uc3m | Universidad Carlos III de Madrid

University Degree in Aerospace Engineering

Academic Year 2024-2025

Bachelor Thesis

Integration of plasma flows for electric propulsion with Trixi.jl

Oscar Mora Hernandez

Mario Merino

Leganes, 06/16/2025

SUMMARY

This final degree project evaluates whether Julia and the open-source framework Trixi.jl can be effectively used to simulate plasma flows in magnetic arch thrusters. To begin with, the study uses well-known test cases, like the 1D Sod shock tube and the 2D Prandtl–Meyer expansion, to validate the numerical setup. Different configurations are tested by varying the polynomial order, mesh resolution, numerical flux, and time marching methods to analyze accuracy and performance.

After this initial validation, the plasma model described in the reference article is implemented in Julia using Trixi.jl. The results obtained are compared with those from the original model to assess their consistency. Finally, a parametric study is carried out to understand how changes in the adiabatic coefficient γ affect the plasma's behavior, particularly in terms of flow expansion, ion dispersion, and the position of the shock wave.

The results show that Julia and Trixi.jl offer a reliable and efficient environment for simulating plasma in electric propulsion systems. Although some limitations were found, such as challenges with visualization and sensitivity to boundary conditions, the approach proves to be a good and accessible alternative for future research in this field.

Keywords: Electric propulsion, Plasma simulation, Julia, Trixi.jl, Magnetic Arc.

DEDICATION

This work has been carried out in the framework of the ZARATHUSTRA project, which received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 950466)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Motivation

The configuration known as the Magnetic Arch (MA), which appears when operating two plasma sources with opposite polarities, is presented as an alternative to traditional magnetic nozzles used in space propulsion. In these conventional systems, such as Helicon or ECR thrusters, two main problems arise. On one hand, the plasma undergoes significant radial expansion, leading to high divergence, and on the other hand, strong magnetic dipole moments appear. This makes integration into satellites difficult, since high divergence reduces thrust, and the magnetic dipole interacts with the Earth's magnetic field, causing unwanted torques on the spacecraft [1].

The MA geometry aims to solve both problems. By connecting the field lines of two magnetic nozzles with opposite polarities, forming a closed arc, the magnetic moment is canceled and a narrower plasma jet is achieved. This reduces lateral momentum losses and the negative effects on the satellite's attitude. These improvements make it easier to incorporate such systems into spacecraft [1].

Additionally, the MA allows combining several EPTs into a single system. This not only increases the total available thrust but also adds advantages such as modularity and redundancy. Each source can operate at lower power, better distributing the thermal load, and if one fails, the system can continue to operate. Another advantage of the MA is that directional control can be achieved without moving parts, simply by adjusting the power of each source [1].

These characteristics make the MA a good option for future space missions that require compact, flexible, and efficient electric propulsion systems.

In this work, it was decided to use the Julia programming language together with the Trixi.jl framework, instead of the internal code developed by the research group. This open-source tool focuses on performing numerical simulations of systems of hyperbolic partial differential equations [2].

The main objective of the project is to evaluate the viability of implementing the plasma model developed by Merino et al. [3] in Julia using Trixi.jl, comparing the results obtained with those of the original model. The goal is to determine whether a solution based on open-source tools with an active community can faithfully reproduce previous results and represent an accessible and scalable alternative for future research. In addition, a parametric study is carried out by varying the adiabatic coefficient, γ, in order to analyze its influence on the plasma dynamics.

1.2. State of the art

In recent years, research on plasma acceleration using magnetic nozzles has made significant progress. One of the key works was by Ahedo and Merino (2010), see [4], where a two-dimensional model was developed to describe the expansion of a plasma inside a magnetic nozzle. In this model, the plasma is nearly neutral, collisionless, and with low pressure compared to the magnetic field. Their study showed that as plasma expands, radial non-uniformities appear, causing a separation between ions and electrons, which generates internal electric currents. They also explained that the thrust generated is not only due to the thermal expansion of the plasma, but also to the electromagnetic forces associated with the Hall currents induced in the system. Additionally, they calculated the energy lost due to radial expansion, which is important for understanding the system's efficiency. Although it was an important step, this model did not address aspects such as plasma detachment from the magnetic field or how the flow becomes supersonic at the nozzle exit.

Based on these results, Merino et al. (2023) [3], presented the concept of the Magnetic Arch (MA), a configuration consisting of two parallel magnetic nozzles with opposite polarities, whose magnetic field lines connect to form a closed magnetic arc. Using a stationary, two-dimensional, collisionless, bi-fluid model, they simulated how the plasma expands in this configuration. They found that the ion beams from each nozzle meet at the center of the domain and merge into a single jet that can propagate downstream, despite the presence of closed field lines. This result confirmed the theoretical feasibility of generating useful thrust in an MA.

They also observed that the thrust was slightly lower than that of a traditional nozzle, partly because the jet expands less laterally and because certain currents appear that slightly slow down the plasma. However, when including the magnetic field induced by the plasma itself, an axial stretching of the arc occurs, which smooths the curvature in the apex region, reducing the contribution of magnetic drag and improving the overall efficiency of the system.

On the other hand, Boyé et al. (2025) [1] carried out the first experimental validation of the MA using two ECR plasma sources with opposite polarities. They compared different configurations: one source, two with the same polarity, and two with opposite polarities, measuring ion current and energy. Their results matched the predictions of Merino et al. [3], the MA configuration produces a narrower jet with higher ion current than the others, although with slightly lower ion energy. Despite this, the net thrust was effective, confirming the practical feasibility of the concept.

Taken together, the works of Ahedo and Merino laid the theoretical foundation for plasma behavior in magnetic nozzles; Merino et al. extended this framework toward a new configuration with closed topology and characterized its main physical mechanisms; and Boyé et al. provided experimental evidence supporting its usefulness in real environments.

This project follows that line of research by reproducing the model of Merino et al. using the Julia programming language and the Trixi.jl framework, and by studying the effect of the adiabatic coefficient on plasma dynamics. The goal is to evaluate the potential of open-source tools as an accessible and reproducible alternative for simulations in the field of electric propulsion.

1.3. Objectives

The overall purpose of this final degree project is to evaluate the feasibility of using Julia and Trixi.jl as an open-source platform for plasma simulation in magnetic arch thrusters, taking as reference the results from the previously mentioned article [3].

To achieve this, the following objectives are proposed:

Obj 1 Numerical verification in known cases
Implementation of the Sod shock tube (1D) and the Prandtl–Meyer expansion (2D) in Trixi.jl, performing a convergence analysis by varying the polynomial degree, number of mesh elements, numerical flux, and explicit and implicit time marching methods. The accuracy with respect to the analytical solution and the computational time associated with each configuration will be evaluated simultaneously.

Obj 2 Integration of the model in Julia and comparison
Integrate the physical model from the reference article in Julia using the Trixi.jl framework, defining a new system of equations within the framework. In addition, the results obtained will be contrasted with the reference results, identifying similarities and possible discrepancies.

Obj 3 Parametric study
Evaluate the influence of the adiabatic constant γ on the plasma dynamics through an analysis that allows identifying its impact on the macroscopic variables of the system.

Obj 4 Conclusions and future work
Establish conclusions on the potential of Julia and Trixi.jl for the design of electric thrusters, specifically magnetic arch thrusters, identifying areas for improvement and possible future implementations.

1.4. Regulatory framework

In Europe, any activity related to the space sector, including electric propulsion and simulation models, is standardized and regulated by the European Cooperation for Space Standardization (ECSS).

For this project, and for the future development of the Magnetic Arch Thruster, three documents are particularly important:

- **ECSS-E-ST-35-01C – Liquid and electric propulsion for spacecraft (15 November 2008)**
  This document defines the design, testing, and acceptance requirements for all electric propulsion systems.
- **ECSS-E-ST-40C Rev.1 – Software (30 April 2025)**
  This standard covers all aspects of space software engineering, including requirements definition, design, development, verification and validation, transfer, operation, and maintenance.
- **ECSS-E-ST-40-07C – Simulation modelling platform (2 March 2020)**
  This document complements the previous one and enables the effective reuse of simulation models within and between space projects and their stakeholders.

Together, these standards require that any simulation supporting the design of an electric thruster must be traceable, verifiable against reference cases, and reproducible under the same numerical parameters.

1.5. Socio-economic environment

The use of Julia and Trixi.jl, both open-source, eliminates the licensing costs associated with most commercial CFD software. This absence of direct expenses is particularly attractive for academic groups and NewSpace companies, as it allows them to allocate resources to hardware or experimentation instead of software licenses.

In addition, the open-source condition of these tools supports training and talent development. Any student or researcher can examine the implementation, reproduce results, and propose improvements, which strengthens a culture of collaboration and lowers the entry barrier for new participants in the field of fluid simulation. This approach helps build a technical community capable of sustaining the development of future versions of the model.

Lastly, there is an environmental component: the high-order methods implemented in Trixi.jl allow for the same numerical accuracy to be achieved with fewer time steps and coarser meshes. In practice, this leads to a significant reduction in CPU-hours and, consequently, in electricity consumption and associated emissions. Although the energy savings from a single bachelor's thesis may be minimal, the widespread adoption of such techniques could become a relevant factor.

The following table presents the budget associated with this project.

TABLE 1.1. PROJECT BUDGET

Item	**Units**	**Unit cost (€)**	**Subtotal (€)**
Student work	600 h	15	9 000
Mid-range computer (8 cores / 32 GB RAM)	1	1 200	1 200
Electricity consumed during simulations	150 kWh	0,25	38
Julia + Trixi.jl license*	—	0	0
		Estimated total cost	**10 238**

** As open-source software, the license cost is 0€, representing a direct saving compared to commercial suites.*

1.6. Methodology

1.6.1. Preliminary study and knowledge acquisition

The development of this Bachelor's Thesis first required an initial phase of knowledge acquisition. During this stage, a review of the fundamentals of plasma physics was carried out. In addition, numerical methods based on the Discontinuous Galerkin (DG) scheme, used by the Trixi.jl simulation framework, were studied.

Since the chosen development environment was the Julia programming language, it was also necessary to acquire a solid basis in it. This was achieved through official documentation, practical examples, and simple tests [5].

1.6.2. Familiarization with the simulation environment

Once a basic level in Julia had been achieved, the next step was to study the functioning of Trixi.jl, a tool for solving hyperbolic partial differential equations. To verify the understanding of the framework and its associated workflows, two test simulations were developed using classical cases:

- The one-dimensional Sod shock tube
- A two-dimensional Prandtl-Meyer expansion over an inclined plane

These simulations served as a foundation for consolidating the implementation of boundary conditions, mesh configuration, numerical flux selection, and result evaluation. Additionally, to assess the numerical behavior of the system, a convergence analysis was carried out for both cases.

Several aspects that influence both the accuracy of the solution and the computational cost were examined, including:

- The number of elements in the mesh.
- The polynomial order.
- The type of numerical flux, such as Lax-Friedrichs or HLL.
- The chosen time marching method

These analyses made it possible to understand how each parameter affects the solution, providing a more detailed view of the simulation behavior.

1.6.3. Simulation of auxiliary variables

Before addressing the full plasma simulation, work was carried out on the evolution of auxiliary variables such as the electron energy (H_e) and the out-of-plane velocity (U_{ye}). The propagation of these variables was studied to ensure they followed the magnetic streamlines, and interpolators were implemented to extend the data defined at the domain boundaries throughout the interior.

In addition, a convergence analysis was performed to verify that the obtained solutions were accurate and consistent with the expected behavior.

1.6.4. Implementation of the plasma model

For this part, it was necessary to define a new type of system of equations in Trixi.jl to capture the properties of the physical system under consideration [3]. In addition, boundary conditions were designed to be compatible with the geometry and physics of the problem, including the enforcement of symmetries and the specification of plasma inflow and outflow conditions.

1.6.5. Main simulation and parametric analysis

As previously mentioned, the ultimate goal was to reproduce and compare the results presented in the original article with those obtained from the implementation in Julia using the Trixi.jl framework. To this end, a geometric and magnetic field configuration similar to that of the article was designed, and the full plasma simulation was initiated.

During this phase, a parametric analysis was also carried out to study the effects of variations in the value of the adiabatic coefficient.

2. PRELIMINARY VALIDATION OF TRIXI.JL

2.1. Fundamentals of Trixi.jl

All the information presented below, along with additional features and advanced functionalities not covered here, can be found in the official Trixi.jl documentation available on its website [2], as well as by exploring the source code on GitHub, where users can also contribute, report issues, ask questions, or suggest improvements [6].

As previously mentioned, the programming environment used in this project is the Julia language in combination with the Trixi.jl framework. Therefore, before proceeding, it is essential to understand what it is and how it works.

Trixi.jl is an open-source framework developed in Julia, designed to solve hyperbolic partial differential equations. To achieve this, it primarily uses the Discontinuous Galerkin (DG) method, providing high accuracy in numerical solutions and allowing the simulation of systems in one, two, and three dimensions.

Its architecture is based on modularity, which enables the separation of different components of the numerical scheme, such as the physical equations, spatial discretization methods, and mesh types, among others. This structure allows for rapid experimentation with various configurations. Additionally, it facilitates extensions to implement custom systems of equations, specific boundary conditions, source terms, or auxiliary functions (callbacks).

Among the key features of Trixi.jl is the ability to use both structured and unstructured meshes, including those generated with external preprocessing tools such as GMSH. It also supports advanced techniques like adaptive mesh refinement (AMR).

The framework includes several predefined physical systems, such as the compressible Euler equations, linear advection equations, and the Navier-Stokes equations, among others. Moreover, Trixi.jl offers extensive documentation to support new users, along with an active community that provides effective assistance in resolving questions and difficulties during its use.

What follows is a description of the steps needed to configure and execute a simulation using Trixi.jl:

2.1.1. Definition of the equations

The first step is to choose the system of equations to be solved. You can either use predefined systems or create a new one. Implementing predefined equations is straightforward:

$$equations = SystemOfEquations(parameters)$$

If the parameters or the name of the system of equations are unknown, they can be found in the official documentation or in the source code available on GitHub [6]. In cases where a custom system is required, as with the plasma model equations described in later chapters, the implementation is more complex. However, the Trixi.jl documentation [2] provides guidance on how to create a new one-dimensional equation, which can be extended to two and three dimensional models.

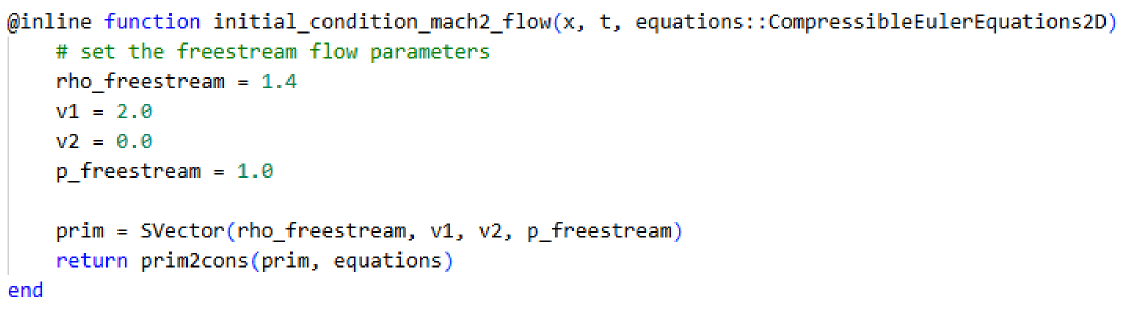
2.1.2. Mesh definition

The next step is to define the domain and the type of mesh. Trixi.jl allows geometries to be specified either by setting the minimum and maximum coordinates of the domain or by using meshes generated with external tools such as GMSH.

If a mesh is generated using preprocessing software, it must contain square or hexagonal elements in Abaqus (.inp) format.

2.1.3. Initial conditions and source terms

Initial conditions can be defined using predefined functions or by creating custom ones. When defined manually, a function is created that takes position, time, and the equation system as parameters, and returns a vector of conserved variables. For example, for the two-dimensional compressible Euler equations:



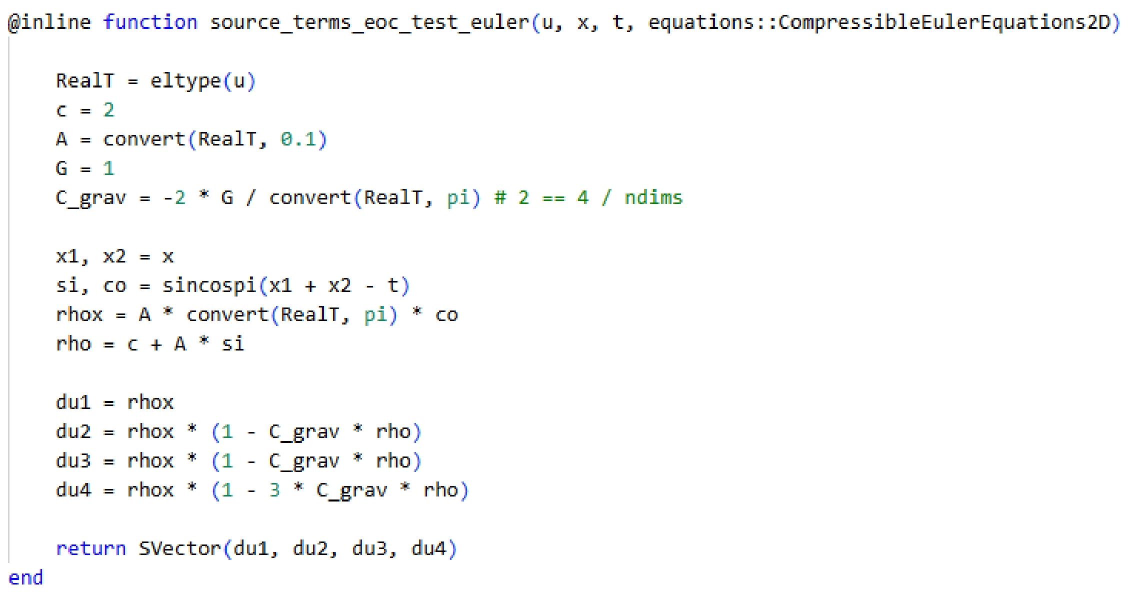
```julia
@inline function initial_condition_mach2_flow(x, t, equations::CompressibleEulerEquations2D)
    # set the freestream flow parameters
    rho_freestream = 1.4
    v1 = 2.0
    v2 = 0.0
    p_freestream = 1.0

    prim = SVector(rho_freestream, v1, v2, p_freestream)
    return prim2cons(prim, equations)
end
```

Fig. 2.1. Initial conditions example.

In this example, when defining the initial conditions for the primitive variables, namely, density, velocities, and pressure, the prim2cons function must be used to convert them into the corresponding conserved variables.

Similarly, source terms require functions that return vectors consistent with the conserved variables. As an example, consider the source terms applied to the two dimensional compressible Euler equations:



```julia
@inline function source_terms_eoc_test_euler(u, x, t, equations::CompressibleEulerEquations2D)

    RealT = eltype(u)
    c = 2
    A = convert(RealT, 0.1)
    G = 1
    C_grav = -2 * G / convert(RealT, pi) # 2 == 4 / ndims

    x1, x2 = x
    si, co = sincospi(x1 + x2 - t)
    rhox = A * convert(RealT, pi) * co
    rho = c + A * si

    du1 = rhox
    du2 = rhox * (1 - C_grav * rho)
    du3 = rhox * (1 - C_grav * rho)
    du4 = rhox * (1 - 3 * C_grav * rho)

    return SVector(du1, du2, du3, du4)
end
```

Fig. 2.2. Source terms example.

2.1.4. Boundary conditions

Trixi.jl provides several predefined boundary conditions, such as periodic, Dirichlet, Neumann, and no-slip. It is important to first check whether these conditions are defined for the chosen system of equations by consulting the documentation [2] or the source code [6].

If the required boundary condition is not defined for the selected system, the available documentation lacks sufficient information, making it challenging for beginners to fluid simulation to implement custom boundary conditions.

2.1.5. Definition of the solver

This step involves selecting the solver (DGSEM or DGMULTI), specifying the polynomial degree, the numerical flux, and determining whether techniques such as shock capturing or adaptive mesh refinement (AMR) will be used. The DGMULTI solver enables the use of triangular meshes in addition to square and hexagonal ones. However, since the geometry required to solve our system is not complex, DGSEM was chosen.

2.1.6. Problem definition

Finally, the SemidiscretizationHyperbolic function is used, which requires the mesh, initial conditions, source terms, boundary conditions, and the solver as parameters. With this function, the time interval is defined and the semidiscretization process is initiated:

semi = SemidiscretizationHyperbolic(...)

```
time_interval = (initial_time, final_time)
ode = semidiscretize(time_interval,semi)
```

2.1.7. Callbacks and simulation execution

Before running the simulation, it is possible to configure callbacks, auxiliary functions that are executed at specific points during the simulation.

Some examples include:

- **Analysis callback**: calculates errors, integrals and also displays and stores the results.
- **Alive callback**: reports the progress of the simulation.
- **Steady-state callback**: stops the simulation once a specified tolerance is reached.
- **AMR callback**: handles adaptive mesh refinement.
- **Save and Restart Callbacks**: the save callback is necessary for post-processing in ParaView, as it generates the files required to convert the output to VTK format. The restart callback allows the simulation to be restarted from a saved state.

Trixi.jl contains all the necessary information to solve the problem except for the time integration. Therefore, the simulation is executed using the solve function, which relies on the OrdinaryDiffEq.jl package to manage time integration. This package offers various time marching methods, including implicit methods such as the trapezoidal and explicit methods like Runge-Kutta.

2.2. 1D Sod shock tube case

As a first step, the one dimensional Sod shock tube problem is analyzed. This study includes a comparison with the analytical solution, evaluating the impact of various numerical parameters: polynomial degree p, time marching method, numerical flux, number of elements and the use of shock capturing techniques. The objective is to understand how each of these parameters affects both the accuracy and the computational cost of the solution.

The accuracy of the solution is evaluated using the root mean square error (RMSE) with respect to the analytical solution.

All analyses start from a baseline configuration consisting of: HLLC numerical flux, a mesh of 500 elements, polynomial order $p = 3$, and no shock capturing. Each parameter is modified in isolation to study its impact in a controlled manner.

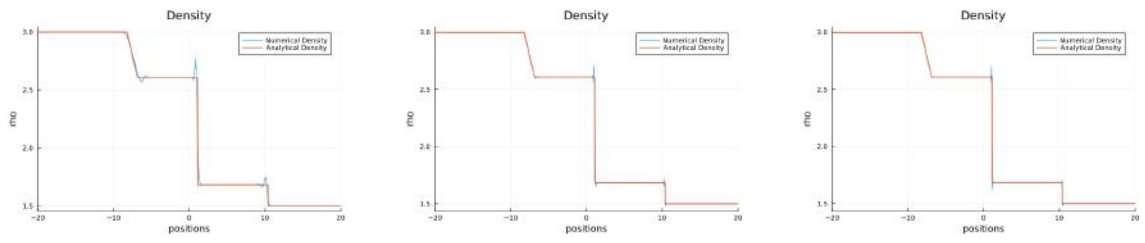
There are two exceptions to the baseline configuration:

- In the comparison of time marching methods, 1 000 elements are used, since with 500 elements the simulation times were too short to clearly differentiate performance.
- In the shock capturing test, 5 000 elements are used to more precisely observe the effect of this technique on the solution.

2.2.1. Influence of the polynomial order

The first parameter analyzed is the polynomial order p, which determines the degree of the polynomial function used to approximate the solution within each element. Higher orders provide greater spatial resolution and an improved ability to capture complex gradients without the need to refine the mesh. However, a high p value results in increased computational cost and may introduce non-physical oscillations in the presence of strong discontinuities if appropriate limiters are not used.

In cases involving smooth discontinuities, such as Prandtl–Meyer expansions or rarefaction waves, high-order polynomials allow accurate representation of transitions using fewer elements.



(a) $p = 1$ (b) $p = 2$ (c) $p = 3$

Fig. 2.3. Influence of polynomial order for Sod shock tube case.

TABLE 2.1. INFLUENCE OF POLYNOMIAL ORDER FOR SOD SHOCK TUBE CASE.

Polynomial order p	**RMSE**	**Simulation time [ms]**
$p = 1$	0.0298	169
$p = 2$	0.0194	238
$p = 3$	0.0139	389

The data presented in the table show an improvement in accuracy as the polynomial degree increases. The error decreases from 0.0297 with $p = 1$ to 0.0139 with $p = 3$, reflecting the scheme's greater ability to approximate the analytical solution. This improvement is particularly noticeable in the rarefaction regions and the strong shock wave.

Although, as noted earlier, high-order p schemes can induce non-physical oscillations near strong discontinuities, no significant artifacts are observed in this case. Therefore, this order yields stable and accurate results without the need for additional techniques.

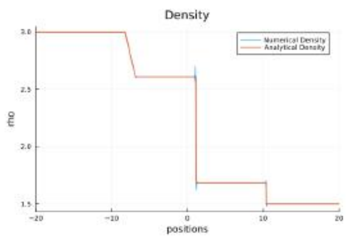
Regarding computational cost, the simulation with $p = 1$ completes in 169 ms, whereas with $p = 3$ the time increases to 389 ms. Although this difference may seem minor in a one dimensional case with few elements, the increase can be substantial in larger-scale simulations or in three dimensional geometries. Thus, the choice of polynomial order should consider both the desired accuracy and the available computational resources. In this case, $p = 3$ appears to be a good choice to balance resolution and efficiency.

2.2.2. Influence of time marching methods

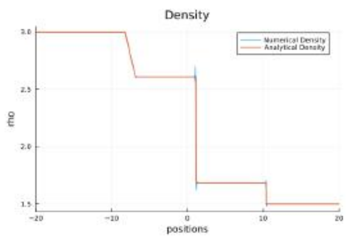
The choice of time marching method also affects the accuracy, stability, and computational cost of a simulation. This temporal scheme defines how the solution evolves between time steps. As previously mentioned, in Trixi.jl, the time integration method is implemented through the OrdinaryDiffEq.jl package, which offers a wide range of predefined methods.

The methods compared in this study are as follows:

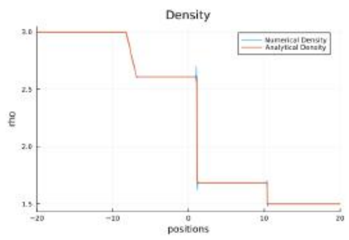
- **Explicit methods**:
  - **RDPK3SpFSAL510:** It is a Runge-Kutta method of fifth order, ten-stage method with embedded error estimator.
  - **SSPRK104:** It is a Runge-Kutta method of fourth order and ten-stage strong stability preserving method.
  - **RK4:** It is a Runge-Kutta method of fourth order.
- **Implicit methods**:
  - **ImplicitEuler:** It is an Euler method of first order.
  - **Trapezoidal:** It is a second order symmetric method.
  - **SDIRK2:** It is a Runge-Kutta implicit method of second order.



(a) RDPK3SpFSAL510

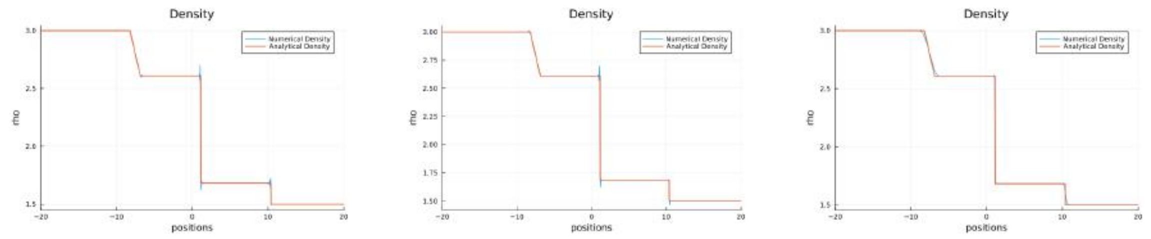


(b) RK4



(c) SSPRK104

Fig. 2.4. Explicit time marching methods.



(a) Trapezoidal (b) SDIRK2 (c) ImplicitEuler

Fig. 2.5. Implicit time marching methods.

TABLE 2.2. INFLUENCE OF TIME MARCHING METHOD FOR SOD SHOCK TUBE CASE.

Time marching method	RMSE	Simulation time [s]
RDPK3SpFSAL510	0.0139	0.338
SSPRK104	0.0139	0.340
RK4	0.0139	0.792
ImplicitEuler	0.0178	338
Trapezoidal	0.0139	0.340
SDIRK2	0.0139	0.792

As shown in the results, both explicit and implicit methods yield similar errors, around 0.014, except for the ImplicitEuler method, whose error increases to 0.018.

Regarding simulation time, the difference between the two groups is significant. Implicit methods are several orders of magnitude slower than explicit ones, without offering any improvement in solution quality. In fact, ImplicitEuler produces a less accurate result.

Additionally, the plots show that the implicit Euler method smooths the solution, both in the Prandtl–Meyer expansion and in the rarefaction region, while exhibiting very similar behavior in the strong shock wave. On the other hand, the other methods more accurately reproduce the analytical solution, although they introduce oscillations near the shock and rarefaction zones.

Therefore, it can be concluded that explicit time marching methods are more efficient in this case, offering shorter simulation times with comparable accuracy to the implicit ones. In particular, the SSPRK104 method stands out as the fastest and most reliable option.

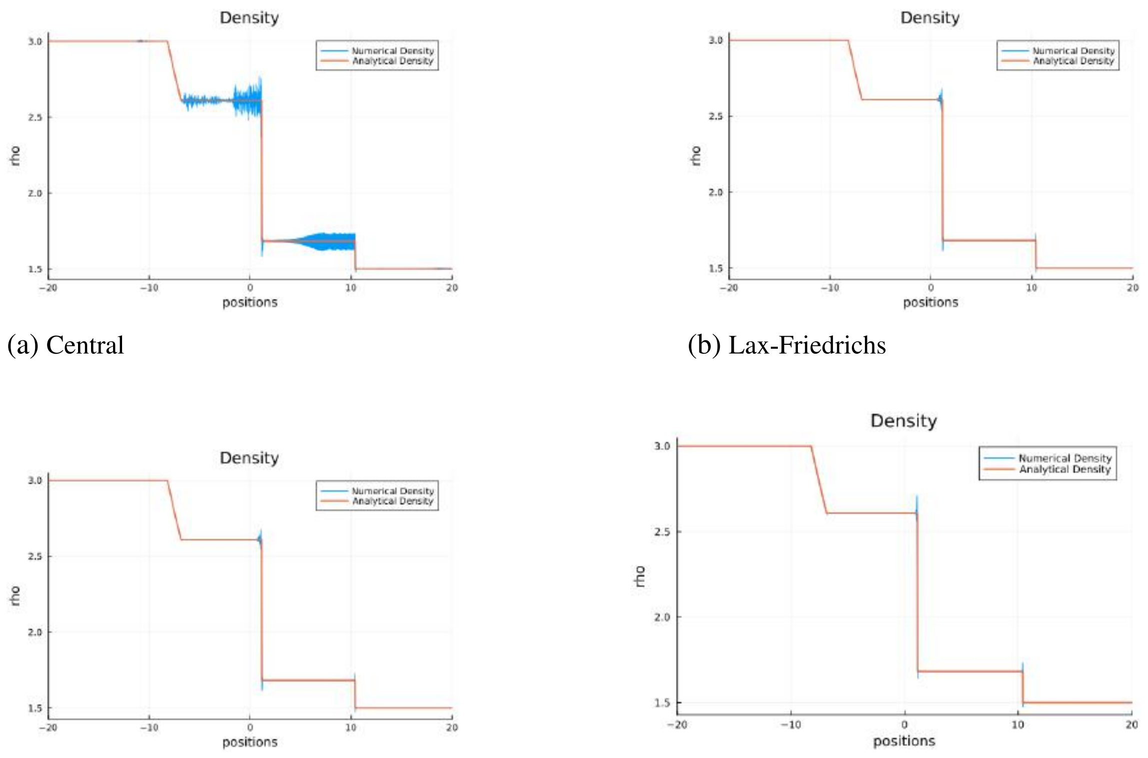
2.2.3. Influence of the numerical flux

After analyzing the impact of polynomial degree and time marching method, the next step is to study the effect of the numerical flux on solution accuracy, a key component in Discontinuous Galerkin methods. Numerical fluxes determine how information is exchanged between adjacent elements and therefore have a direct impact on the stability, diffusion, and fidelity of the solution, especially in the presence of discontinuities or steep gradients.

In Trixi.jl, numerical fluxes are defined in a modular way, allowing the selection of different schemes depending on the characteristics and equations of the problem. In this study, the following fluxes were compared:

- **Central**: a purely centered flux with no numerical diffusion. Due to this, it may produce oscillations in the presence of shocks and discontinuities.
- **Lax–Friedrichs**: introduces artificial diffusion proportional to the jump between cells, stabilizing the solution at the expense of smoothing it.
- **HLL**: an approximate Riemann flux with intermediate diffusion, which preserves discontinuities reasonably well.
- **HLLC**: an improved version of HLL, reconstructs the contact wave and offers higher accuracy in regions with multiple waves.

This analysis makes it possible to observe how the solution varies depending on the flux used, while keeping all other parameters constant. The goal is to evaluate each scheme's ability to resolve shocks, rarefaction waves, and Prandtl–Meyer expansions without introducing significant numerical errors.



(a) Central (b) Lax-Friedrichs

(c) HLL (d) HLLC

Fig. 2.6. Influence of numerical flux for Sod shock tube case.

TABLE 2.3. INFLUENCE OF NUMERICAL FLUX FOR SOD SHOCK TUBE CASE.

Numerical flux	**RMSE**	**Simulation time [s]**
Central flux	0.0219	1.40
Lax-Friedrichs flux	0.0093	1.26
HLL flux	0.0092	1.33
HLLC flux	0.0093	1.07

The plots show that the central flux is unable to handle discontinuities, introducing significant oscillations in the contact region and the rarefaction zone, which results in a considerably higher error.

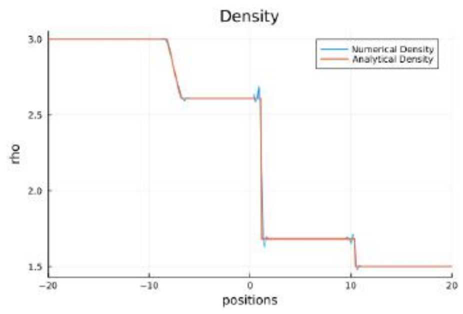
In contrast, the Lax–Friedrichs, HLL, and HLLC fluxes produce nearly identical results, with errors around 0.0093 and similar simulation times. The HLLC flux is particularly noteworthy, as it maintains comparable accuracy while achieving the shortest simulation time. Additionally, a reduction in oscillations is observed, likely due to the lower diffusion introduced in the contact region.

Although the differences among the last three fluxes are subtle in this case study, the use of the HLLC flux may be more advantageous in complex simulations, where accuracy in regions with discontinuities is critical.

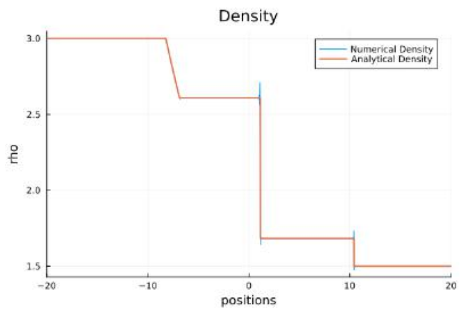
2.2.4. Influence of the number of elements

Mesh refinement is one of the most direct ways to increase the accuracy of a numerical solution. Increasing the number of elements allows for a more detailed representation of both abrupt changes and smooth gradient regions in the flow field, which leads to a reduction in numerical error. However, this improvement in resolution comes at the cost of increased computational effort, as the number of operations per time step rises proportionally.

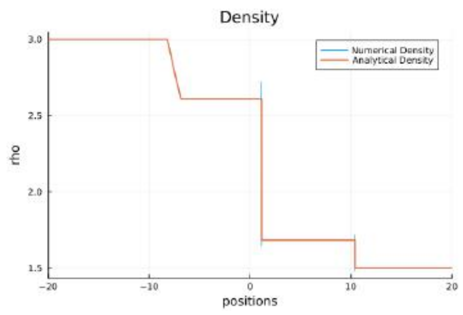
The following section presents a comparison between different mesh configurations:



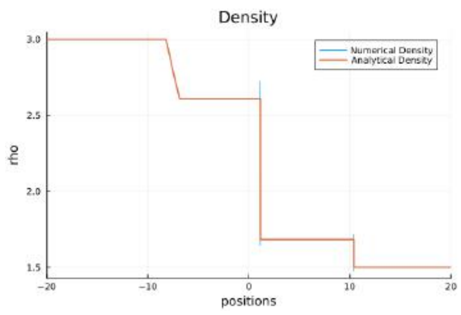
(a) 100 elements



(b) 1 000 elements



(c) 10 000 elements



(d) 15 000 elements

Fig. 2.7. Influence of number of elements for Sod shock tube case.

TABLE 2.4. INFLUENCE OF NUMBER OF ELEMENTS FOR SOD SHOCK TUBE CASE.

Number of elements	**RMSE**	**Simulation time [s]**
100	0.0254	0.041
1 000	0.0093	1.28
10 000	0.0036	81.2
15 000	0.0034	189

The table reveals a clear decreasing trend in error, which stabilizes around 0.003. The most significant drop occurs between 100 and 1 000 elements, with a 63% reduction in error. Accuracy continues to improve up to 10 000 elements, at which point the decrease in error relative to the number of elements becomes negligible.

In the plots, oscillations are visible in the contact and rarefaction regions when using a mesh with 100 elements. In contrast, with 1 000, 10 000, and 15 000 elements, the differences in the solutions are virtually imperceptible.

Although increasing the number of elements improves accuracy, the associated computational cost must be considered. For example, the simulation time with 15 000 elements reaches 3 minutes, compared to just 41 milliseconds with only 100 elements. Since accuracy gains become marginal beyond a certain point, a trade-off must be considered between accuracy and cost.

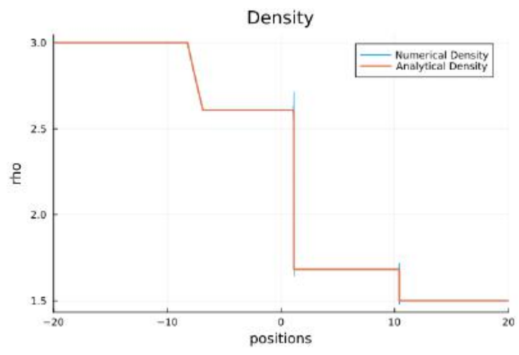
In this case, using a mesh with 1 000 elements appears to offer a good compromise, providing a sufficiently accurate and stable solution with a low computational cost.

2.2.5. Influence of the use of shock capturing

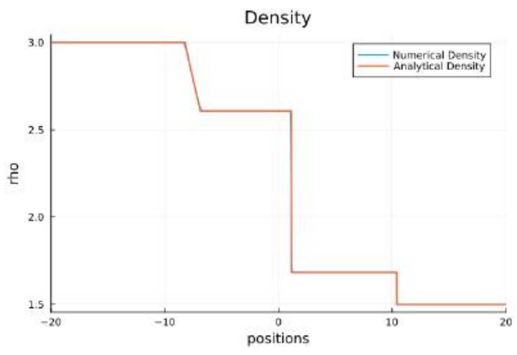
To conclude the analysis of the one dimensional case, the impact of applying shock capturing techniques is examined. In the presence of shock waves or steep gradients, numerical solutions may exhibit significant errors or even instabilities if appropriate stabilization mechanisms are not introduced. These techniques add localized diffusion in regions with discontinuities, aiming to stabilize the solution without significantly affecting the rest of the domain.

Trixi.jl allows these techniques to be activated through specific functions that incorporate adaptive diffusion based on indicators and user-defined parameters.

The results obtained are presented below:



(a) Without Shock Capturing



(b) With Shock Capturing

Fig. 2.8. Influence of shock capturing for Sod shock tube case.

TABLE 2.5. INFLUENCE OF SHOCK CAPTURING FOR SOD SHOCK TUBE CASE.

Shock capturing	RMSE	Simulation time [s]
ON	0.0147	121
OFF	0.0048	22

The solution obtained without using shock capturing maintains high accuracy with respect to the analytical solution, without introducing significant numerical oscillations. In contrast, when the technique is enabled, additional numerical diffusion is introduced in the discontinuity regions, which smooths the transitions and artificially broadens the affected areas.

This increase in diffusion explains the rise in error, as accuracy is sacrificed in favor of greater robustness. In this specific case, the use of shock capturing does not provide clear improvements and results in a significant increase in computational cost, with the simulation time rising from 22 to 121 seconds, an increase of more than five times.

However, these techniques are essential in more complex scenarios, such as multi-dimensional problems or those involving irregular geometries, where controlling non-physical oscillations is critical to ensure numerical stability. Therefore, before applying shock capturing techniques, one must carefully balance the need for stability against the potential loss of accuracy and increase in computational cost.

2.3. 2D Prandtl-Meyer expansion case

Once the analysis of the one dimensional Shock Sod tube is completed, the study proceeds with a two dimensional problem: the Prandtl–Meyer expansion

This problem presents several differences compared to the previous one. First, no strong shock waves are observed, instead, it presents a Prandtl–Meyer expansion characterized by a smooth discontinuity region. Second, it is a two dimensional problem, which allows for the evaluation of practical aspects related to the implementation of Trixi.jl in more complex configurations.

For this study, the geometry was created using the preprocessing tool GMSH. This made it possible to verify Trixi.jl's compatibility with externally generated meshes, demonstrating its ability to integrate into more general and realistic engineering workflows.

2.3.1. Problem statement and analytical solution development

This problem consists of a Prandtl-Meyer expansion on a 32 degree inclined plane with respect to the horizontal. The velocity of the flow at the expansion entry corresponds to a Mach number of 3. With these data, we proceed with the development of the analytical solution.

To determine the Mach number after the turn (M_2), it is necessary to relate it to the initial Mach number (M_1) and the turn angle (θ). The expression that links them is:

$$\theta = \nu(M_2) - \nu(M_1) \tag{2.1}$$

where the Prandtl-Meyer function $\nu(M)$ is defined as:

$$\nu(M) = \sqrt{\frac{\gamma+1}{\gamma-1}} \cdot \tan^{-1}\left(\sqrt{\frac{(\gamma-1)}{(\gamma+1)} \cdot (M^2-1)}\right) - \tan^{-1}\left(\sqrt{M^2-1}\right). \tag{2.2}$$

From this relation, considering $\gamma = 1.4$, $M_1 = 3$, and $\theta = 32$ degrees (0.5585 rad), we obtain:

$$\nu(M_1) = 0.86842 \text{ rad}, \quad \nu(M_2) = 1.42692. \tag{2.3}$$

Finally, solving for M_2, we obtain:

$$M_2 = 5.56. \tag{2.4}$$

Once the analytical solution is determined, a comparison is made between the polynomial degree p, the number of elements, the numerical flux, and the use of adaptive mesh refinement (AMR). In this case, no theoretical explanations will be provided for each parameter evaluated, as they were already discussed in the previous section. Only the analysis of AMR will be developed in more detail, since it is a new tool introduced in this study. The parameters used—except for the one being varied—are: polynomial degree $p = 3$, 2 842 elements, Lax–Friedrichs numerical flux, and the SSPRK104 time marching method.

This case includes one exception:

- In the comparison involving AMR, 722 elements are used, as simulations with 2,842 elements result in excessively high runtimes due to the computational cost of this technique.

2.3.2. Influence of the polynomial order

Following the same approach used in the Shock Sod tube case, the results are analyzed for polynomial degrees $p = 1$, $p = 2$, and $p = 3$.

The following table presents the Mach number obtained, the simulation time, and the relative error:

TABLE 2.6. INFLUENCE OF POLYNOMIAL ORDER FOR PRANDTL-MEYER EXPANSION CASE

Polynomial order p	**Mach number**	**Simulation time [s]**	**Relative Error**
1	4.490	7.08	0.1932
2	4.791	18.9	0.1391
3	4.950	40.3	0.1105

As expected, increasing the polynomial degree improves the accuracy of the solution, with a reduction in relative error from 19.32% for $p = 1$ to 11.05% for $p = 3$. However, this increase in accuracy comes with a considerable rise in computational cost, resulting in a simulation time that is more than five times longer between the two extremes analyzed.

2.3.3. Influence of the number of elements

This section compares different meshes generated using the GMSH preprocessing tool. Specifically, meshes with 322, 722, 2 842, and 6 952 elements are analyzed.

The results obtained are as follows:

TABLE 2.7. INFLUENCE OF NUMBER OF ELEMENTS FOR PRANDTL-MEYER EXPANSION CASE

Number of elements	Mach number	Simulation time [s]	Relative error
322	4.327	2.1	0.2224
722	4.572	5.7	0.1785
2 842	4.950	40.3	0.1105
6 952	5.110	256	0.0813

As the number of elements increases, a clear decreasing trend in relative error is observed, dropping from 22.24% with the coarsest mesh to 8.13% with the finest one.

However, this improvement in accuracy comes at a considerable computational cost, with the simulation time increasing from 2.1 to 256 seconds. The change in accuracy and runtime between 2 842 and 6 952 elements shows that, for a 2.6% reduction in error, the simulation time becomes six times longer.

Due to this behavior, it is expected that beyond 6 952 elements, the gain in accuracy would be negligible, while the time required to solve the problem would increase significantly, leading to an inefficient computational cost.

2.3.4. Influence of the numerical flux

Before addressing the use of AMR techniques, the impact of numerical fluxes is examined. In this case, the Lax–Friedrichs, HLL, and HLLC fluxes are considered. The central flux, analyzed in the previous section, is not suitable for this case, as it produces a large number of oscillations that lead to numerical instabilities and, consequently, the interruption of the simulation.

The following table summarizes the results obtained:

TABLE 2.8. INFLUENCE OF NUMERICAL FLUXES FOR PRANDTL-MEYER EXPANSION CASE

Numerical flux	Mach number	Simulation time [s]	Relative error
Lax-Friedrichs	4.950	40.3	0.1105
HLL	4.959	48.0	0.1089
HLLC	4.966	40.3	0.1077

All three fluxes yield similar performance, with relative errors around 11% and comparable simulation times. The slight differences in accuracy are mainly due to the level of numerical diffusion each flux introduces to capture discontinuities. Greater diffusion results in more smoothing of gradients, which can negatively impact solution accuracy.

In this case, the HLLC flux appears to be the most suitable option, as it provides the highest accuracy while maintaining a simulation time equivalent to the others.

2.3.5. Influence of AMR

Adaptive mesh refinement (AMR) is an adaptive meshing technique in which the mesh is dynamically updated at user-defined time intervals. In regions of the domain where gradients are high, the mesh is automatically refined, while in smoother areas it remains unchanged. This technique allows for higher resolution where it is most needed, improving the behavior of the solution.

It is important to note that AMR involves a very high computational cost, making it useful primarily in problems that require fine resolution in regions with steep gradients.

The following is a comparison between two simulations: one with AMR enabled and one without it.

TABLE 2.9. INFLUENCE OF AMR TECHNIQUES FOR PRANDTL-MEYER EXPANSION CASE

AMR	Mach number	Simulation time [s]	Relative error
ON	5.496	835.0	0.0124
OFF	4.571	16.5	0.1785

As previously mentioned, the use of AMR comes with a very high computational cost, being approximately 50 times slower. On the other hand, the relative error is reduced from 17.85% to 1.24%, representing a 93% improvement in accuracy.

Therefore, although AMR involves significant computational costs, its use is justified in problems that require high resolution and precision, provided that sufficient resources are available to handle such demands.

2.4. Conclusions associated with the convergence studies

Both analyses explored key numerical parameters that directly affect the quality and stability of the solution obtained with Trixi.jl. The simulations conducted lead to a fundamental conclusion: improving accuracy or increasing robustness always results in higher computational cost, leading to longer simulation times.

Whether by increasing the polynomial order, refining the mesh, selecting more complex numerical fluxes, or enabling techniques such as shock capturing or AMR, every decision aimed at enhancing solution accuracy and stability translates into greater simulation time and resource demand.

However, greater accuracy does not always lead to a significantly better final result. There are thresholds beyond which the gains in precision become negligible compared to the increase in cost.

Therefore, this study highlights the need to find a balance between three fundamental factors:

- The solution must be sufficiently accurate.
- The simulation must remain stable, especially in the presence of strong discontinuities.
- The computational cost must be reasonable, allowing for efficient use of available resources.

This balance is essential in engineering applications, where the goal is to perform multiple simulations, optimize designs, or integrate them into more complex workflows. Choosing an appropriate numerical configuration is not only a matter of accuracy, but also of computational feasibility and sustainability.

3. STUDY OF PLASMA IN A MAGNETIC ARC

The objective of this section is to define the physical model to be simulated using Trixi.jl, for later comparison with the results presented in the reference article by Merino et al., from which the following information on the physical model has been obtained [3]. This model falls under the category of Magnetic-Arch Thruster (MAT) devices and proposes a two dimensional, bifluid formulation with ion species i and electron species e.

The following assumptions are established:

- Fully ionized, collisionless, and quasi-neutral plasma.
- Perfectly magnetized, quasi-Maxwellian electrons with no inertia and a polytropic closure.
- Cold ions with unit charge and arbitrary magnetization. It is assumed that ions remain cold downstream, neglecting any shock-like discontinuity in temperature or distribution.
- Planar symmetry geometry, as an intermediate step toward a three dimensional configuration.

The conservative form of the equations, defined in the article [3] and implemented in the Julia-based model, is as follows:

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \mathbf{F} = \mathbf{R} \tag{3.1}$$

where,

$$\mathbf{Q} = \begin{bmatrix} n \\ nu_{zi} \\ nu_{xi} \\ nu_{yi} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} nu_{zi} & nu_{xi} \\ nu_{zi}^2 + n^\gamma & nu_{zi}u_{xi} \\ nu_{zi}u_{xi} & nu_{xi}^2 + n^\gamma \\ nu_{zi}u_{yi} & nu_{xi}u_{yi} \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 0 \\ -n(H_e' + u_{yi})B_x \\ n(H_e' + u_{yi})B_z \\ n(u_{zi}B_x - u_{xi}B_z) \end{bmatrix}. \tag{3.2}$$

The boundary conditions are defined as follows:

- **Symmetry plane at** $x = 0$: the flux perpendicular to this plane is zero.
- **Plasma inlet:**: located between $x = 6$ and $x = 4$.Although this inlet range is specified in the article, it is extended up to the position of the current-carrying wires to avoid regions with zero density, following the same approach as in the original work.

- **Plasma outlets**: located along the right, top, and left edges of the domain, excluding the inlet region.

The magnetic field is generated by four wires located at $x = 7$, $x = 3$, $x = 3$, and $x = 7$, such that the total current is zero. According to the article, the magnetic field is normalized so that its magnitude equals 1 at the center of the plasma outlet ($x = 5$), which matches the ion magnetization level. To avoid singularities near the wires, the decision was made to displace them by 0.125 units in the negative z-direction, in order to prevent Trixi.jl from encountering instabilities in the regions close to the wires. This decision may introduce slight inaccuracies, as the variables defined at the inlet may not exactly match the imposed conditions. Nevertheless, this offset is considered small, and the results are assumed to remain valid for comparison purposes.

The electron properties, such as the energy H_e and the out-of-plane velocity U_{ye}, are defined throughout the entire domain starting from the inlet plane ($z = 0$). This is possible because, given that the electrons are fully magnetized, the value of H_e is conserved along magnetic field lines. Therefore, once the inlet conditions are known, it is possible to propagate both H_e and U_{ye} throughout the domain.

H_e is calculated using the following expression:

$$H_e(\psi_B) = \frac{\gamma}{\gamma - 1}\left(n^{\gamma-1} - 1\right) - \phi. \tag{3.3}$$

Assuming that the electrostatic potential is zero at the plasma inlet, the initial equation propagated by interpolation along the magnetic lines is:

$$H_e(\psi_B) = \frac{\gamma}{\gamma - 1}\left(n(0, x)^{\gamma-1} - 1\right) - \phi \tag{3.4}$$

Once H_e is known throughout the domain, the out-of-plane velocity component of the electrons, U_{ye}, can be determined using the following expression:

$$u_{ye}(\psi_B) = -\frac{1}{B}\frac{\partial H_e}{\partial l_\perp} = -\frac{\mathrm{d}H_e}{\mathrm{d}\psi_B} = -H_e'. \tag{3.5}$$

Another way to obtain U_{ye} is to compute its value at $z = 0$ and propagate it along the magnetic lines, in an analogous way to the approach used for H_e. The equation used for its computation at $z = 0$ is:

$$u_{ye}(0, x) = \frac{1}{B_z} \cdot \frac{\gamma}{\gamma - 1} \cdot \frac{\partial n(0, x)^{\gamma-1}}{\partial x}. \tag{3.6}$$

The initial conditions are defined as follows:

- **Density** $n(x, y)$: a Gaussian profile centered at the plasma inlet region, decreasing by three orders of magnitude toward the edges. The functional form used is:

$$n(x, y) = 10^{-3(x-5)^2} \tag{3.7}$$

- **Axial velocity** v_z: Defined using the local sound speed, such that the ions are sonic at the magnetic throat.

$$v_z = \sqrt{\gamma n^{\gamma-1}} \tag{3.8}$$

- **Radial velocity, out-of-plane velocity and electrostatic potential**: all of them are initially assumed to be zero.

The derivation of these equations is detailed in the reference model [3], therefore, only the final expressions used for the code implementation are included in this document.

Before running the full model simulation, an intermediate step is carried out by computing H_e and U_{ye} using the advection equation.

3.1. Advection of the H_e and U_{ye} variables

The behavior of the variables H_e and U_{ye} is essential for properly understanding plasma dynamics. In particular, U_{ye} determines the magnetic force that appears in the source terms of the system of equations. Moreover, this variable depends directly on the electron energy, H_e.

Given the assumption of a perfectly magnetized and collisionless plasma, these variables can be considered constant along magnetic field lines. This would justify computing them using interpolation, propagating their values from the plasma inlet throughout the domain.

However, in this work, it was decided to explicitly solve the advection equation for these variables. This decision is based on two main reasons:

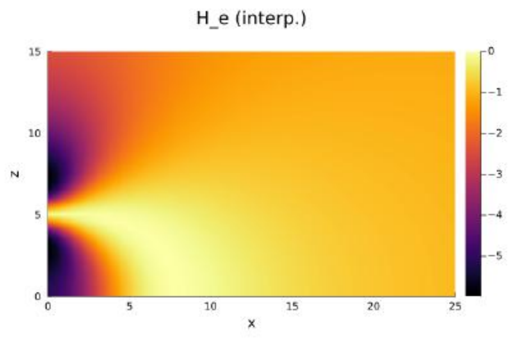
- **Generality and extensibility of the model:** by formulating H_e and U_{ye} as advected variables, it enables future implementation of effects such as collisions or other physical phenomena that may break conservation along magnetic field lines. This approach makes the model more versatile and applicable to more complex or realistic configurations.

- **Modification of a predefined equation:** Since Trixi.jl provides a predefined 2D advection equation, its behavior was tested. However, it only accepted a constant advection velocity, so it was modified to adapt it to the case under study, where the advection velocity is the magnetic field and depends on the position within the domain.

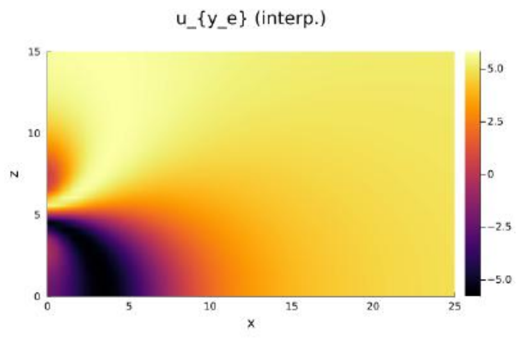
Nonetheless, although this numerical advection-based analysis was carried out, the model used for direct comparison with the results of the reference article [3] relies on the interpolated version of H_e and U_{ye}. This is because that approach allows for an analytically precise definition of these variables, suitable for ideal, collisionless cases, consistent with the physical conditions of the model being compared. The introduction of collisions or non-conservative effects is beyond the scope of this work.

A convergence analysis was performed between the simulated values and the analytical ones, the latter being obtained through interpolation along the magnetic field lines. Unlike the analyses in previous sections, this study focuses solely on the influence of the number of elements and the polynomial degree.

The accuracy of the solutions was evaluated using the Root Mean Square Error (RMSE). The reference solutions used for comparison are presented below.



(a) He interpolated



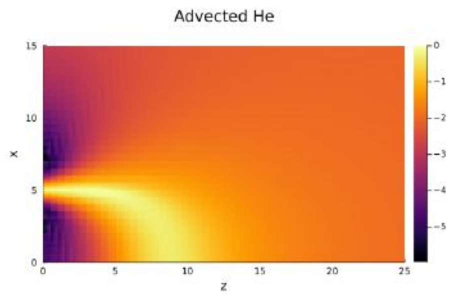
(b) Uye interpolated

Fig. 3.1. Interpolated solution for H_e and U_{ye}.

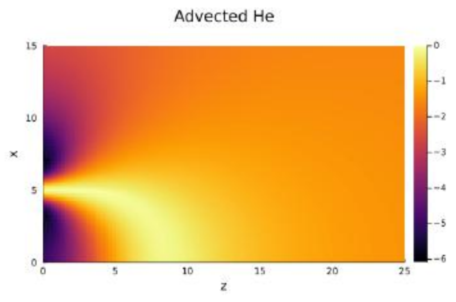
3.1.1. Influence of number of elements

In this case, a Lax–Friedrichs numerical flux was used, with a polynomial degree of $p = 1$ and the SSPRK104 time marching method. The meshes evaluated consist of 1 500, 6 000, 13 500, and 37 500 elements. The time step (dt) was chosen as the largest possible value that ensures the solution remains as fast and stable as possible.

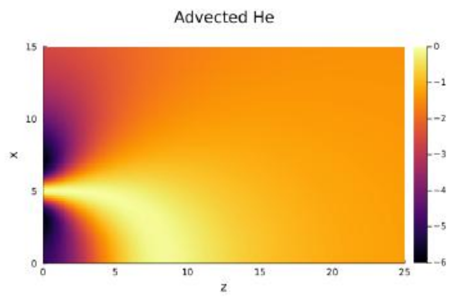
Results obtained for H_e:



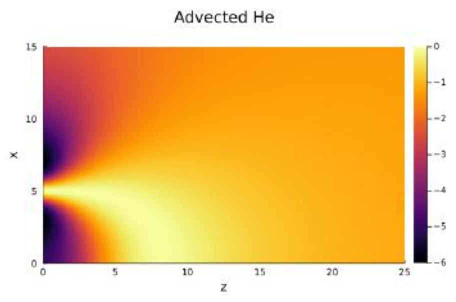
(a) 1 500 elements



(b) 6 000 elements



(c) 13 500 elements



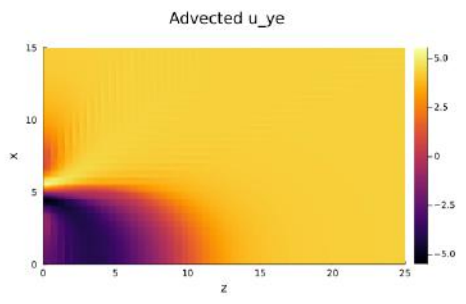
(d) 37 500 elements

Fig. 3.2. Influence of number of elements for H_e.

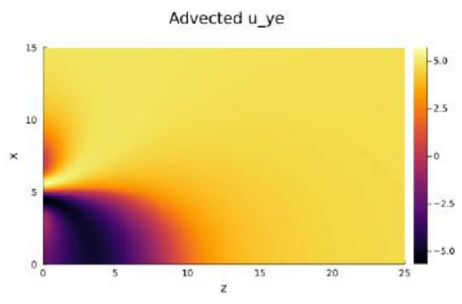
TABLE 3.1. INFLUENCE OF NUMBER OF ELEMENTS FOR H_E.

Number of elements	**RMSE**	**Simulation time [s]**	**dt**
1 500	0.8467	10.8	1.0
6 000	0.4219	69.9	0.5
13 500	0.2848	220	0.35
37 500	0.1732	1058	0.2

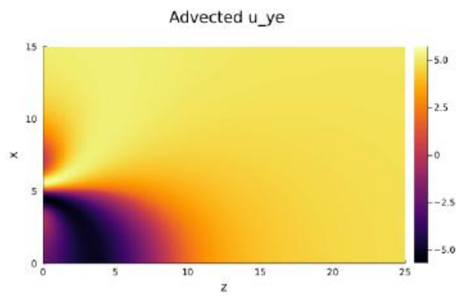
Results obtained for U_{ye}:



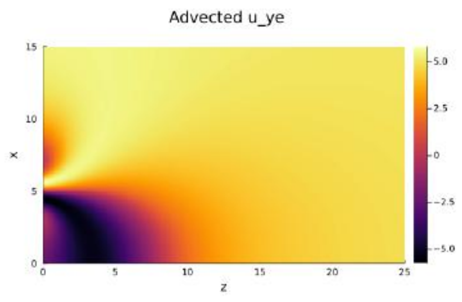
(a) 1 500 elements



(b) 6 000 elements



(c) 13 500 elements



(d) 37 500 elements

Fig. 3.3. Influence of number of elements for U_{ye}.

TABLE 3.2. INFLUENCE OF NUMBER OF ELEMENTS FOR U_{YE}.

Number of elements	**RMSE U_{ye}**	**Simulation time [s]**	**dt**
1 500	0.8106	10.2	1.0
6 000	0.4402	70.4	0.5
13 500	0.3092	222	0.35
37 500	0.1935	1051	0.2

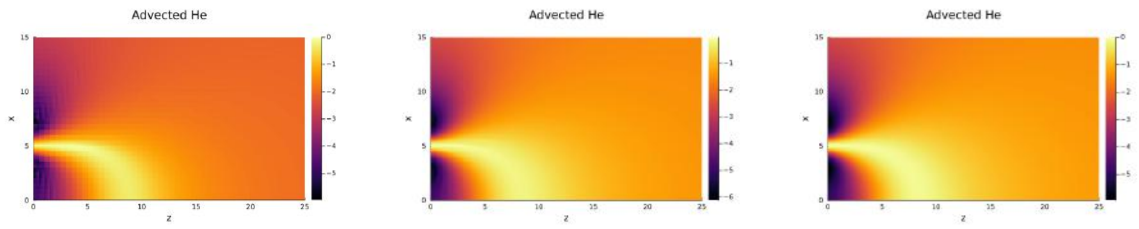
As shown in the table, increasing the number of elements improves the accuracy of the simulation. However, even with the finest mesh, the errors remain relatively high. This may be due to the propagation of local errors throughout the domain, an effect that can be amplified by the artificial diffusion introduced by the Lax–Friedrichs flux. In regions with higher gradients, this leads to smoothing of those gradients and a widening of such areas, contributing to a loss of accuracy in regions far from the initial values.

This behavior can be seen more clearly in the images: at the plasma outlet, the values are higher than in the rest of the domain, deviating from the theoretical behavior in which the quantities are conserved along magnetic field lines. This contradicts the assumption of fully magnetized electrons, under which perfect advection of these quantities along the magnetic lines would be expected.

3.1.2. Influence of polynomial order p

For this second analysis, the effect of the polynomial order p is evaluated, using a fixed mesh of 1 500 elements. The remaining parameters are the same as in the previous analysis.

Results obtained for H_e:



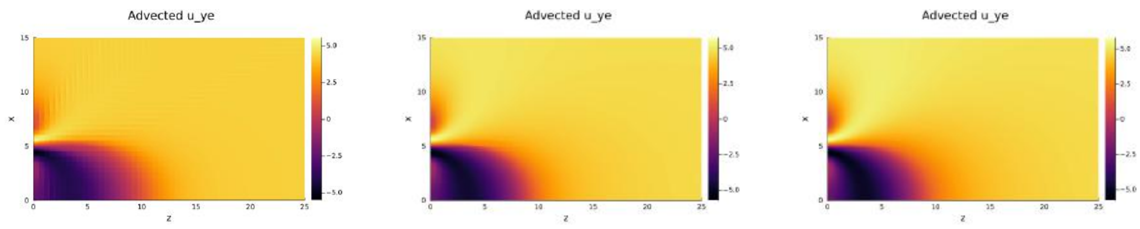
(a) $p = 1$ (b) $p = 2$ (c) $p = 3$

Fig. 3.4. Influence of polynomial order for H_e.

TABLE 3.3. INFLUENCE OF POLYNOMIAL ORDER FOR H_E.

Polynomial order p	**RMSE**	**Simulation time [s]**	**dt**
1	0.8467	10.8	1.0
2	0.4078	49.3	0.4
3	0.3005	172	0.2

Results obtained for U_{ye}



(a) $p = 1$ (b) $p = 2$ (c) $p = 3$

Fig. 3.5. Influence of polynomial order for U_{ye}.

TABLE 3.4. INFLUENCE OF POLYNOMIAL ORDER FOR U_{YE}.

Polynomial order p	**RMSE**	**Simulation time [s]**	**dt**
1	0.8106	10.2	1.0
2	0.4607	48.7	0.4
3	0.3354	172	0.2

As the polynomial order increases, a significant improvement in error is observed, specifically, a 51.8% reduction when moving from $p = 1$ to $p = 2$, with a simulation time that is 4.5 times longer. Despite the increase in runtime, the solution with $p = 2$ is more accurate and efficient than the one obtained with 6 000 elements and $p = 1$, suggesting that increasing the polynomial order is more efficient than refining the mesh.

For $p = 3$, although the error decreases by an additional 26% compared to $p = 2$, the simulation time triples. Nevertheless, a runtime of 172 seconds with 1 500 elements and $p = 3$ yields an error comparable to that obtained with 13 500 elements and $p = 1$, but with 1.3 times less computational time. This comparison reinforces the earlier conclusion that, for this type of problem, increasing the polynomial order is more efficient than increasing the number of elements.

Two main conclusions can be drawn from this study: First, the model based on explicitly solving the advection equations, in which H_e and U_{ye} serve as useful approximations, allows for the future inclusion of electron–ion collisions, which would be modeled via source terms. Second, the results show that accumulated numerical errors significantly affect solution accuracy, especially when using numerical fluxes that introduce artificial diffusion, such as Lax–Friedrichs.

Therefore, to advance toward more realistic models where electron–ion collisions influence the solution, it would be necessary to improve the numerical scheme, for example, by using fluxes with lower artificial diffusion, increasing both the polynomial order and mesh resolution, or incorporating adaptive refinement techniques. These improvements would help ensure greater fidelity in solving the problem.

3.2. Comparison with the reference model

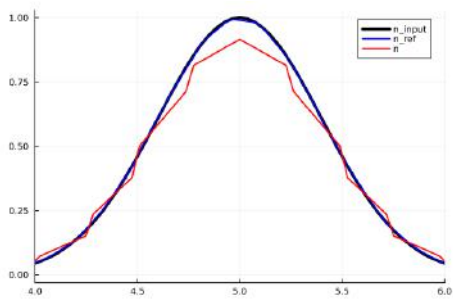
The next step is to compare the solution obtained using the model developed in Julia with the Trixi.jl framework against the reference model [3]. For this simulation, a mesh of 6 000 elements was used, with 100 elements in the z-direction and 60 in the x-direction, along with a Lax–Friedrichs numerical flux, the SSPRK104 time marching method, and a polynomial order of $p = 1$.

The choice of the SSPRK104 method is due to its stability-preserving properties compared to other methods. On the other hand, the polynomial order was limited by computational cost, higher order requires smaller time steps to ensure stability, which significantly increases the overall cost. The choice of the remaining parameters is detailed in Appendix A.

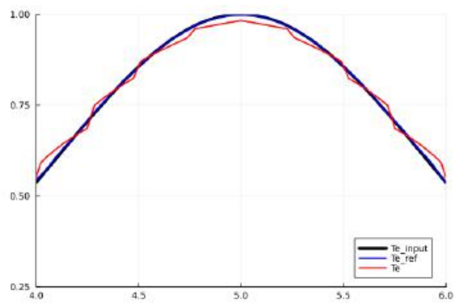
To compare both models, two regions are considered: the plasma outlet region, defined between $x = 4$ and $x = 6$ at $z = 0$; and the line located at $x = 5$, which spans the entire domain.

The first region is analyzed only at the plasma outlet, as both the Julia-based model and the reference model exhibit errors in the upper and lower areas near the outlet, regions without plasma, resulting in solutions that are not physically meaningful.

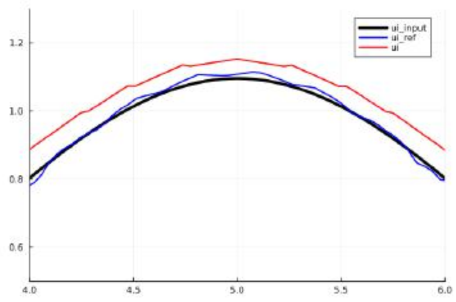
Evaluating the solution along the centerline at the plasma outlet allows for the analysis of variables throughout the expansion. This analysis is necessary to assess the quality of the expansion produced by the Trixi.jl model in comparison with the reference model [3].



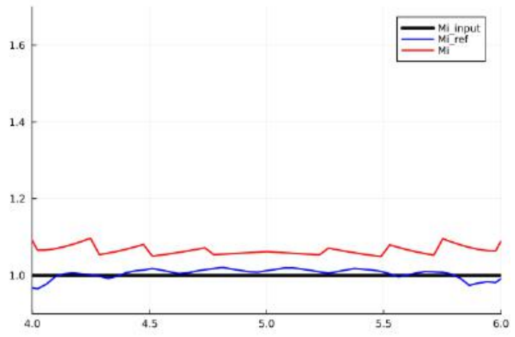
(a) Plasma density



(b) Electron temperature



(c) In-plane Ion velocity



(d) In-plane Mach number

Fig. 3.6. Plasma properties at z = 0.

TABLE 3.5. RMSE FOR $Z = 0$

Variable	**RMSE** ($z = 0$)
Density	0.0181
Electron Temperature	0.1917
Electrostatic Potential	0.2132
In-plane Ion Velocity	0.5485
In-plane Mach Number	–

The variables to be compared are: density, temperature, velocity, and Mach number, as shown in Figure 3.6. In addition to comparing these variables with each other, they are also compared against the expected inlet values, plotted with black lines.

Due to the relationship between the four variables, one of them, the density, has a direct influence on all the others. In the Trixi.jl simulation, the inlet density reaches around 0.9, whereas it should reach 1 at the center of the outlet, resulting in an error of

0.0181. This error in the inlet density, which is defined at the start of the simulation, may be attributed to how boundary conditions are handled in Trixi.jl: through ghost nodes, where the transfer of information from these ghost nodes to the inlet nodes occurs via the numerical flux, thereby altering the imposed inlet conditions relative to those initialized in the code.

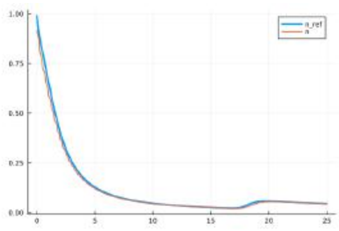
To address this, a numerical flux capable of properly handling numerical dissipation between ghost nodes and inlet nodes would be required. Another possible solution is to reduce the size of the elements, in other words, to increase the number of elements in order to achieve higher accuracy at the boundary. However, this would also result in a significantly higher computational cost.

As for the temperature, which is directly linked to the density, the solution obtained with Trixi.jl approximates the expected values, while the reference solution exactly matches the imposed values.

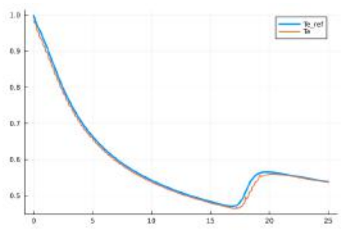
The previously mentioned deviations in density and temperature also lead to errors in velocity. Since both density and temperature are lower, the ion velocity in the simulated case is higher than expected. This affects ion expansion: starting from a higher velocity leads to a slight deviation throughout the domain compared to the reference solution.

It is also important to note that, as mentioned at the beginning of the chapter, in the Julia-based model the current-carrying wires that generate the magnetic field were shifted 0.125 units to the left, outside the domain. This decision, made to avoid singularities, introduces slight geometric differences with respect to the reference model [3].

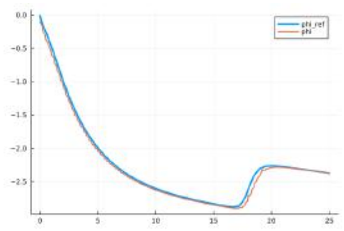
Due to this displacement, the exact positions where the wires should be located no longer coincide with the true geometric center. As a result, the magnetic field exhibits a non-zero B_x component, whereas under ideal conditions only a B_z component should exist. This deviation introduces discrepancies between the reference model and the one implemented in Julia, and may be one of the reasons for the observed differences between both models.



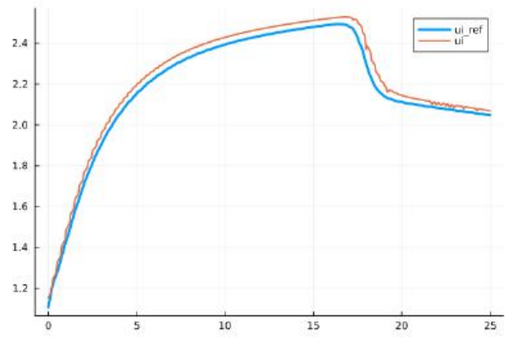
(a) Plasma density



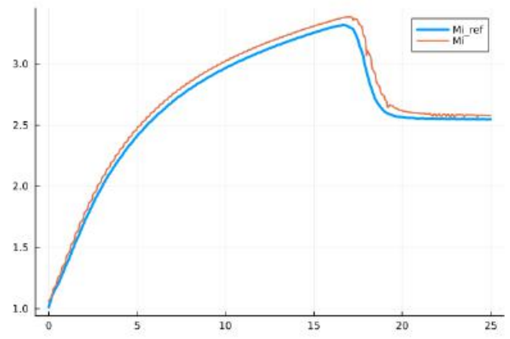
(b) Electron temperature



(c) Electrostatic potential



(d) In-plane ion velocity



(e) In-plane Mach number

Fig. 3.7. Plasma properties at x = 5.

TABLE 3.6. RMSE FOR X = 5

Variable	**RMSE** ($x = 5$)
Density	0.0159
Electron Temperature	0.4739
Electrostatic Potential	0.0490
In-plane Ion Velocity	0.0418
In-plane Mach Number	0.0699

By analyzing the downstream variables, now including the electrostatic potential as one of the variables under consideration, it can be observed that the shock wave, generated by the collision of the two ion beams at the symmetry plane, occurs slightly earlier in the reference solution than in the one obtained using Trixi.jl. This difference may be due to the fact that, starting with a higher initial velocity, the ions in the Trixi.jl simulation take longer to reach the symmetry plane, causing the shock wave to appear later.

After the shock wave, the expected behavior is observed: electron density, electron temperature, and electrostatic potential increase, while ion velocity and Mach number decrease. Although differences between both solutions remain after the shock, they follow the same physical trend, which is consistent with the initial discrepancy. What would be more concerning is a case where, after the shock wave, the velocity and Mach number drop below the reference values, while the electron density, temperature, and electrostatic potential rise above them. Such behavior would suggest that the flux being used introduces errors upstream of the shock.

On the other hand, the table shows that all variables exhibit low errors, except for the electron temperature, which presents an error of 0.4739, primarily caused by the region at the entrance to the shock wave.

Conclusions

The results show that the model implemented in Julia using the Trixi.jl framework consistently reproduces the plasma dynamics in a Magnetic Arch Thruster, in accordance with the reference model.

The differences observed are mainly due to the displacement of the current-carrying wires and the inlet conditions, which introduce errors but do not alter the overall plasma behavior. Therefore, it can be concluded that the developed tool is valid for studying and predicting the plasma behavior in a MAT based on the reference model.

3.3. Parametric analysis

Once the model implemented in Julia was developed and verified, a study of plasma behavior was conducted by modifying different parameters. Initially, three studies were planned: the first by varying γ, the second by increasing ion magnetization, and the third by changing the position of the wires.

The last two studies could not be completed due to the requirement of very small time steps, a high computational cost, and the fact that the solution failed to converge in any case. After thoroughly reviewing the code for potential errors, no specific cause was identified. Therefore, the analysis focuses solely on how plasma dynamics change with variations in the adiabatic constant γ. For this purpose, three cases are compared: a nearly isothermal flow with $\gamma = 1.01$, and adiabatic flows with $\gamma = 1.2$ and $\gamma = 1.67$.

3.3.1. Comparison of plasma dynamics under varying adiabatic coefficients

This study analyzes the impact of varying the adiabatic constant γ on the behavior of the plasma generated in a Magnetic Arch Thruster (MAT), allowing for a better understanding of how the thermodynamics of the gas influence plasma dynamics.

Before analyzing the results obtained, it is important to theoretically examine how changes in the adiabatic constant are expected to affect plasma behavior.

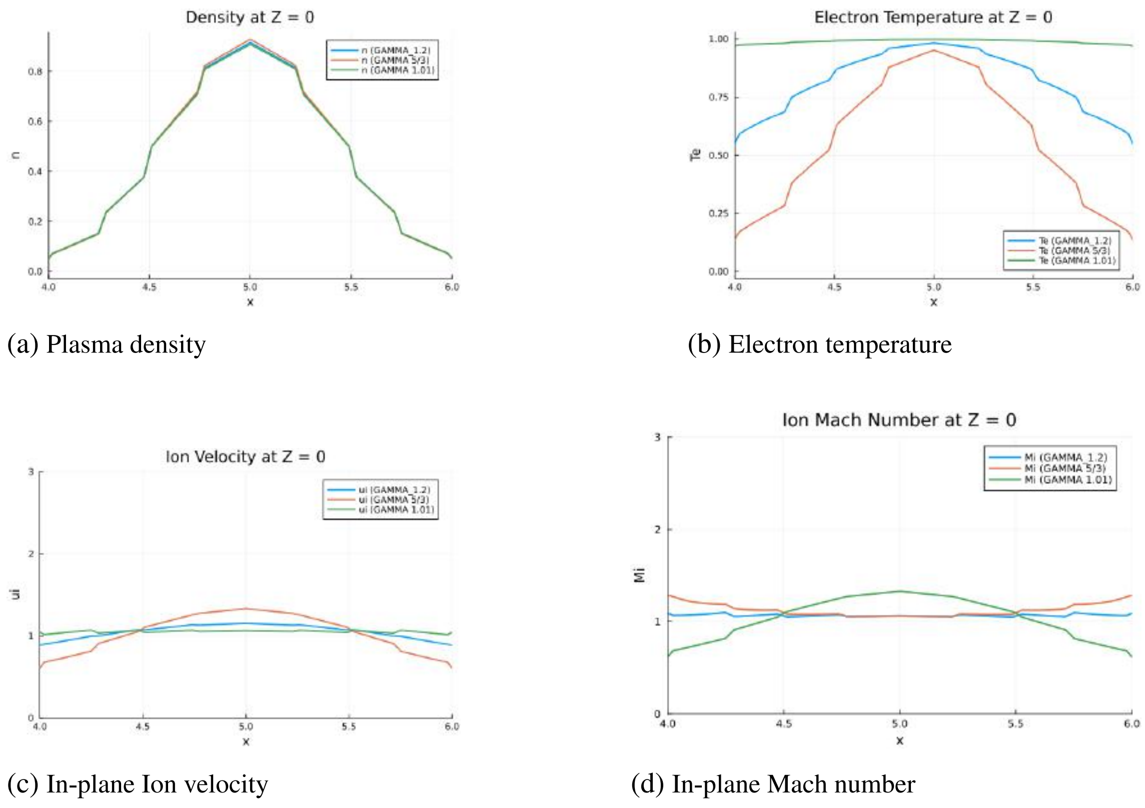
The adiabatic constant γ governs the relationship between pressure and density through the polytropic equation of state adopted in the model, expressed as $p = n^\gamma$, due to the absence of an energy equation. From this relation, the pressure gradient can be expressed as $\nabla p = \gamma n^{\gamma-1}\nabla n$, showing that, for the same variation in density, a higher γ value will result in a stronger pressure gradient.

This pressure gradient acts as the source of the ambipolar field that arises in quasi-neutral plasmas, responsible for accelerating ions and decelerating electrons to maintain quasi-neutrality. This ambipolar electric field can be expressed as $E = -\nabla\phi$.

Based on this information, we can deduce that in a nearly isothermal state, with $\gamma = 1.01$, the electron temperature remains nearly constant throughout the domain, resulting in smoother gradients and, therefore, lower ion acceleration. This should cause ions to collide sooner along the symmetry axis, generating a shock wave closer to the plasma outlet. Additionally, due to lower axial acceleration, the ions exhibit a broader transverse spread, resulting in increased dispersion across the domain.

On the other hand, for $\gamma = 1.67$, which corresponds to a monoatomic gas, temperature variations are more pronounced, resulting in stronger pressure gradients and, consequently, higher axial velocities. This generates the shock wave farther from the outlet and leads to reduced ion dispersion.

The intermediate case, with $\gamma = 1.2$, represents a value that can be empirically observed in magnetic nozzles [7]. In this regime, both ion acceleration and pressure gradients take on intermediate values, resulting in a mixed behavior, with shock wave position and ion dispersion falling between those of the other two cases.



(a) Plasma density (b) Electron temperature

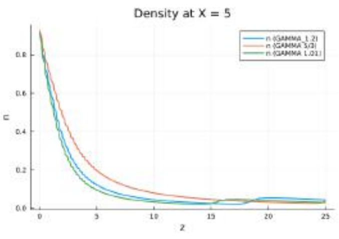
(c) In-plane Ion velocity (d) In-plane Mach number

Fig. 3.8. Plasma properties at z = 0.

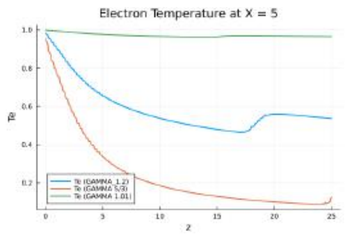
In graphs 3.8, the profiles of plasma density, electron temperature, and both the in-plane velocity and Mach number of the ions at the outlet can be observed. As expected, since the same numerical flux is used, the densities for the three cases have similar values.

Regarding the temperature, we can see that for $\gamma = 1.01$, the temperature is practically constant, representing an ideal case where $\gamma = 1.01$, whereas for the cases of $\gamma = 1.2$ and $\gamma = 1.67$, a progressive decrease is observed at the edges.

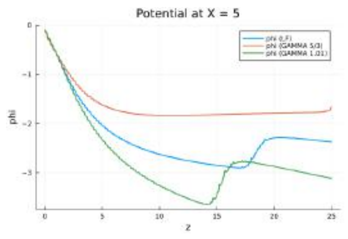
The difference in the temperature profile is reflected in the ion velocity profile. For $\gamma = 1.67$, the ions reach higher velocities at the beginning; this happens because the initial ion velocity is defined as the speed of sound, so if there is no significant temperature change at the center of the outlet, the speed of sound, and therefore the ion velocity, will be higher for greater adiabatic coefficients. In contrast, for a more pronounced change, as occurs at the edges of the outlet, a lower adiabatic coefficient will result in a higher velocity.



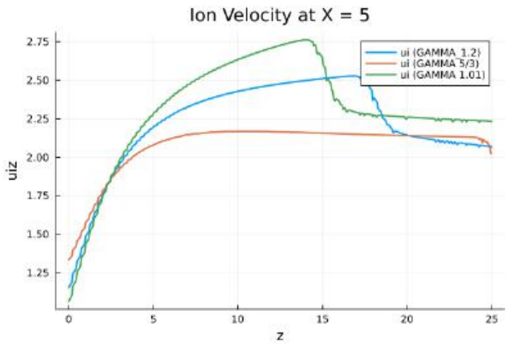
(a) Ion density



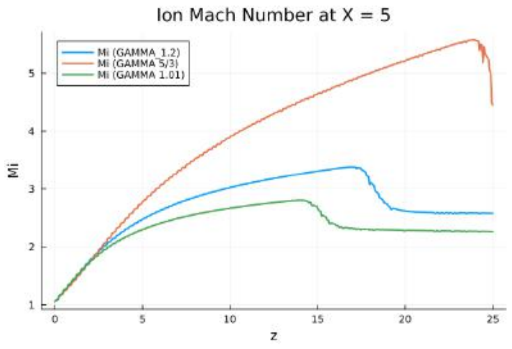
(b) Ion temperature



(c) Potential



(d) Ion velocity



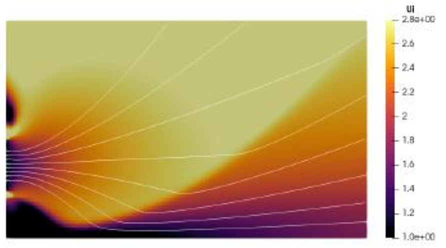
(e) Mach number

Fig. 3.9. Plasma properties at x = 5.

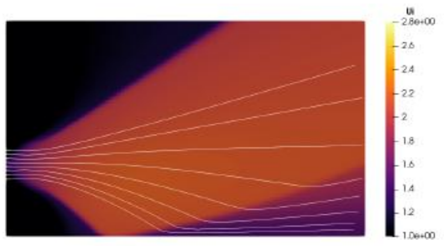
In Figure 3.9, the same variables are shown, except for the electrostatic potential, which has been introduced for this case.

First, it is observed that the electron temperature undergoes a greater change as the adiabatic coefficient increases. For the case of $\gamma = 1.67$, although there is a significant exchange of thermal energy from the electrons to the kinetic energy of the ions, the ambipolar field will be smaller for this case, becoming more pronounced for lower values of γ.

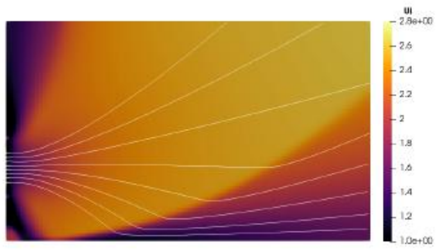
This means that the acceleration of the ions is lower, with a null acceleration around $z = 7$, where the ions have a constant velocity. Having a lower ion velocity does not imply a lower Mach number; this is calculated as $M_i = \frac{u_i}{\sqrt{\gamma T_e}}$. Therefore, at lower temperature, the speed of sound is lower and the Mach number is higher for a given velocity. This is why, although lower adiabatic coefficients imply higher ion velocities, we also have a relation with a lower speed of sound, resulting in higher Mach numbers for higher adiabatic coefficients, see Figures 3.9d and 3.9e.



(a) Streamlines for $\gamma = 1.01$



(b) Streamlines for $\gamma = 1.67$



(c) Streamlines for $\gamma = 1.2$

Fig. 3.10. Streamlines and in-plane ion velocity.

Finally, as seen in Figures 3.10, the dispersion of the ions is greater for lower adiabatic coefficients. This behavior has an effect on the distance at which the shock wave is produced; the higher the γ, the farther from the plasma exit region it will be generated.

3.3.2. Conclusions

The analysis carried out demonstrates how the adiabatic coefficient influences the behavior of the plasma. During the analysis, patterns consistent with the previously explained theory have been identified, which validates both the physical model and its numerical implementation in Julia using the Trixi.jl framework.

First, it has been confirmed that, the higher the γ, the lower the dispersion of the ions, caused by a higher initial axial velocity, which generates a shock wave farther from the plasma exit. Moreover, it has been observed that as this parameter increases, the electrostatic potential and, consequently, the ion velocity tend to stabilize.

On the other hand, the value of the Mach number stands out for showing high values even under low-velocity conditions, due to the reduction of the plasma temperature.

In conclusion, this study confirms that the adiabatic constant not only determines the thermodynamics of the plasma, but also directly affects macroscopic variables such as the ion geometry, the position of the shock wave, and the intensity of the ambipolar field.

4. CONCLUSIONS

During the development of this project, both known cases and the simulation of plasma flows in a Magnetic Arch Thruster (MAT) have been implemented and evaluated using the Julia programming language and the Trixi.jl framework, where the main objective has been to verify the capability of this open-source tool to accurately reproduce the plasma physics of the model described in the reference article [3], validating the results through the comparison of both solutions.

The results obtained show that Trixi.jl is capable of simulating the plasma dynamics. All variables have shown behavior consistent with the reference model. Furthermore, through the parametric study carried out by varying the adiabatic coefficient, it can be observed that the model integrated in Julia behaves physically and predictably, showing a trend in the analyzed variables consistent with the theoretical behavior described.

In terms of computational performance, Julia has great efficiency, combining the readability of a high-level language with execution speeds close to those of the C language [5]. On the other hand, Trixi.jl has offered flexibility, extensibility, and good documentation, facilitating both the development of simulations and the post-processing of results.

Nevertheless, it is worth highlighting some specific limitations of using Trixi.jl that have been identified during the course of the project. The first to note is that it does not allow querying the solution at an arbitrary point in the domain, which limits the implementation of some visualization techniques. For example, to represent streamlines or carry out more detailed post-processing, the use of external tools such as ParaView is necessary, which introduces an extra layer of complexity. On the other hand, although the active development of Trixi.jl is an advantage in terms of continuous improvement, it also implies long-term instability, as some functions may change their syntax, parameters, or disappear overnight. This can break previously functional implementations if one intends to use the most up-to-date version of the framework and requires constant attention to its updates.

Despite these limitations, which are usability issues rather than numerical capability issues, this work has shown that Julia and Trixi.jl are viable tools for plasma simulation, combining efficiency, accuracy, and flexibility. Being open-source, modular, and supported by a growing community of users and developers makes them a strong candidate for future research, especially in early design phases. Moreover,during the development of this project, several questions were raised regarding functionalities such as how to stop the simulation when steady state was reached. Responses took between one and two days and were very helpful. This community engagement reinforces Trixi.jl's suitability for future research, especially in early-stage academic and design environments.

In short, this work has demonstrated that it is possible to integrate plasma simulations such as those in the reference article [3] using Julia and the Trixi.jl framework, open-source and constantly evolving tools, forming a foundation for future developments in the field of electric propulsion.

5. FUTURE WORK

Once it has been demonstrated that Julia and Trixi.jl can accurately reproduce the described dynamics, it is important to mention potential improvements and future implementations that could be developed.

Firstly, the implementation of the energy equation could be considered in order to avoid relying on the polytropic closure used in the current model. Among other factors, this would increase the computational cost, since adding an extra variable raises the degrees of freedom in the system and, therefore, the number of operations required to obtain the solution. However, it would allow a more realistic behavior of plasma dynamics.

On the other hand, the current model does not include collisions between ions and electrons, so adding them may be interesting to observe the interaction between these species, as well as the effects of dissipation and momentum transfer.

Finally, moving from a 2D to a 3D model appears to be a reasonable step. Although it would require modifying the equations to be solved, leading to an increase in computational cost, requiring more powerful hardware, and a more complex postprocessing procedure to visualize the data, it would also represent the real nozzle geometry in a more reliable way than in the 2D case.

Implementing the modifications outlined above would not only allow for a deeper understanding of plasma dynamics in the MAT, but also help improve the Trixi.jl codebase, by adding new systems of equations or even creating predefined functions for visualizing results without the need for external tools.

BIBLIOGRAPHY

[1] C. Boyé, J. Navarro-Cavallé, and M. Merino, “Ion current and energy in the magnetic arch of a cluster of two ecr plasma sources,” *Journal of Electric Propulsion*, vol. 4, no. 10, 2025. DOI: 10.1007/s44205-025-00100-w.

[2] Trixi Framework. “Trixi.jl documentation (stable branch).” Accessed: June 14, 2025. [Online]. https://trixi-framework.github.io/TrixiDocumentation/stable/. (2025).

[3] M. Merino, D. García-Lahuerta, and E. Ahedo, “Plasma acceleration in a magnetic arch,” *Plasma Sources Science and Technology*, vol. 32, no. 6, p. 065 005, 2023. DOI: 10.1088/1361-6595/acd476.

[4] E. Ahedo and M. Merino, “Two-dimensional supersonic plasma acceleration in a magnetic nozzle,” *Physics of Plasmas*, vol. 17, p. 073 501, 2010. DOI: 10.1063/1.3442736.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. “The julia programming language.” (2025), [Online]. Available: https://julialang.org/.

[6] M. Schlottke-Lakemper *et al.*, *Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*, https://github.com/trixi-framework/Trixi.jl, 2025. DOI: 10.5281/zenodo.3996439.

[7] M. Merino and E. Ahedo, “Influence of electron and ion thermodynamics on the magnetic nozzle plasma expansion,” *IEEE Transactions on Plasma Science*, vol. 43, no. 1, pp. 244–251, 2015. DOI: 10.1109/TPS.2014.2316020.

APPENDIX A: SIMULATION PARAMETERS SELECTION

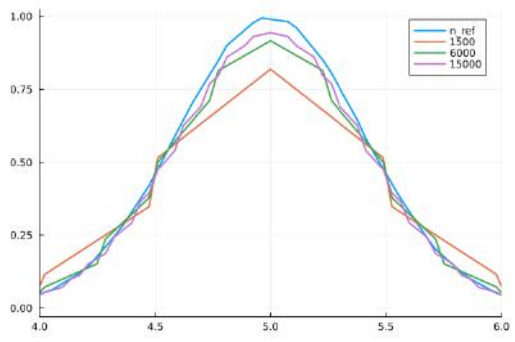
To obtain the configuration that will be used, a convergence study has been conducted, analyzing the number of elements and the numerical flux used.

Both the analysis carried out in this appendix and those in sections 3.2 and 3.3 compare the relevant variables at $x = 5$, that is, from the plasma exit to the downstream region, as well as at the plasma exit, at $z = 0$. This has been fixed in this way since what matters is to study the behavior of the plasma at the thruster exit and what happens downstream from its center.

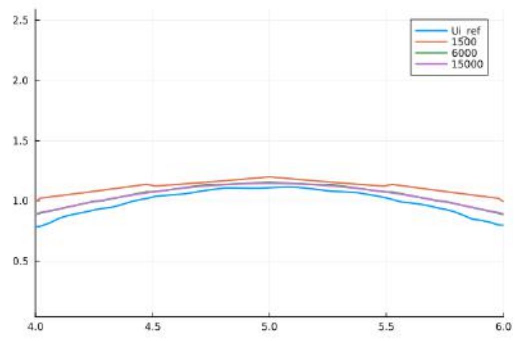
In addition to the plots at $z = 0$, a table is shown with simulation times and errors calculated using the Root Mean Square Error (RMSE). The variables considered for the analysis are density, Mach number at $x = 5$, and ion velocity at $z = 0$.

Number of elements

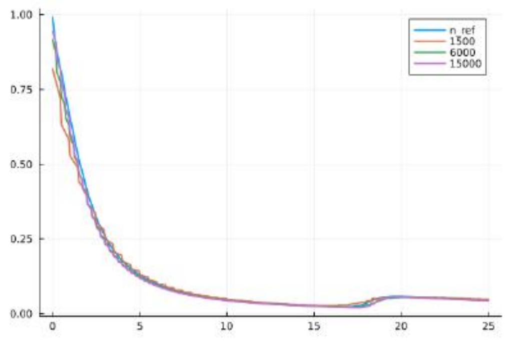
The first analysis is the comparison of the number of elements. For this, a polynomial degree $p = 1$, the time marching method SSPRK104, and Lax-Friedrichs as the numerical flux have been fixed. The number of elements compared are: 1 500, 6 000, and 15 000 elements.



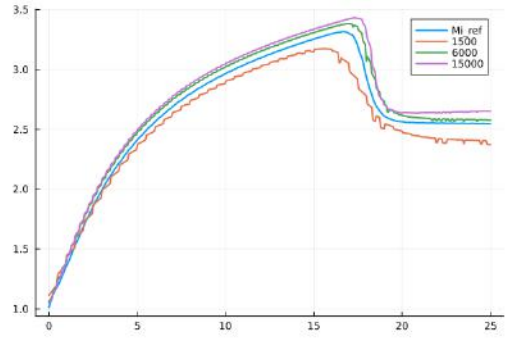
(a) Density (z = 0)



(b) Ui (z = 0)



(c) Density (x = 5)



(d) Mach (x = 5)

Fig. A.1. Impact of the number of elements compared with reference solution.

TABLE A.1. ERRORS AND SIMULATION TIME FOR $Z = 0$.

Number of elements	**RMSE density** ($z = 0$)	**RMSE** u_i ($z = 0$)	**Simulation time [s]**
1 500	0.0412	0.9680	46.6
6 000	0.0181	0.5485	287
15 000	0.0120	0.6060	1736

TABLE A.2. ERRORS AND SIMULATION TIMES FOR $X = 5$.

Number of elements	**RMSE density** ($x = 5$)	**RMSE Mach** ($x = 5$)	**Simulation time [s]**
1 500	0.0302	0.1117	46.6
6 000	0.0159	0.0699	287
15 000	0.0122	0.1064	1736

The first thing we can notice is that the density at the inlet conditions is lower than the reference one, which will produce errors in the other variables. This error in the inlet density, which is defined at the beginning of the simulation, may be due to the fact that boundary conditions in Trixi.jl are handled through ghost nodes, where the transfer from these ghost nodes to the inlet nodes is done through the numerical flux, which modifies the inlet conditions compared to those initialized in the code.

Continuing with the density, along the domain at $x = 5$, we observe that the behavior is very close to the reference, and moreover, the solution for 6 000 and 15 000 appears to be practically identical. This similarity can be seen in the table, where we observe that the difference in error is 0.0037.

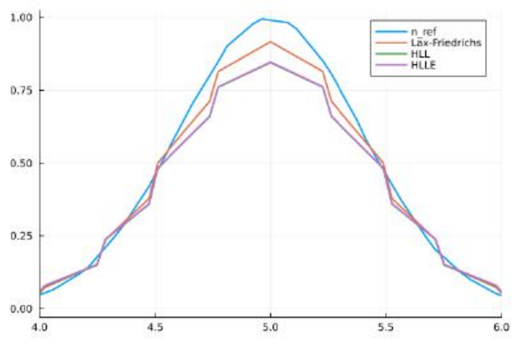
The behavior of the ion velocity is the same as for the density; the difference between 1 500 and 6 000 elements is 43.3%, while between 6 000 and 13 500 elements we observe an increase in error.

Comparing the Mach number, we see that the results for 15 000 and 6 000 elements are similar, whereas the one with 1 500 deviates from the reference solution.

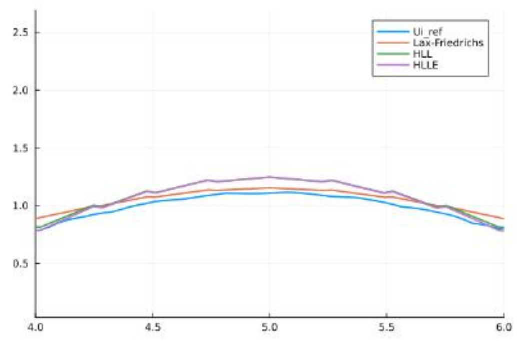
All of this, added to the fact that with 15 000 elements the simulation time is 6 times longer than with 6000 elements, allows us to conclude that the number of elements to be used is 6 000.

Numerical flux

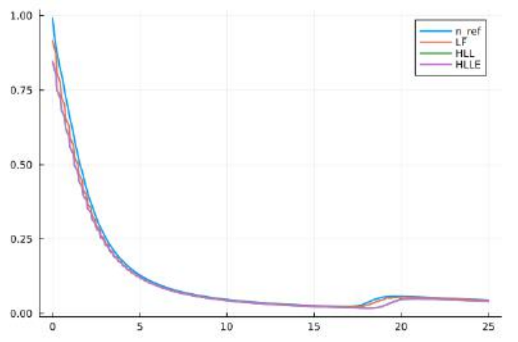
For the comparison of different numerical fluxes, as in the previous analysis, a polynomial order $p = 1$, a time marching method SSPRK104, and a mesh of 6 000 elements have been used. The fluxes compared are: Lax-Friedrichs, HLL, and HLLC.



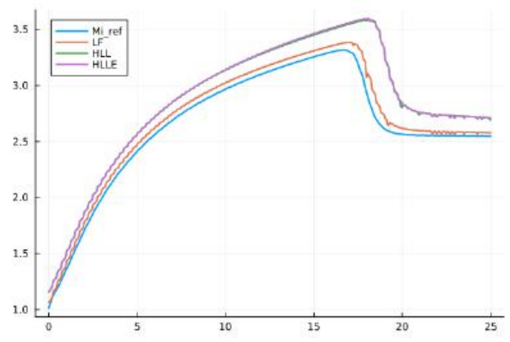
(a) Density (z = 0)



(b) Ui (z = 0)



(c) Density (x = 5)



(d) Mach (x = 5)

Fig. A.2. Impact of the numerical flux compared with reference solution.

TABLE A.3. ERRORS AND SIMULATION TIMES FOR $Z = 0$.

Numerical flux	**RMSE density** ($z = 0$)	**RMSE Mach** $z = 0$	**Simulation time [s]**
Lax-Friedrichs	0.0181	0.5485	287
HLL	0.0309	0.6901	244
HLLE	0.0305	0.5704	230

TABLE A.4. ERRORS AND SIMULATION TIMES FOR $X = 5$.

Numerical flux	**RMSE density** ($x = 5$)	**RMSE Mach** ($x = 5$)	**Simulation time [s]**
Lax-Friedrichs	0.0159	0.0699	287
HLL	0.0280	0.2567	244
HLLE	0.0276	0.2594	230

As can be observed both in the graphs and in the tables, the behavior of the HLL and HLLE numerical fluxes is practically identical. However, both show worse performance than the Lax-Friedrichs flux. Regarding the density at the outlet, the latter approaches unity better than the others, with an error improvement of 70.17%. For the ion velocity, a similar behavior to that of the density is observed, where the HLLE and HLL fluxes behave similarly, while the Lax-Friedrichs flux provides a better approximation to the reference solution.

As for the variables at $x = 5$, it can be seen that the HLL and HLLE fluxes enter the shock wave with more delay compared to the Lax-Friedrichs flux, the latter being the one that gives the best results when compared to the reference.

Given these results, the flux used for the comparison between the model implemented in Julia and the reference model will be the Lax-Friedrichs flux.

Conclusions

Analyzing the data obtained, it is concluded that, in order to perform the comparison between the numerical solution obtained using Trixi.jl and the reference solution extracted from the reference article, the most suitable configuration consists of using a mesh of 6 000 elements, the Lax-Friedrichs numerical flux, a polynomial order $p = 1$, and the SSPRK104 time marching method.

Although the simulation times for Lax-Friedrichs are slightly higher, being able to use 6 000 elements mitigates this drawback, providing a balance between computational cost and accuracy.

APPENDIX B: DECLARATION OF USE OF GENERATIVE IA IN BACHELOR THESIS (TFG)

I have used Generative AI in this work

[✓ **YES**]	**NO**

Part 1: Reflection on ethical and responsible behaviour

Please be aware that the use of Generative AI carries some risks and may generate a series of consequences that affect the moral integrity of your performance with it. Therefore, we ask you to answer the following questions honestly (*please tick all that apply*):

Question		
1. In my interaction with Generative AI tools, I have submitted **sensitive data** with the consent of the data subjects.		
YES, I have used this data with permission	NO, I have used this data without authorisation	[✓ NO, I have not used sensitive data]
2. In my interaction with Generative AI tools, I have submitted **copyrighted materials** with the permission of those concerned.		
YES, I have used these materials with permission	NO, I have used these materials without permission	[✓ NO, I have not used protected materials]
3. In my interaction with Generative AI tools, I have submitted **personal data** with the consent of the data subjects.		
YES, I have used this data with permission	NO, I have used this data without authorisation	[✓ NO, I have not used personal data]

4. My use of the Generative AI tool has **respected its terms of use**, as well as the essential ethical principles, not being maliciously oriented to obtain an inappropriate result for the work presented, that is to say, one that produces an impression or knowledge contrary to the reality of the results obtained, that supplants my own work or that could harm people.	
[✓ YES]	NO

If you **did NOT** have the permission of those concerned in any of questions 1, 2 or 3, briefly explain why *(e.g. "the materials were protected but permitted use for this purpose" or "the terms of use, which can be found at this address (...), prevent the use I have made, but it was essential given the nature of the work".*

Part 2: Declaration of technical use

I declare that I have made use of the Generative AI system ChatGPT for:

Documentation and drafting:

- *Revision or rewriting of previously drafted paragraphs*

I have asked for paragraphs to be rewritten to improve the style and make the content easier to follow for the reader.

Part 3: Reflection on utility

Please provide a personal assessment (free format) of the strengths and weaknesses you have identified in the use of Generative AI tools in the development of your work. Mention if it has helped you in the learning process, or in the development or drawing conclusions from your work.

Generative artificial intelligence has proven to be very useful at times, as it has helped me correct errors and improve the way certain parts of the text were written, making the reading experience clearer and easier to follow.