Warsaw University of Technology

Faculty of Power and Aeronautical Engineering
Institute of Heat Engineering

Computational Methods in Combustion

Simulation of Premixed Methane-Air Combustion in a 2D Chamber Using XiFoam Solver in OpenFOAM

Author:
Franciszek Tkaczyk

Supervisor:
Dr. inż. Mateusz Żbikowski

Warsaw, June 2025

Contents

Abstract

This report presents a two-dimensional numerical simulation of the combustion of a stoichiometric methane-air mixture using the `XiFoam` solver within the OpenFOAM v2012 framework. The main objective is to evaluate the behavior of key thermodynamic variables, such as temperature, pressure, and density, over time in a simplified geometry. The simulation aims to replicate flame propagation phenomena under idealized conditions with a fixed flame speed model. Particular focus is placed on the configuration and performance of a homogeneous premixed mixture, with constant initial conditions and standard atmospheric parameters. Post-processing of the computed fields is performed with ParaView and Python to produce time-resolved visualizations and diagnostic plots.

1 Introduction and Physical Background

Combustion of methane in air is one of the most extensively studied phenomena in energy and thermal engineering due to its relevance in domestic, industrial, and power-generation applications. From a theoretical standpoint, the combustion of a premixed methane-air mixture is governed by chemical kinetics, transport phenomena, and turbulent mixing processes. In this project, a simplified approach is adopted to allow for an efficient computational representation of premixed flame propagation.

The process analyzed herein corresponds to the reaction:

$$\mathrm{CH_4} + 2(\mathrm{O_2} + 3.76\mathrm{N_2}) \rightarrow \mathrm{CO_2} + 2\mathrm{H_2O} + 7.52\mathrm{N_2}$$

This reaction assumes a stoichiometric ratio between methane and oxygen and reflects the composition of atmospheric air by including molecular nitrogen as a non-reactive diluent. The molar ratio of air to methane is approximately 9.5:1, leading to an effective molar mass of the reactive mixture around 29 g/mol.

Modeling this process using the `XiFoam` solver allows for the simulation of premixed turbulent combustion by employing a reaction progress variable Ξ. The solver tracks the flame front without explicitly resolving chemical species or reaction rates, which reduces computational cost while capturing the essential flame dynamics.

The study focuses on a two-dimensional domain representing an idealized combustion chamber, in which the mixture is ignited at one end and allowed to propagate freely. The aim is to extract diagnostic data (e.g. maximum temperature, pressure evolution) to characterize flame behavior and detect any irregularities in propagation or oscillatory phenomena.

2 Objectives and Scope of Simulation

The scope of the simulation is strictly limited to the investigation of one reactive case: a premixed stoichiometric methane-air mixture. The objective is to provide detailed temporal analysis of:

- peak temperature development,
- minimum and maximum density fields,
- average and maximum pressure evolution.

These quantities are selected due to their relevance in combustion diagnostics and system design. No chemical reaction mechanism is explicitly solved; instead, the flame front is prescribed through the turbulent flame speed S_u. The use of a constant flame speed model permits isolating geometric and transport effects in the solution.

The study does not consider detailed pollutant formation, incomplete combustion, heat losses, or three-dimensional effects. This enables a clean, controlled assessment of the solver's capabilities in tracking flame propagation and associated thermodynamic responses.

The computational results will be evaluated via post-processing tools. The simulation series includes a single execution of the solver, with no parametric variation between test cases.

3 Computational Setup

3.1 Geometry and Mesh

The computational domain is a two-dimensional rectangular chamber of size 100 cm × 30 cm. The mesh consists of 1000 × 300 cells, ensuring sufficient spatial resolution to capture the flame front and flow structures.

3.2 Solver and Physical Models

Simulations are performed using the `XiFoam` solver available in OpenFOAM v2012. This solver resolves the Favre-averaged compressible Navier–Stokes equations supplemented with a transport equation for the reaction progress variable Ξ, characterizing the evolution of the flame front in premixed combustion.

Turbulence is modeled using the standard $k-\epsilon$ approach. The fluid is treated as a perfect gas, and the Sutherland law is employed for dynamic viscosity. Thermophysical properties are assigned via the JANAF polynomial coefficients.

Combustion is modeled as a single-step premixed reaction with flame speed prescribed in the file `Su`, and the turbulent flame speed is computed internally by the solver based on the progress variable.

3.3 Initial and Boundary Conditions

The computational domain is initially filled with a quiescent reactive mixture at atmospheric pressure (101325 Pa) and temperature (298.15 K). The reactant molar mass is defined in `thermophysicalProperties`.

The boundary conditions are defined as follows:

- **Inlet and outlet (left/right)**: zeroGradient for pressure and temperature, fixed value for velocity and Ξ
- **Top and bottom walls**: slip or symmetry condition

3.4 Simulation Workflow

Simulations were conducted following a structured workflow, which ensured repeatability and consistency across cases. The following operations were executed in sequence:

1. Mesh generation:
   ```
   blockMesh
   ```
2. Domain decomposition for parallel computation:
   ```
   decomposePar
   ```
3. Parallel execution of the solver on 16 processors:
   ```
   mpirun -oversubscribe -np 16 XiFoam -parallel
   ```
4. Reconstruction of decomposed results:
   ```
   reconstructPar
   ```
5. Optional removal of processor directories to save disk space:
   ```
   rm -r processor*
   ```

Each case was configured in its own directory, containing subfolders `0/`, `constant/`, and `system/`. The configuration included solver settings (`controlDict`), discretization schemes (`fvSchemes`), and solution controls (`fvSolution`). Thermodynamic data and species properties were specified in `thermophysicalProperties`, `chemistryProperties`, and `combustionProperties`. The flame speed model was parameterized in `Su`.

Post-processing was carried out in ParaView for visual verification and using a dedicated Python script for extraction and analysis of scalar fields such as pressure, temperature, and density.

4 Python Post-Processing

To enable automated extraction and visualization of scalar fields from simulation output directories, a custom Python script was developed. This script performs the following key operations:

- iterates through all time-step folders (excluding metadata like `0/`, `constant/`),
- reads scalar field data for temperature (T), pressure (p), and density (rho),
- calculates: maximum temperature, minimum and maximum density, average and maximum pressure for each time step,
- generates time-resolved plots using `matplotlib`,
- saves plots in a dedicated `plots/` subfolder within the simulation directory.

The script is designed to be reusable for other case folders, provided that field data are exported in OpenFOAM scalar format (nonuniform `List<scalar>`).

Required Python Packages

The following packages must be installed and available in the Python 3.9+ environment:

- `numpy` — for efficient numerical operations on arrays,
- `matplotlib` — for plotting and saving diagrams,
- `os`, `sys` — standard libraries for file handling and argument parsing.
- `tqdm` — for progress bars and ETA estimation during time-step loop execution.

These can be installed using:

```
pip install numpy matplotlib tqdm
```

Script Listing

The full content of the script `plot_combustion_results.py` is presented below:

```python
import os
import sys
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

def read_scalar_field(path):
    try:
        with open(path, 'r') as f:
            lines = f.readlines()
    except FileNotFoundError:
        return None

    n_values, start_idx = None, None
    for i, line in enumerate(lines):
        if 'nonuniform' in line and 'List<scalar>' in line:
            try:
                n_values = int(lines[i + 1].strip())
                start_idx = i + 3
                break
            except ValueError:
                continue

    if start_idx is None or n_values is None:
        return None

    values = []
    for line in lines[start_idx:]:
        if ')' in line:
            break
        stripped = line.strip().strip(';')
        if stripped:
            try:
                values.append(float(stripped))
            except ValueError:
                continue

    if len(values) != n_values:
        return None

    return np.array(values)

def analyze_series(case_dir):
    times = []
    t_max_list = []
    rho_min_list = []
    rho_max_list = []
    p_max_list = []
    p_avg_list = []

    time_dirs = []
    for d in os.listdir(case_dir):
        path = os.path.join(case_dir, d)
        if os.path.isdir(path):
            try:
                t_val = float(d)
                if t_val > 0.0:
                    time_dirs.append(d)
            except ValueError:
                continue

    for time_dir in tqdm(sorted(time_dirs, key=float), desc="Processing steps"):
        time_path = os.path.join(case_dir, time_dir)

        t_field = read_scalar_field(os.path.join(time_path, 'T'))
        rho_field = read_scalar_field(os.path.join(time_path, 'rho'))
        p_field = read_scalar_field(os.path.join(time_path, 'p'))

        if t_field is None or rho_field is None or p_field is None:
            continue

        times.append(float(time_dir))
        t_max_list.append(np.max(t_field))
        rho_min_list.append(np.min(rho_field))
        rho_max_list.append(np.max(rho_field))
        p_max_list.append(np.max(p_field))
        p_avg_list.append(np.mean(p_field))

    return (np.array(times), np.array(t_max_list), np.array(rho_min_list),
            np.array(rho_max_list), np.array(p_max_list), np.array(p_avg_list))

def plot_results(times, t_max, rho_min, rho_max, p_max, p_avg, output_dir):
    os.makedirs(output_dir, exist_ok=True)

    plt.figure()
    plt.plot(times, t_max, label='Max Temperature')
    plt.xlabel('Time [s]')
    plt.ylabel('Temperature [K]')
    plt.title('Maximum Temperature Over Time')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'Tmax_vs_time.png'))

    plt.figure()
    plt.plot(times, rho_min, label='Minimum Density', color='green')
    plt.plot(times, rho_max, label='Maximum Density', color='red')
    plt.xlabel('Time [s]')
    plt.ylabel('Density [kg/m ]')
    plt.title('Density Extremes Over Time')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'rho_min_max_vs_time.png'))

    plt.figure()
    plt.plot(times, p_max, label='Maximum Pressure', color='purple')
    plt.plot(times, p_avg, label='Average Pressure', color='orange')
    plt.xlabel('Time [s]')
    plt.ylabel('Pressure [Pa]')
    plt.title('Pressure Evolution Over Time')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'pressure_over_time.png'))

    plt.close('all')

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python plot_combustion_results.py <case_directory>")
        sys.exit(1)

    case_dir = sys.argv[1]
    output_dir = os.path.join(case_dir, "plots")

    times, t_max, rho_min, rho_max, p_max, p_avg = analyze_series(
        case_dir)
    plot_results(times, t_max, rho_min, rho_max, p_max, p_avg,
        output_dir)
```

Listing 1: Python Code for processing OpenFOAM combustion output

4.1 Code description

The Python script begins by importing necessary modules: `numpy` for numerical computations, `matplotlib` for plotting, `os` and `sys` for file system interaction, and `tqdm` for displaying a progress bar with estimated remaining time.

The main logic is divided into three functional blocks:

1. **Scalar field reader:** a function `read_scalar_field()` opens a given field file (e.g. `T`, `p`, `rho`) and parses values from the OpenFOAM format — skipping headers and recognizing nonuniform scalar lists.
2. **Series analyzer:** the function `analyze_series()` loops over all numeric time-step folders, loading field values, computing extremal and average values for each time step, and collecting time stamps.
3. **Plotting function:** the function `plot_results()` generates three figures:
   - maximum temperature vs time,
   - minimum and maximum density vs time,
   - maximum and average pressure vs time,

   each saved as a PNG in the `plots/` subdirectory.

The script is executed from the command line with a single argument: the path to the simulation case folder.

```
python plot_combustion_results.py ./Seria1
```

Its modular design ensures that it can be reused for multiple series or simulation cases, provided that time-step directories and scalar fields are present. The use of `tqdm` makes long processing runs transparent to the user by reporting real-time progress and ETA.

5 Results

The simulation produced complete time-series data for temperature, density, and pressure in a two-dimensional combustion chamber. The analysis was performed both visually using ParaView and numerically using a custom Python script that extracts scalar field extrema at each simulation time step.

Thermodynamic Time Series

Figures 1 through 3 present the evolution of maximum temperature, minimum and maximum density, and both maximum and average pressure over time. These plots provide a diagnostic snapshot of the system response to ignition and flame propagation.

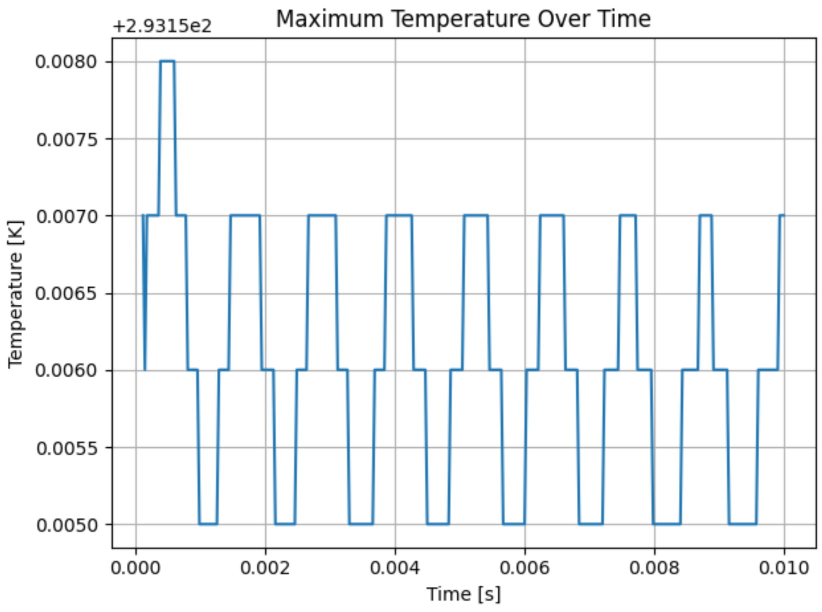


Figure 1: Maximum temperature evolution over time.

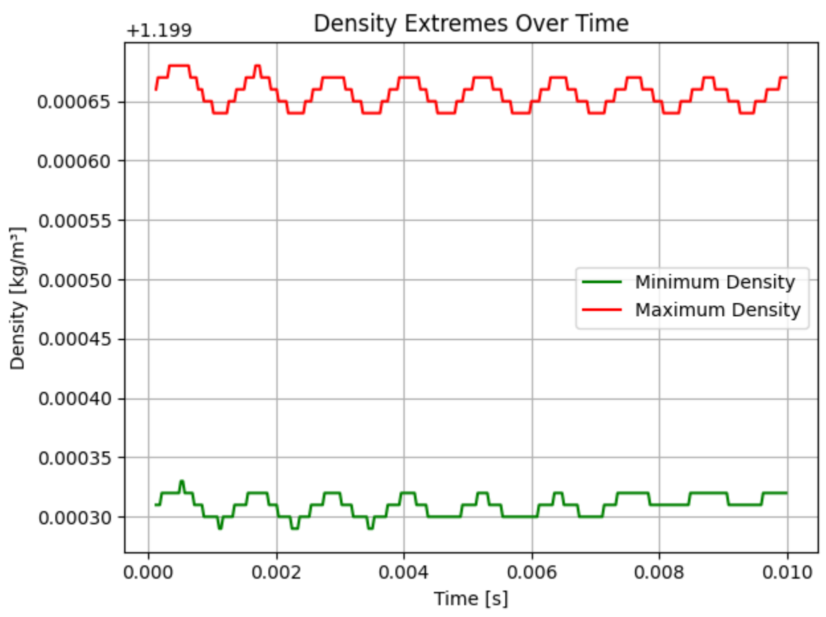


Figure 2: Minimum and maximum density over time.

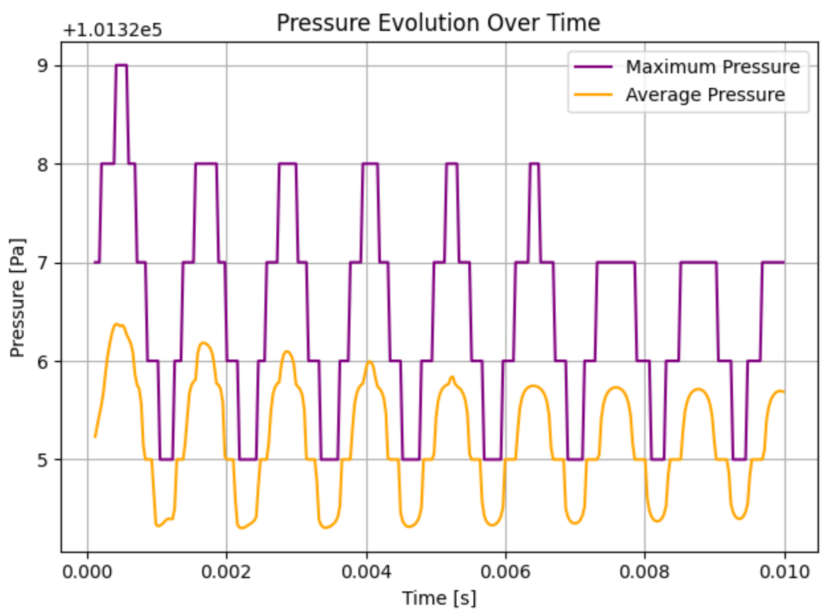
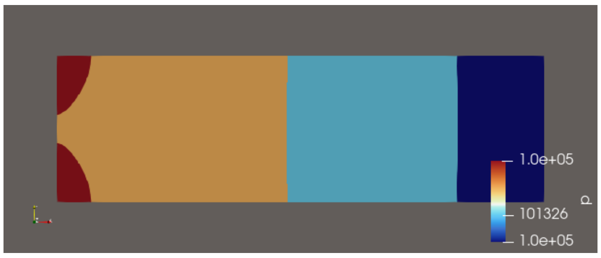


Figure 3: Maximum and average pressure evolution over time.

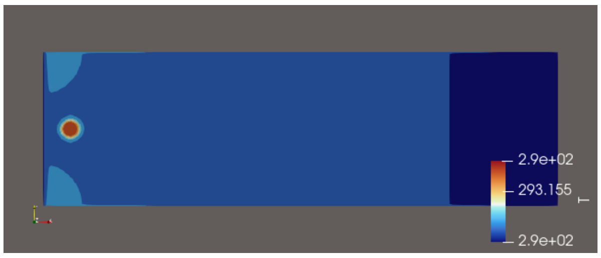
All variables display an oscillatory trend, which is particularly visible in the pressure and density plots. These oscillations are non-physical and stem from numerical artifacts rather than any physical wave reflection or instability. The root cause is suspected to be related to the configuration of the flame stretch factor model (`XiModel = transport`) and inadequate numerical damping in the solution scheme, especially at early stages of ignition.

Flow Field Snapshots

Figures 4a and 4b present visualizations of the computed density and temperature fields at a selected mid-simulation time. These snapshots are obtained using ParaView and illustrate the flame propagation front, with sharp gradients and compressible effects captured by the solver.



(a) Pressure field from ParaView



(b) Temperature field from ParaView

Figure 4: Visualization of field values in ParaView at selected time.

Optionally, an animated GIF was also produced to visualize the flame development dynamically. Such animations help correlate scalar oscillations with spatial flame propagation patterns and may help isolate error sources.

6 Conclusions

This project successfully implemented and executed a premixed methane-air combustion simulation using the `XiFoam` solver. The solver was configured with the `transport` model for flame stretch factor Ξ and used ignition sources defined via `ignitionSites`.

Despite correct mesh resolution and physical model selection, the results revealed significant non-physical oscillations in the time evolution of key thermodynamic fields: pressure, density, and temperature. These oscillations were not artifacts of the post-processing routine and were confirmed in ParaView visualizations.

The likely cause is the inadequate damping and time-step management in the initial flame growth stage. The flame front propagates too rapidly or with excessive stiffness in the governing equations, particularly when turbulent flame speed is exaggerated by high `Xi` values immediately after ignition. Additionally, no active chemical kinetics were resolved — only a flame surface tracking model was applied.

Recommendations for Future Work

To improve numerical stability and reduce artificial fluctuations, the following enhancements are recommended:

- Implement dynamic flame speed models (e.g. based on local temperature or turbulence),
- Calibrate ignition parameters (e.g. reduce `strength`, increase `duration`),
- Use higher-order discretization schemes for the transport equation of Ξ,
- Lower the initial `Xi` and `b` values outside of the ignition zone,
- Introduce time-step control using `adjustTimeStep = yes;` and set a low `maxCo`.

The report highlights both the strengths and limitations of simplified flame modeling in OpenFOAM. While the solver can reproduce realistic flame structures, its sensitivity to initial condition tuning and numerical schemes must not be underestimated.

References

[1] OpenFOAM Foundation. OpenFOAM User Guide v2012.
`https://www.openfoam.com/documentation/`

[2] Turns, S. R.
An Introduction to Combustion: Concepts and Applications. 3rd ed., McGraw-Hill, 2012.

[3] Poinsot, T. and Veynante, D.
Theoretical and Numerical Combustion. 2nd ed., R.T. Edwards, 2005.

[4] Versteeg, H. K., and Malalasekera, W.
An Introduction to Computational Fluid Dynamics: The Finite Volume Method. 2nd ed., Pearson, 2007.

[5] Ahrens, J., Geveci, B., Law, C.
ParaView: An End-User Tool for Large Data Visualization.
`https://www.paraview.org/`