APC 524 Final Project:

Implementing a Navier-Stokes Solver and Physics Informed Neural Network for Simulating Two-Dimensional Fluid Flow Around a Cylinder

Fairuz Ishraque[1], Joseph Lockwood[1], and Aaron Spaulding[2]

[1]Department of Geosciences
[2]Department of Civil and Environmental Engineering

Derivation Implementation of NS Solver for 2D Cylinder Wake Flow

- Implemented finite-element Navier-Stokes (NS) solver.
- Focus on incompressible, viscous fluid flow.
- No-slip boundary conditions for real fluid behavior.
- Investigated flow separation, wake formation, and vortex shedding.

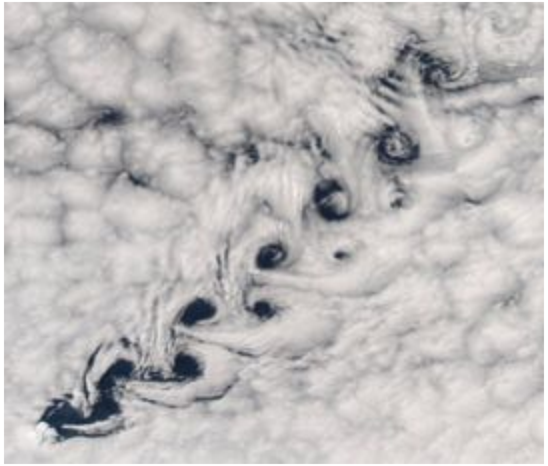


Fig: Real world wake formation NASA 2015.

Joseph's Contributions

Unit Testing: Ensuring Code Reliability

- Comprehensive Unit Tests
  - Robust suite of tests covering various aspects such as Class and method functionality
- Initialization Tests
  - Validates proper class initialization with default values
  - Ensures correct setup of the simulation environment
- Mechanics Validation
  - Validates internal mechanics of the Environment class
  - Essential for computational algorithm accuracy
- Verification and Validation
  - Verification: Code meets requirements and functions correctly
  - Validation: Requirements make sense and serve intended purpose
  - Enhances code reliability and effectiveness
- Adversarial Testing
  - Rigorous challenges to identify potential defects
  - Documentation of code expectations and requirements
  - Builds user confidence in code reliability

Joseph's Contributions

Speed and Efficiency Analysis

- **Profiling with cProfile**
  - Identifying performance bottlenecks
  - Gathering essential performance data
- **SnakeViz for Analysis**
  - Using SnakeViz, a graphical profiler viewer
  - Visualizing and decoding performance bottlenecks
  - Targeting areas for optimization
- **Benefits**
  - Pinpoint resource-intensive functions/methods
  - Informed optimization decisions

Implementation of a General Navier-Stokes Solver with the Finite Difference Method

- Implemented a customizable environment class that allows arbitrary environments and environmental conditions
- Defined modular boundary conditions allowing for many environment types to be investigated
- Created composable objects that allow for complex environments to be modeled and simulated

Aaron Spaulding's Contributions

Implementation of a General Navier-Stokes Solver with the Finite Difference Method

Environmental Variables:

- Spatial Resolution
- Temporal Resolution
- Density
- Kinematic Viscosity
- Environment Size

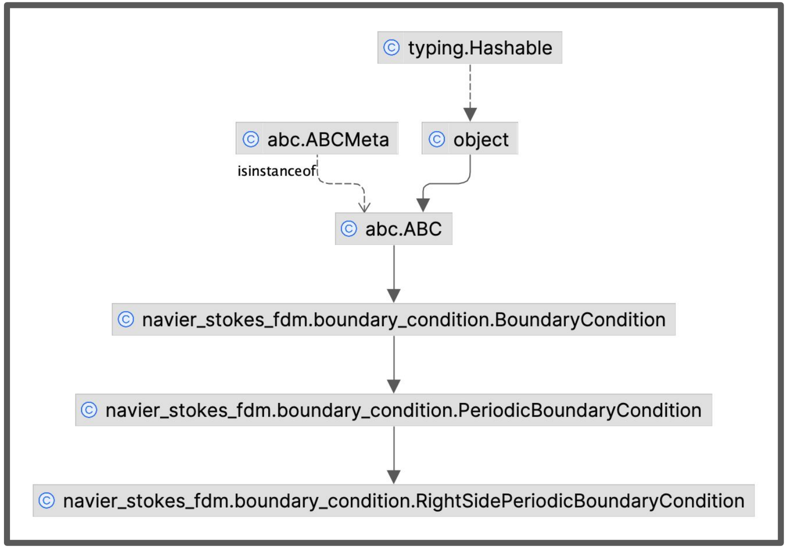
Boundary Conditions:

- No-Slip Boundary Condition:
- Fixed Velocity Boundary Condition
- Periodic Boundary Condition
- Free Slip Boundary Condition

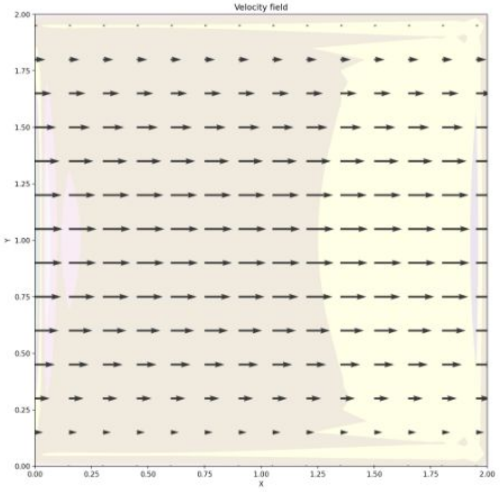
Implemented Objects:

- Rectangle
- Circle

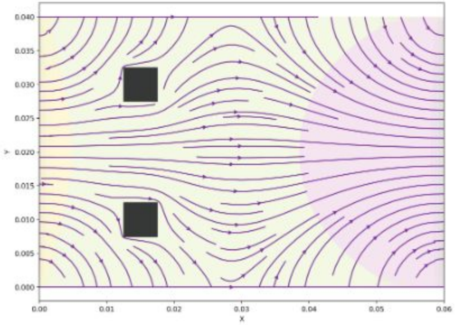
Inheritance Tree For the Right Side Periodic Boundary Condition



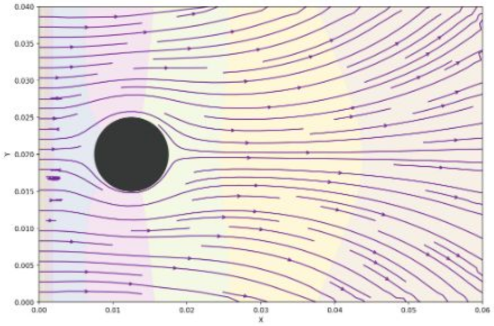
Aaron Spaulding's Contributions



(a) Simulation of fluid flowing in a pipe. The top and bottom boundary conditions are set as no-slip conditions, while the left and right are set as periodic boundary conditions.



(b) Simulation of a fluid flowing around two boxes. Boundary conditions for all sides are set as fixed velocity conditions. The boundary conditions for the boxes are set as no-slip conditions are updated dynamically as each box is added to the environment.



(c) A simulation of fluid flow around a cylinder. Here the right, top, and bottom sides have no slip conditions while the left side has a fixed velocity boundary condition. The boundary conditions around the cylinder are automatically updated at simulation time.

Figure 2: The modular boundary conditions, customizable environment, and composable objects allows for easy simulation of complex environments with very different conditions and requirements.

Aaron Spaulding's Contributions

Example Simulation Setup

Python Module Imports

Modular Boundary Conditions

Composable Objects

Customizable Environment

Automatic Plotting

```
from navier_stokes_fdm import Environment
from navier_stokes_fdm import Rectangle
import navier_stokes_fdm.boundary_condition as bc


U = 1  # m/s
dimension = 0.005

boundary_conditions = [
    bc.TopSideFixedVelocityBoundaryCondition(u_value=U, v_value=0),
    bc.BottomSideFixedVelocityBoundaryCondition(u_value=U, v_value=0),
    bc.LeftSideFixedVelocityBoundaryCondition(u_value=U, v_value=0),
    bc.RightSideFixedVelocityBoundaryCondition(u_value=U, v_value=0),
]


x1, y1 = 0.0125, (0.04 / 2) - (dimension / 2)
objects = [Rectangle(x1, y1, x1 + dimension, y1 + dimension)]


a = Environment(
    F=(1.0, 0.0),
    len_x=0.06,
    len_y=0.04,
    dt=0.00000015,
    dx=0.0001,
    boundary_conditions=boundary_conditions,
    objects=objects,
    rho=0.6125  # kg/m³
    nu=3e-5  # m²/s
)

a.run_many_steps(480)
a.plot_streamline_plot(title="", filepath="../Figures/box_example_streamline.png")
```

Aaron Spaulding's Contributions

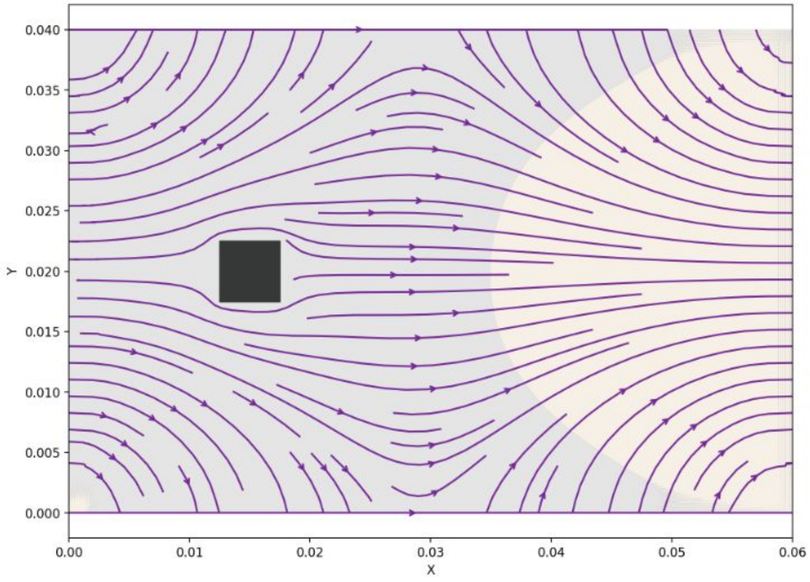
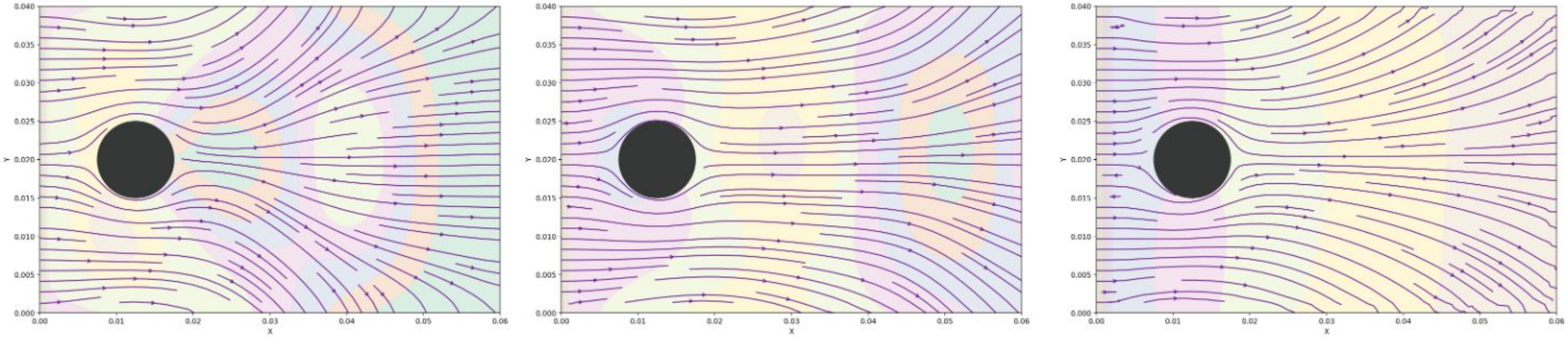


Figure 3: Example streamline plot of a fluid flow around a box. Each boundary is assigned a fixed velocity. Shading represents the pressure field with lighter colors indicating regions of lower pressure. Streamlines are shown in purple.

Aaron Spaulding's Contributions



(a) $9\mu s$ (b) $13.5\mu s$ (c) $45\mu s$

Figure 4: Three time steps of the FD simulation of fluid flow around a a cylinder. The pressure field is shown by the shading while streamlines are shown in purple.

Aaron Spaulding's Contributions

Implementation of a General Navier-Stokes Solver with the Finite Difference Method

- I implemented tests using “pytest” for boundary conditions.
- Implemented automatic testing with GitHub Actions to test every commit for changes that might break core functionality

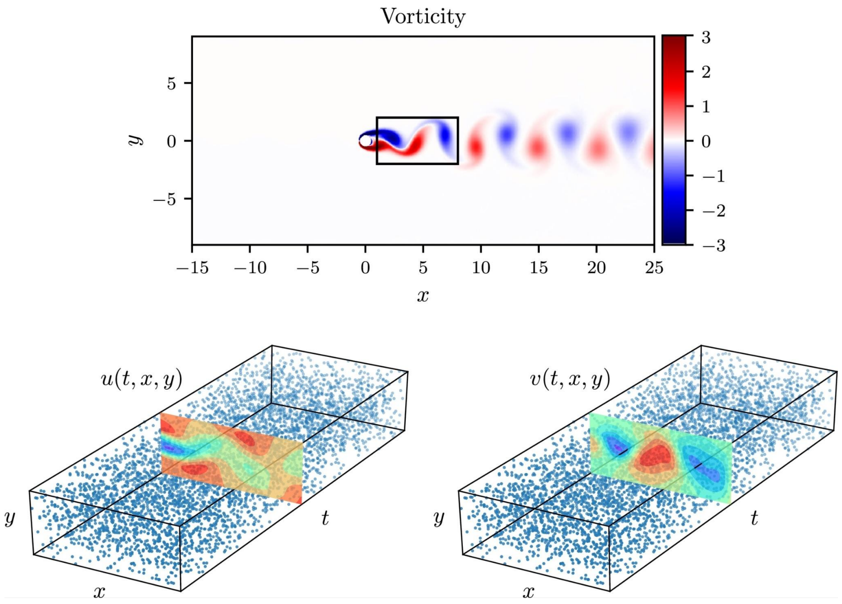
Aaron Spaulding’s Contributions

Implementation of a Physics Informed Neural Network as Navier-Stokes Solver

- Implemented a modular PINN class using OOP principles in Tensorflow2
- Implemented a Input-Output manager class for PINN to better handle training, testing, and prediction data
- Implemented a Plotting Manager to easily plot and save PINN predictions
- Wrote unit tests for every PINN functionality
- Implemented data save functionality in the numerical Navier-Stokes solver so the data from the simulation can be used for PINN training

Fairuz Ishraque's Contributions

Implementation of a Physics Informed Neural Network as Navier-Stokes Solver



Raissi et al. (2019)

PINN Model Training

- Already established simulation data of cylinder wake flow was used (from Raissi et al. 2019)
- Randomly selected 5000 data points (blue dots) to use for training
- Model was trained for 200000 iterations, but loss-curve analysis suggests 100000 should be enough

Fairuz Ishraque's Contributions

Example PINN Setup

Python Module Imports

Data Loading and Prep

Select and Prep Training Data

Model Initialization

Model Training

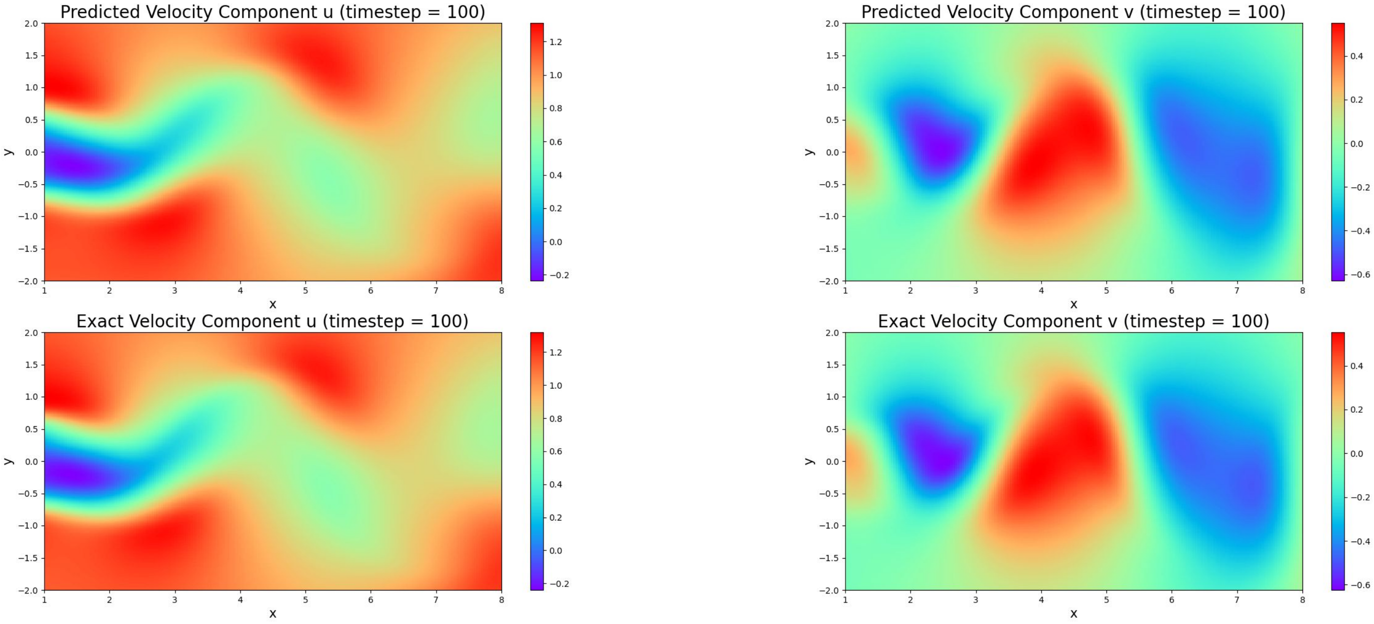
Model Inference and Save Data

Plot Model Predictions

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from navier_stokes_pinn.PINN import PhysicsInformedNN
from navier_stokes_pinn.input_output import NavierStokesPINN_IO
from navier_stokes_pinn.plotting import NavierStokesPINN_Plotter


IO_manager = NavierStokesPINN_IO("navier_stokes_pinn/data", "navier_stokes_pinn/output")
IO_manager.parse_data_file('cylinder_nektar_wake.mat')

# Selecting training data
IO_manager.select_training_data(N_train=5000)

# Extract training data from IO_manager
training_data = IO_manager.training_data

# Casting the training data into tensorflow
x_train = tf.cast(training_data['x_train'], dtype=tf.float32)
y_train = tf.cast(training_data['y_train'], dtype=tf.float32)
t_train = tf.cast(training_data['t_train'], dtype=tf.float32)
u_train = tf.cast(training_data['u_train'], dtype=tf.float32)
v_train = tf.cast(training_data['v_train'], dtype=tf.float32)

# Setting model architechture
layers = [3, 20, 20, 20, 20, 20, 20, 20, 20, 2]

# Setting Reynold's Number
Re = 100

# Initializing the PINN model
# Model training support TensorFlow 2 and GPU acceleration
model = PhysicsInformedNN(x_train, y_train, t_train, u_train, v_train, Re, layers)
# Train PINN model
model.train(2000, learning_rate=1e-3)
# Select test data at a time snapshot to test run inference and test the trained model
time_snapshot = 100
IO_manager.select_test_data(time_snapshot)
test_data = IO_manager.test_data
# Run inference and save predicted data
IO_manager.save_predict_data(model, 'example_prediction_100.npz')

# Plot the predicted data and save the plot with plotting class
Plot_manager = NavierStokesPINN_Plotter("navier_stokes_pinn/data", "navier_stokes_pinn/plots", IO_manager)
# Saves the u, v. and p values at the timestep 100
Plot_manager.plot_compare_predictions('example_prediction_100.npz', time_snapshot)
```

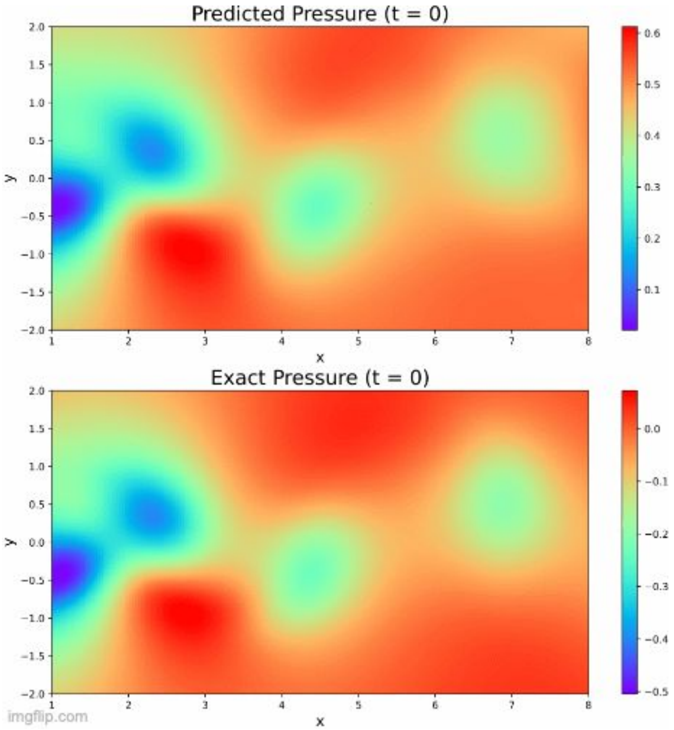
Fairuz Ishraque's Contributions

PINN Predictions (Velocity Fields)



Fairuz Ishraque's Contributions

PINN Predictions (Pressure Field)



[View this GIF]

PINN Results

- u and v predictions nearly exact to ground truth
- Pressure predictions off by a constant that is due to the nature of the Navier-Stokes system
- Pressure field predictions qualitatively quite accurate

Fairuz Ishraque's Contributions

References

- NASA / GSFC / Jeff Schmaltz / MODIS Land Rapid Response Team. English: Cloud vortices in the cloud layer off Heard Island, south Indian Ocean. The Moderate Resolution Imaging Spectroradiometer (MODIS) aboard NASA’s Aqua satellite captured this true-color image of sea ice off Heard Island on Nov 2, 2015 at 5:02 AM EST (09:20 UTC). Nov. 2015. url: https: / / commons . wikimedia . org / wiki / File : Heard _ Island _ Karman _ vortex _ street . jpg
- M. M. Zdravkovich. Flow around Circular Cylinders: A Comprehensive Guide through Flow Phenomena, Experiments, Applications, Mathematical Models, and Computer Simulations. Vol. 1. Oxford University Press, 1997. 8
- M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: Journal of Computational Physics 378 (Feb. 2019),