A Generic Fenics-Framework For Moment Approximations of Boltzmann equation

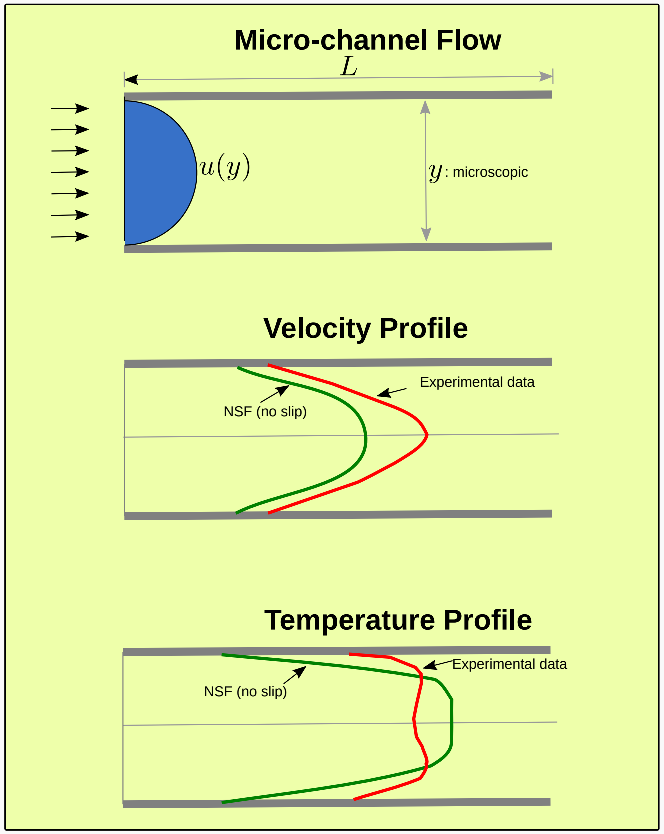
Edilbert Christhuraj and Prof. Manuel Torrilhon

Applied and Computational Mathematics
RWTH Aachen
Germany

FEniCS 2021

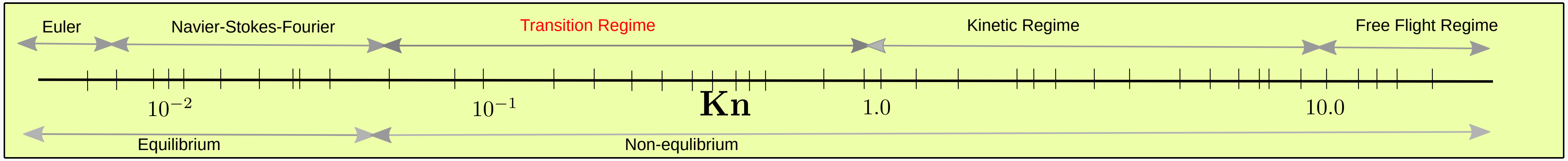
Setting The Scene

1. Need for new models
   - We want to model non-equilibrium gas flows.
   - E.g. consider a gas flow with temperature θ and velocity u.
     - Gas velocity follows *Stokes* problem, $\nabla \cdot \mathbf{u} = 0, \quad \nabla p = \nabla \mathbf{u}$.
     - Gas temperature follows *Poisson* problem, $\Delta\theta = 0$
   - Classical models use equilibrium assumptions in their closure equations. E.g.
     - Fourier's heat condution law $q \sim \nabla\theta$
     - Stoke's law $\sigma \sim (\nabla \mathbf{u})_{\mathrm{dev}}$
   - In non-equilibrium scenarios, these models are inadequate. (E.g., see micro-channel flow example)



2. **Kn**udsen number
   - is a dimensionless quntity in gas kinetic theory.
   - is defined as $\frac{\text{Mean free path}}{\text{Characteristic length scale}} = \frac{\lambda}{L}$.
   - is a good indicator of gas flow characterisation.
   - When **Kn** is large,
     - it implies non-equilibrium gas flow.
     - λ is very large, e.g. atmospheric entry flows, vacuum devices, or L is very small, e.g. mico-electro-mechanical devices.

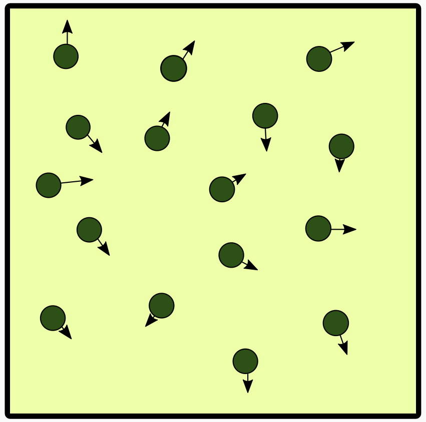
Find a continuum mechanics theory based on a system of partial differential equations to describe non-equilibrium gas flows accurately.



Theoritical Foundation In A Nutshell

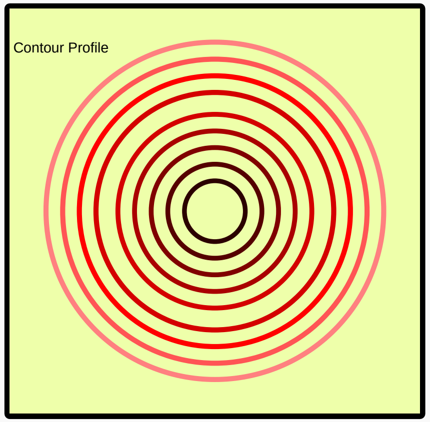
- Gas kinetic theory approach
- Gas as collection of particles
- Each particle has attributes
  - mass m
  - position $\mathbf{x}$
  - velocity $\mathbf{c}$
- Computationally expensive

Microscopic Setting



Continuum Limit

Macroscopic Setting

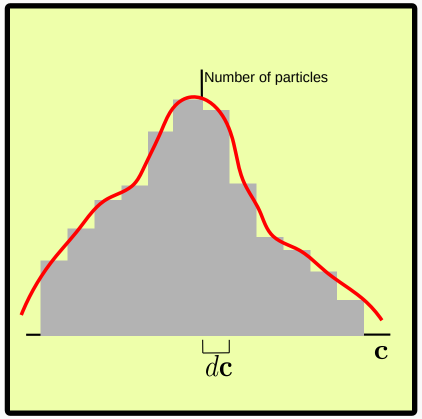


- Continuum equations
- Macroscopic variables
  - fluid velocity $\mathbf{u}$
  - temperature θ
  - density ρ
- Typically, partial differential equations with boundary conditions

Statistics

Moment Method

- Probability density function $f(\mathbf{x}, \mathbf{c}, t)$
- Equilibrium distribution $f^{(\mathrm{eq})}(\mathbf{c}) = \frac{\rho/m}{\sqrt{2\pi\theta}^3}\exp(\frac{(c_i-u_i)^2}{2\theta})$
- Boltzmann equation $\frac{\partial f}{\partial t} + c_i\frac{\partial f}{\partial x_i} = Q(f, f)$



Mesoscopic Setting

1. Moment method is an approximative method to Boltzmann equation
2. It connects microscopic setting to macroscopic setting through statiscal description

Moment Method - An Overview

1. Choose a constant background process with reference values of our choice.
2. Approximate the distribution function f by

$$\tilde{f} \approx \sum_{\mathcal{I}} w_{\mathcal{I}} \psi_{\mathcal{I}}(\mathbf{c}) f_r(\mathbf{c}; \rho_r, \theta_r, u_i^{(r)}), \qquad (1)$$

   where

   - $\mathcal{I}$ is an index set,
   - $w_{\mathcal{I}}$ are the coefficients,
   - $\psi_{\mathcal{I}}$ are the associated legendre polynomials,
   - f_r is a reference maxiwellian (pseudo-equilibrium).

3. Relations between moment variables $w_{\mathcal{I}}$ and macroscopic variables are possible:

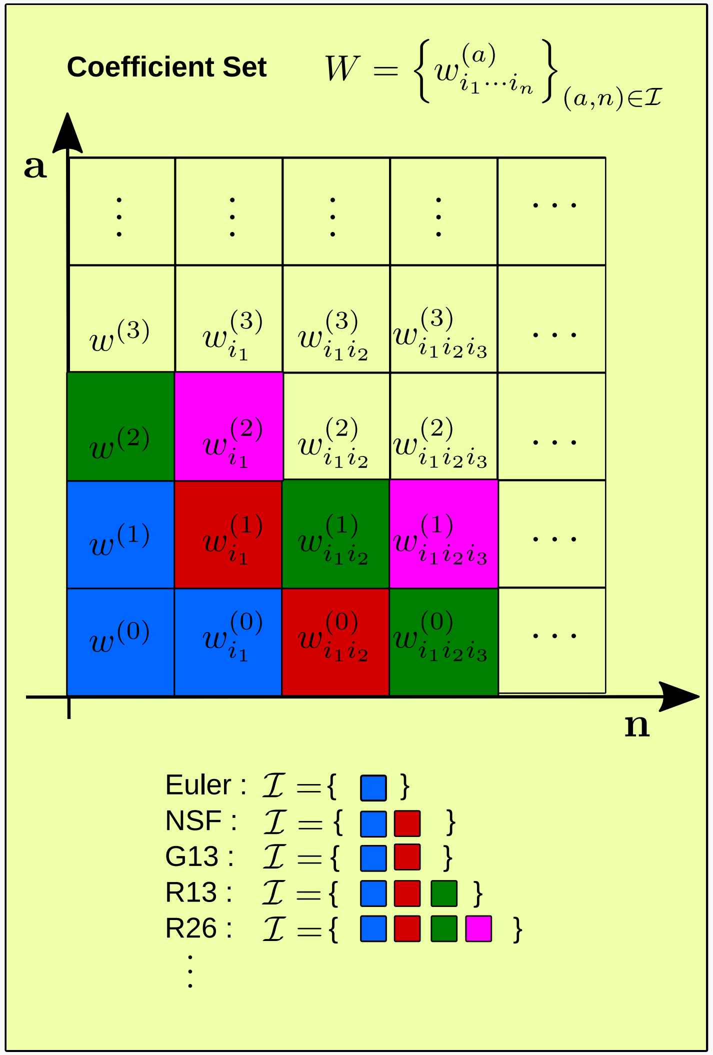
$$\left\{ w^{(0)}, w^{(0)}_{i_1}, w^{(1)}, w^{(0)}_{i_1 i_2} w^{(1)}_{i_1}, \cdots \right\} \Longleftrightarrow \{\rho, u_{i_1}, \theta, \sigma_{i_1 i_2}, q_{i_1}, \cdots\}.$$

4. Galerkin-type orthogonal projection on the Boltmann equation leads to moment equations.

$$\int_{\mathbb{R}^3} \bullet \left(\frac{\partial \tilde{f}}{\partial t} + c_i \frac{\partial \tilde{f}}{\partial x_i} \right) \mathrm{d}c = \int_{\mathbb{R}^3} \bullet \; S(\tilde{f}, \tilde{f})\, \mathrm{d}c,$$

   where $\bullet$ is any suitable test function.

5. Choose different ansatz and index set $\mathcal{I}$ to obtain different moment models.



A Generic First order Formulation

- Toy example: Consider the following Poisson equation

$$-\Delta\theta = f \quad \longrightarrow \quad \begin{matrix} \nabla \cdot q = f \\ \nabla\theta = -q \end{matrix} \quad \longrightarrow \quad \begin{matrix} \partial_x q_x + \partial_y q_y = f \\ \partial_x \theta = -q_x \\ \partial_y \theta = -q_y \end{matrix}$$

- The above system can be written as a system of first order equations.

$$W = \begin{pmatrix} \theta \\ q_x \\ q_y \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \partial_x \begin{pmatrix} \theta \\ q_x \\ q_y \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \partial_y \begin{pmatrix} \theta \\ q_x \\ q_y \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \theta \\ q_x \\ q_y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

- **Key idea**: Formulate moment system as first order system. In two dimensions (x, y), a system of m moment variables reads

$$A^{(x)}\, \partial_x W + A^{(y)}\, \partial_y W + P\, W = F,$$

where

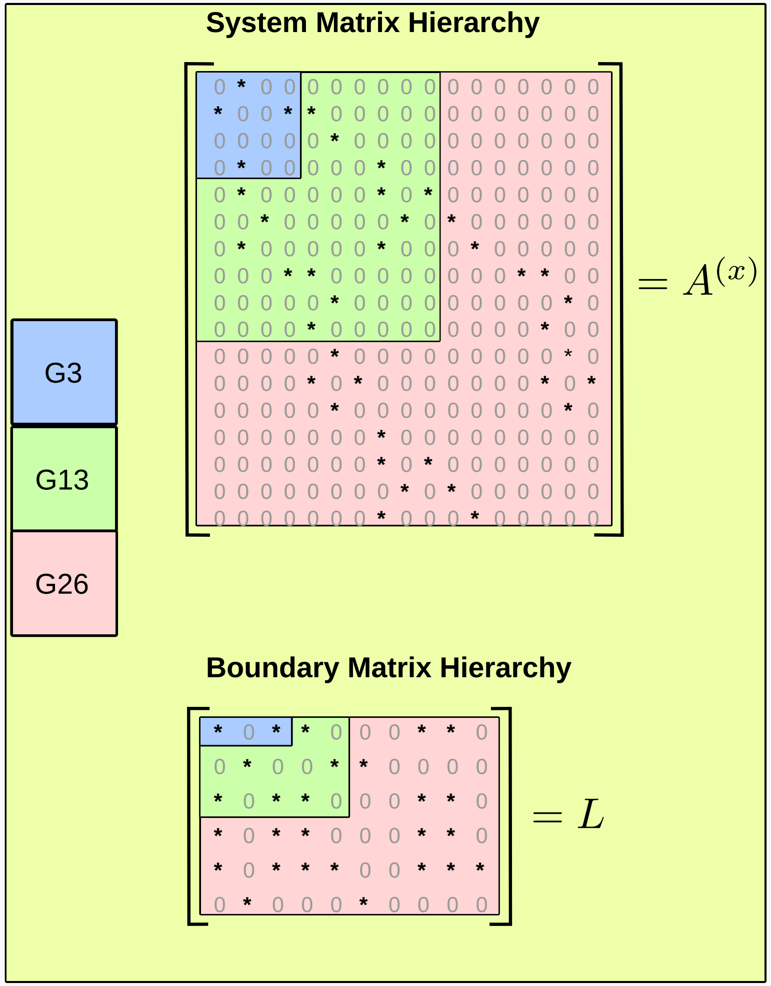
  - $U \in \mathbb{R}^m$ is the vector of m scalar moment variables,
  - $A^{(i)} \in \mathbb{R}^{m\times m}$ is the transport matrix in i^{th} direction,
  - $P \in \mathbb{R}^{m\times m}$ is the coefficient matrix of collision terms,
  - $F \in \mathbb{R}^m$ is the vector of external forces.

- Corresponding boundary conditions read

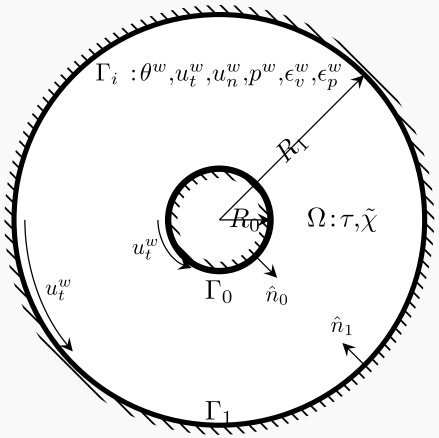
$$W^{(o)} = L\mathbb{A}\, W^{(e)} + g,$$

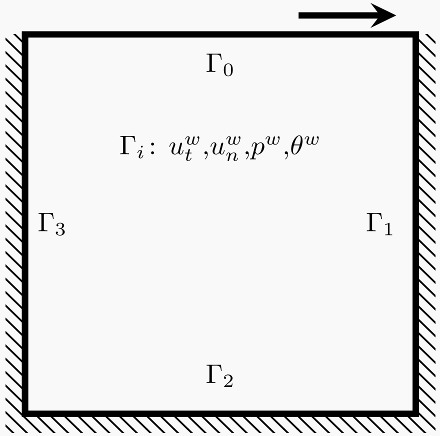
where

  - $W^{(o)} \in \mathbb{R}^p$ is the vector of odd moments,
  - $L \in \mathbb{R}^{p\times q}$ is the given boundary matrix,
  - $W^{(e)} \in \mathbb{R}^q$ is the vector of even moments,
  - $g \in \mathbb{R}^p$ is the vector of inhomogeneities.
  - $\mathbb{A} \in \mathbb{R}^{p\times q}$ is defined in the slide 7



Problem Setup - User's Perspective





Process	:	Flow over a cyliner
Quantities of interest	:	Velocity $\mathbf{u}$, Temperature θ, Heat flux $\mathbf{q}$
Geometry boundary	:	inner cylinder Γ_0, outer cylinder Γ_1

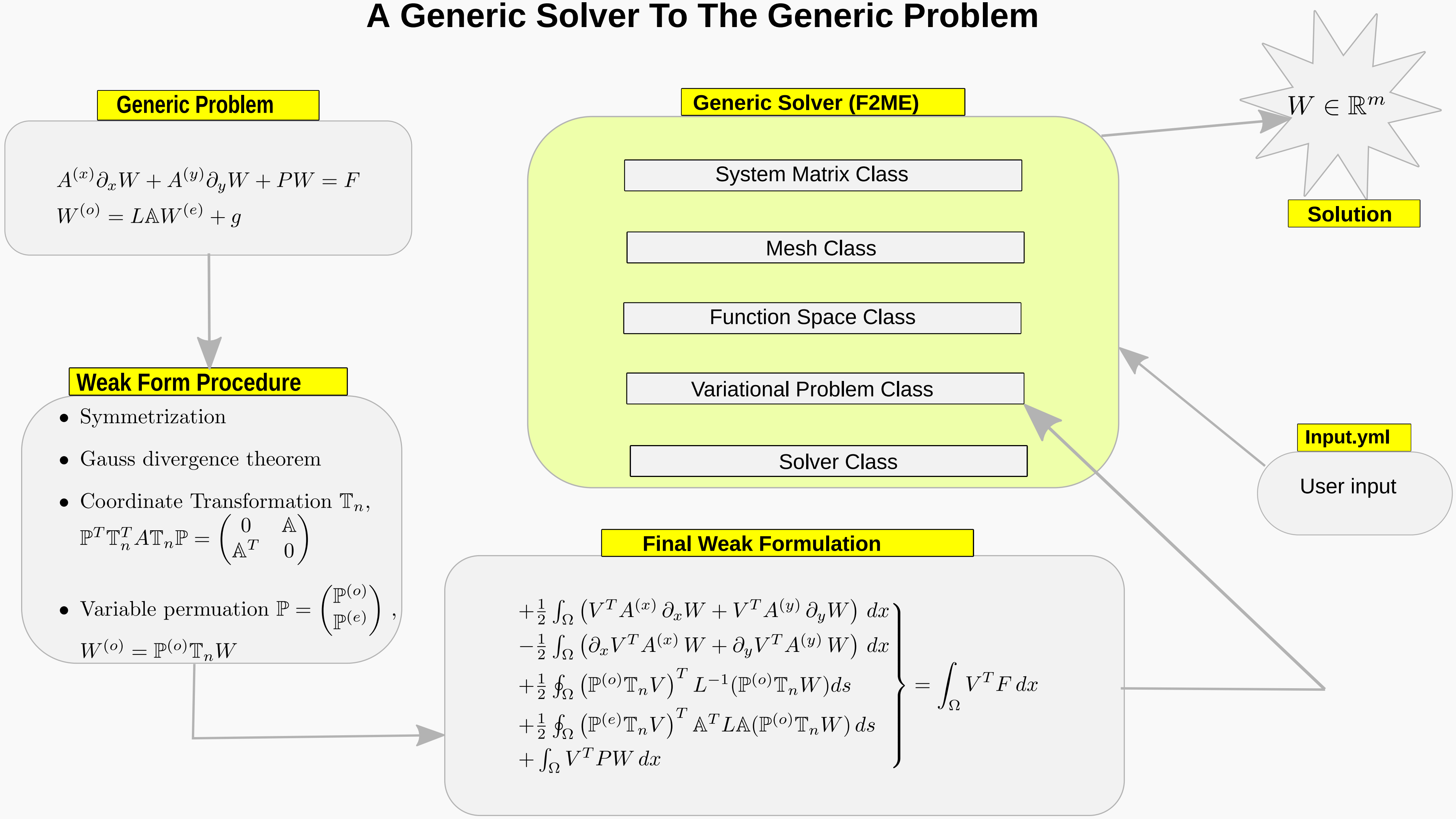
Process	:	Lid driven cavity
Quantities of interest	:	Velocity $\mathbf{u}$, Temperature θ, pressure $\mathbf{p}$
Geometry boundary	:	moving wall Γ_0, stationary walls $\Gamma_1, \Gamma_2, \Gamma_3$

Typical questions a user may ask:
Given some values, e.g., pressure p, inflow, heat flux $\mathbf{q}$, tangential velocity u_t, on a boundary Γ_i,
what is the velocity distibution?
what is the temperature distribution ?
⋮

1. Processes are equipped with **Kn** number.
   - When **Kn** is low, use classical models, e.g., NSF.
   - When **Kn** is (moderately) large, no need to solve full Boltzmann. Instead, solve moment models.

   Choose a model so that it is accurate enough to describe the process.
2. Prescribe desired values on different boundaries.

A Generic Solver To The Generic Problem



F2ME

Name — **F**enics **F**or **M**oment **E**quations , hence the abbreviation **F2ME**

Location — *https://www.gitlab.com/19ec94/f2me*
or
https://git.rwth-aachen.de/19ec94/f2me

Usage

```
$ git clone git@gitlab.com:19ec94/f2me.git
$ cd f2me/f2me
$ python3 f2me.py input.yml
```

Features

- Built upon FEniCS
- Generalised solver
- Highly modularised
- Easily extendable to other moment models
- Easily customisable to needs of an user
- Docker support available

f2me.py (main file)

Import custom modules / Read user input / Mesh using Gmsh / VectorFunctionSpace() / Prepare system matrices / Create & assemble system / Inbuilt solver (mumps)

```python
...
import dolfin as df
...
df.parameters['ghost_mode']= 'shared_facet'

#Import custom modules
from modules.SystemMatrix import SystemMatrix
from modules.Mesh import H5Mesh
from modules.FunctionSpace  import FunctionSpace
from modules.FormulateVariationalProblem import \
        FormulateVariationalProblem
from modules.Solver import Solver

# Read and Store user input
if len(sys.argv) >1 :
    user_given_input_file= sys.argv[1]
else:
    print("Provide an input file")
    quit()
with open(user_given_input_file,'r') as input_file:
    my_input_cls = yaml.load(input_file, \
            Loader=yaml.FullLoader)

# Main code starts
for current_mesh in range(len(my_input_cls["mesh_list"])):
    my_current_mesh = my_input_cls['mesh_list'][current_mesh]

    # Read and Process MESH info
    my_mesh_cls = H5Mesh(my_current_mesh)

    # Set up FUNCTION SPACE
    problem_type = my_input_cls['problem_type']
    number_of_moments = my_input_cls['number_of_moments']
    my_function_space_cls = FunctionSpace(my_input_cls,\
            my_mesh_cls)
    my_function_space_cls.set_function_space()

    # Create SYSTEM MATRIX
    my_system_matrix_cls = SystemMatrix(my_input_cls,\
            my_mesh_cls)
    my_system_matrix_cls.convert_to_ufl_form()

    # Create and Assemble VARIATIONAL PROBLEM
    my_var_prob_cls = FormulateVariationalProblem(
        my_input_cls, my_mesh_cls,
        my_system_matrix_cls, my_function_space_cls
        )
    my_var_prob_cls.create_lhs() #Left Hand Side
    my_var_prob_cls.create_rhs() #Right Hand Side

    # Call SOLVER
    my_solver_cls = Solver(my_input_cls,\
            my_function_space_cls, my_var_prob_cls)
    if problem_type == 'nonlinear':
        my_solver_cls.inbuilt_newton_solver()
        u = my_solver_cls.u
    else:
        my_solver_cls.inbuilt_linear_solver()
        u = my_solver_cls.u_Function

    # Start POST-PROCESSING
    sol = u.split()
```

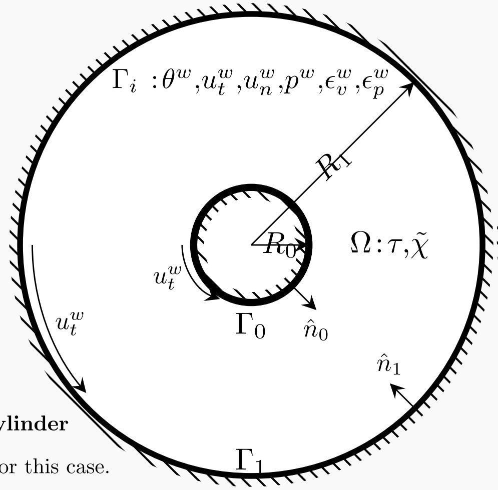
Simulation Results

input.yml

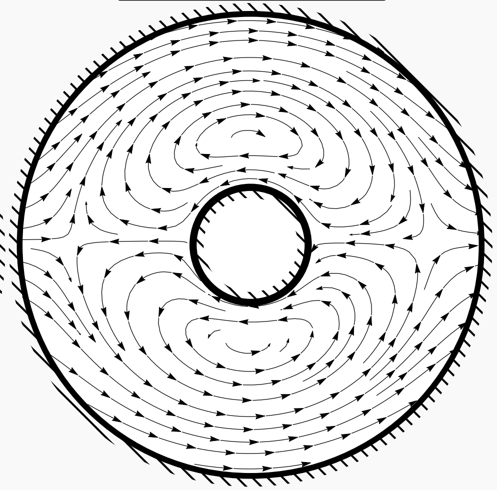
Model specific · Problem specific · Stabilisation · Problem specific · Solver specific · Model specific · Problem specific

```
problem_type: 'linear' # 'linear' or 'nonlinear'
moment_order: 13 # 3 or 6 or 13 or 'grad13' or 'ns'
number_of_moments: 17 # 6 or 9 or 10 or 17
mesh_list: #Provide mesh list
  - ../mesh/ring0.h5
  - ../mesh/ring1.h5
  - ../mesh/ring2.h5
  - ../mesh/ring3.h5
  - ../mesh/ring4.h5
  - ../mesh/ring5.h5
  - ../mesh/ring6.h5
stabilization: # Set FEM stabilisation paramters
  enable: True
  stab_type: cip
  cip:
    DELTA_T: 1.0
    DELTA_P: 0.01
    DELTA_U: 1.0
    ht: 3
    hp: 3
    hu: 3
  gls:
    h_power: 1
    const: 10
Kn: 0.1 # Knudsen number
Ma: 0.0 # Mach number
newton_abs_tol: 0.00 # Solver paramters
newton_rel_tol: 0.0
newton_max_itr: 0
newton_relaxation_parameter: 0.0
chi: 1.0
epsilon_w: 1.0
bc: # Boundary conditions
  3000:
    theta_w: 1.0
    u_t_w: -10.0 * sin(phi)
    u_n_w: 0.0
    p_w: 0.0

  3100:
    theta_w: 0.5
    u_t_w: -1.0 * sin(phi)
    u_n_w: +1.0 * cos(phi)
    p_w: -0.27 * cos(phi)
```



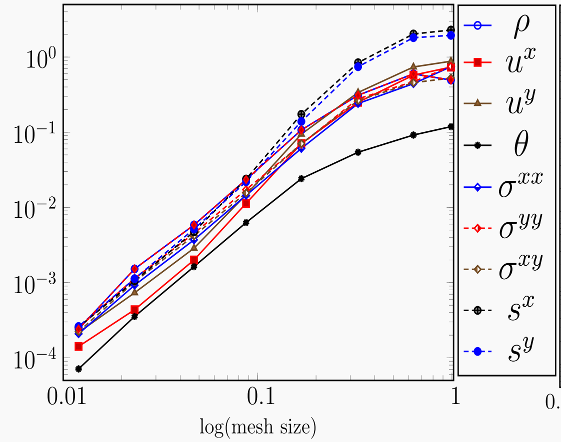
Velocity Profile



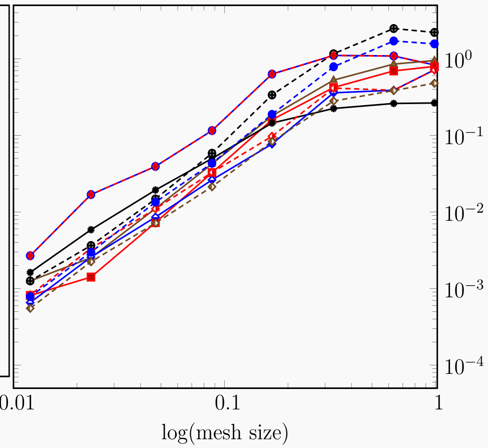
Moment System R13

- Test case: **Flow over a cylinder**
- Analytical solutions exist for this case.
- Convergence tests are performed.
- Order of convergence is found to be between first and second order for all variables.

Normalised L^2 Errors



Normalised L^∞ Errors



Simulation Results

input.yml

- Model specific
- Problem specific
- Stabilisation
- Problem specific
- Solver specific
- Model specific
- Problem specific

```yaml
problem_type: 'nonlinear' # linear or  nonlinear
moment_order: 'grad13' # 6 or 9 or 13 or 'grad13' or 'ns'
number_of_moments: 9 # 6 or 9 or 10 or 17
mesh_list: # Provide mesh list
  - ../mesh/lid08.h5
stabilization: # Set FEM stabilisation paramters
  enable: True
  stab_type: gls
  cip:
    DELTA_T: 1.0
    DELTA_P: 0.01
    DELTA_U: 1.0
    ht: 3
    hp: 3
    hu: 3
  gls:
    h_power: 1
    const: 10
Kn: 0.00001 # Knudsen number
Ma: 0.01 # Mach number
newton_abs_tol: 0.001 # Solver paramters
newton_rel_tol: 0.01
newton_max_itr: 5000
newton_relaxation_parameter: 0.3
chi: 1.0
epsilon_w: 0.0000001
bc: # Boundary conditions
  3000:
    theta_w: 0.0
    u_t_w: -1.0
    u_n_w: 0.0
    p_w: 0.0

  3100:
    theta_w: 0.0
    u_t_w: 0.0
    u_n_w: 0.0
    p_w: 0.0
```

Moment System G13

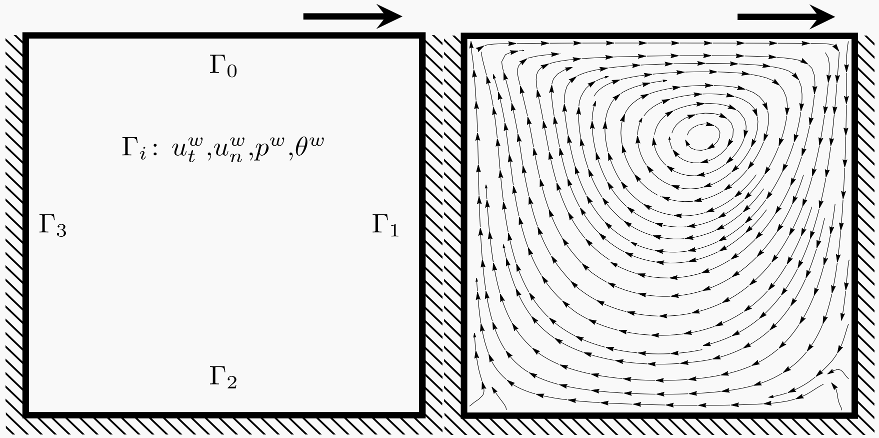
* Test case: **Lid driven cavity**

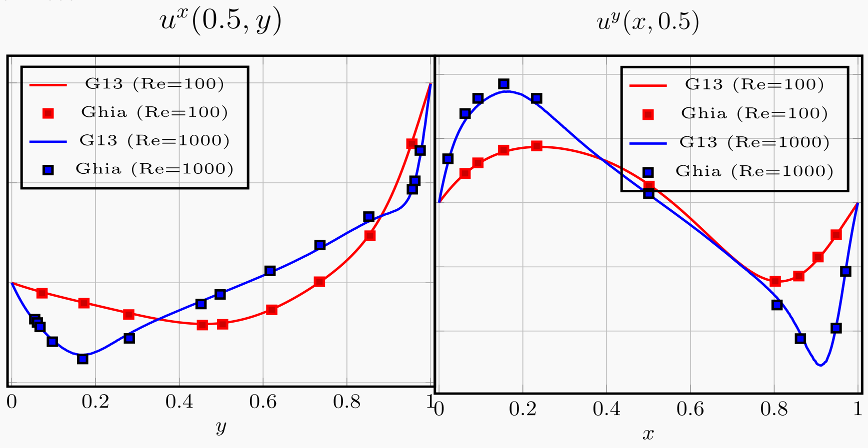
* Comparison with Ghia et al. is performed.

* Results are presented for two different Reynolds numbers (Re), namely Re=100 and Re=1000.

* G13 results are in good agreement with the results presented in Ghia's paper.

Velocity Profile





Outlook

Summary

- Non-equilibrium gas flows
- Kinetic picture - Boltzmann equation
- Moment approximation to Boltzmann equation
- A generic solver to generic moment equations (Many moment models but one solver for all)

What is next ?

- Extend to other moment models
- Include more realistic physics such as body force, gravity
- Make some parameters space dependent
- Implement model refinement

⋮