s Jupyter running thingy with access to NXCals). Can share if you would like!

Reflections

Matlab

- Slow
- One function per file = difficult to organize code
- Everything is a matrix = easy but limiting
- Array manipulation is world class (⇒ performance penalty)
- <3 backslash operator

Python

- Slow
- Many datatypes to learn (lists, tuples, dicts, dicts-of-dicts, np arrays, pandas…)
- Easier to organise code. Modules are very useful
- Numpy syntax is ugly.
- Array manipulation less trivial
- Matplotlib is amazing

Thank You.