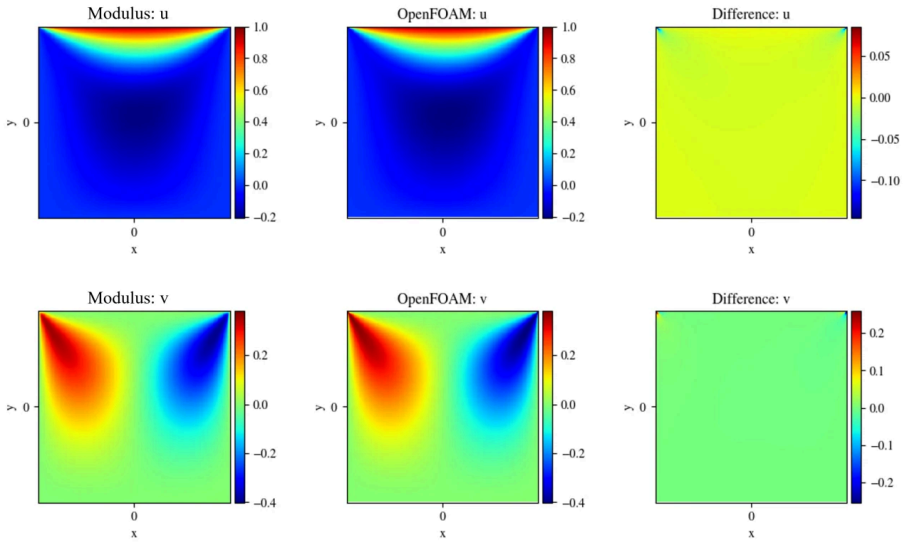
NVIDIA Modulus Part 3: Turbulent physics: Zero Equation Turbulence Model

Part 2: Introducing Lid Driven Cavity (LDC)Example

https://docs.nvidia.com/deeplearning/modulus/modulus-sym/user_guide/basics/lid_driven_cavity_flow.html

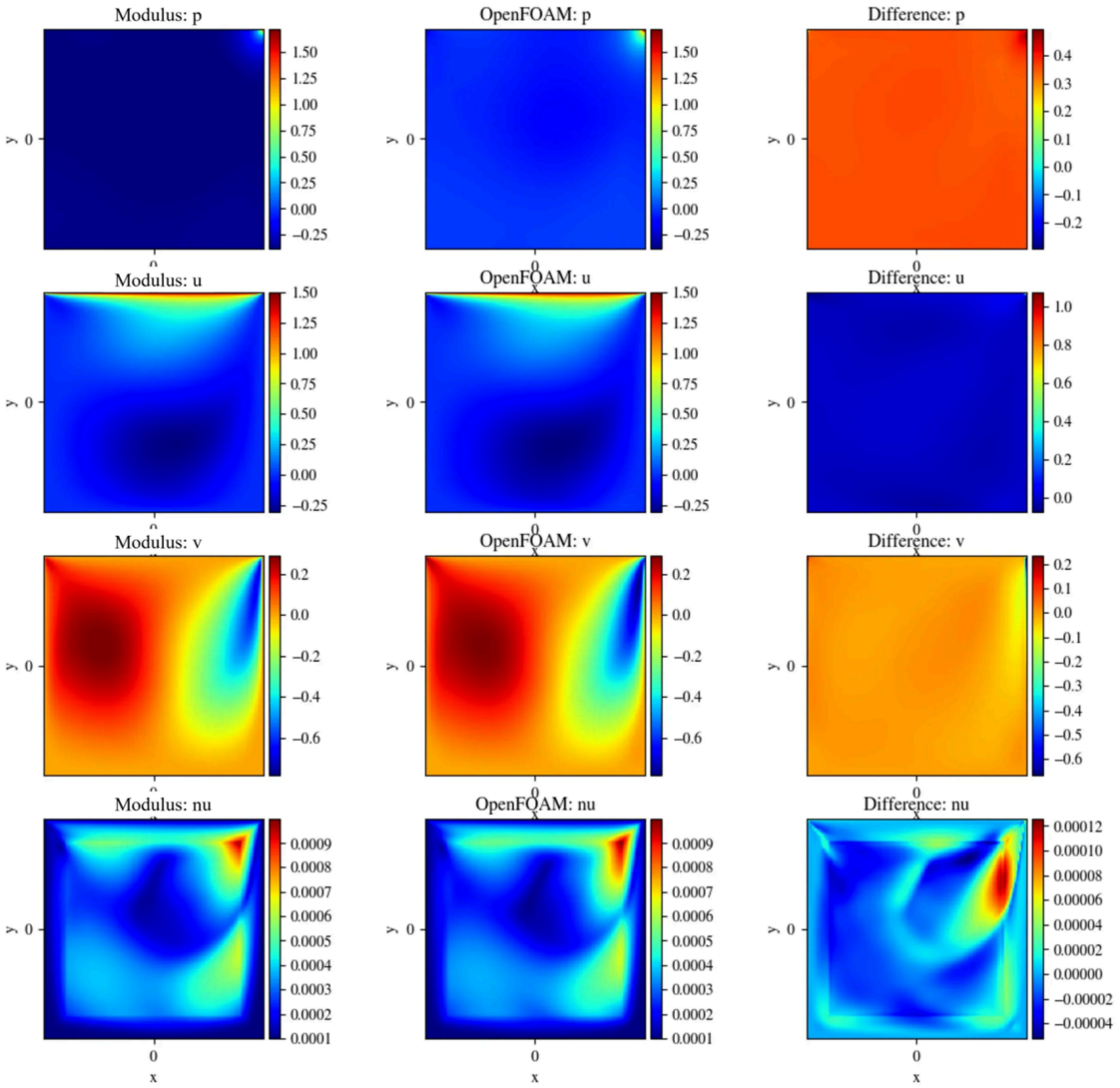


Part 3: Turbulent physics: Zero Equation Turbulence Model

- **I**ntegrating the **Zero Equation turbulence model** into a **Lid-Driven Cavity (LDC) flow**
- Code walkthrough
- Visualize outputs

  https://github.com/AI-ME-Ben/NVIDIA-Modulus-Reproduce/tree/main

  https://catalog.ngc.nvidia.com/orgs/nvidia/teams/modulus/resources/modulus_sym_examples_supplemental_materi



Part 1: Integrating the Zero Equation turbulence model into a Lid-Driven Cavity (LDC) flow

- The previous **LDC case had Re = 10**, but here it's increased to **1000**.
- The model computes **effective viscosity** using the mean strain rate tensor.

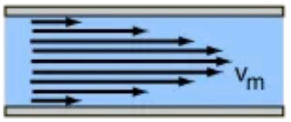
1. Reynolds Number Formula

$$Re = \frac{UL}{\nu}$$

where:

- U = characteristic velocity
- L = characteristic length
- ν = molecular (kinematic) viscosity

This equation shows that Re is directly affected by molecular viscosity (ν), but it does not include turbulent viscosity (ν_t).



Laminar flow, Reynolds low

Turbulence, Reynolds high

1. Standard 2D Steady-State Navier-Stokes Equations (Incompressible)

For an incompressible flow ($\nabla \cdot \mathbf{u} = 0$):

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

2. Modified Navier-Stokes Equations with Effective Viscosity

In turbulence models, we replace **molecular viscosity** ν with **effective viscosity** ν_{eff}:

$$\nu_{\text{eff}} = \nu + \nu_t$$

where:

- ν = molecular kinematic viscosity
- ν_t = turbulent viscosity (depends on turbulence model)

Now, the modified equations become:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \nu_{\text{eff}}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \nu_{\text{eff}}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

where ν_{eff} accounts for both **laminar and turbulent** diffusion effects.

Understanding Effective Viscosity in the Zero Equation Turbulence Model

The **Zero Equation Turbulence Model** (also called the **Algebraic Turbulence Model**) estimates turbulence effects using an **effective (turbulent) viscosity** ν_{eff}. Instead of solving additional transport equations (like in **k-epsilon** or **k-omega models**), it computes turbulent viscosity directly using local velocity gradients and wall distances.

1. Definition of Effective Viscosity

In the `ZeroEquation` class, the **effective viscosity** ν_{eff} is computed as:

$$\nu_{\text{eff}} = \nu + \rho \cdot l_m^2 \cdot \sqrt{G}$$

where:

- ν = **kinematic viscosity** (molecular viscosity of the fluid)
- ρ = **density** of the fluid
- l_m = **mixing length** (describes turbulence scale, dependent on wall distance)
- G = **strain rate magnitude**, which represents turbulence production

2. Breakdown of Key Components

(A) Mixing Length l_m

The **mixing length model** is a simplified approach to turbulence modeling. It assumes that the turbulent viscosity is proportional to the velocity gradient and a characteristic length scale. The mixing length is given by:

$$l_m = \min\left(\kappa d, \beta d_{\max}\right)$$

where:

- $\kappa = 0.419$ (Von Kármán constant, empirical value)
- d = **normal distance to the nearest wall** (function `"sdf"` in code)
- $\beta = 0.09$ (scaling factor for maximum distance)
- $d_{\max}$ = **maximum reference distance** in the flow field

This equation ensures that the mixing length **scales correctly near walls** and does not grow indefinitely in free shear flows.

- **Near walls**: l_m is **small** due to the wall distance constraint, leading to **lower turbulent viscosity**.
- **Away from walls**: l_m can grow, allowing **higher turbulent viscosity**, capturing the effects of larger eddies.
- **In free shear flows** (e.g., mixing layers, jets): The model adapts dynamically using velocity gradients.

(B) Strain Rate Magnitude G

The **strain rate magnitude** represents how fast the velocity gradients change in a given region, and it plays a crucial role in turbulence generation.

$$G = 2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + 2\left(\frac{\partial w}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)^2$$

This formula sums the squares of all velocity gradients, effectively capturing **how much "shearing" is present** in the flow.

- If **velocity gradients are large**, G increases, leading to a **higher turbulent viscosity**.
- If **velocity gradients are small**, G is low, and the flow behaves more like a **laminar flow**.

3. Physical Interpretation

The effective viscosity ν_{eff} is a combination of:

1. **Molecular viscosity** ν: Represents standard fluid resistance (important in laminar flows).
2. **Turbulent viscosity** $\nu_t = \rho l_m^2 \sqrt{G}$: Accounts for additional momentum transfer due to turbulence.

4. Advantages of the Zero Equation Model

✅ **No additional transport equations**: Unlike k-ε or k-ω, this model avoids solving extra PDEs.
✅ **Fast computation**: Since it relies only on algebraic calculations, it's suitable for real-time simulations.
✅ **Good for simple flows**: Works well for **boundary layer flows** and **shear-driven turbulence** (e.g., lid-driven cavity).
❌ **Limited accuracy**: Does not capture complex turbulence effects like separation or wake flows.

5. How Modulus Sym Uses This Model

In **Modulus Sym**, the `ZeroEquation` model is implemented as an additional **PDE constraint** in the computational graph:

```python
ze = ZeroEquation(nu=1e-4, max_distance=height / 2, dim=2, time=False)
```

- `nu=1e-4`: Specifies the **base kinematic viscosity**.
- `max_distance=height / 2`: Sets the reference maximum wall distance.
- `dim=2`: Configures for **2D turbulence**.
- `time=False`: Uses a **steady-state formulation**.

Once added, the solver ensures that the **effective viscosity** is computed at each point dynamically.

```python
from modulus.sym.eq.pdes.turbulence_zero_eq import ZeroEquation
```

https://help.autodesk.com/view/SCDSE/2019/ENU/?guid=GUID-BBA4E008-8346-465B-9FD3-D193CF108AF0
https://help.autodesk.com/view/SCDSE/2019/ENU/?guid=GUID-4148B3AE-8B8E-49AA-A604-1DAE92CD8D02
https://turbmodels.larc.nasa.gov/

Part 2: Code Walkthrough

Geometry and Governing Equations

- Governing Equations:
  - **Navier-Stokes Equations** (with turbulence model viscosity)
  - **Zero Equation Model** (for effective viscosity calculation)

Importing Required Packages

```
import os
import warnings
from sympy import Symbol, Eq, Abs
import torch
import modulus.sym
from modulus.sym.hydra import to_absolute_path, instantiate_arch, ModulusConfig
from modulus.sym.utils.io import csv_to_dict
from modulus.sym.solver import Solver
from modulus.sym.domain import Domain
from modulus.sym.geometry.primitives_2d import Rectangle
from modulus.sym.domain.constraint import PointwiseBoundaryConstraint, PointwiseInteriorConstraint
from modulus.sym.domain.validator import PointwiseValidator
from modulus.sym.eq.pdes.navier_stokes import NavierStokes
from modulus.sym.eq.pdes.turbulence_zero_eq import ZeroEquation
```

Defining Geometry

```
# Define cavity geometry
height, width = 0.1, 0.1
x, y = Symbol("x"), Symbol("y")
rec = Rectangle((-width / 2, -height / 2), (width / 2, height / 2))
```

Adding Turbulence Model

- The **Zero Equation Model** is instantiated to calculate the **effective viscosity (v_t)**.
- The kinematic viscosity for Navier-Stokes equations is replaced with the one computed by **ZeroEquation**.

```
# Define Zero Equation turbulence model
ze = ZeroEquation(nu=1e-4, dim=2, time=False, max_distance=height / 2)

# Define Navier-Stokes equations with turbulence model viscosity
ns = NavierStokes(nu=ze.equations["nu"], rho=1.0, dim=2, time=False)
```

Boundary Conditions

- **Top Wall:** Moving with constant velocity **u = 1.5, v = 0**.
- **Other Walls:** No-slip condition **u = 0, v = 0**.

```
# Create computational domain
ldc_domain = Domain()

# Top moving wall (Lid)
top_wall = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=rec,
```

```
    outvar={"u": 1.5, "v": 0},
    batch_size=cfg.batch_size.TopWall,
    lambda_weighting={"u": 1.0 - 20 * Abs(x), "v": 1.0},  # Reduce weight at edges
    criteria=Eq(y, height / 2),
)
ldc_domain.add_constraint(top_wall, "top_wall")

# No-slip walls
no_slip = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=rec,
    outvar={"u": 0, "v": 0},
    batch_size=cfg.batch_size.NoSlip,
    criteria=y < height / 2,
)
ldc_domain.add_constraint(no_slip, "no_slip")
```

Interior Constraints (Enforcing PDEs)

- The **continuity** and **momentum equations** must be satisfied in the domain.
- **Signed Distance Field (SDF)** is used for turbulence viscosity derivatives.

```
interior = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=rec,
    outvar={"continuity": 0, "momentum_x": 0, "momentum_y": 0},
    batch_size=cfg.batch_size.Interior,
    compute_sdf_derivatives=True,  # Required for turbulence viscosity gradients
    lambda_weighting={
        "continuity": Symbol("sdf"),
        "momentum_x": Symbol("sdf"),
        "momentum_y": Symbol("sdf"),
    },
)
ldc_domain.add_constraint(interior, "interior")
```

Validation Against OpenFOAM Data

https://catalog.ngc.nvidia.com/orgs/nvidia/teams/modulus/resources/modulus_sym_examples_supplemental_materials

```
file_path = "openfoam/cavity_uniformVel_zeroEqn_refined.csv"

if os.path.exists(to_absolute_path(file_path)):
    mapping = {
        "Points:0": "x",
        "Points:1": "y",
        "U:0": "u",
        "U:1": "v",
        "p": "p",
        "d": "sdf",
        "nuT": "nu",
    }
    openfoam_var = csv_to_dict(to_absolute_path(file_path), mapping)
    openfoam_var["nu"] += 1e-4  # Effective viscosity correction

    # Create validator
```

```
    openfoam_validator = PointwiseValidator(
        nodes=nodes,
        invar={k: v for k, v in openfoam_var.items() if k in ["x", "y", "sdf"]},
        true_outvar={k: v for k, v in openfoam_var.items() if k in ["u", "v", "nu"]},
        batch_size=1024,
    )
    ldc_domain.add_validator(openfoam_validator)
```

Running the Solver and Visualizing Results

Solver Setup

```
# Create solver
slv = Solver(cfg, ldc_domain)

# Run solver
slv.solve()
```

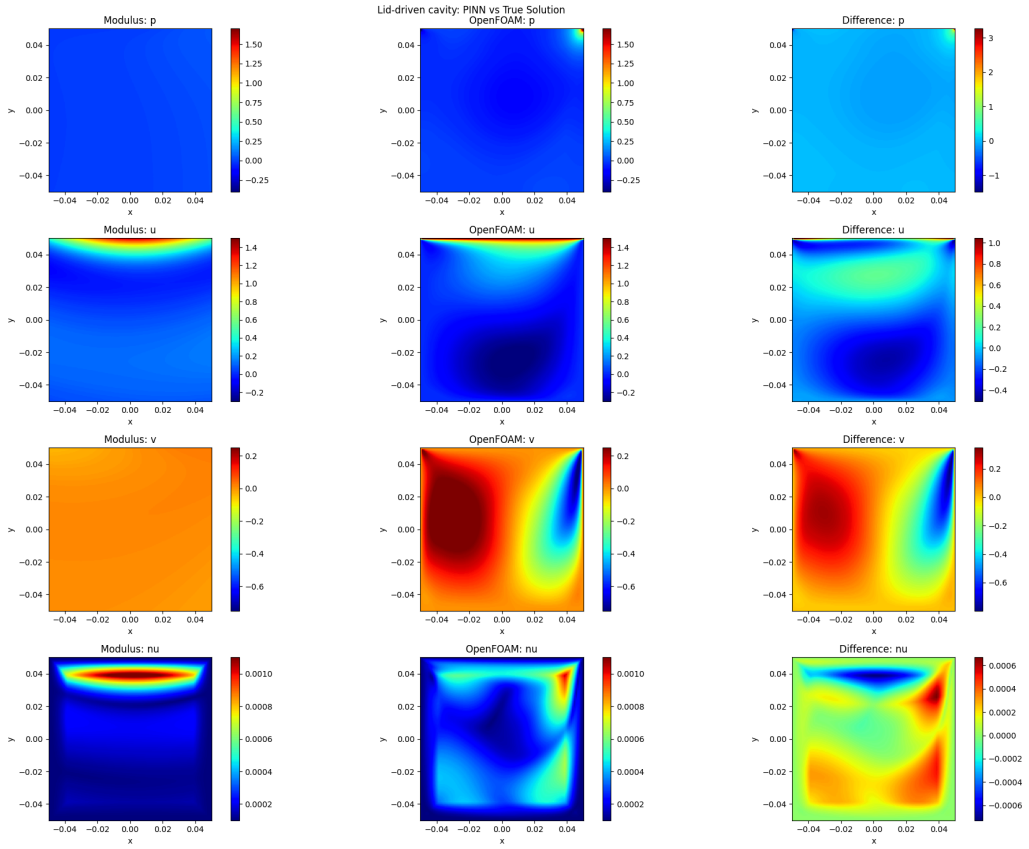
Part 3: Results Visualization

https://github.com/AI-ME-Ben/NVIDIA-Modulus-Reproduce/tree/main

- **Velocity and Pressure fields** are visualized from the inference domain.
- **Comparison with OpenFOAM** simulation results.

https://catalog.ngc.nvidia.com/orgs/nvidia/teams/modulus/resources/modulus_sym_examples_supplemental_materials

```
rec_validation_freq: 100
```



Summary

✅ **Defined the Zero Equation Turbulence Model** in Modulus Sym.

✅ **Set up the Lid-Driven Cavity Flow problem** with Re = 1000.

✅ **Implemented boundary conditions, domain constraints, and validation.**

✅ **Ran the solver and compared with OpenFOAM results.**