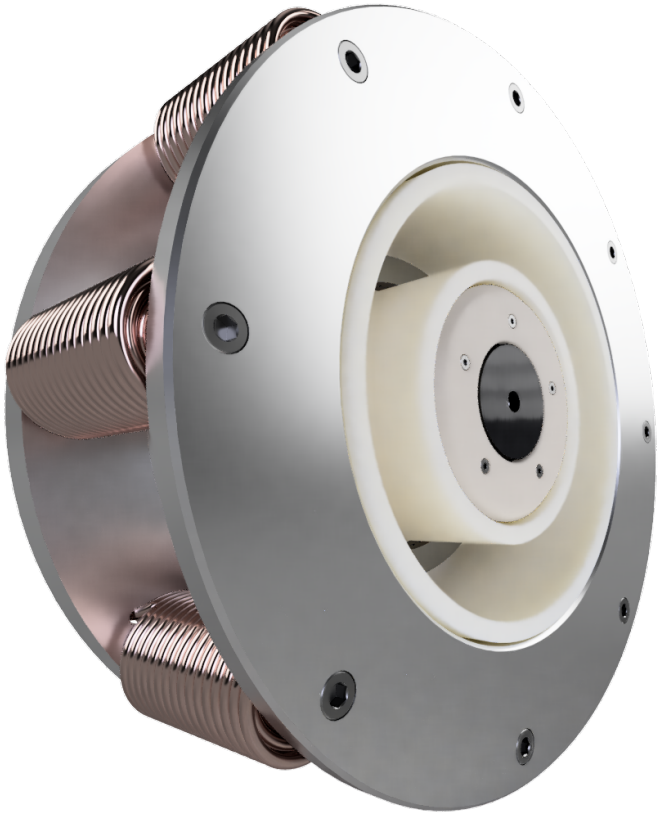
# Development of a 2kW Krypton Hall Effect Thruster

— A Portfolio for Dan Card —

By Dan Card[†]



[†]M.S.E. Aerospace Engineering, University of Michigan - Ann Arbor
B.S.E. Aerospace Engineering, University of Michigan - Ann Arbor

# Contents

# List of Figures

# 1 About Me

My name is Dan Card, I am a recent graduate from The University of Michigan - Ann Arbor with an Undergraduate Degree in Aerospace Engineering (Aug 2020) as well as a Master's Degree in Aerospace Engineering with the focus being Gas Dynamics (May 2021). I am writing this portfolio to highlight my skills as an engineer, as well as areas that interest me that in my pursuit of a professional career in the development of affordable space travel.

Its become apparent that my lack of industry experience was hindering my profile and that I should uptake a role in a project team to allow for the gained experience. As such I have written this portfolio to highlight my skills as a designer and to show my capability as an engineer. Here in this portfolio, you will find drawings for every single component, custom-written CFD analysis and supporting Python3 code, manufacturing drawings, and renderings that were entirely done on my own over three months of free time.

## 1.1 Prior Work and Experience

As I am a fairly recent college graduate, I have included my previous internships to expand further my knowledge set and how I could contribute to a fast-paced work environment and aid in engine design. Each subsection will provide a short description of my contribution to the company/research with more technical information found on my resume.

### 1.1.1 Northrop Grumman Corpration

Currently, I am working full-time as a Tooling and Design Engineer at Northrop Grumman in the Clearfield, Utah facility aiding operations and manufacturing across all of Northrop Grumman's solid rocket programs. These such programs include the aging legacy program of the Trident II D-5 marine missile which I have aided in their modernization of this program. Such modernization efforts include updating/designing supporting equipment to help aid operations during their manufacturing process of producing rocket motor cases.

Additionally, as a tooling engineer, I often work with discrepant parts and tools and work to develop fixes, and often update our designs to prevent these repairs from stopping manufacturing. It is not too uncommon for me to go on the floor and make changes on my own to increase the turnaround time of such repairs.

With this position although I am a tooling engineer I very much see myself as an all-around jack-of-all-trade systems engineer working on the floor with operations, designing in NX, making tool repairs, and flexing across a multitude of scenarios whenever the need arises.

### 1.1.2 ExoTerra LLC

While in college I held an internship at a small aerospace start-up in Littleton, CO as a propulsion engineer. In this internship, I worked to aid the development of a low-power hall effect thruster (a smaller-scale version of the one shown in this portfolio). Additionally, I worked to design supporting laboratory equipment such as an ion beam dump to prolong the vacuum chamber lifespan as well as a full re-design of our propellant flow panel to adjust the speeds of the cathode/anode gas flow rates.

### 1.1.3 The University of Michigan's Plasmadynamics and Electric Propulsion Laboratory (PEPL)

I held a research position as a mechanical and electrical engineering intern at the University of Michigan's Plasmadynamics and Electric Propulsion Laboratory (PEPL). Here I aided the development of supporting laboratory equipment such as repairing a high-speed linear translational stage and designing a PCB interfaced with an Arduino to input data from an incremental encoder to display position in real-time.

## 1.2 Areas of Interest

My areas of interest are fairly broad but I enjoy aerospace propulsion as a whole and very much enjoy the design aspects of these systems. I wish to gain experience in the design of propulsion components and better myself both as a design engineer as well as a propulsion engineer so that I may one day gear myself towards a propulsion systems engineer position.

I also wish to gain more experience in propulsion systems to gain better working knowledge in gas/liquid systems while using my knowledge of combustion processes to advance technology to enable human interplanetary travel. With my current position, although I have learned a lot on the side of manufacturing and operations, is limiting with my expertise and I feel that my skills would better be suited to directly working on the design aspect of propulsion systems.

I have working knowledge in the electric propulsion field with my prior research and internships and this highlighted portfolio. The first two sections will describe my thoughts and rationale which will drive the development of this hall effect thruster, the remainder of this will strictly be designs and simulations — skip to Section 4 for strict design. If there ever arises any questions feel free to contact me either by email at dcard@umich.edu or by phone at (810) 728-6754, I appreciate the time and consideration.

Sincerely,

Dan Card

# 2 Introduction and Motivation

I had an idea of using low-thrust satellites to mitigate space debris/non-functional satellites and reached out to DARPA for serious consideration of this avenue in space. I wanted to hone my skills in CAD while learning how to align my thinking as a systems engineer. In this I decided to accurately design a Hall Effect Thruster, then design a satellite as well (Attached and shown in the Appendix B) for the mass approximations then use Python3 scripts to simulate low-thrust burns and ultimately determine the propellant mass requirements and the feasibility of such missions.

The mission outline would consist of a conventional launch into Earth's upper atmosphere where a small satellite would be deployed. From here, a low-thrust Hall Effect Thruster would turn on and slowly raise the orbit and adjust the inclination of the satellite to match that of the de-functional satellite. Then with some sort of mechanical arm, grapple the de-functional satellite and perform a retro-burn to slowly de-orbit both itself and the de-functional satellite. a small diagram can be seen below in Figure 1.

1.) Low-Thrust Orbit Raise
2.) Rendezvous with De-functional Satellite
3.) De-orbit Satellite and De-functional Satellite

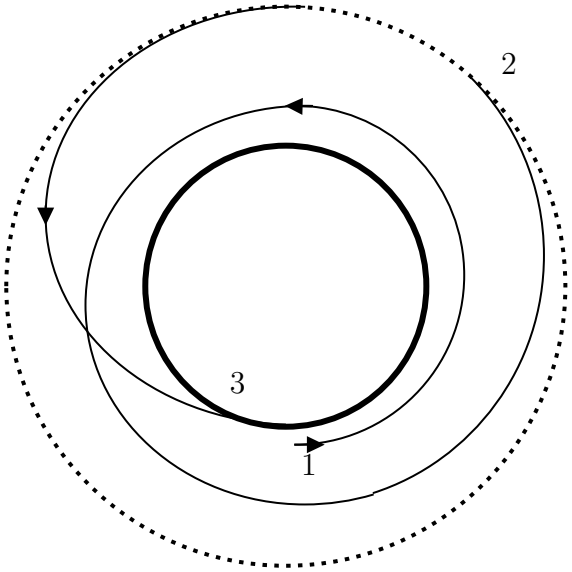



**Figure 1:** Mission outline for DARPA proposal.

In order to determine the feasibility of such a mission, I would need to develop an algorithm that would use the target mass and altitude and ultimately determine the fuel requirements and determine whether or not the results were physical or not. However, this algorithm requires an initial starting mass of the satellite to capture the space debris to provide the initial weights of the satellite.

# 3 Driving Constraints

During my undergraduate degree at the University of Michigan, I took Electric Propulsion and learned about the rule of thumb design aspects that are commonly used in industry. As such to determine an accurate model I would use them as well to ease design time and focus on top-level design aspects (manufacturability, and ease of installation). As such these are the design constraints/choices I made while developing this Hall Effect Thruster.

## 3.1 Channel Design

Common in the Aerospace industry, the scaling of the Hall Effect Thruster channel widths goes as $b/d = 1/5$ where $b$ is the channel width and $d$ is the diameter of the Hall Effect Thruster. A simple visualization can be seen below in Figure 2.

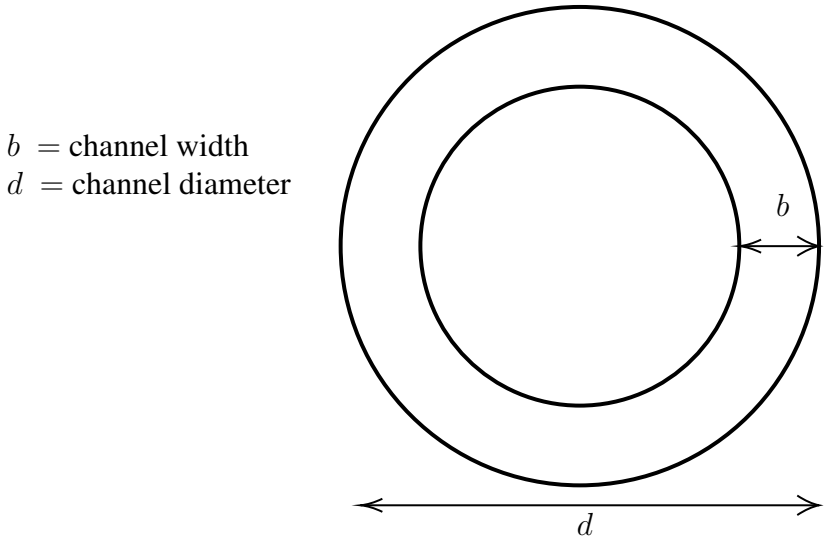



**Figure 2:** Commonly used industry standard channel scaling.

## 3.2 Magnet Design

From my Graduate Electric Propulsion class, I know that industry standards and several Thrusters use a field strength of approximately 200 Gauss and that this would be something to aim for. By invoking simple magnet relationships I can gauge what the magnet should be able to produce within the channel of the Hall Effect Thruster and in simple magnet, field simulations determine how close this field strength is to industry standard.

## 3.3 Anode/Cathode Design

There are several design considerations that I made for the cathode design. I used the separation distance that was on PEPL's experimental cathode, as well as a LaB6 pressed sleeve to emit electrons. Additionally, I used Dan Goebel's book for the reference of what a hollow cathode design should entail, and thus Figure 3, can be found below of which I centered my design around.

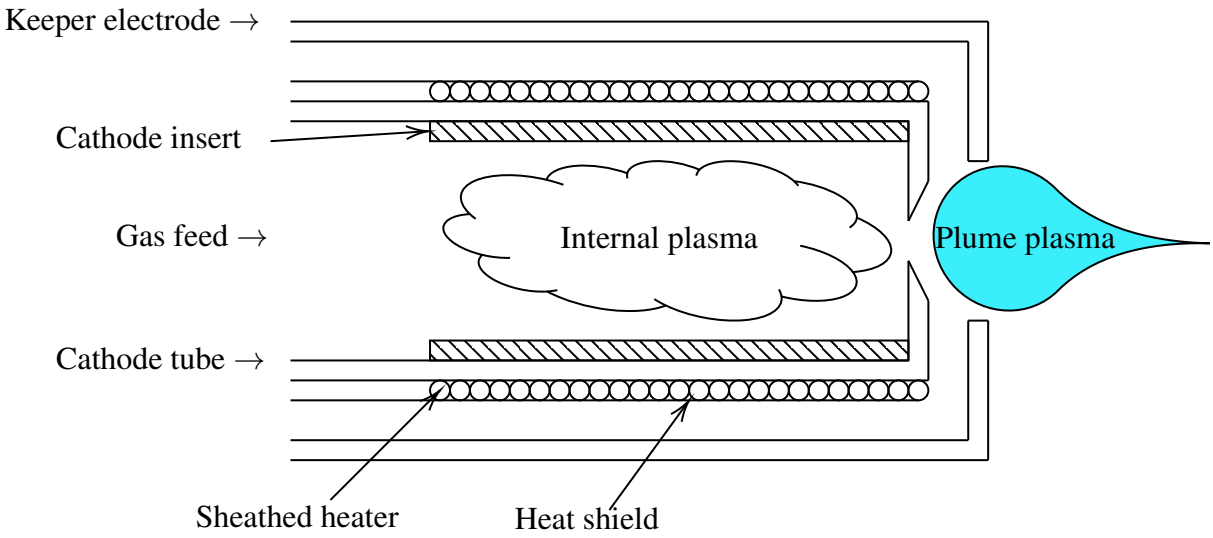



**Figure 3:** Schematic of hollow cathode to be coaxially located within a HET.

## 3.4 General Material Choices

There are common materials used with electric propulsion, and these are the materials often used in industry:

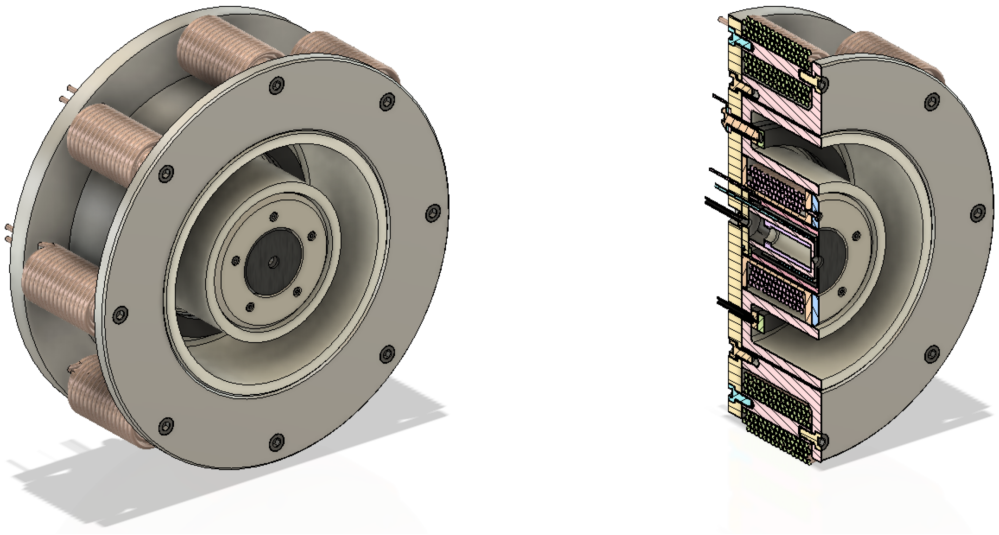
- Boron Nitride for the channel walls
  - Insulating properties — as it is a ceramic
  - Low sputtering properties — which extends the lifetime of the thruster
- Tungsten for the cathode
  - High melting point — good for dealing with internal hot plasma
  - Good conductivity — aids to create a voltage potential to accelerate plasma
- Aluminum for general metals
  - Has good conductivity for creating a common ground
  - Good weight properties

# 4 Preliminary Design

Throughout this design, I used Fusion 360, due to the availability of the software and its free use for non-commercial projects. Although the free use does hinder the flexibility and often limits performance it is nonetheless a good option for me to show my potential for component design as this software has limitations with large assemblies.

## 4.1 Assembly Overview

Shown below in Figure 4 is a screenshot of the CAD model within Fusion 360 in an isometric view with and without the cross-sectional view enabled. With everything mainly built around a central axis, the design remains fairly simple to implement and design accordingly.



**(a)** Isometric view. **(b)** Cross-sectional isometric view.

**Figure 4:** Fusion 360 isometric design view with cross section.

As seen above in Figure 4, this is an accurate model that has taken into account the wiring of the inner/outer magnets, the keeper, the cathode, the heater, and gas inlet to the anode and cathode. Additionally, to improve mass estimates and modeling of the design, I have created solid models of the magnet windings as well that can be seen around the annular of the Hall Effect Thruster.

To show more a more detailed cross-section, Figure 5 can be found below. This highlights the modeled magnet windings, the inner channel, the anode, as well as the inner workings of the hollow cathode.

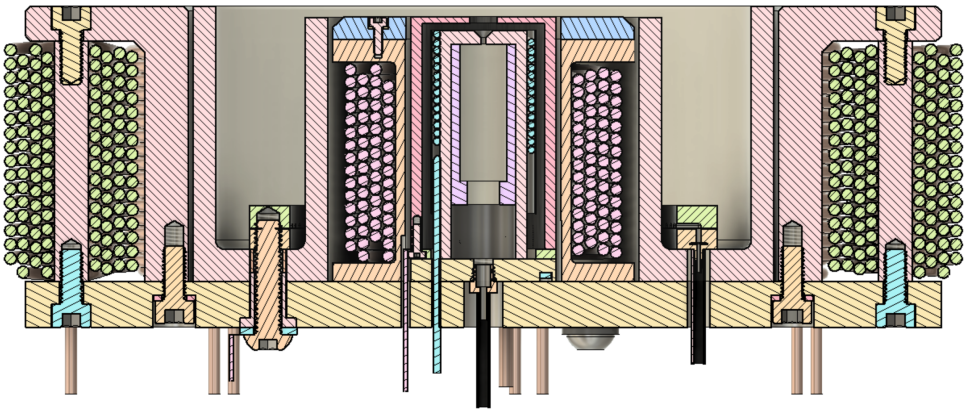


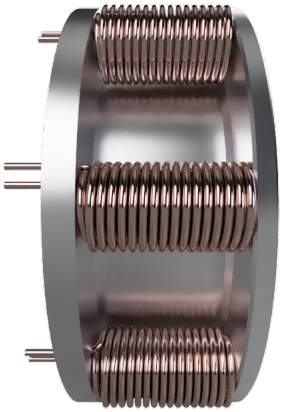
**Figure 5:** Fusion 360 cross-sectional view of interior Hall Effect Thruster.

Looking above to Figure 5, the centrally coaxially located hollow cathode can be seen and compared across Figure 3 from the prior section. Further analysis will be done on the channel.

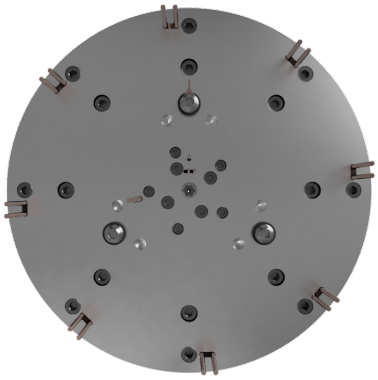
## 4.2 Renderings

To better highlight what the finalized thruster would look like I used Fusion 360's built-in Render tool to create some simple renderings of the finished Hall Effect Thruster.

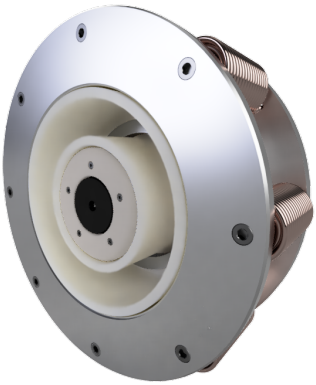
**(a)** Front view.



**(b)** Side view.



**(c)** Back view.

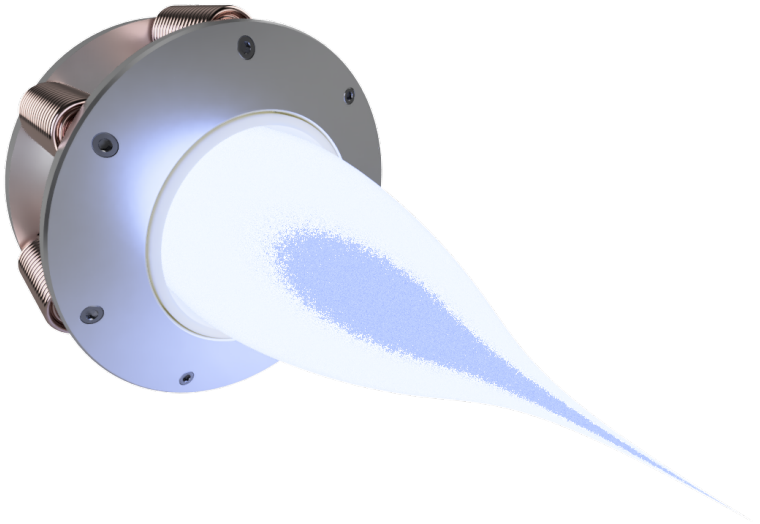


**(d)** Isometric view.

**Figure 6:** Fusion 360 renderings of Hall Effect Thruster.

Finally, to visualize the ion plume I created a spline and revolved it around the central axis, and gave it a blue hue and high luminosity to create the visible glow while in operation. This visualization of the difference of the Hall Effect Thruster during operation can be found below in Figure 7.

**(a)** Unlit Hall Effect Thruster.



**(b)** Lit Hall Effect Thruster.

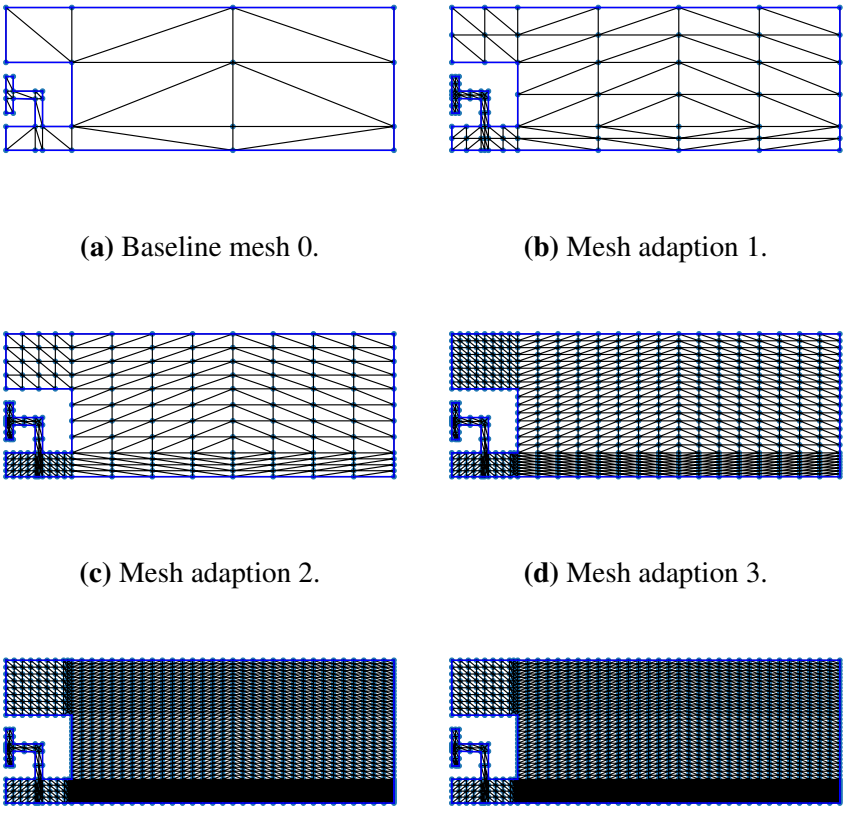
**Figure 7:** Fusion 360 renderings of Hall Effect Thruster during operation.

## 4.3 Simulations

In this subsection, I wanted to highlight some simple simulated analyses of the Hall Effect Thruster and my corresponding skills in Python3. In these simulations, I wanted to analyze the flow through the anode and into the inner channel, as well as the magnetic contours from the magnets.

### 4.3.1 Python CFD Analysis

To do the CFD analysis I needed a mesh to store the state values, but to have an accurate solution I would have to adapt the baseline mesh of the cross-section such that it was refined enough for simulation. Shown below in Figures 8a - 8f are the adapted meshes to refine the results from the baseline.



**(a)** Baseline mesh 0.

**(b)** Mesh adaption 1.

**(c)** Mesh adaption 2.

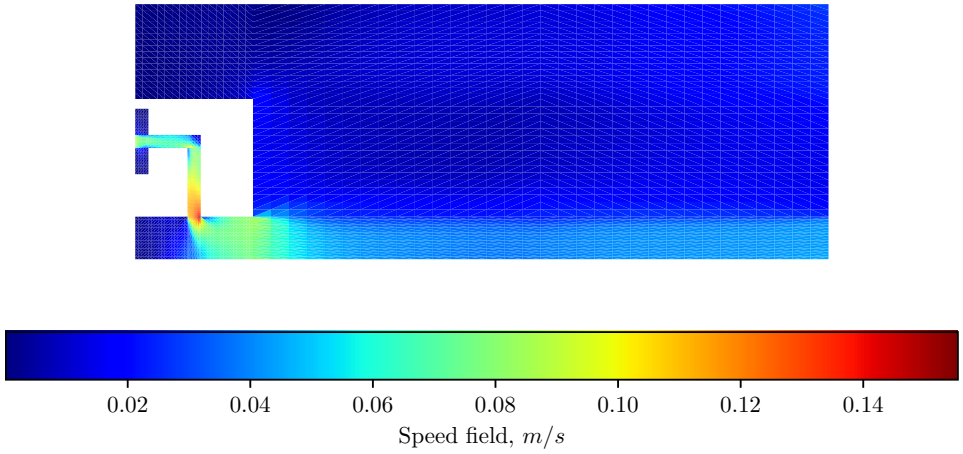
**(d)** Mesh adaption 3.

**(e)** Mesh adaption 4.

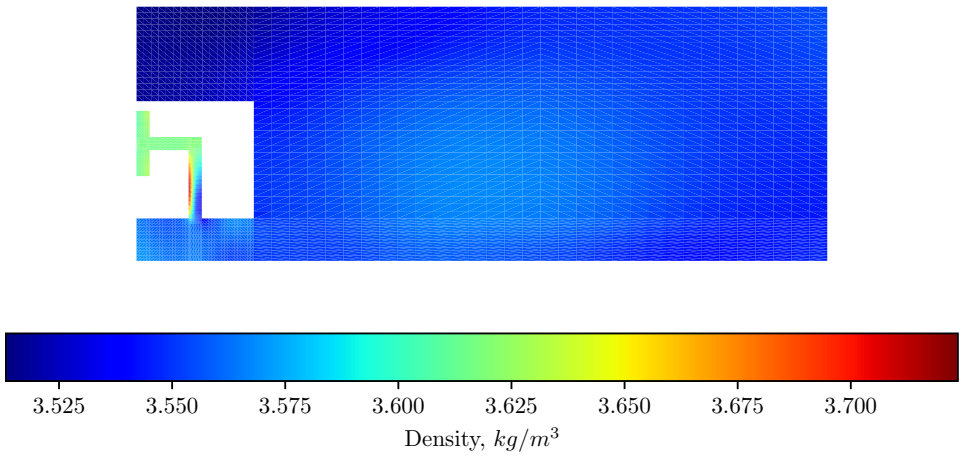
**(f)** Mesh adaption 5.

**Figure 8:** Python3 mesh adaptions/refinements of the anode and inner channel.

After adapting and refining the meshes shown above, I used finite-element methods to approximate the state value inside an element(triangle) and iterate until the total accumulated error across the entire region was below a certain threshold ($\mathcal{O}\ 10^{-3}$). After completing these iterations I generated Figure 9 below.




**(a)** Python3 analysis of speed field.

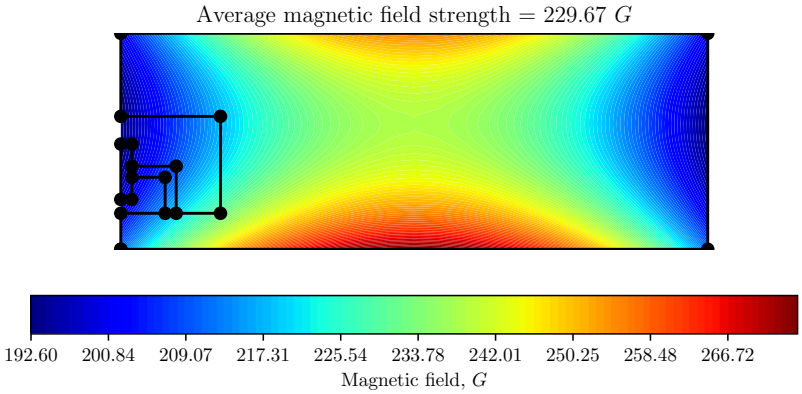



**(b)** Python3 analysis of density field.

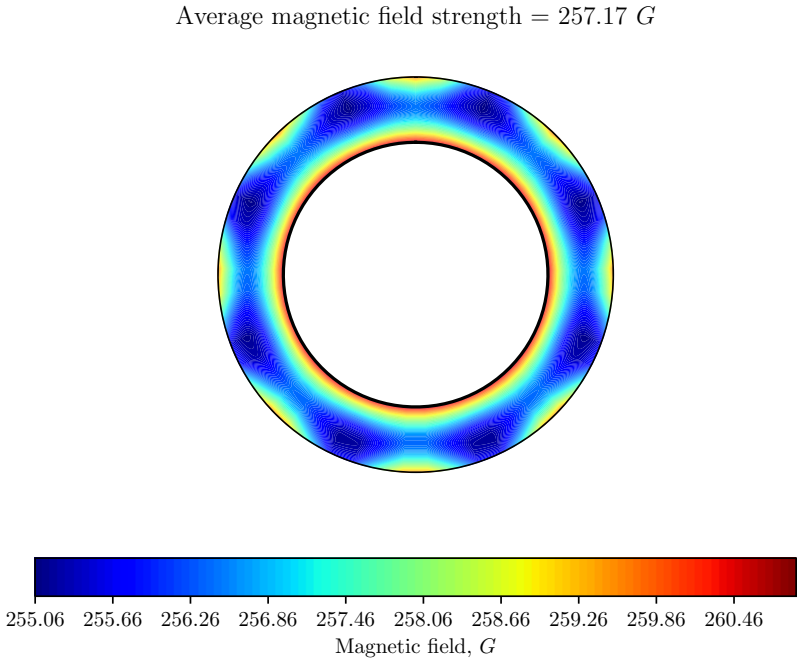
**Figure 9:** Python3 simulation results of the anode and interior channel.

### 4.3.2 Python Magnetic Field Analysis

Finally, to determine if I was around the target specification of the magnetic field strength of 200 Gauss, I used superposition of magnetic fields to determine the magnetic field lines within the cross-section and again of the entire Hall Effect Thruster shown below in Figure 10.




**(a)** Magnetic field within interior channel.




**(b)** Front-view of magnetic field along the entire Hall Effect Thruster.

**Figure 10:** Python3 magnet analysis.

# 5 Design Caveats

While working on multi-disciplinary projects, there are many unknowns that I cannot control and am not sure how to approach. These problems are those that are often convoluted in electrical engineering, thermal analysis, electrodynamics, and theoretical aerospace engineering applications. As such, I've tried my best to approximate with first principles to get a good understanding of the underlying physics to create a preliminary design to better myself as an engineer and present my skills.

In this section, I will discuss some of these design caveats and why I choose to stick with the choices I have made but acknowledging that these areas are outside of my expertise. I often feel like a great engineer is one that knows their areas of expertise, tries to expand upon it but ultimately is open to criticism on possible shortcomings — these following subsections are the areas in which I know will need to be revised if I were to update in the future.

## 5.1 Anode Sizing

I struggled to find any academic papers discussing the location/width of the anode, and as such, this would most likely be subjected to change. Most important to the anode is that the flow field is even and distributed to result in the plasma being accelerated evenly to create an even thrust profile. Additionally, I have that the gas is injected only along with the interior, I believe — as I am not certain — that this would aid to increase the ionization efficiency as the electrons located closer radially to the inner magnets would spiral at a greater speed and as such increase the likelihood of an ionization event.

## 5.2 Cathode-Keeper Separation

This seems like a simple problem but there are many variables at play with heavy theoretical applications that I am unsure of how to incorporate. First, there is the temperature of the plasma, fluxing temperature from the heater, the speed of the plasma, applied voltage from the cathode, then magnetic field contouring through the acceleration region all resulting in a very complex problem I'm not sure of solving. If the separation length is increased too far, then the emitted electrons will be outside of the near field and there will be drops in cathode efficiency and troubles lighting the cathode. Decrease the separation too much, and the cathode risks arcing and shorting. As such I have decided to use a small separation of 1.5 mm with room to adjust as needed.

## 5.3 Design Drawings

I do not have an incredible amount of expertise in drawings, and as such the drawings attached at the end of this portfolio may be lacking engineering "*standards*." However, this is an area I am very excited to improve upon (additionally this is an exceptional skill to have that in my opinion) and I did these drawings to practice and hopefully someday obtain feedback on how to improve.

## 5.4 Fasteners

I do not have much expertise with the correct fasteners that should be used in space-ready components and as such I'd like these to be double-checked for which fasteners are best suited for vibration that is seen in the launch and add any that see thermal loading.

## 5.5 Flow Field

Even though my Python CFD script has shown that the flow is being accelerated downstream and hugs the inner wall, I'd ideally prefer to see professional CFD software confirm this to be true.

## 5.6 Magnet Windings

Like the prior section my Python scripts have returned that I am generating the correct value for the magnetic fields, but using a more sophisticated magnetic simulation software may result in different values. Due to my uncertainty and lack of knowledge in magnetics, this is something I would place that would need to get revisited and possibly revised.

# 6 Conclusion

I have made a preliminary design that is very much suited to be improved upon provided the correct feedback. I have considered all the shortcomings stemming from my lack of expertise in lesser-known areas and am excited to improve upon the design and my background. This has taken a considerable amount of time, but I have very much enjoyed all of it — as the final step I am compiling all the performance metrics below in Table 1.

**Table 1:** Hall Effect Thruster approximated performance metrics.

| Parameter | Units |
|---|---|
| Thrust | 100 mN |
| Power | 2kW |
| Exit velocity | 20 km/s |
| Specific Impulse | 2000 s |
| Discharge Voltage | 691 V |
| Keeper Voltage | 300 V |
| Mass flow rate | 5 $\mu$g/s |

## 6.1 DARPA Application

As I mentioned at the beginning of this portfolio, this initially started as a small side project aimed at gaining the attention of DARPA if deemed a suitable area of research. I designed not only the Hall Effect Thruster but also a small satellite as well to obtain accurate mass approximations to determine the feasibility of such a mission. These can be found in Appendix B.

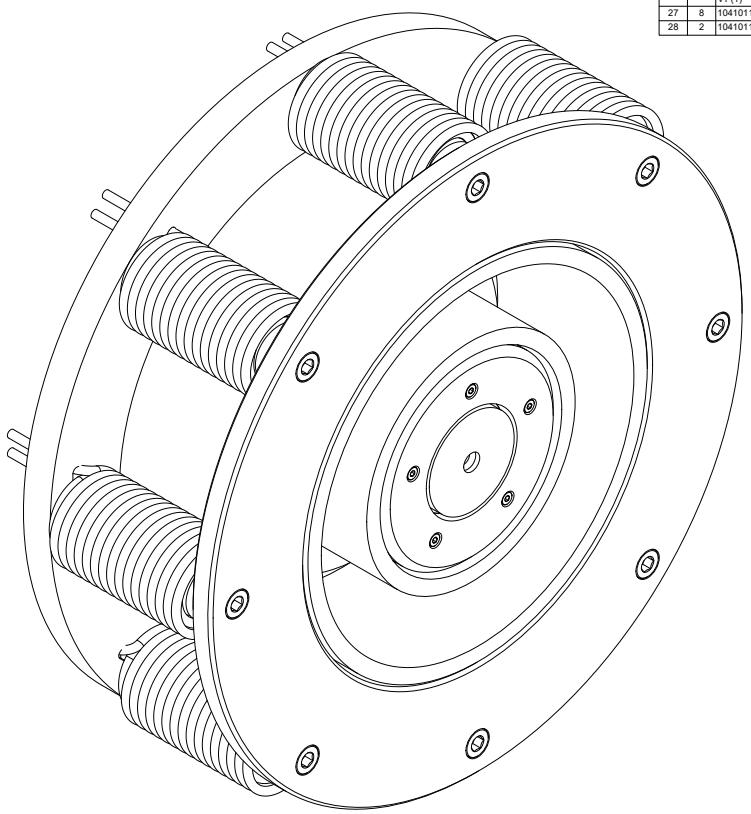
## 6.2 Going Forward

I initially wanted to reach out to my previous Professor that had both taught and mentored my research, Associate Professor Benjamin Jorns, but shortly after concluding the drawings, I received an email entailing that The University of Michigan was considering raising his status to Tenured Associate Professor and to avoid conflicts of interest I decided that an experts feedback would have to wait sometime. Additionally, I wanted to obtain real quotes from vendors on the approximate cost to manufacture and assemble this... but I was wary about reaching out to a machine shop to provide actual quotes to a project that I have no interest in developing shortly. Going forward, I will most likely do both of these given the right conditions. Additionally, I would like to thank you for taking the time to read this.

# Appendices

## A Design Drawings

On the following pages, I will attach my preliminary drawing designs to demonstrate my skills in engineering design and perseverance to hone my skills in technical drawing. I apologize for not being able to label radii and diameters in ANSI callouts and provide installation lines, however, these are features that are not supported by Fusion 360.



Parts List

| Item | Qty | Part Number | Description | Material |
|---|---|---|---|---|
| 21.11 | 1 | M5 x 0.8 x 12 303SS HP BARBED - MMC 4406T662 v1 | | Steel |
| 21.12 | 1 | 104101116-037 | KEEPER_WIRE | Copper |
| 22 | 1 | 104101116-039 | ANODE_WIRE | Copper |
| 23 | 1 | 104101116-050 | | |
| 24 | 8 | 104101116-022 | OUTER_MAGNET_STANDOFF | Aluminum |
| 25 | 8 | M5 x 0.8 x 12 - MMC 91290A228 v1 | | Steel |
| 26 | 8 | M5 x 0.8 x 12 - MMC 91290A228 v1 (1) | | Steel |
| 27 | 8 | 104101116-023 | OUTER_MAGNET_WINDING | Copper |
| 28 | 2 | 104101116-046 | INLET_GAS_TUBE | Rubber |

Parts List

| Item | Qty | Part Number | Description | Material |
|---|---|---|---|---|
| 1 | 1 | 104101116-002 | MOUNT_PLATE | Aluminum 7075 |
| 2 | 1 | 104101116-004 | OUTER_CHANNEL | Aluminum 7075 |
| 3 | 1 | 104101116-020 | INNER_MAGNET | Aluminum 7075 |
| 4 | 1 | 104101116-005 | MAGNET_COVER | Boron Nitride |
| 5 | 1 | 104101116-021 | INNER_MAGNET_WINDING | Copper |
| 6 | 8 | M5 x 0.8 x 12 - MMC 91290A228 | | Steel |
| 7 | 5 | M2 x 0.4 x 6 - MMC 91290A013 | | Steel |
| 8 | 7 | M3 x 0.5 x 10- MMC 91290A115 | | Steel |
| 9 | 8 | M5 18-8SS SW - MMC 92148A170 | | Stainless Steel |
| 10 | 1 | 104101116-003 | HET_CHANNEL | Boron Nitride |
| 11 | 7 | M3 18-8SS SW - MMC 92148A150 v1 (1) | | Stainless Steel |
| 12 | 5 | M2 18-8SS SW - MMC 92148A050 v1 | | Stainless Steel |
| 13 | 5 | M3 x 0.5 x 10- MMC 91290A115 v1 | | Steel |
| 14 | 5 | M3 18-8SS SW - MMC 92148A150 v1 | | Stainless Steel |
| 15 | 3 | Electrical-Insulating Ceramic Sleeve Washer - MMC 92107A120 v2 | | Beryllium oxide (BeO) |
| 16 | 3 | 104101116-044 | MMC 94669A165 | Aluminum |
| 17 | 3 | Electrical-Insulating Nylon Washer - MMC 90718A180 v2 | | Nylon 6 - BLACK |
| 18 | 3 | M6 x 1 x 25 BUTTON HEAD - MMC 92095A242 v2 | | Stainless Steel |
| 19 | 4 | M4 x 0.7 x 3 SHC - MMC 91290A140 v1 | | Steel |
| 20 | 1 | 104101116-045 | ANODE | |
| 20.1 | 1 | 104101116-040 | ANODE_WITH_BAFFLES | Steel |
| 20.2 | 1 | 104101116-041 | ANODE_DISTRIBUTOR | Steel |
| 20.3 | 1 | 104101116-042 | OUTER_GASKET | Rubber |
| 20.4 | 1 | 104101116-043 | INNER_GASKET | Rubber |
| 21 | 1 | 104101116-038 | CATHODE | |
| 21.1 | 1 | 104101116-030 | KEEPER | Conductive Graphite |
| 21.2 | 1 | 104101116-031 | CATHODE_TUBE | Tungsten |
| 21.3 | 1 | 104101116-032 | CATHODE_INSERT | Lanthanum Hexaboride (LaB6) |
| 21.4 | 1 | 104101116-033 | HEATER | Copper |
| 21.5 | 1 | 104101116-034 | RADIATION_SHIELD | Stainless Steel AISI 309 |
| 21.6 | 1 | 104101116-035 | KEEPER_ISOLATER | Ceramic |
| 21.7 | 1 | 104101116-036 | CATHODE_MOUNT_PLATE | Aluminum |
| 21.8 | 5 | M2 x 0.4 x 6mm LOW PROFILE - MMC 93070A276 v1 | | Steel |
| 21.9 | 5 | M2 x 0.4 x 4 LOW PROFILE - MMC 93070A274 v1 | | Steel |
| 21.10 | 4 | M2 x 0.4 x 8 LOW PROFILE - MMC 93070A277 v1 | | Steel |

**NOTES**

- CLEAN ALL SURFACES WITH ISOPROPYL ALCOHOL (IPA) PRIOR TO ASSEMBLY

| | | | |
|---|---|---|---|
| DC LABS | DRAWN BY | Dan Card | PROJECT HET - Hall Effect Thruster |
| | DATE | 8/4/2021 | DWG NO 104101116-001 |
| UNLESS OTHERWISE SPECIFIED DIMENSIONS IN MM TOLERANCING X ± 0.2 0.X ± 0.1 0.XX ± .05 ANGULAR = ±0.5° REMOVE BURRS AND SHARP EDGES SURFACE ROUGHNESS 3.2 | | | TITLE HALL_EFFECT_THRUSTER |
| | SIZE A1 | MATERIAL | |
| | SCALE 1:2 | | SHEET 1/1 |

# The rest of the technical drawings have been omitted to comply with ITAR Regulations, these can be provided if requested

# B DARPA Design Outline

As previously mentioned, part of this project was to determine the mass estimates of a small satellite to be used to de-orbit non-functional satellites in LEO to help mitigate the risk of orbital collisions. As such, below in Figure 11 was the preliminary design of the satellite to provide mass approximations into a Python simulation that would ultimately determine whether or not my method was physically possible.

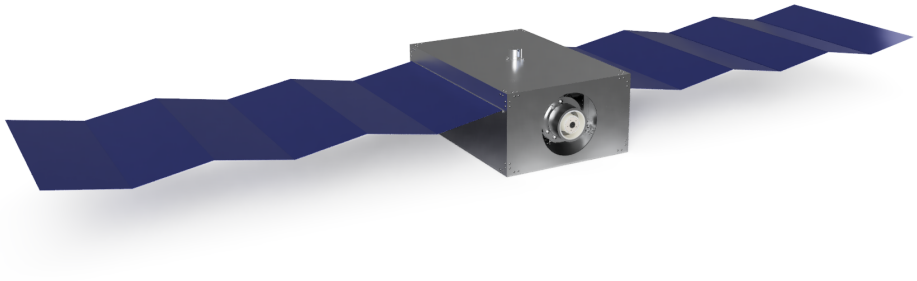


**Figure 11:** Preliminary design of space scrubber.

With the small satellite generated above, and using its mass estimates I was able to generate its operational parameters that would ultimately determine the feasibility of such a mission. As such Figure 12 below was generated to determine where these missions are best suited and the diminishing returns of more ambitions de-orbits.

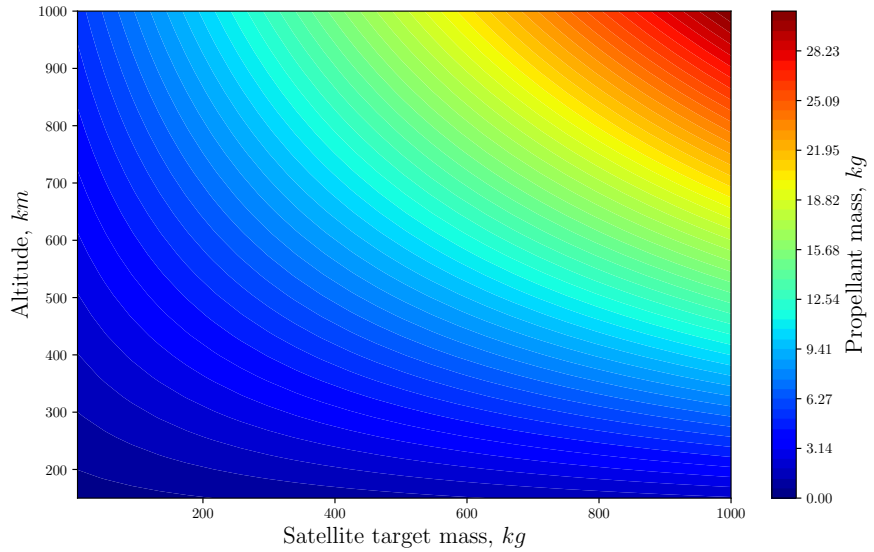



**Figure 12:** Operational parameters of space scrubber for the target satellite.

# C Python Implementation

In this section, I will list all relevant code implementations that I have written that has aided in this preliminary design.

**Listing 1:** Channel magnet simulation.

```
import numpy as np
import matplotlib.pyplot as plt

plt.rc('text', usetex=True)
plt.rc('font', family='serif')


def calcB(xlin, ylin, wirey, turns):
    wirex = xlin[0,:]
    mu0 = 1.256637e-6
    cur = 20
    num = xlin.shape[0]
    magdat = xlin*0
    for k in range(num):
        for i in range(num):
            for j in range(num):
                r = np.sqrt((wirex[k] - xlin[i,j])**2 + (wirey[k] - ylin[i,j])**2)
                magdat[i,j] += mu0*cur*turns/(2*np.pi * r)*52.5e-3

    return magdat

def main():
    num = 25
    xlin = np.linspace(0, 53, num) * 10**-3
    ylin = np.linspace(0, 19.5, num) * 10**-3
    xlin, ylin = np.meshgrid(xlin, ylin)

    magdat = calcB(xlin, ylin, np.ones((num,1))*(-11.25e-3), 60) + calcB(xlin, ylin, np.ones((
        num,1))*(19.5e-3+15.5e-3), 64)

    f = plt.figure(figsize=(6,3))
    plt.contourf(xlin*1e3, ylin*1e3, magdat*1e4, np.linspace(np.min(magdat.flatten()), np.max(
        magdat.flatten()), 100)*1e4,cmap='jet')
    plt.plot(np.array([0,53, 53, 0, 0, 9, 9, 5, 5, 1, 1, 0, 0, 1, 1, 4, 4, 0, 0]), np.array([0,0,
        19.5, 19.5, 12, 12, 3.25, 3.25, 7.5, 7.5, 9.5, 9.5, 4.5, 4.5, 6.5, 6.5, 3.25, 3.25,
        0]), color='k', marker='o')
    plt.colorbar(label=r'Magnetic field, $G$', orientation='horizontal')
    titl = r'Average magnetic field strength =' + str("{0:.2f}").format(np.mean(magdat.flatten()
        ) * 1e4) + r' $G$'
    plt.title(titl)
    plt.axis('equal'); plt.axis('off')
    f.tight_layout()
    plt.savefig('magField.pdf', bbox_inches='tight')
    plt.show()



if __name__ == "__main__":
    main()
```

**Listing 2:** Radial magnet simulation.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.core.fromnumeric import transpose

plt.rc('text', usetex=True)
plt.rc('font', family='serif')

def calcB(r, theta, mloc, turns, inner):
    mu0 = 1.256637e-6
    cur = 18

    rx = r*np.cos(theta) - mloc[0]
    ry = r*np.sin(theta) - mloc[1]
    r = np.sqrt(rx**2 + ry**2)

    return mu0*cur*turns/(2*np.pi * r)

def main():
    num = 50
    zeniths = np.linspace(39, 58.5, num) * 1e-3 # Radial positions
    azimuths = np.linspace(0, 2*np.pi, num)
    values = np.zeros((num**2, num**2)) # Z-data

    magloc = np.zeros((9,2)); tempthet = np.linspace(0, 2*np.pi, 8, endpoint=False)
    for i in range(1,9):
        magloc[i,0] = np.cos(tempthet[i-1])*94e-3
        magloc[i,1] = np.sin(tempthet[i-1])*94e-3

    values = np.zeros((len(azimuths), len(zeniths)))
    r, theta = np.meshgrid(zeniths, azimuths)

    for k in range(9):
        for i in range(num):
            for j in range(num):
                if k == 0:
                    values[j,i] += calcB(zeniths[i], azimuths[j], magloc[k,:], 60, True)
                else:
                    values[j,i] += calcB(zeniths[i], azimuths[j], magloc[k,:], 64, False)


    fig, ax = plt.subplots(subplot_kw=dict(projection='polar'))
    ax.set_theta_zero_location("N")
    ax.set_theta_direction(-1)
    cax = ax.contourf(theta, r, values*1e4, 100, cmap='jet')
    ax.axis('off')
    ax.plot(0,0)
    ax.plot(np.linspace(0, 2*np.pi, 100), np.ones((100,1))*39.0e-3, color='k')
    ax.plot(np.linspace(0, 2*np.pi, 100), np.ones((100,1))*58.5e-3, color='k')
    titl = r'Average magnetic field strength = ' + str("{0:.2f}").format(np.mean(values.flatten()) * 1e4) + r' $G$'
    plt.title(titl)
    cb = fig.colorbar(cax, orientation='horizontal', label=r'Magnetic field, $G$')
    plt.savefig('radialmagField.pdf', bbox_inches='tight')
    plt.show()



if __name__ == "__main__":
    main()
```

**Listing 3:** Adapting mesh code.

```
import numpy as np
from numpy import linalg as LA
from readgri import writegri
from edgehash import edgehash
import matplotlib.pyplot as plt

def mach_perp(u, nhat):
    uvel = u[1]/u[0]; v = u[2]/u[0]          # Calculate the velocity
    q = np.dot(np.array([uvel, v]), nhat)    # Determine the perpendicular speed
    P = (1.4 - 1)*(u[3] - 0.5*u[0]*q**2)     # Calculate pressure
    H = (u[3] + P)/u[0]                      # Calculate enthalpy
    c = np.sqrt(0.4*(H - 0.5*q**2))          # Calculate speed of sound
    mach = q/c                               # Calculate the Mach number

    return mach

def check_vert(Vvec, x):
    check = True
    # Loop over the vertices
    for i in range(Vvec.shape[0]):
        # If this vertex exists return False
        if '%.5f'%x[0] == '%.5f'%Vvec[i,0] and '%.5f'%x[1] == '%.5f'%Vvec[i,1]:
            check = False
            break

    return check

def isCCW(a, b, c):
    # Princeton implementation of CCW logical
    cross_val = (b[0] - a[0])*(c[1] - a[1]) - (c[0] - a[0])*(b[1] - a[1])

    # Conditional statements
    if cross_val > 0:
        cross_val = 1
    elif cross_val < 0:
        cross_val = -1
    else:
        cross_val = 0

    return cross_val

def vert_ind(Vvec, x):
    check = False; ind = np.array([])
    # Loop over the vertices
    for i in range(Vvec.shape[0]):
        # If this vertex exists return False

        if '%.5f'%x[0] == '%.5f'%Vvec[i,0] and '%.5f'%x[1] == '%.5f'%Vvec[i,1]:
            check = True; ind = np.append(ind, np.array([i]))

    return check, ind

def genflags(u, mach, V, E, IE, BE):

    # Pre-allocate flag array
    flags = np.zeros(((IE.shape[0] + BE.shape[0]),2)); flags[:,0] = np.arange(flags.shape[0]);k =
        0

    k = 0
    for i in range(IE.shape[0]):
        ig, ig, e1, e2 = IE[i,:]
        for j in np.array([e1,e2]):
            n1, n2, n3 = E[j,:]
```

```
63                  x1 = V[n1,:]; x2 = V[n2,:]; x3 = V[n3,:]
64
65                  l1 = np.sqrt((x1[0] - x2[0])**2 + (x1[1] - x2[1])**2)
66                  l2 = np.sqrt((x1[0] - x3[0])**2 + (x1[1] - x3[1])**2)
67                  l3 = np.sqrt((x3[0] - x2[0])**2 + (x3[1] - x2[1])**2)
68                  area = 0.5*(x1[0]*(x2[1]- x3[1]) + x2[0]*(x3[1] - x1[1]) +x3[0]*(x1[1] - x2[1]))
69                  if l1>2e-3 or l2>2e-3 or l3>2e-3:
70                      flags[k,1] = area
71              k += 1
72          for i in range(BE.shape[0]):
73              ig, ig, e1, e2 = BE[i,:]
74              for j in np.array([e1,e2]):
75                  n1, n2, n3 = E[j,:]
76                  x1 = V[n1,:]; x2 = V[n2,:]; x3 = V[n3,:]
77
78                  l1 = np.sqrt((x1[0] - x2[0])**2 + (x1[1] - x2[1])**2)
79                  l2 = np.sqrt((x1[0] - x3[0])**2 + (x1[1] - x3[1])**2)
80                  l3 = np.sqrt((x3[0] - x2[0])**2 + (x3[1] - x2[1])**2)
81                  area = 0.5*(x1[0]*(x2[1]- x3[1]) + x2[0]*(x3[1] - x1[0]) +x3[0]*(x1[1] - x2[1]))
82                  if l1>2e-3 or l2>2e-3 or l3>2e-3:
83                      flags[k,1] = area
84              k += 1
85
86          # Sort from largest to smallest errors
87          flags = flags[flags[:,1].argsort()]; flags = np.flipud(flags)
88          # Remove all outliers to be refined
89          ind = int(np.ceil(flags.shape[0]))
90          flags[ind:(flags.shape[0]-1),1] = 0
91
92          # Sort the errors increasing the edge number to iterate
93          flags = flags[flags[:,0].argsort()]
94
95          return flags
96
97      def genV(flags, V, E, IE, BE):
98          Vcopy = V.copy(); k = 0
99          for i in range(IE.shape[0]):
100             err = flags[k,1]
101             if err > 0:
102                 ig, ig, e1, e2 = IE[i,:]
103                 for j in np.array([e1,e2]):
104                     n1, n2, n3 = E[j,:]
105                     x1 = V[n1,:]; x2 = V[n2,:]; x3 = V[n3,:]
106
107                     # Conditionals to prevent duplicate nodes
108                     if check_vert(Vcopy, (x2-x1)/2 +x1):
109                         Vcopy = np.append(Vcopy, np.array([(x2-x1)/2 +x1]), axis=0)
110                     if check_vert(Vcopy, (x3-x1)/2 +x1):
111                         Vcopy = np.append(Vcopy, np.array([(x3-x1)/2 +x1]), axis=0)
112                     if check_vert(Vcopy, (x3-x2)/2 +x2):
113                         Vcopy = np.append(Vcopy, np.array([(x3-x2)/2 +x2]), axis=0)
114             k += 1
115
116         for i in range(BE.shape[0]):
117             err = flags[k,1]
118             if err > 0:
119                 ig, ig, e1, ig = BE[i,:]
120                 n1, n2, n3 = E[e1,:]
121                 x1 = V[n1,:]; x2 = V[n2,:]; x3 = V[n3,:]
122
123                 # Conditionals to prevent duplicate nodes
124                 if check_vert(Vcopy, (x2-x1)/2 +x1):
125                     Vcopy = np.append(Vcopy, np.array([(x2-x1)/2 +x1]), axis=0)
126                 if check_vert(Vcopy, (x3-x1)/2 +x1):
127                     Vcopy = np.append(Vcopy, np.array([(x3-x1)/2 +x1]), axis=0)
```

```
128                 if check_vert(Vcopy, (x3-x2)/2 +x2):
129                     Vcopy = np.append(Vcopy, np.array([(x3-x2)/2 +x2]), axis=0)
130             k += 1
131
132     return Vcopy
133
134 def genUE(u, Vcopy, V, E, IE, BE):
135     Ecopy = E.copy(); Ucopy = u.copy()
136     for i in range(Ecopy.shape[0]):
137         # Grab the node values of each given element
138         n1, n2, n3 = Ecopy[i,:]
139         x1 = V[int(n1),:]; x2 = V[int(n2),:]; x3 = V[int(n3),:]
140         vals = np.array([(x2-x1)/2 +x1, (x3-x1)/2 +x1, (x3-x2)/2 +x2])
141
142         # Generate nodes for each element
143         nodes = np.array([])
144         for k in vals:
145             check, ind = vert_ind(Vcopy, k)
146             if check:
147                 nodes = np.append(nodes, ind)
148
149         # If three flags have been flagged
150         if nodes.shape[0] == 3:
151             # Ensure that the nodes are CCW
152             if isCCW(Vcopy[int(nodes[0]),:], Vcopy[int(nodes[1]),:], Vcopy[int(nodes[2]),:]) !=
                    1:
153                 nodes = np.flip(nodes)
154
155             nodeint = np.array([n1, n2, n3])
156             if isCCW(Vcopy[int(nodeint[0]),:], Vcopy[int(nodeint[1]),:], Vcopy[int(nodeint[2])
                    ,:]) != 1:
157                 nodeint = np.flip(nodeint)
158
159             # Loop through the nodes
160             for k in range(3):
161                 # Start at the nodes N1 -> N2 for consistency
162                 if '%.5f'%Vcopy[int(nodes[k]),0] == '%.5f'%vals[0,0] and '%.5f'%Vcopy[int(nodes[
                        k]),1] == '%.5f'%vals[0,1]:
163                     Ecopy[i,:] = np.array([n1, nodes[k], nodes[(k+2)%3]])  # Replace the ith
                            element with new element
164
165                     # Start the indices based from generalized case
166                     ind1 = np.array([nodes[k], n2, nodes[(k+1)%3]])
167                     ind2 = np.array([nodes[k], nodes[(k+1)%3], nodes[(k+2)%3]])
168                     ind3 = np.array([nodes[(k+1)%3], nodes[(k+2)%3], n3])
169
170                     # Ensure that the nodes are CCW
171                     if isCCW(Vcopy[int(ind1[0]),:], Vcopy[int(ind1[1]),:], Vcopy[int(ind1[2]),:])
                            != 1:
172                         ind1 = np.flip(ind1)
173                     if isCCW(Vcopy[int(ind2[0]),:], Vcopy[int(ind2[1]),:], Vcopy[int(ind2[2]),:])
                            != 1:
174                         ind2 = np.flip(ind2)
175                     if isCCW(Vcopy[int(ind3[0]),:], Vcopy[int(ind3[1]),:], Vcopy[int(ind3[2]),:])
                            != 1:
176                         ind3 = np.flip(ind3)
177
178                     # Append new elements
179                     Ecopy = np.append(Ecopy, np.transpose(np.array([[ind1[0]], [ind1[1]], [ind1
                            [2]]])), axis=0)
180                     Ecopy = np.append(Ecopy, np.transpose(np.array([[ind2[0]], [ind2[1]], [ind2
                            [2]]])), axis=0)
181                     Ecopy = np.append(Ecopy, np.transpose(np.array([[ind3[0]], [ind3[1]], [ind3
                            [2]]])), axis=0)
182
```

```
183                     # Append values of U to the updated U for initial starting condition
184                     for l in range(3):
185                         Ucopy = np.append(Ucopy, np.transpose(np.array([[u[i,0]], [u[i,1]], [u[i
                                ,2]], [u[i,3]]])), axis=0)
186                     break
187
188             # If two edges have been flagged
189             elif nodes.shape[0] == 2:
190                 node_ind = np.array([n1, n2, n3])
191
192                 # Ensure CCW order
193                 if isCCW(Vcopy[int(node_ind[0]),:], Vcopy[int(node_ind[1]),:], Vcopy[int(node_ind[2])
                        ,:]) != 1:
194                     node_ind = np.flip(node_ind)
195
196                 # Ensure CCW order
197                 ccw_count = 0
198                 for k in range(3):
199                     if isCCW(Vcopy[int(node_ind[k]),:], Vcopy[int(nodes[0]),:], Vcopy[int(nodes[1])
                            ,:]) != 1:
200                         ccw_count += 1
201                 if ccw_count == 2:
202                     nodes = np.flip(nodes)
203
204                 # Determine which node can be omitted
205                 for k in range(3):
206                     xn1 = Vcopy[int(node_ind[k]),:]; xn2 = Vcopy[int(node_ind[(k+1)%3]),:]; xn3 =
                            Vcopy[int(node_ind[(k-1)%3]),:]
207                     test1 = (xn2-xn1)/2 + xn1; test2 = (xn1-xn3)/2 + xn3
208
209                     # Conditional to determine the starting nodes for triangle
210                     if '%.5f'%test1[0] == '%.5f'%Vcopy[int(nodes[1]),0] and '%.5f'%test1[1] == '%.5f'
                            %Vcopy[int(nodes[1]),1] and '%.5f'%test2[0] == '%.5f'%Vcopy[int(nodes[0])
                            ,0] and '%.5f'%test2[1] == '%.5f'%Vcopy[int(nodes[0]),1]:
211                         ind1 = np.array([node_ind[k], nodes[1], nodes[0]])
212                         newnodes = np.array([node_ind[(k+1)%3], node_ind[(k+2)%3]])
213
214                 # Initialize values for test
215                 vn1 = Vcopy[int(nodes[0]),:]; vn2 = Vcopy[int(nodes[1]),:]
216                 xn1 = Vcopy[int(newnodes[0]),:]; xn2 = Vcopy[int(newnodes[1]),:]
217
218                 # Determine the angle of the first corner
219                 temp1 = (vn1-xn1)/2 + xn1; temp2 = (vn2-xn1)/2 + xn1
220                 theta1 = np.arccos(np.dot(temp1, temp2)/(LA.norm(temp1)*LA.norm(temp2)))
221
222                 # Determine the angle of the second corner
223                 temp1 = (vn1-xn2)/2 + xn2; temp2 = (vn2-xn2)/2 + xn2
224                 theta2 = np.arccos(np.dot(temp1, temp2)/(LA.norm(temp1)*LA.norm(temp2)))
225
226                 # Conditional to ensure no edge overlaps
227                 if theta1 >= theta2:
228                     # Re-arrangement of indices to ensure connectivity
229                     ind2 = np.array([newnodes[1], nodes[0], nodes[1]])
230                     ind3 = np.array([newnodes[0], newnodes[1], nodes[1]])
231                 else:
232                     # Re-arrangement of indices to ensure connectivity
233                     ind2 = np.array([newnodes[0], nodes[0], nodes[1]])
234                     ind3 = np.array([newnodes[0], newnodes[1], nodes[0]])
235
236                 # Ensure that they are CCW
237                 if isCCW(Vcopy[int(ind1[0]),:], Vcopy[int(ind1[1]),:], Vcopy[int(ind1[2]),:]) != 1:
238                     ind1 = np.flip(ind1)
239                 if isCCW(Vcopy[int(ind2[0]),:], Vcopy[int(ind2[1]),:], Vcopy[int(ind2[2]),:]) != 1:
240                     ind2 = np.flip(ind2)
241                 if isCCW(Vcopy[int(ind3[0]),:], Vcopy[int(ind3[1]),:], Vcopy[int(ind3[2]),:]) != 1:
```

```
242                     ind3 = np.flip(ind3)
243
244                 # Overwrite E, and append to E
245                 Ecopy[i,:] = np.array([ind1[0], ind1[1], ind1[2]])
246
247                 Ecopy = np.append(Ecopy, np.transpose(np.array([[ind2[0]], [ind2[1]], [ind2[2]]])),
                        axis=0)
248                 Ecopy = np.append(Ecopy, np.transpose(np.array([[ind3[0]], [ind3[1]], [ind3[2]]])),
                        axis=0)
249
250                 # Append to U for initial starting guess
251                 for k in range(2):
252                     Ucopy = np.append(Ucopy, np.transpose(np.array([[u[i,0]], [u[i,1]], [u[i,2]], [u[
                            i,3]]])), axis=0)
253
254             # If ones edges has been flagged
255             elif nodes.shape[0] == 1:
256                 # Determine the starting node
257                 for k in range(3):
258                     if '%.5f'%vals[k,0] == '%.5f'%Vcopy[int(nodes[0]),0] and '%.5f'%vals[k,1] == '
                            %.5f'%Vcopy[int(nodes[0]),1]:
259                         # Rearrange the nodes according to the node value
260                         if k == 0:
261                             ind1 = np.array([n1, nodes[0], n3])
262                             ind2 = np.array([n2, n3, nodes[0]])
263                         elif k == 1:
264                             ind1 = np.array([n1, nodes[0], n2])
265                             ind2 = np.array([n3, n2, nodes[0]])
266                         elif k == 2:
267                             ind1 = np.array([n2, nodes[0], n1])
268                             ind2 = np.array([n3, n1, nodes[0]])
269
270                 # Ensure CCW orientation
271                 if isCCW(Vcopy[int(ind1[0]),:], Vcopy[int(ind1[1]),:], Vcopy[int(ind1[2]),:]) != 1:
272                     ind1 = np.flip(ind1)
273                 if isCCW(Vcopy[int(ind2[0]),:], Vcopy[int(ind2[1]),:], Vcopy[int(ind2[2]),:]) != 1:
274                     ind2 = np.flip(ind2)
275
276                 # Overwrite and append E
277                 Ecopy[i,:] = np.array([ind1[0], ind1[1], ind1[2]])
278                 Ecopy = np.append(Ecopy, np.transpose(np.array([[ind2[0]], [ind2[1]], [ind2[2]]])),
                        axis=0)
279
280                 # Append to U for initial start
281                 Ucopy = np.append(Ucopy, np.transpose(np.array([[u[i,0]], [u[i,1]], [u[i,2]], [u[i
                        ,3]]])), axis=0)
282
283     # Double check to make sure CCW orientation
284     for i in range(Ecopy.shape[0]):
285         n1, n2, n3 = Ecopy[i,:]
286         ind1 = np.array([n1, n2, n3])
287         if isCCW(Vcopy[int(ind1[0]),:], Vcopy[int(ind1[1]),:], Vcopy[int(ind1[2]),:]) != 1:
288             Ecopy[i,:] = np.array([Ecopy[i,2], Ecopy[i,1], Ecopy[i,0]])
289
290     # Return E (as an integer array)
291     Ecopy = Ecopy.astype(int)
292
293     return Ucopy, Ecopy
294
295 def genB(u, V, Vcopy, BE):
296     Bcopy = BE.copy()
297     for i in range(Bcopy.shape[0]):
298         # Node locations of boundary edges
299         n1, n2, e1, bgroup = BE[i,:]
300         xl = V[n1,:]; xr = V[n2,:]
```

```
301
302             # Call function to determine if the vertex exists
303             check, ind = vert_ind(Vcopy, (xl+xr)/2)
304             if check:
305                 # If this vertex exists re-write B
306                 Bcopy[i,:] = np.array([n1, ind[0], e1, bgroup])
307                 Bcopy = np.append(Bcopy, np.transpose(np.array([[ind[0]],[n2], [Bcopy.shape[0]+1], [
                        bgroup]])), axis=0)
308
309         # Re-arrange B for input to edgehash()
310         B0 = np.array([[-1,-1]]); B1 = B0.copy(); B2 = B0.copy(); B3 = B0.copy()
311         for i in range(Bcopy.shape[0]):
312             # Node vales
313             n1, n2, e, bname = Bcopy[i,:]
314             # Given the values of Bname append to the corresponding group
315             if bname == 0:
316                 B0 = np.append(B0, np.transpose(np.array([[n1], [n2]])), axis=0)
317             if bname == 1:
318                 B1 = np.append(B1, np.transpose(np.array([[n1], [n2]])), axis=0)
319             if bname == 2:
320                 B2 = np.append(B2, np.transpose(np.array([[n1], [n2]])), axis=0)
321             if bname == 3:
322                 B3 = np.append(B3, np.transpose(np.array([[n1], [n2]])), axis=0)
323
324         # Output B#'s to B for input to edgehash()
325         B0 = B0[1:,:]; B1 = B1[1:,:]; B2 = B2[1:,:]; B3 = B3[1:,:];
326         B = [B0.astype(int), B1.astype(int), B2.astype(int), B3.astype(int)]
327
328         return B
329
330 def adapt(u, mach, V, E, IE, BE, filepath):
331
332         flags = genflags(u, mach, V, E, IE, BE)          # Flag edges along the interior and exterior
333         Vcopy = genV(flags, V, E, IE, BE)                # With the flags determine the nodes on the
                    elements to split
334         Ucopy, Ecopy = genUE(u, Vcopy, V, E, IE, BE)     # Determine the new Elements and with them
                    the new U
335         B = genB(u, V, Vcopy, BE)                        # With the old boundary edges determine
                    which are on the edges
336         IEcopy, BEcopy = edgehash(Ecopy, B)
337
338
339         # Prepare for input to writegri
340         Mesh = {'V':Vcopy, 'E':Ecopy, 'IE':IEcopy, 'BE':BEcopy, 'Bname':['Engine', 'Exit', 'Outflow',
                    'Inflow'] }
341         writegri(Mesh, filepath)
342
343         return Ucopy, Vcopy, Ecopy, IEcopy, BEcopy
```

**Listing 4:** Calculating flux code.

```
import numpy as np
from numpy import linalg as LA

gam = 1.4

def RoeFlux(Ul, Ur, n):
    # Left side arguments
    rhol = Ul[0]; ul = Ul[1]/rhol; vl = Ul[2]/rhol; rhoEl = Ul[3]
    pl = (gam-1)*(rhoEl-0.5*rhol*(ul**2 + vl**2))
    Hl = (rhoEl + pl)/rhol

    # Right side arguments
    rhor = Ur[0]; ur = Ur[1]/rhor; vr = Ur[2]/rhor; rhoEr = Ur[3]
    pr = (gam-1)*(rhoEr-0.5*rhor*(ur**2 + vr**2))
    Hr = (rhoEr + pr)/rhor

    # Left and Right side fluxes
    FL = np.array([np.dot([Ul[1],Ul[2]], n), np.dot([Ul[1]*ul+pl, Ul[2]*ul], n), np.dot([Ul[1]*vl,
        Ul[2]*vl+pl], n), Hl*np.dot([Ul[1],Ul[2]], n)])
    FR = np.array([np.dot([Ur[1],Ur[2]], n), np.dot([Ur[1]*ur+pr, Ur[2]*ur], n), np.dot([Ur[1]*vr,
        Ur[2]*vr+pr], n), Hr*np.dot([Ur[1],Ur[2]], n)])

    # Roe-Averages
    RHS, ls = ROE_Avg(ul,vl,rhol,Hl,rhoEl, ur,vr,rhor,Hr,rhoEr, n)
    F = 0.5*(FL + FR) - 0.5*RHS

    return F, FL, FR, ls

def ROE_Avg(ul,vl,rhol,Hl,rhoEl, ur,vr,rhor,Hr,rhoEr, n):
    vell = np.array([ul, vl]); velr = np.array([ur, vr])

    # Calculating Roe average
    v = (np.sqrt(rhol)*vell + np.sqrt(rhor)*velr)/(np.sqrt(rhol) + np.sqrt(rhor))
    H = (np.sqrt(rhol)*Hl + np.sqrt(rhor)*Hr)/(np.sqrt(rhol) + np.sqrt(rhor))

    # Calculating eigenvalues
    q = LA.norm(v)
    c = np.sqrt((gam-1.0)*(H - 0.5*q**2))
    u = np.dot(v, n)
    ls = abs(np.array([u+c, u-c, u]))

    # Apply the entropy fix
    ls[abs(ls) < 0.1*c] = ((0.1*c)**2 + ls[abs(ls) < 0.1*c]**2)/(2*0.1*c)

    delrho = rhor - rhol; delmo = np.array([rhor*ur - rhol*ul, rhor*vr - rhol*vl]); dele = rhoEr
        - rhoEl
    s1 = 0.5*(abs(ls[0]) + abs(ls[1])); s2 = 0.5*(abs(ls[0]) - abs(ls[1]))
    G1 = (gam-1.0)*(0.5*q**2*delrho - np.dot(v, delmo) + dele); G2 = -1.0*u*delrho + np.dot(
        delmo, n)
    C1 = G1*(c**-2)*(s1 - abs(ls[2])) + G2*(c**-1)*s2; C2 = G1*(c**-1)*s2 + (s1 - abs(ls[2]))*G2

    RHS = np.array([ls[2]*delrho+C1, ls[2]*delmo[0]+C1*v[0]+C2*n[0], ls[2]*delmo[1]+C1*v[1]+C2*
        n[1], ls[2]*dele+C1*H+C2*u])

    return RHS, max(ls)
```

**Listing 5:** Finite volume method code.

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
from flux import RoeFlux
from readgri import readgri, writegri

gam = 1.4

def getUinf():
    u0 = 1.0695
    rho = 3.74; p = 101e3; a = np.sqrt(gam*p/rho)
    uinf = np.transpose(np.array([rho, rho*u0/a, 0, 1/(gam*(gam-1)) + (u0/a)**2/2]))

    return uinf

def solve(u0, mesh):
    V = mesh['V']; E = mesh['E']; BE = mesh['BE']; IE = mesh['IE']

    uinf = getUinf()

    u = u0.copy()
    R = np.zeros((E.shape[0], 4)); dta = R.copy(); err = np.array([1]); itr = 0


    while err[err.shape[0]-1] > 2.5*10**(-3):
        R *= 0; dta *= 0
        for i in range(IE.shape[0]):
            n1, n2, e1, e2 = IE[i,:]
            xl = V[n1,:]; xr = V[n2,:]
            ul = u[e1,:]; ur = u[e2,:]

            dx = xr - xl; deltal = LA.norm(dx)
            nhat = np.array([dx[1], -dx[0]])/deltal
            F, FL, FR, ls = RoeFlux(ul, ur, nhat)
            R[e1,:] += F*deltal; R[e2,:] -= F*deltal
            dta[e1,:] += ls*deltal; dta[e2,:] += ls*deltal

        for i in range(BE.shape[0]):
            n1, n2, e1, bgroup = BE[i,:]
            xl = V[n1,:]; xr = V[n2,:]
            uedge = u[e1,:]

            dx = xr - xl; deltal = LA.norm(dx)
            nhat = np.array([dx[1], -dx[0]])/deltal

            if bgroup == 2: # Engine - Invscid
                vp = np.array([uedge[1], uedge[2]])/uedge[0]
                vb = vp - np.dot(vp, nhat)*nhat
                pb = 0.4*(uedge[3] - 0.5*uedge[0]*(vb[0]**2 + vb[1]**2))
                ignore, FL, FR, ls = RoeFlux(uedge, uinf, nhat)

                F = pb*np.array([0, nhat[0], nhat[1], 0])
            elif bgroup == 1: # Exit/Outflow - Supersonic Outflow
                F, FL, FR, ls = RoeFlux(uedge, uedge, nhat)
            elif bgroup == 0: # Inflow
                F, FL, FR, ls = RoeFlux(uedge, uinf, nhat)

            R[e1,:] += F*deltal
            dta[e1,:] += ls*deltal

        dta = 2/dta
        u -= np.multiply(dta, R)
        err = np.append(err, sum(sum(abs(R))))
```

```
64
65
66              print(' Iteration :␣%4d,␣Error:␣%.3e'%(itr, err[err.shape[0]-1])); itr += 1
67
68      return u, err[1:], V, E, BE, IE
```

**Listing 6:** Main driving code of CFD.

```
import numpy as np
from numpy import linalg as LA
import random
import matplotlib.pyplot as plt
import time

# Project specific functions
from readgri import readgri, writegri
from flux import RoeFlux
from fvm import solve
from adapt import adapt

plt.rc('text', usetex=True)
plt.rc('font', family='serif')

gam = 1.4

def getIC(Ne, u0):
    rho = 3.74; p = 10e3; a = np.sqrt(gam*p/rho)
    uinf = np.array([rho, rho*u0/a, 0, 1/(gam*(gam-1)) + (u0/a)**2/2])

    u0 = np.zeros((Ne, 4))
    for i in range(4):
        u0[:,i] = uinf[i]
    u0[abs(u0) < 10**-10]

    return u0

def post_process(u):
    uvel = u[:,1]/u[:,0]; v = u[:,2]/u[:,0]

    q = np.zeros(u.shape[0])
    for i in range(u.shape[0]):
        q[i] = LA.norm(np.array([uvel[i], v[i]]))

    P = (1.4 - 1)*(u[:,3] - 0.5*u[:,0]*q**2)
    H = (u[:,3] + P)/u[:,0]
    c = np.sqrt(0.4*(H - 0.5*q**2))
    mach = q/c

    Pt = P*(1 + 0.5*0.4*mach**2)**(1.4/0.4)

    return mach, Pt

def plotmesh(V, BE, E, saves, fname):

    f = plt.figure(figsize=(8,5))
    plt.triplot(V[:,0], V[:,1], E, 'k-')
    plt.scatter(V[:,0], V[:,1])
    for i in range(BE.shape[0]):
        plt.plot(V[BE[i,0:2],0], V[BE[i,0:2],1], '-', linewidth=2, color='blue')
    plt.axis('equal'); plt.axis('off')
    f.tight_layout()
    if saves:
        plt.savefig(fname, bbox_inches = 'tight')
    plt.show()

def genMesh():
    # Edit later for easier adaptability
    cords = np.array([[0,0], [4,0], [4, 3.25], [0, 3.25], [5, 0], [5, 3.25], [9,0],
    [9, 3.25], [31, 0], [31, 3.25], [53, 0], [53, 3.25], [53, 12], [31, 12], [9, 12],
    [53, 19.5], [31, 19.5], [9, 19.5], [0, 19.5], [0, 12], [4, 7], [5, 7.075],
    [5, 8.075], [4, 8.075], [1, 8.075], [1, 10.075], [0, 10.075], [0, 8.075],
```

```
64      [0,  7.075],  [0,  5.075],  [1,  5.075],  [1,  7.075]]) * 10**-3
65
66
67      fid = open('mesh0.gri', 'w')
68      fid.write('32 32 2\n')
69      for i in np.arange(len(cords)):
70          fid.write(str(cords[i,0]) + ' ' + str(cords[i,1]) + '\n')
71
72      # Define Edges
73      fid.write('3\n1 2 inflow\n28 29\n3 2 outflow\n11 12\n12 13\n13 16')
74      fid.write('\n26 2 edges\n1 2\n2 5\n5 7\n7 9\n9 11\n16 17\n17 18\n18 19\n19 20\n20 15\
            n15 8\n8 6\n6 22\n22 23\n23 24\n24 25\n25 26\n26 27\n27 28\n29 30\n30 31\n31
            32\n32 21\n21 3\n3 4\n4 1')
75      fid.write('\n32 1 TriLagrange\n')
76      fid.write('1 3 4\n1 2 3\n2 5 3\n3 5 6\n5 7 6\n6 7 8\n7 9 8\n8 9 10\n9 12 10\n9 11 12\
            n10 12 14\n12 13 14\n8 10 14\n8 14 15\n13 16 17\n13 17 14\n14 17 15\n15 17 18\
            n15 18 19\n15 19 20')
77      fid.write('\n3 6 21 \n6 22 21\n21 22 24\n22 23 24\n21 24 25\n21 25 32\n25 26 27\n25
            27 28\n25 28 32\n28 29 32\n29 31 32\n29 30 31')
78      fid.close()
79
80
81      mesh = readgri('mesh0.gri')
82      V = mesh['V']; E = mesh['E']; BE = mesh['BE']; IE = mesh['IE']
83      plotmesh(V, BE, E, True, 'figs/mesh0.pdf')
84
85
86      for i in range(5):
87          mesh = readgri('mesh' + str(i) + '.gri')
88          V = mesh['V']; E = mesh['E']; BE = mesh['BE']; IE = mesh['IE']
89
90          u = getIC(E.shape[0], 0.05)
91          Ucopy, V, E, IE, BE = adapt(u, 0.05, V, E, IE, BE, 'mesh' + str(i+1) + '.gri')
92          print('Num Vertices ' + str(V.shape[0]) + ',\t Num of Triangles ' + str(E.shape[0]))
93          plotmesh(V, BE, E, True, 'figs/mesh' + str(i+1) + '.pdf')
94
95
96  def main():
97      starttime = time.time()
98      saves = True
99      u0 = 1.065
100     mesh = readgri('mesh4.gri')
101     V = mesh['V']; E = mesh['E']; BE = mesh['BE']; IE = mesh['IE']
102
103     u = getIC(E.shape[0], u0)
104     u, err, V, E, BE, IE = solve(u, mesh)
105     mach, pt = post_process(u); endtime = time.time(); print('Elapsed Time = ' + str((endtime -
            starttime)/60) + ' min')
106
107     f = plt.figure(figsize=(6,3))
108     plt.tripcolor(V[:,0], V[:,1], triangles=E, facecolors=u[:,0], cmap='jet')
109     plt.axis('equal'); plt.axis('off')
110     cbar = plt.colorbar(orientation='horizontal')
111     cbar.set_label(r'Density, $kg/m^3$')
112     f.tight_layout();
113     if saves: plt.savefig('figs/density.pdf', bbox_inches='tight')
114     plt.show()
115
116     unorms = np.sqrt(u[:,1]**2 + u[:,2]**2)
117     f = plt.figure(figsize=(6,3))
118     plt.tripcolor(V[:,0], V[:,1], triangles=E, facecolors=unorms, cmap='jet')
119     plt.axis('equal'); plt.axis('off')
120     cbar = plt.colorbar(orientation='horizontal')
121     cbar.set_label(r'Speed field , $m/s$')
122     f.tight_layout();
```

```
123         if saves: plt.savefig('figs/speed.pdf', bbox_inches='tight')
124         plt.show()
125
126 if __name__ == "__main__":
127         genMesh()
128         main()
```