gas_dynamics

Release 0.4.2

Fernando de la Fuente

Sep 30, 2022

INTRODUCTION

Package containing functions for working with compressible flow.

```
pip install gas-dynamics
```

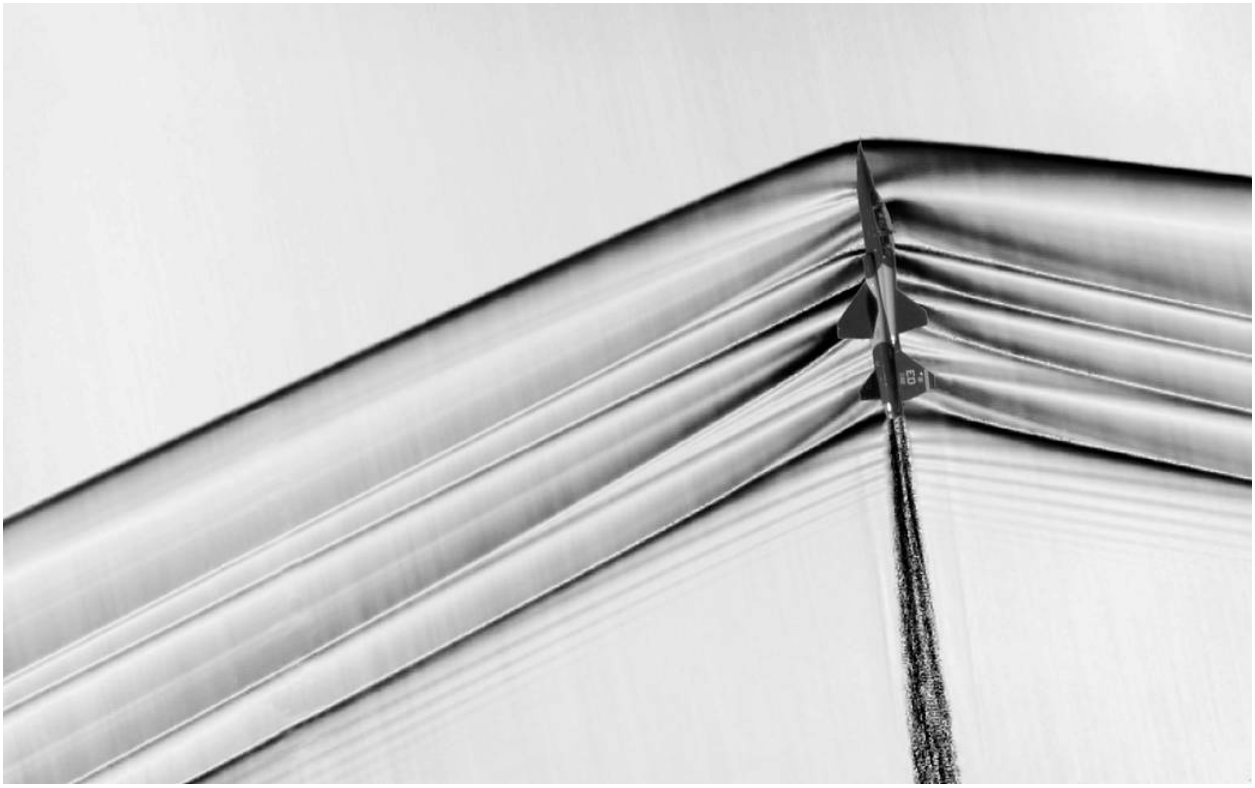


Fig. 1: Credits: NASA Images

Last edited: Sep 30, 2022

CHAPTER

ONE

GETTING STARTED

Gas dynamics equations, table generators, oblique shock chart generators and more.

To install with pip, use

```
pip install gas_dynamics
```

Included are functions to solve problems relating to compressible flow, from stagnation relations to determining the mach number from changes in local properties. Tables can be made for any gas and its respective ratio of specific heats, as well as plots and charts of relationships.

All functions contain an argument to specify the fluid so as to obtain the appropriate ratio of specific heats and gas constant. If the fluid is not specified, the default argument is for air with a ratio of specific heats of 1.4 and a gas constant of 286.9 J / kg K. Alternatively, if you want to specify the ratio of specific heats and gas constant directly, a class for fluids exists to initialize a fluid and its properties, as well as keep track of units.

```
>>> import gas_dynamics as gd

Mach number after a normal shock for air
>>> M2 = gd.shock_mach(mach=1.25)
>>> M2
0.8126360553720011
>>>

Mach number after a normal shock for argon
>>> from gas_dynamics.fluids import argon
>>> M2 = gd.shock_mach(mach=1.25, gas=argon)
>>> M2
0.8184295177443512
>>>

Mach number for a user-defined fluid
>>> methane = gd.fluid(name='methane', gamma=1.33, R=96.4, units='ft-lbf/lbm-R')
>>> methane.name, methane.gamma, methane.R, methane.units
('methane', 1.33, 96.4, 'ft-lbf/lbm-R')
>>> M2 = gd.shock_mach(mach=1.25, gas=methane)
>>> M2
0.810574008582977
>>>
```

Generate the isentropic flow tables for a range of Mach numbers and for a given gas.

```
>>> from gas_dynamics.fluids import nitrogen
>>> gd.stagnation_ratios(range=[.1,5], step=.2, gas=nitrogen)
Isentropic Flow Parameters for  = 1.4
M: 0.100   |   P/Pt: 0.993    |    T/Tt: 0.998    |    A/A*: 5.822    |   rho/rho_t: 0.
↪995
M: 0.300   |   P/Pt: 0.939    |    T/Tt: 0.982    |    A/A*: 2.035    |   rho/rho_t: 0.
↪956
M: 0.500   |   P/Pt: 0.843    |    T/Tt: 0.952    |    A/A*: 1.340    |   rho/rho_t: 0.
↪885
M: 0.700   |   P/Pt: 0.721    |    T/Tt: 0.911    |    A/A*: 1.094    |   rho/rho_t: 0.
↪792
M: 0.900   |   P/Pt: 0.591    |    T/Tt: 0.861    |    A/A*: 1.009    |   rho/rho_t: 0.
↪687
M: 1.100   |   P/Pt: 0.468    |    T/Tt: 0.805    |    A/A*: 1.008    |   rho/rho_t: 0.
↪582
M: 1.300   |   P/Pt: 0.361    |    T/Tt: 0.747    |    A/A*: 1.066    |   rho/rho_t: 0.
↪483
M: 1.500   |   P/Pt: 0.272    |    T/Tt: 0.690    |    A/A*: 1.176    |   rho/rho_t: 0.
↪395
M: 1.700   |   P/Pt: 0.203    |    T/Tt: 0.634    |    A/A*: 1.338    |   rho/rho_t: 0.
↪320
M: 1.900   |   P/Pt: 0.149    |    T/Tt: 0.581    |    A/A*: 1.555    |   rho/rho_t: 0.
↪257
M: 2.100   |   P/Pt: 0.109    |    T/Tt: 0.531    |    A/A*: 1.837    |   rho/rho_t: 0.
↪206
M: 2.300   |   P/Pt: 0.080    |    T/Tt: 0.486    |    A/A*: 2.193    |   rho/rho_t: 0.
↪165
M: 2.500   |   P/Pt: 0.059    |    T/Tt: 0.444    |    A/A*: 2.637    |   rho/rho_t: 0.
↪132
M: 2.700   |   P/Pt: 0.043    |    T/Tt: 0.407    |    A/A*: 3.183    |   rho/rho_t: 0.
↪106
M: 2.900   |   P/Pt: 0.032    |    T/Tt: 0.373    |    A/A*: 3.850    |   rho/rho_t: 0.
↪085
M: 3.100   |   P/Pt: 0.023    |    T/Tt: 0.342    |    A/A*: 4.657    |   rho/rho_t: 0.
↪069
M: 3.300   |   P/Pt: 0.017    |    T/Tt: 0.315    |    A/A*: 5.629    |   rho/rho_t: 0.
↪056
M: 3.500   |   P/Pt: 0.013    |    T/Tt: 0.290    |    A/A*: 6.790    |   rho/rho_t: 0.
↪045
M: 3.700   |   P/Pt: 0.010    |    T/Tt: 0.268    |    A/A*: 8.169    |   rho/rho_t: 0.
↪037
M: 3.900   |   P/Pt: 0.008    |    T/Tt: 0.247    |    A/A*: 9.799    |   rho/rho_t: 0.
↪030
M: 4.100   |   P/Pt: 0.006    |    T/Tt: 0.229    |    A/A*: 11.715    |   rho/rho_t: 0.
↪025
M: 4.300   |   P/Pt: 0.004    |    T/Tt: 0.213    |    A/A*: 13.955    |   rho/rho_t: 0.
↪021
M: 4.500   |   P/Pt: 0.003    |    T/Tt: 0.198    |    A/A*: 16.562    |   rho/rho_t: 0.
↪017
M: 4.700   |   P/Pt: 0.003    |    T/Tt: 0.185    |    A/A*: 19.583    |   rho/rho_t: 0.
↪015
M: 4.900   |   P/Pt: 0.002    |    T/Tt: 0.172    |    A/A*: 23.067    |   rho/rho_t: 0.
↪012
```

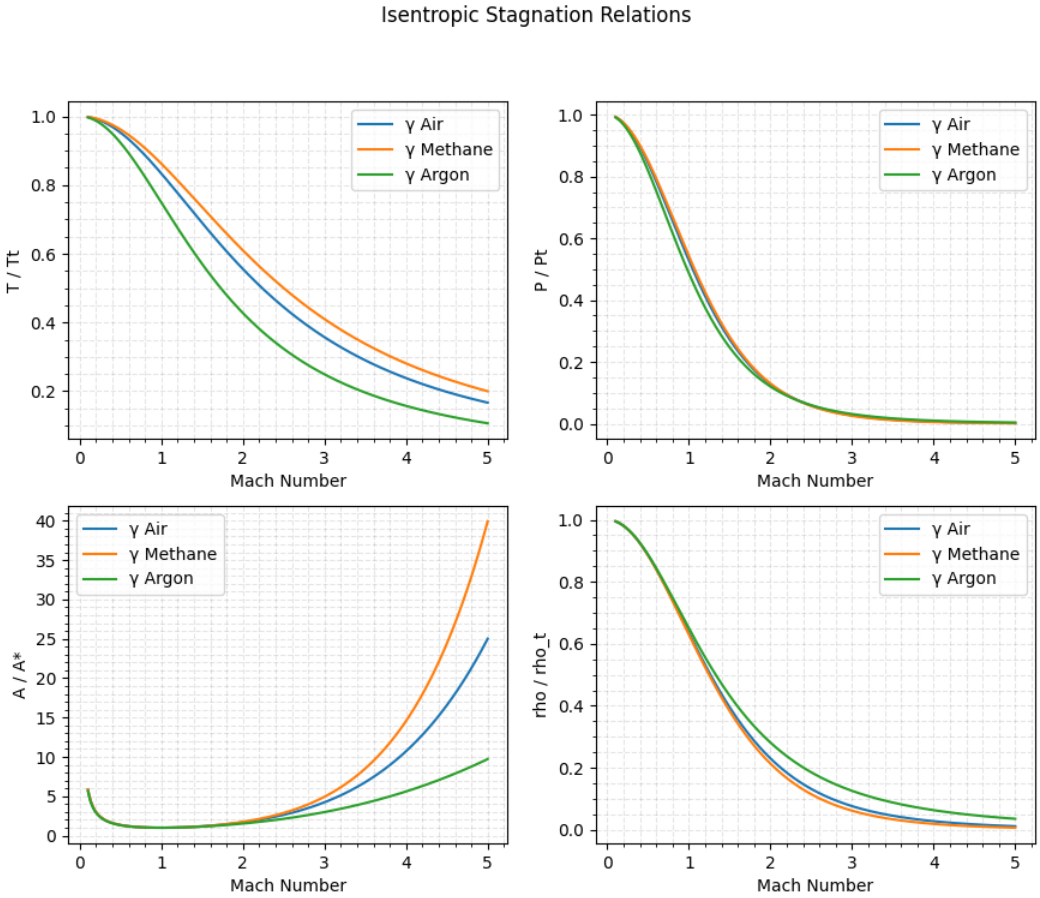
(continues on next page)

(continued from previous page)

```
M: 5.100   |   P/Pt: 0.002    |    T/Tt: 0.161    |    A/A*: 27.070    |   rho/rho_t: 0.
↪010
```

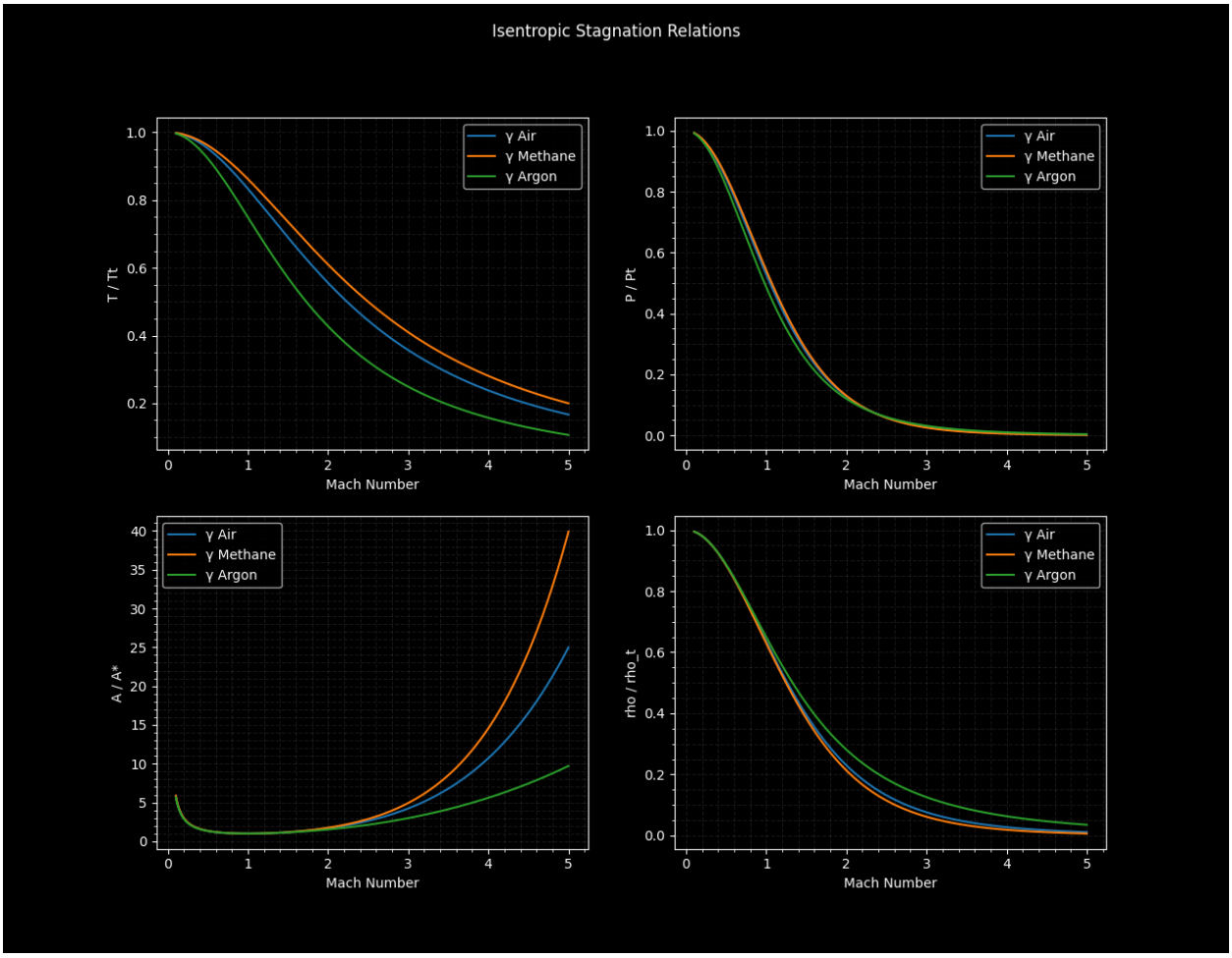
Plotting Stagnation relations versus mach number for different gammas. Arguments are the mach number range, increment, and a list of gasses.

```
plot_stagnation_ratios(dark=False)
```



Additionally, plots are available in dark mode.

```
plot_stagnation_ratios(dark=True)
```



All of the stagnation ratios are available, for example: Return the area ratio required to accelerate a flow to M = 3 and the corresponding stagnation pressure and temperature ratio

```
>>> import gas_dynamics as gd
>>> A_Astar =gd.mach_area_ratio_choked(mach=3)
>>> A_Astar
4.23456790123457
>>> p_pt = gd.stagnation_pressure_ratio(mach=3)
>>> p_pt
0.027223683703862817
>>> Tt = gd.stagnation_temperature_ratio(mach=3)
>>> Tt
0.35714285714285715
>>>
```

For the stagnation pressure and stagnation temperature relations, if two of the three necessary arguments are provided, the function will return the missing argument.

```
>>> pt = gd.stagnation_pressure(pressure=10, mach=1)
>>> pt
18.92929158737854
>>> M = gd.stagnation_pressure(pressure=10, stagnation_pressure=pt)
>>> M
```

(continues on next page)

(continued from previous page)

```
1.0
>>>
>>> Tt = gd.stagnation_temperature(temperature=300, mach=1)
>>> Tt
360.0
>>> M = gd.stagnation_temperature(temperature=300, stagnation_temperature=Tt)
>>> M
1.0
```

Some miscellaneous valuable functions are also included to calculate flow rates or areas required for choked flow

```
>>> mdot = 5 #kg/s
>>> mdot_per_area = gd.mass_flux_max(1000000, 300) #units are in Pascals
>>> mdot_per_area
2333.558560606226
>>> throat_area = mdot / mdot_per_area
>>> throat_area             #units are in meters squared
0.0021426503214477164
>>>
```

Determine the Mach number before and after a normal shock

```
>>> M2 = gd.shock_mach(mach=1.5)
>>> M2
0.7010887416930995
>>> M1 = gd.shock_mach_before(M2)
>>> M1
1.4999999999999998
>>>
```

Generate the shock tables using

```
>>> gd.shock_tables(range=[1,2], step=.1)
Normal Shock Parameters for Air,  = 1.4
M: 1.00   |   M2: 1.0000   |    p2/p1: 1.0000   |    T2/T1: 1.0000   |   dV/a: 0.0000  ␣
↪|   pt2/pt1: 1.000000
M: 1.10   |   M2: 0.9118   |    p2/p1: 1.2450   |    T2/T1: 1.0649   |   dV/a: 0.1591  ␣
↪|   pt2/pt1: 0.998928
M: 1.20   |   M2: 0.8422   |    p2/p1: 1.5133   |    T2/T1: 1.1280   |   dV/a: 0.3056  ␣
↪|   pt2/pt1: 0.992798
M: 1.30   |   M2: 0.7860   |    p2/p1: 1.8050   |    T2/T1: 1.1909   |   dV/a: 0.4423  ␣
↪|   pt2/pt1: 0.979374
M: 1.40   |   M2: 0.7397   |    p2/p1: 2.1200   |    T2/T1: 1.2547   |   dV/a: 0.5714  ␣
↪|   pt2/pt1: 0.958194
M: 1.50   |   M2: 0.7011   |    p2/p1: 2.4583   |    T2/T1: 1.3202   |   dV/a: 0.6944  ␣
↪|   pt2/pt1: 0.929787
M: 1.60   |   M2: 0.6684   |    p2/p1: 2.8200   |    T2/T1: 1.3880   |   dV/a: 0.8125  ␣
↪|   pt2/pt1: 0.895200
M: 1.70   |   M2: 0.6405   |    p2/p1: 3.2050   |    T2/T1: 1.4583   |   dV/a: 0.9265  ␣
↪|   pt2/pt1: 0.855721
M: 1.80   |   M2: 0.6165   |    p2/p1: 3.6133   |    T2/T1: 1.5316   |   dV/a: 1.0370  ␣
↪|   pt2/pt1: 0.812684
```

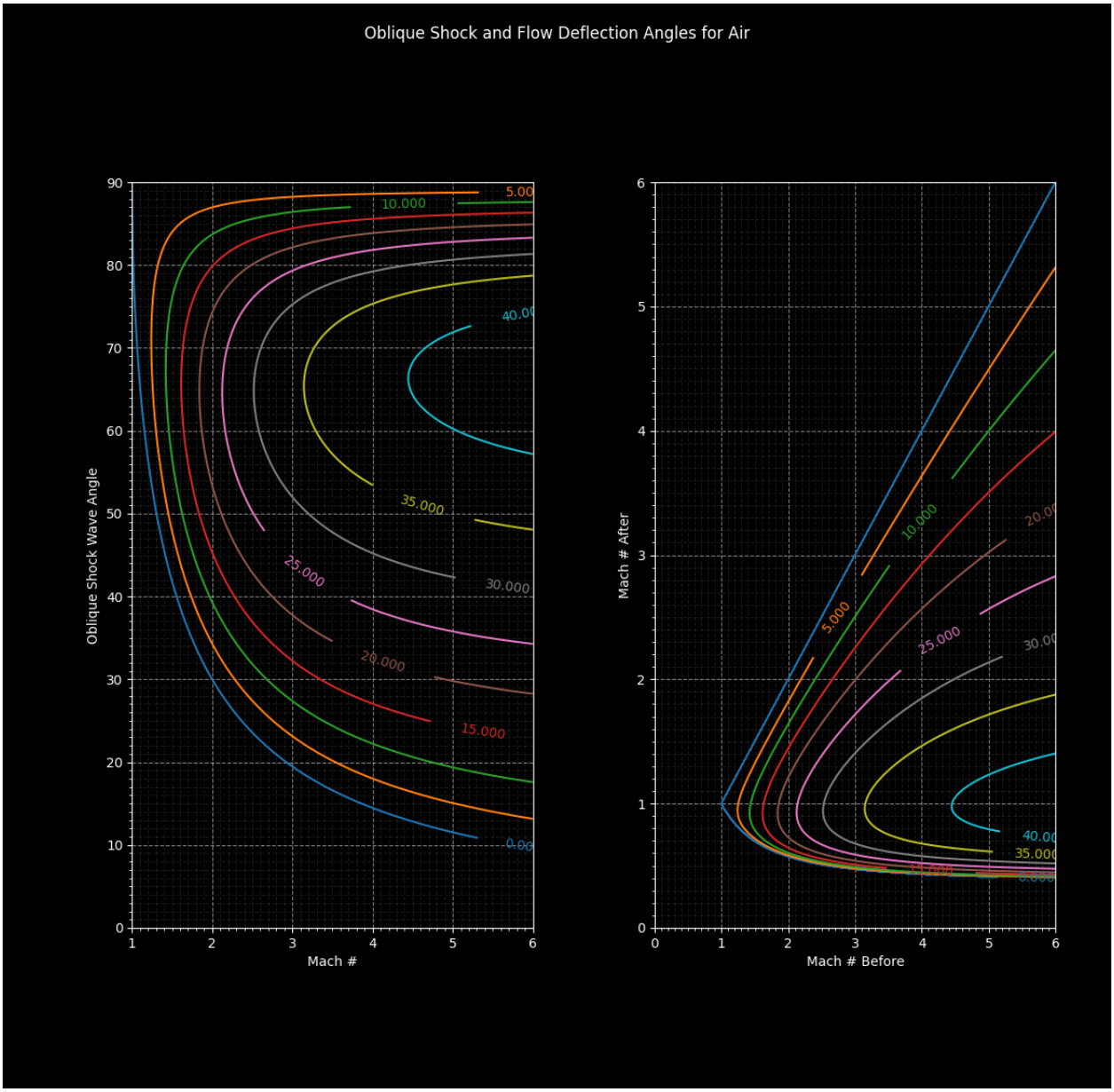
(continues on next page)

(continued from previous page)

```
M: 1.90   |   M2: 0.5956   |    p2/p1: 4.0450   |    T2/T1: 1.6079   |   dV/a: 1.1447  ␣
↪|   pt2/pt1: 0.767357
M: 2.00   |   M2: 0.5774   |    p2/p1: 4.5000   |    T2/T1: 1.6875   |   dV/a: 1.2500  ␣
↪|   pt2/pt1: 0.720874
```

Extremely useful in solving flow deflection problems are the oblique shock charts, so those are provided. For more precise solutions, equation solvers are embedded in the functions to find the exact values for the strong and weak shock solutions.

```
gd.shock_oblique_charts()
```




```
>>> deflect = gd.shock_flow_deflection(mach=2, shock_angle=40)
>>> deflect
```

(continues on next page)

(continued from previous page)

```
10.62290962494955
>>>
```

Get the strong and weak shock solution for a flow deflection

```
>>> shock_angles = gd.shock_angle(mach=2, flow_deflection=10)
>>> shock_angles
[39.31393184481759, 83.70008037574696]
>>>
```

Solve for the Mach number

```
>>> M = gd.shock_mach_given_angles(shock_angle=40, flow_deflection=10)
>>> M
1.9667966166259971
>>>
```

CHAPTER

TWO

THERMO AND FLUID SCIENCES REVIEW

A brief review of thermodynamics.

2.1 The Laws

Before discussing the 0th, 1st, and 2nd laws, it is important to acknowledge the existence of a relation betweeen the properties of state. This is sometimes referred to as the 00 law. Behold the equation of state

$$p = \rho RT$$

2.1.1 0

The zeroth law states that two systems in thermal equilibrium with a third are in thermal equilbrium with each other.

2.1.2 1

The first law of thermodynamics deals with the conservation of energy and can be stated in a variety of ways. The simplest statement for a closed system is by far

$$\Sigma Q = \Sigma W$$

where ΣQ is the heat transferred into the system, and ΣW is the work transferred from the system.

On a unit mass basis, one can also say

$$\delta q = \delta w + de$$

We take special care to note δq and δw as inexact differentials, or path dependent, while de is an exact differential and represents the change only between an initial and final state.

We can continue to expand the expression for the first law knowing that $e \equiv u + \frac{v^2}{2} + gz$ and say

$$\delta q = \delta w + u + \frac{v^2}{2} + gz$$

Finally we arrive at

$$0 = \delta q + \delta w + \Delta u + \frac{\Delta v^2}{2} + g\Delta z$$

All the equations above are different ways to say the same fundamental thing - energy is conserved. Heat and work are two types of energy in transit, heat being the transfer of energy due to a temperature gradient (and always from the higher temperature system to the lower), and work being the effect of elevating a mass in a gravity field.

2.1.3 2

The second law of thermodynamics points to the very direction of all processes in our universe. Carnot, Clausius, and Lord Kelvin all came to this significant conclusion while working to understand things like steam engines and machines. Kelvin and Planck stated that it is impossible for an engine operating in a cycle to produce net work when exchanging heat with only one temperature source. In laymans terms, this can be thought as a machine using energy from a thermal reservoir cannot operate unless there is another, lower, thermal reservoir.

Clausius determined that heat can never pass from a cold body to a warmer body without the aid of an external process, giving recognition of the irreversibility of certain processes such as fluid friction and heat transfer. All real processes are irreversible. For a fictitious reversible process, we can quantify this degredation of energy quality by defining entropy.

$$\Delta S \equiv \int \frac{\delta Q_r}{T}$$

$$ds \equiv \frac{\delta q_r}{T}$$

This defines a change in entropy in terms of a reversible addition of heat δq (which we know is an impossible process) for a system temperature. We can write the definition of entropy as such because entropy is a state, and is independent of the process. Regardless, it may be more prudent to write the definition of entropy as

$$ds \geq \frac{\delta q_r}{T}$$

or

$$ds = \frac{\delta q}{T}$$

Where the first expression shows we understand that we are always trending towards greater entropy, and the second stating that the change in entropy is the heat we add in addition to irreversible contributions from friction, viscosity, thermal conductivity, and so on.

2.2 Property Relations, Entropy, and Perfect Gases

Enthalpy is defined as the specific internal energy of a system and the product of pressure and volume, or the sum of thermal energy and "flow work".

$$h = u + pv$$

Recalling our definition of work and our definition entropy on a unit mass basis for a reversible system.

$$\delta q = Tds$$

$$\delta w = pdv$$

we can come to the result

$$Tds = du + pdv$$

We can further manipulate this result by differentiating the definition of enthalpy and replacing the du term.

$$dh = du + pdv + vdp$$

$$Tds = dh - vdp$$

When it comes to perfect gases (substances that follow the perfect gas equation of state), specific heat at constant volume and specific heat at constant pressure are defined as follows:

$$c_v \equiv \left(\frac{\partial u}{\partial T}\right)_v$$

$$c_p \equiv \left(\frac{\partial h}{\partial T}\right)_p$$

Since both of these are functions of temperature only, we can lose the partial derivative.

$$\Delta u = c_v \Delta T$$

$$\Delta h = c_p \Delta T$$

An very common term used in gas dynamics is the ratio of specific heats γ

$$\gamma \equiv \frac{c_p}{c_v}$$

γ can be thought of the ratio of the fluids specific internal energy and pressure volume ability's to take or remove heat away from a system to just the internal energy's ability to do so.

2.3 The Liquid Vapor Dome

2.4 Example Derivations

2.4.1 Pumps

The energy equation is a great way to understand pumps and their operation.

$$w_1 + q_1 + h_1 + \frac{{v_1}^2}{2} + gz_1 = w_2 + q_2 + h_2 + \frac{{v_2}^2}{2} + gz_2$$

We can then multiply through by $\dot{m}$, and remove the terms for heat in, heat out, and work out. The work is being imparted to our fluid and heat in and out are negligent compared to the other forms of energy in this process.

$$W = \dot{m}\left[\left(h_1 + \frac{{v_1}^2}{2} + gz_1\right) - \left(h_2 + \frac{{v_2}^2}{2} + gz_2\right)\right]$$

We know that enthalpy is defined as $h = u + pv$, and we rearrange to solve for the common "pump head" equation.

$$\frac{W}{\dot{m}} = g\Delta H + \Delta u$$

where W is the shaft power, u is specific internal energy, and H is known as the pump head, defined as

$$H = \frac{p}{\rho g} + \frac{V^2}{2g} + Z$$

Of course there will probably be losses in our system, such as the marginal increase of heat imparted to the fluid to slippage from our pump, so our equation should actually read something like

$$g\Delta H < \frac{W}{\dot{m}}$$

The pump efficiency can then be defined as

$$\eta = \frac{g\Delta H\dot{m}}{W}$$

The efficiency term can further be broken down to capture efficiency of various aspects of the pump, such as mechanical efficiency (capturing external drag from bearings and seals), hydraulic efficiency (ratio of output head to input head), and volumetric efficiency (leakage of the fluid).

CHAPTER

THREE

COMPRESSIBLE FLOW

3.1 Compressibility

3.2 The Mach Number

CHAPTER

FOUR

STANDARD EQUATIONS

4.1 Equation Map - Standard Equations

sonic_velocity

$$a = \sqrt{\gamma R T}$$

4.1.1 Stagnation Relations and Ratios

stagnation_pressure will return P, P_t, or M depending on parameters given .

$$p_t = p\left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}}$$

stagnation_temperature will return T, T_t, or M depending on parameters given.

$$T_t = T\left(1 + \frac{\gamma - 1}{2} M^2\right)$$

stagnation_pressure_ratio returns $\frac{P}{P_t}$

$$\frac{p}{p_t} = \frac{1}{\left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}}}$$

stagnation_temperature_ratio returns $\frac{T}{T_t}$

$$\frac{T}{T_t} = \frac{1}{\left(1 + \frac{\gamma - 1}{2} M^2\right)}$$

stagnation_density_ratio returns $\frac{\rho}{\rho_t}$.

$$\frac{\rho}{\rho_t} = \left(\frac{1}{1 + \frac{\gamma - 1}{2} M^2}\right)^{\frac{1}{\gamma - 1}}$$

4.1.2 Mach Number and Property Relations

mach_from_pressure_ratio solves for M_2 from the following equation, while *pressure_from_mach_ratio* will solve for p_2.

$$\frac{p_2}{p_1} = \left(\frac{1 + \frac{\gamma-1}{2}M_1^2}{1 + \frac{\gamma-1}{2}M_2^2}\right)^{\frac{\gamma}{\gamma-1}} e^{\frac{-\Delta s}{R}}$$

mach_from_temperature_ratio solves for M_2 from the following equation, while *temperature_from_mach_ratio* will solve for T_2.

$$\frac{T_2}{T_1} = \frac{1 + \frac{\gamma-1}{2}M_1^2}{1 + \frac{\gamma-1}{2}M_2^2}$$

mach_area_star_ratio returns the ratio of $\frac{A}{A*}$.

$$\frac{A}{A*} = \frac{1}{M}\left(\frac{1 + \frac{\gamma-1}{2}M^2}{\frac{\gamma+1}{2}}\right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

mach_area_ratio returns the ratio of $\frac{A_2}{A_1}$ given two Mach numbers, whi;e *mach_from_area_ratio* will return the possible mach numbers that satisfy the area ratio.

$$\frac{A_2}{A_1} = \frac{M_1}{M_2}\left(\frac{1 + \frac{\gamma-1}{2}M_2^2}{1 + \frac{\gamma-1}{2}M_1^2}\right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

4.1.3 Mass Flux

mass_flux returns the flow rate per unit area while `mass_flux_max` will return the maximum flow rate per unit area, where $M = 1$.

$$\frac{\dot{m}}{A} = M\left(1 + \frac{\gamma-1}{2}M^2\right)^{\frac{-(\gamma+1)}{2(\gamma-1)}} \sqrt{\left(\frac{\gamma}{R}\right)}\frac{p_t}{\sqrt{T_t}}$$

$$\frac{\dot{m}}{A^*} = \sqrt{\frac{\gamma}{R}\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}}\frac{p_t}{\sqrt{T_t}}$$

4.2 Worked Example

The year is 1970. You are an eager, young, bright eyed engineer working at Aerojet Rocketdyne tasked with designing the nozzle for the Space Shuttle main engine, the RS-25. The design team is not sure when or where maximum dynamic pressure will occur, so to play it safe they decide that all engines (and consequently, their nozzles) should be operating ideally at around 55,000 ft above sea level where the air pressure is approximately 1.3 psi. Additionally you know that the combustion chamber temperature and pressure are 6000 Fahrenheit and 3000 psi. You are asked to deliver several items:

- The exit Mach number.
- The area ratio.
- The maximum flow rate per unit area (mass flux), and the areas and diameters of the throat and exit, given a flowrate of 1100 lbm/s.

Fig. 1: Credits: NASA Images

- A plot of position in the nozzle versus temperature and pressure to deliver to the thermal and structural engineers.

Solution

Let us assume the fluid velocity inside the chamber is non-moving and therefore is at stagnation temperature and pressure, $T_t = 6000\,^\circ F$, and $p_t = 3000\,psi$. Assuming that our nozzle is functioning ideally, we also know what the ratio of stagnation pressures from the chamber to the throat and from the throat to the nozzle exit is unity. We come to the following conclusion.

$$\frac{p_{air}}{p_t} = \frac{1.3}{3000} = .000433$$

There are a couple of ways to go about this. We generate a coarse view of the tables (because we are feeling nostalgic and old school) and seek the Mach number that corresponds to the stagnation pressure ratio.

```
>>> import gas_dynamics as gd
>>> gd.stagnation_ratio_table([1,8], step=.2)
Isentropic Flow Parameters for Air,  = 1.4
M: 1.000   |   P/Pt: 0.52828    |    T/Tt: 0.8333    |    A/A*: 1.000    |   rho/rho_t:␣
↪0.634
M: 1.200   |   P/Pt: 0.41238    |    T/Tt: 0.7764    |    A/A*: 1.030    |   rho/rho_t:␣
↪0.531
M: 1.400   |   P/Pt: 0.31424    |    T/Tt: 0.7184    |    A/A*: 1.115    |   rho/rho_t:␣
↪0.437
M: 1.600   |   P/Pt: 0.23527    |    T/Tt: 0.6614    |    A/A*: 1.250    |   rho/rho_t:␣
↪0.356
M: 1.800   |   P/Pt: 0.17404    |    T/Tt: 0.6068    |    A/A*: 1.439    |   rho/rho_t:␣
↪0.287
M: 2.000   |   P/Pt: 0.12780    |    T/Tt: 0.5556    |    A/A*: 1.688    |   rho/rho_t:␣
↪0.230
M: 2.200   |   P/Pt: 0.09352    |    T/Tt: 0.5081    |    A/A*: 2.005    |   rho/rho_t:␣
↪0.184
M: 2.400   |   P/Pt: 0.06840    |    T/Tt: 0.4647    |    A/A*: 2.403    |   rho/rho_t:␣
↪0.147
M: 2.600   |   P/Pt: 0.05012    |    T/Tt: 0.4252    |    A/A*: 2.896    |   rho/rho_t:␣
↪0.118
M: 2.800   |   P/Pt: 0.03685    |    T/Tt: 0.3894    |    A/A*: 3.500    |   rho/rho_t:␣
↪0.095
M: 3.000   |   P/Pt: 0.02722    |    T/Tt: 0.3571    |    A/A*: 4.235    |   rho/rho_t:␣
↪0.076
M: 3.200   |   P/Pt: 0.02023    |    T/Tt: 0.3281    |    A/A*: 5.121    |   rho/rho_t:␣
↪0.062
M: 3.400   |   P/Pt: 0.01512    |    T/Tt: 0.3019    |    A/A*: 6.184    |   rho/rho_t:␣
↪0.050
M: 3.600   |   P/Pt: 0.01138    |    T/Tt: 0.2784    |    A/A*: 7.450    |   rho/rho_t:␣
↪0.041
M: 3.800   |   P/Pt: 0.00863    |    T/Tt: 0.2572    |    A/A*: 8.951    |   rho/rho_t:␣
↪0.034
M: 4.000   |   P/Pt: 0.00659    |    T/Tt: 0.2381    |    A/A*: 10.719    |   rho/rho_t:␣
↪0.028
M: 4.200   |   P/Pt: 0.00506    |    T/Tt: 0.2208    |    A/A*: 12.792    |   rho/rho_t:␣
↪0.023
M: 4.400   |   P/Pt: 0.00392    |    T/Tt: 0.2053    |    A/A*: 15.210    |   rho/rho_t:␣
↪0.019
M: 4.600   |   P/Pt: 0.00305    |    T/Tt: 0.1911    |    A/A*: 18.018    |   rho/rho_t:␣
```

(continues on next page)

(continued from previous page)

```
↪0.016
M: 4.800   |   P/Pt: 0.00239    |    T/Tt: 0.1783    |    A/A*: 21.264    |   rho/rho_t:␣
↪0.013
M: 5.000   |   P/Pt: 0.00189    |    T/Tt: 0.1667    |    A/A*: 25.000    |   rho/rho_t:␣
↪0.011
M: 5.200   |   P/Pt: 0.00150    |    T/Tt: 0.1561    |    A/A*: 29.283    |   rho/rho_t:␣
↪0.010
M: 5.400   |   P/Pt: 0.00120    |    T/Tt: 0.1464    |    A/A*: 34.175    |   rho/rho_t:␣
↪0.008
M: 5.600   |   P/Pt: 0.00096    |    T/Tt: 0.1375    |    A/A*: 39.740    |   rho/rho_t:␣
↪0.007
M: 5.800   |   P/Pt: 0.00078    |    T/Tt: 0.1294    |    A/A*: 46.050    |   rho/rho_t:␣
↪0.006
M: 6.000   |   P/Pt: 0.00063    |    T/Tt: 0.1220    |    A/A*: 53.180    |   rho/rho_t:␣
↪0.005
M: 6.200   |   P/Pt: 0.00052    |    T/Tt: 0.1151    |    A/A*: 61.210    |   rho/rho_t:␣
↪0.004
M: 6.400   |   P/Pt: 0.00042    |    T/Tt: 0.1088    |    A/A*: 70.227    |   rho/rho_t:␣
↪0.004
M: 6.600   |   P/Pt: 0.00035    |    T/Tt: 0.1030    |    A/A*: 80.323    |   rho/rho_t:␣
↪0.003
M: 6.800   |   P/Pt: 0.00029    |    T/Tt: 0.0976    |    A/A*: 91.594    |   rho/rho_t:␣
↪0.003
M: 7.000   |   P/Pt: 0.00024    |    T/Tt: 0.0926    |    A/A*: 104.143    |   rho/rho_
↪t: 0.003
M: 7.200   |   P/Pt: 0.00020    |    T/Tt: 0.0880    |    A/A*: 118.080    |   rho/rho_
↪t: 0.002
M: 7.400   |   P/Pt: 0.00017    |    T/Tt: 0.0837    |    A/A*: 133.520    |   rho/rho_
↪t: 0.002
M: 7.600   |   P/Pt: 0.00014    |    T/Tt: 0.0797    |    A/A*: 150.585    |   rho/rho_
↪t: 0.002
M: 7.800   |   P/Pt: 0.00012    |    T/Tt: 0.0759    |    A/A*: 169.403    |   rho/rho_
↪t: 0.002
M: 8.000   |   P/Pt: 0.00010    |    T/Tt: 0.0725    |    A/A*: 190.109    |   rho/rho_
↪t: 0.001
```

On second thought, let us be a little more precise and get the *exact* Mach number. From the table, we are expecting something around Mach = 6.4, and then an area ratio around 70.

```
>>> M_exit = gd.mach_from_pressure_ratio(pressure_initial=3000, pressure_final=1.3,mach_
↪initial=0)
>>> M_exit
6.379339932707969
>>> A_Astar = gd.mach_area_star_ratio(M_exit)
>>> A_Astar
69.24755332876032
>>>
```

$$M_{exit} = 6.379$$

$$\frac{A}{A*} = 69.25$$

We got something! Cool. Now lets tackle the nozzle area. Knowing that our flowrate is 1100 lbm/s, lets solve for $\frac{\dot{m}}{A*}$

and then divide out flowrate to get A*. Lets double check the function inputs while we're here.

```
>>> help(gd.mass_flux_max)
Help on function mass_flux_max in module gas_dynamics.standard.standard:

mass_flux_max(stagnation_pressure: float, stagnation_temperature: float, gas=<gas_
↪dynamics.fluids.fluid object at 0x00000240BB661D60>) -> float
    Returns the maximum flow rate per unit choked area

    Notes
    -----
    Given stagnation pressure, stagnation temperature, and the fluid,
    return the flow rate for a Mach number equal to 1. Default fluid
    is air.

    **Units**:

    J / kg-K and Pa return kg/m^2

    kJ / kg-K and kPa returns kg/m^2

    ft-lbf / lbm-R and psi returns lbm/in^2


    Parameters
    ----------
    stagnation_pressure : `float`
        The stagnation pressure.

    stagnation_temperature : `float`
        The stagnation temperature.

    gas : `fluid`
        A user defined fluid object. Default is air

    metric : `bool`
        Use metric or US standard.


    Returns
    -------
    float
        The maximum mass flux
```

It looks like our output is going to be in lbm/in^2. Our temperature should also be in Rankine instead of Fahrenheit, and we should be using air with the US standard properties.

```
>>> from gas_dynamics.fluids import air_us
>>> Temp_rankine = 6000 + 459.67
>>> chamber_pressure = 3000
>>> mdot = 1100
>>> flux = gd.mass_flux_max(stagnation_pressure=chamber_pressure, stagnation_
↪temperature=Temp_rankine, gas=air_us)
>>> flux
```

(continues on next page)

(continued from previous page)

```
19.857532983568127
>>> throat_area = flux**-1 * 1100
>>> throat_diameter = (throat_area*4/3.14159)**.5
>>> exit_area = A_Astar*throat_area
>>> exit_diameter = (exit_area*4/3.14159)**.5
>>> throat_area, throat_diameter
(55.394595134765076, 8.398252714991807)
>>> exit_area, exit_diameter
(3835.9401807197314, 69.88615638831808)
>>>
```

Let us reflect on some these results:

$$\frac{\dot{m}}{A^*} = 19.857\, lbm/in^2$$

$$A^* = 55.39\, in^2$$

$$d_{throat} = 8.39\, in$$

$$A_{exit} = 3835.94\, in^2$$

$$d_{exit} = 69.88\, in$$

At a temperature and pressure of 6459.67 Rankine and 3000 psi, the RS-25 will be pushing around 20 pound masses of combustion gases through a square inch every second. The throat needs to be approximately 8 inches diameter and the nozzle exit needs to be around 70 inches diameter in order to accelerate our fluid to the design Mach number.

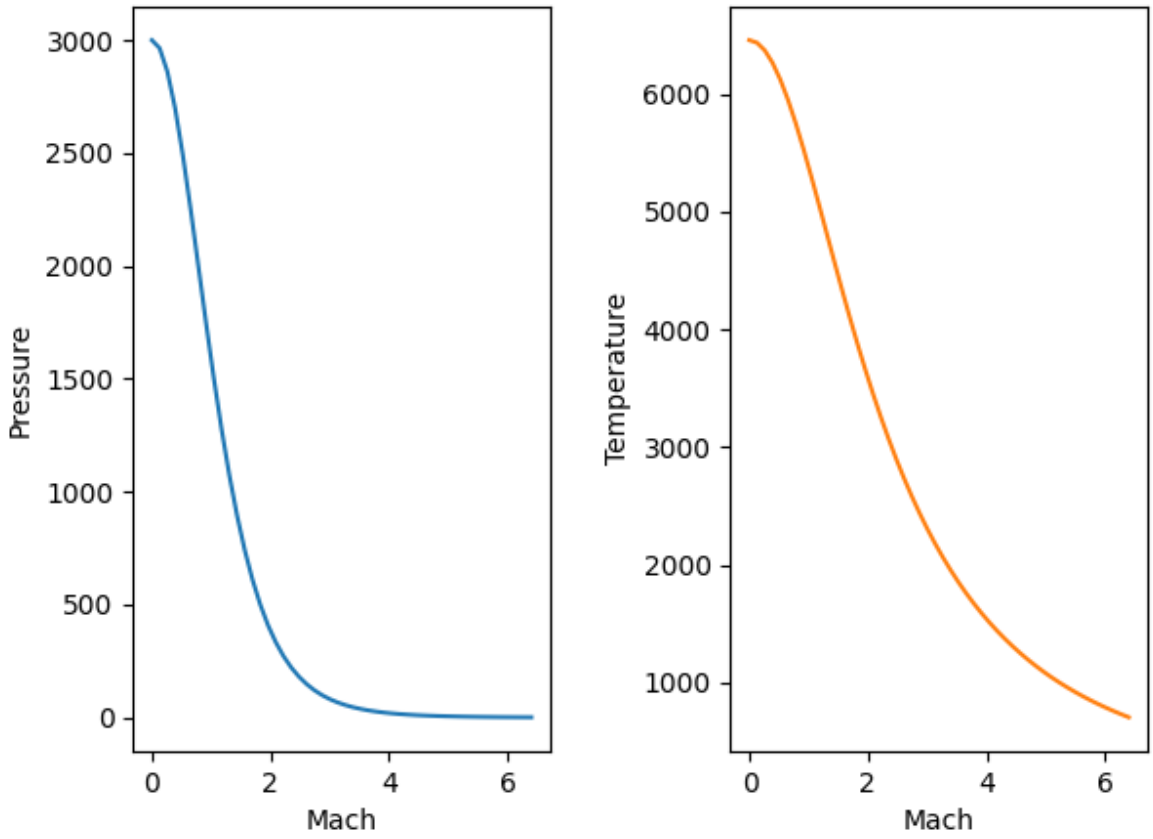
Finally, lets make those plots.

```
>>> import matplotlib as plt
>>> import numpy as np
>>> machs = np.linspace(0,6.4,100)
>>> pressures=[]
>>> temperatures=[]
>>> for m in machs:
        pressures.append(gd.stagnation_pressure(stagnation_pressure=3000,mach=m))
        temperatures.append(gd.stagnation_temperature(stagnation_temperature=6459.67,
↪mach=m))
>>>
>>> fig, (ax1,ax2) = plt.subplots(1,2)
>>> ax1.plot(machs,pressures,color='tab:blue')
>>> ax1.set_xlabel('Mach')
>>> ax1.set_ylabel('Pressure')
>>> ax2.plot(machs,temperatures,color='tab:orange')
>>> ax2.set_xlabel('Mach')
>>> ax2.set_ylabel('Temperature')
>>> fig.tight_layout(pad=2.0)
>>> plt.show()
```

We observe that the pressure and temperature sink quite drastically as we progress and accelerate through the nozzle. The structural engineer can take the day off but it looks like the thermal engineer has quite a lot of work to do to find out ways to cool the nozzle.

gas_dynamics.standard.standard.**sonic_velocity**(*temperature=273.15*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the local speed of sound.


Pressure
3000
2500
2000
1500
1000
500
0
0
2
4
6
Mach
Temperature
6000
5000
4000
3000
2000
1000
0
2
4
6
Mach

Notes

Given a ratio of specific heats, gas constant, and temperature this function returns the locoal speed of sound. Default fluid is air.

Parameters

- **gas** (*fluid*) – A user defined fluid object. Default is air
- **temperature** (*float*) – The temperature

Returns
The local speed of sound

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> gd.sonic_velocity(air, temperature=500)
448.2186966202994
>>> gd.sonic_velocity(gas=Argon, temperature=300)
```

gas_dynamics.standard.standard.**stagnation_pressure**(*stagnation_pressure=None*, *mach=None*, *pressure=None*, *gas=<gas_dynamics.fluids.fluid object>*, *output=False*) → float

Returns the stagnation pressure given pressure and Mach number.

Notes

Given a pressure, Mach number, and a ratio of specific heats return the stagnation pressure. Alternatively, provided two arguments the function will return the missing one. Default fluid is air.

Parameters

- **stagnation_pressure** (*float*) – The stagnation pressure.
- **pressure** (*float*) – The pressure.
- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **output** (*bool*) – Print out a string to verify the output is the parameter desired

Returns
The stagnation pressure, static pressure, or mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> pt = gd.stagnation_pressure(pressure=10, mach=1)
>>> pt
18.92929158737854
>>> M = gd.stagnation_pressure(pressure=10, stagnation_pressure=pt)
>>> M
1.0
>>>
```

gas_dynamics.standard.standard.**stagnation_temperature**(*temperature=None*, *stagnation_temperature=None*, *mach=None*, *gas=<gas_dynamics.fluids.fluid object>*, *output=False*) → float

Returns the stagnation temperature given temperature and Mach number.

Notes

Given a temperature, Mach number, and a ratio of specific heats this function returns the stagnation temperature. Alternatively, provided two arguments the function will return the missing one. Default fluid is air.

Parameters

- **stagnation_temperature** (*float*) – The stagnation temperature
- **temperature** (*float*) – The temperature
- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **output** (*bool*) – Print out a string to verify the output is the parameter desired

Returns
The stagnation temperature, temperature, or mach number

Return type
float

Examples

```
>>> Tt = gd.stagnation_temperature(temperature=300, mach=1)
>>> Tt
360.0
>>> M = gd.stagnation_temperature(temperature=300, stagnation_temperature=Tt)
>>> M
1.0
>>>
```

gas_dynamics.standard.standard.**stagnation_density**(*density=None*, *stagnation_density=None*, *mach=None*, *gas=<gas_dynamics.fluids.fluid object>*, *output=False*) → float

Returns the stagnation density given density and Mach number.

Notes

Given a density, Mach number, and a ratio of specific heats this function returns the stagnation density. Alternatively, provided two arguments the function will return the missing one. Default fluid is air.

Parameters

- **stagnation_density** (*float*) – The stagnation density
- **density** (*float*) – The density
- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **output** (*bool*) – Print out a string to verify the output is the parameter desired

Returns
The stagnation temperature, temperature, or mach number

Return type
float

Examples

gas_dynamics.standard.standard.**stagnation_pressure_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the pressure ratio of p / p_t

Notes

Given a Mach number and ratio of specific heats return the relation of pressure over stagnation pressure. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The stagnation pressure ratio

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> p_pt = gd.stagnation_pressure_ratio(mach=3)
>>> p_pt
0.027223683703862817
>>>
```

gas_dynamics.standard.standard.**stagnation_temperature_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the temperature ratio of T / T_t

Notes

Given a Mach number and ratio of specific heats return the relation of temperature over stagnation temperature. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The stagnation temperature ratio

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> T_Tt = gd.stagnation_temperature_ratio(mach=1.5)
>>> T_Tt
0.6896551724137931
>>>
```

gas_dynamics.standard.standard.**stagnation_density_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the density ratio rho / rho_t

Notes

Given a Mach number and ratio of specific heats, return the relation of density over stagnation density. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach #
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The stagnation density ratio

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> rho_rho_t = gd.stagnation_density_ratio(mach=1.5)
>>> rho_rho_t
0.39498444639115327
>>>
```

gas_dynamics.standard.standard.**stagnation_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → list

Return stagnation pressure, temperature, density, and choked area ratio for a mach number

Notes

Given a mach number and the fluid, return the three stagnation ratios and the ratio of the area to the choked area. Default fluid is air.

Parameters

- **mach** (*float*) – The mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The stagnation pressure, stagnation temperature, stagnation density ratio, and choked area ratio.

Return type
list

Examples

```
>>> import gas_dynamics as gd
```

gas_dynamics.standard.standard.**stagnation_ratio_table**(*range=[0, 5], step=0.1, gas=<gas_dynamics.fluids.fluid object>*) → str

Returns the isentropic flow tables in the given range.

Notes

Given a ratio of specific heats, print out the stagnation temperature ratio, stagnation pressure ratio, the area to choked area ratio, and the stagnation density ratio for every incremental Mach number.

Parameters

- **range** (*list*) – The starting and ending Mach numbers in a list, ex: [0, 5]
- **step** (*float*) – The step size between min and max mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The isentropic flow table

Return type
str

Examples

```
>>> import gas_dynamics as gd
>>> gd.stagnation_ratios(range=[0,2], step=.2, gas='nitrogen')
M: 0.000   |   P/Pt: 1.000    |    T/Tt: 1.000    |    A/A*: inf
M: 0.200   |   P/Pt: 0.972    |    T/Tt: 0.992    |    A/A*: 2.964
M: 0.400   |   P/Pt: 0.896    |    T/Tt: 0.969    |    A/A*: 1.590
M: 0.600   |   P/Pt: 0.784    |    T/Tt: 0.933    |    A/A*: 1.188
M: 0.800   |   P/Pt: 0.656    |    T/Tt: 0.887    |    A/A*: 1.038
M: 1.000   |   P/Pt: 0.528    |    T/Tt: 0.833    |    A/A*: 1.000
M: 1.200   |   P/Pt: 0.412    |    T/Tt: 0.776    |    A/A*: 1.030
M: 1.400   |   P/Pt: 0.314    |    T/Tt: 0.718    |    A/A*: 1.115
M: 1.600   |   P/Pt: 0.235    |    T/Tt: 0.661    |    A/A*: 1.250
M: 1.800   |   P/Pt: 0.174    |    T/Tt: 0.607    |    A/A*: 1.439
M: 2.000   |   P/Pt: 0.128    |    T/Tt: 0.556    |    A/A*: 1.688
>>>
```

gas_dynamics.standard.standard.**mach_from_pressure_ratio**(*pressure_initial: float*, *pressure_final: float*, *mach_initial: float*, *entropy=0*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number given a Mach number and the local pressures

Notes

Given the local pressure in two regions and the Mach number in one, return the Mach number in the second region. Default arguments are for air and isentropic flow.

Parameters

- **pressure_initial** (*float*) – Pressure in region 1
- **pressure_final** (*float*) – Pressure in region 2
- **mach_initial** (*float*) – Mach number in region 1
- **entropy** (*float*) – Change in entropy, if any
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The Mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M2 = gd.mach_from_pressure_ratio(pressure_initial=10, pressure_final=2, mach_
↪initial=1)
>>> M2
2.1220079294384067
>>>
```

gas_dynamics.standard.standard.**mach_from_temperature_ratio**(*temperature_initial: float*, *temperature_final: float*, *mach_initial: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number given a Mach number and two local temperatures

Notes

Given the local temperatures in two regions and the mach number in one, return the Mach number in the second region. Default fluid is air.

Parameters

- **temperature_initial** (*float*) – Temperature in region 1
- **temperature_final** (*float*) – Temperature in region 2
- **mach_final** (*float*) – Mach number in region 1
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_final = gd.mach_from_temperature_ratio(temperature_initial=300,␣
↪temperature_final=150, mach_final=1)
>>> mach_final
2.6457513110645907
>>>
```

gas_dynamics.standard.standard.**pressure_from_mach_ratio**(*mach_initial: float*, *mach_final: float*, *pressure_initial: float*, *entropy=0*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the pressure given a pressure in one region and the two Mach numbers

Notes

Given the Mach numbers in two regions and the pressure in one, return the missing pressure from the second region. Default arguments are for air and isentropic flow.

Parameters

- **mach_initial** (*float*) – Mach number in region 1
- **mach_final** (*float*) – Mach number in region 2
- **pressure_initial** (*float*) – Pressure in region 1
- **entropy** (*float*) – Change in entropy, if any

- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The local pressure

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> p_final = gd.pressure_from_mach_ratio(mach_initial=1, mach_final=2, pressure_
↪initial=10)
>>> p_final
2.4192491286747444
>>>
```

gas_dynamics.standard.standard.**temperature_from_mach_ratio**(*mach_initial: float*, *mach_final: float*, *temperature_initial: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the temperature given a temperature in one region and the two Mach numbers

Notes

Given the local Mach number in two regions and the temperature in one, return the missing temperature from the second region. Default fluid is air.

Parameters

- **mach_initial** (*float*) – Mach number in region 1
- **mach_final** (*float*) – Mach number in region 2
- **temperature_initial** (*float*) – Temperature in region 1
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The local temperature

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> T_final = gd.temperature_from_mach_ratio(mach_initial=1, mach_final=2,␣
↪temperature_final=297.15)
>>> T_final
198.10000000000002
>>>
```

gas_dynamics.standard.standard.**entropy_produced**(*stagnation_pressure_initial: float*, *stagnation_pressure_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the change in specific entropy from the stagnation pressure ratio

Notes

Given two stagnation pressures and the fluid, determine the entropy produced per unit mass.

Parameters

- **stagnation_pressure_initial** (*float*) – Stagnation pressure in region 1
- **stagnation_pressure_final** (*float*) – Stagnation pressure in region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The specific entropy

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> gd.entropy_produced(pt_initial=10, pt_final=9, gas='air')
30.238467993796142 #J / kg K
>>>
```

gas_dynamics.standard.standard.**mach_area_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the ratio of A / A* given the Mach number.

Notes

Given the Mach number and the ratio of specific heats, return the area ratio of the Mach number given to the area where Mach number is equal to 1. Default fluid is air.

Parameters

- **mach** (*float*) – Mach Number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of area over choked area

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> A_Astar =gd.mach_area_ratio_choked(mach=3)
>>> A_Astar
4.23456790123457
>>>
```

gas_dynamics.standard.standard.**mach_area_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*, *entropy=0*) → float

Return the area ratio given the two Mach numbers

Notes

Given two mach numbers, return the area ratio required to accelerate or deaccelerate the flow accordingly. Default fluid is air.

Parameters

- **mach_initial** (*float*) – Mach number in region 1
- **mach_final** (*float*) – Mach number in region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **ds** (*float*) – Entropy produced, if any

Returns
The area ratio

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> A2_A1 = gd.mach_area_ratio(mach_initial=1.5, mach_final=2.5)
>>> A2_A1
2.241789331255894   #area ratio
>>>
```

gas_dynamics.standard.standard.**mach_from_area_ratio**(*area_ratio: float*, *gas=<gas_dynamics.fluids.fluid object>*) → list

Return the possible mach numbers given a choked area ratio A / A*

Notes

Given a ratio of area over an area where Mach = 1, return the subsonic and supersonic Mach numbers for the change area.

Parameters

- **area_ratio** (*float*) – The ratio of area over choked area
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The subsonic and supersonic mach numbers for the area ratio

Return type
list

Examples

```
>>> import gas_dynamics as gd
>>> gd.mach_from_area_ratio(2)
[0.30590383418910816, 2.197198121652187]
>>>
```

gas_dynamics.standard.standard.**mass_flux_max**(*stagnation_pressure: float*, *stagnation_temperature: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the maximum flow rate per unit choked area

Notes

Given stagnation pressure, stagnation temperature, and the fluid, return the flow rate for a Mach number equal to 1. Default fluid is air.

Units:

J / kg-K and Pa return kg/m^2

kJ / kg-K and kPa returns kg/m^2

ft-lbf / lbm-R and psi returns lbm/in^2

Parameters

- **stagnation_pressure** (*float*) – The stagnation pressure.
- **stagnation_temperature** (*float*) – The stagnation temperature.
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **metric** (*bool*) – Use metric or US standard.

Returns
The maximum mass flux

Return type
float

Examples

```
>>> mdot = 5 #kg/s
>>> mdot_per_area = gd.choked_mdot(1000000, 300) #units are in Pascals
>>> mdot_per_area
2333.558560606226
>>> throat_area = mdot / mdot_per_area
>>> throat_area             #units are in meters squared
0.0021426503214477164
>>>
#alternatively, we can use the english system; psi, rankine and
get lbm/s/in^2
>>> from gas_dynamics.fluids import air_us
>>> air_us.units
'Btu / lbm-R'
>>> flux = gd.mass_flux_max( stagnation_pressure=500, stagnation_temperature=500,␣
↪gas=air_us)
>>> flux
2.097208828890205
>>>
```

gas_dynamics.standard.standard.**mass_flux**(*mach: float*, *stagnation_pressure: float*, *stagnation_temperature: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Determine mass flow rate for a mach number up to 1

Notes

Given stagnation pressure, stagnation temperature, and the fluid, return the flow rate per unit area for the given Mach number. Default fluid is air.

Units:

J / kg-K and Pa return kg/s/m^2

kJ / kg-K and kPa returns kg/s/m^2

ft-lbf / lbm-R and psi returns lbm/s/in^2

Btu / lbm-R and psi returns lbm/s/in^2

Parameters

- **mach** (*float*) – The mach number. Should not exceed 1
- **stagnation_pressure** (*float*) – The stagnation pressure
- **stagnation_temperature** (*float*) – The stagnation temperature
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mass flux

Return type
float

Examples

```
>>> #metric, input units are Pa, K, output is kg/s/m^2
>>> flux = gd.mass_flux(mach=.8, stagnation_pressure=1e6, stagnation_
↪temperature=500)
>>> flux
1741.3113452036841
>>> #us standard, input units are psi and Rankine, output units are lbm/s/in^2
>>> from gas_dynamics.fluids import air_us
>>> air_us.units
'Btu / lbm-R'
>>> flux = gd.mass_flux(mach=.8, stagnation_pressure=500, stagnation_
↪temperature=500, gas=air_us)
>>> flux
2.01998480961849
>>>
```

gas_dynamics.standard.standard.**plot_stagnation_ratios**(*range=[0.1, 5], step=0.01, gasses=[<gas_dynamics.fluids.fluid object>, <gas_dynamics.fluids.fluid object>, <gas_dynamics.fluids.fluid object>], dark=True*)

Plot the isentropic stagnation relationships for different gasses

Notes

Plots Mach number vs T/T, P/Pt, A/A*, rho/rho_t, for a list of specific heat ratios.

Parameters

- **range** (*list*) – The starting and ending Mach # in a list, ex: [.01,5]
- **step** (*float*) – The increment between min and max
- **gasses** (*list*) – A list of the user defined gas objects to be plotted

  ex: gasses = [air, methane, argon]
- **dark** (*bool*) – Use a dark mode plot. Default true.

Examples

```
>>> import gas_dynamics as gd
>>> gd.plot_stagnation_ratios()
>>>
```

CHAPTER

FIVE

SHOCKS

5.1 Equation Map - Shocks

shock_mach

$$M_2 = \left[\frac{M_1^2 + 2/(\gamma - 1)}{[2\gamma/(\gamma - 1)]\, M_1^2 - 1} \right]^{\frac{1}{2}}$$

shock_mach_before

$$M_2 = \left[\frac{M_2^2 + 2/(\gamma - 1)}{[2\gamma/(\gamma - 1)]\, M_2^2 - 1} \right]^{\frac{1}{2}}$$

shock_pressure_ratio

$$\frac{p_2}{p_1} = \frac{2\gamma}{\gamma + 1} M_1^2 - \frac{\gamma - 1}{\gamma + 1}$$

shock_mach_from_pressure_ratio

$$M = \left[\frac{\gamma + 1}{2\gamma} \left(\frac{p_2}{p_1} \right) + \frac{\gamma - 1}{\gamma + 1} \right]^{\frac{1}{2}}$$

shock_temperature_ratio

$$\frac{T_2}{T_1} = \frac{\left(1 + [(\gamma - 1)/2]\, M_1^2\right) \left([2\gamma/(\gamma - 1)]\, M_1^2 - 1\right)}{\left[(\gamma + 1)^2 / 2\,(\gamma - 1)\right] M_1^2}$$

shock_dv_a

$$\frac{dV}{a_1} = \left(\frac{2}{\gamma + 1} \right) \left(\frac{M_1^2 - 1}{M_1} \right)$$

shock_stagnation_pressure_ratio

$$\frac{p_{t2}}{p_{t1}} = \left(\frac{[(\gamma + 1)/2]\, M_1^2}{1 + [(\gamma - 1)/2]\, M_1^2} \right)^{\frac{\gamma}{\gamma - 1}} \left[\frac{2\gamma}{\gamma + 1} M_1^2 - \frac{\gamma - 1}{\gamma + 1} \right]^{\frac{1}{1 - \gamma}}$$

shock_flow_deflection

$$\delta = \arctan\left[2 \cot(\theta) \left(\frac{M_1^2 \sin^2(\theta) - 1}{M_1^2 (\gamma + \cos 2\theta) + 2} \right) \right]$$

shock_angle , *shock_mach_given_angles* , and *shock_flow_deflection_from_machs* all employ equation solvers with combinations of the above functions to return the desired values.

gas_dynamics.shocks.shocks.**shock_mach**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the Mach number after a standing normal shock

Notes

Given a starting Mach number M1 and the ratio of specific heats, return the Mach number M2 that immediately follows the shock. The default fluid is air.

Parameters

- **mach** (*float*) – The Mach number before the shock
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The Mach number after the shock

Return type
float

Examples

```
>>> M2 = gd.shock_mach(mach=1.5)
>>> M2
0.7010887416930995
>>>
```

gas_dynamics.shocks.shocks.**shock_mach_before**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the Mach number before a standing normal shock

Notes

Given the Mach number after the shock and the ratio of specific heats, return the Mach number that immediately precedes the shock. Default fluid is air.

Parameters

- **M** (*float*) – The Mach number after the shock
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mach number before the shock

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M2 = 0.7010887416930995
>>> M1 = gd.shock_mach_before(mach = M2)
>>> M1
1.4999999999999998
>>>
```

gas_dynamics.shocks.shocks.**shock_pressure_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the pressure ratio after a standing normal shock for a given Mach number

Notes

Given a starting Mach number and a ratio of specific heats, this function returns the ratio of pressure 2 over pressure 1 across a standing normal shock.

Parameters

- **mach** (*float*) – The starting Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The pressure ratio p2/p1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> p2_p1 = gd.shock_pressure_ratio(mach=1.5)
>>> p2_p1
2.4583333333333335
>>>
```

gas_dynamics.shocks.shocks.**shock_mach_from_pressure_ratio**(*pressure_ratio: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the mach number across a standing normal shock given a pressure ratio

Notes

Given the ratio of pressure behind the shock over pressure before the shock and the ratio of specific heats, this function returns the Mach number before the shock. Default fluid is air.

Parameters

- **pressure_ratio** (*float*) – The pressure ratio p2 / p1
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mach number prior the shock

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> p2_p1 = 6
>>> M1 = gd.shock_mach_from_pressure_ratio(pressure_ratio = p2_p1)
>>> M1
2.29906813420444
>>>
```

gas_dynamics.shocks.shocks.**shock_temperature_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the temperature ratio after a standing normal shock for a given Mach number

Notes

Given a starting Mach number and a ratio of specific heats, this function returns the ratio of temperature 2 over temperature 1 across a standing normal shock. Default fluid is air.

Parameters

- **mach** (*float*) – The starting Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The temperature ratio T2/T1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> T2_T1 = gd.shock_temperature_ratio(mach=1.5)
>>> T2_T1
1.320216049382716
>>>
```

gas_dynamics.shocks.shocks.**shock_dv_a**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns change in velocity over the local speed of sound after a normal shock.

Notes

Given a starting Mach number and a ratio of specific heats, this function returns the velocity change across a standing normal shock divided by the local speed of sound. Default fluid is air

Parameters

- **mach** (*float*) – The starting Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The change in Mach number

Return type
float

Examples

gas_dynamics.shocks.shocks.**shock_stagnation_pressure_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns stagnation pressure ratio after a normal shock.

Notes

Given a starting Mach number and a ratio of specific heats, this function returns the ratio of stagnation presure 2 over stagnation pressure 1 across a standing normal shock. Default fluid is air.

Parameters

- **mach** (*float*) – The starting Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The stagnation pressure ratio pt2/pt1

Return type
float

Examples

```
>>> pt2_pt1 = gd.shock_stagnation_ratio(mach=2)
>>> pt2_pt1
0.7208738614847454
>>>
```

gas_dynamics.shocks.shocks.**shock_tables**(*range=[1, 5], step=0.01, gas=<gas_dynamics.fluids.fluid object>*) → str

Returns shock tables for a range of Mach numbers.

Notes

Given a range of Mach numbers and a ratio of specific heats, generate the standing normal shock tables for every incremental Mach number in between.

Parameters

- **range** (*list*) – The starting and ending Mach # in a list, ie: [1,5].
- **step** (*float*) – The step size for the tables.
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The shock table

Return type
str

Examples

```
>>> import gas_dynamics as gd
>>> gd.shock_tables(range=[1,2], step=.1)
Normal Shock Parameters for  = 1.4
M: 1.00   |   M2: 1.0000   |   p2/p1: 1.0000   |   T2/T1: 1.0000   |   dV/a: 0.
 ↪0000   |   pt2/pt1: 1.000000
M: 1.10   |   M2: 0.9118   |   p2/p1: 1.2450   |   T2/T1: 1.0649   |   dV/a: 0.
 ↪1591   |   pt2/pt1: 0.998928
M: 1.20   |   M2: 0.8422   |   p2/p1: 1.5133   |   T2/T1: 1.1280   |   dV/a: 0.
 ↪3056   |   pt2/pt1: 0.992798
M: 1.30   |   M2: 0.7860   |   p2/p1: 1.8050   |   T2/T1: 1.1909   |   dV/a: 0.
 ↪4423   |   pt2/pt1: 0.979374
M: 1.40   |   M2: 0.7397   |   p2/p1: 2.1200   |   T2/T1: 1.2547   |   dV/a: 0.
 ↪5714   |   pt2/pt1: 0.958194
M: 1.50   |   M2: 0.7011   |   p2/p1: 2.4583   |   T2/T1: 1.3202   |   dV/a: 0.
 ↪6944   |   pt2/pt1: 0.929787
M: 1.60   |   M2: 0.6684   |   p2/p1: 2.8200   |   T2/T1: 1.3880   |   dV/a: 0.
 ↪8125   |   pt2/pt1: 0.895200
M: 1.70   |   M2: 0.6405   |   p2/p1: 3.2050   |   T2/T1: 1.4583   |   dV/a: 0.
 ↪9265   |   pt2/pt1: 0.855721
M: 1.80   |   M2: 0.6165   |   p2/p1: 3.6133   |   T2/T1: 1.5316   |   dV/a: 1.
 ↪0370   |   pt2/pt1: 0.812684
M: 1.90   |   M2: 0.5956   |   p2/p1: 4.0450   |   T2/T1: 1.6079   |   dV/a: 1.
 ↪1447   |   pt2/pt1: 0.767357
M: 2.00   |   M2: 0.5774   |   p2/p1: 4.5000   |   T2/T1: 1.6875   |   dV/a: 1.
 ↪2500   |   pt2/pt1: 0.720874
>>>
```

gas_dynamics.shocks.shocks.**shock_flow_deflection**(*mach: float*, *shock_angle: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns flow deflection angle from Mach number and oblique shock angle

Notes

Given the Mach number prior to the oblique shock, the angle of the oblique shock in degrees, and the ratio of specific heats, this function returns the angle that the flow is turned. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach number before the shock
- **shock_angle** (*float*) – The shock angle in degrees
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The deflection angle

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> deflect = gd.shock_flow_deflection(mach=2, shock_angle = 157.5)
>>> deflect
10.856560004139958
>>>
```

gas_dynamics.shocks.shocks.**shock_angle**(*mach: float*, *flow_deflection: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the shock angle given the Mach number prior to the shock and the deflection angle

Notes

Given the Mach number prior to the oblique shock, the angle of the flow deflection, and the ratio of specific heats, this functions returns the angle that is formed by the shock. Default ratio of specific heats is for air

Parameters

- **mach** (*float*) – The Mach number before the shock
- **flow_deflection** (*float*) – The flow deflection angle in degrees
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The shock angle

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> shocks = gd.shock_angle(mach=2, flow_deflection = 10)
>>> shocks
[23.014012220565785, 96.29991962425305]
>>>
```

gas_dynamics.shocks.shocks.**shock_mach_given_angles**(*shock_angle: float*, *flow_deflection: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number before a shock given shock angle and flow deflection

Notes

Given the angle of the shock and the angle that the flow has turned, return the mach number that preceded the shock.

Parameters

- **theta** (*float*) – The shock angle in degrees
- **delta** (*float*) – The flow deflection angle in degrees
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = gd.shock_mach_given_angles(theta=22.5, delta=10)
>>> M
3.9293486839798955
>>>
```

gas_dynamics.shocks.shocks.**shock_oblique_charts**(*mach_max=6*, *gas=<gas_dynamics.fluids.fluid object>*, *points=40000*, *dark=True*)

Generate 2-D Oblique Shock Charts

Notes

Displays two plots, 1) Mach number versus oblique shock wave angle and the corresponding deflection angles for each case. 2) Mach number versus resulting Mach number after an oblique shock and the corresponding deflection angles for each case. Default ratio of specific heats is for air.

Parameters

- **mach_max** (*float*) – The upper limit Mach number for the chart
- **gas** (*fluid*) – A user defined fluid object. Default is air
- **points** (*int*) – The number of points to evaluate on the mesh
- **dark** (*bool*) – Dark mode for the plots. Default is true.

Examples

```
>>> gd.shock_oblique_charts()
>>>
```

gas_dynamics.shocks.shocks.**shock_flow_deflection_from_machs**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the flow deflection angle given the mach number before and after the oblique shock

Notes

Given two mach numbers, iteratively solve for the shock angle and flow deflection to satisfy the system.

Parameters

- **mach_initial** (*float*) – The initial mach number
- **mach_final** (*float*) – The mach number after the event
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The flow deflection angle

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> flow_deflect = gd.shock_flow_deflection_from_machs(M1=3, M2=1.5)
>>> flow_deflect
28.589471710648365
>>>
```

CHAPTER

SIX

PRANDTL-MEYER

6.1 Equation Map - Prandtl-Meyer

`prandtl_meyer_turn`

$$\nu = \left(\frac{\gamma+1}{\gamma-1}\right)^{\frac{1}{2}} \tan^{-1}\left[\frac{\gamma-1}{\gamma+1}(M^2-1)\right]^{\frac{1}{2}} - \tan^{-1}(M^2-1)^{\frac{1}{2}}$$

`prandtl_meyer_mach` employs an equation solver to return the Mach number in the equation above.

`mach_wave_angle`

$$\mu = \tan^{-1}\left(\frac{1}{(M^2-1)^{1/2}}\right)$$

6.2 Worked Example

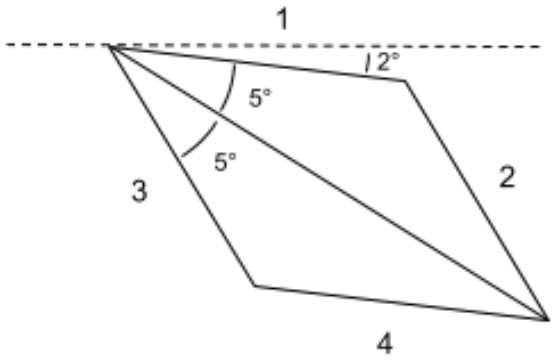
The mighty F-22 raptor is traveling at its supercruise speed of Mach 1.8 when it turns up at an angle of 7 degrees. The airfoil can be modeled as a symmetrical rhombus with 5 degree half angles as pictured below. Calculate the Mach numbers on surface 1, 2, 3, and 4.

Solution

From the figure we come to the conclusion that we will have Prandtl-Meyer expansion fans from the freestream to surface 1, and from surface 1 to surface 2. Using the functions *prandtl_meyer_angle_from_mach* and *prandtl_meyer_mach_from_angle*, we can get the Mach number information pretty quickly.

```
>>> import gas_dynamics as gd
>>> mach_initial = 1.8
>>> angle_init = gd.prandtl_meyer_angle_from_mach(mach_initial)
>>> angle_init
20.72506424776447
>>> angle_surface_1 = angle_init + 2
>>> mach_surface_1 = gd.prandtl_meyer_mach_from_angle(angle_surface_1)
>>> mach_surface_1
1.869672760055039
>>> angle_surface_2 = angle_surface_1 + 10
>>> mach_surface_2 = gd.prandtl_meyer_mach_from_angle(angle_surface_2)
>>> mach_surface_2
2.2385201020582
>>>
```

Fig. 1: Credits: F-22 Demonstration Team



We can also get the corresponding pressures on these surfaces using *pressure_from_mach_ratio*

```
>>> pressure_initial = 1
>>> pressure_surface_1 = gd.pressure_from_mach_ratio(pressure_initial=pressure_initial,␣
↪mach_initial=mach_initial, mach_final=mach_surface_1)
>>> pressure_surface_1
0.8985710141625418
>>> pressure_surface_2 = gd.pressure_from_mach_ratio(pressure_initial=pressure_surface_1,
↪ mach_initial=mach_surface_1, mach_final=mach_surface_2)
>>> pressure_surface_2
0.5059159131613694
>>>
```

For surfaces 3 and 4, we notice that we will first have a shock as the flow is turning into itself, followed by another Prandtl-Meyer expansion from surface 3 to 4. Before proceeding with shock case, lets import the sine function that works in degrees from the "extra" module.

```
>>> from gas_dynamics.extra import sind
>>> shock_angle = gd.shock_angle(mach=mach_initial, flow_deflection=12)
>>> shock_angle
[46.685991854146, 80.21459032863389]
>>> mach_initial_normal = mach_initial * sind(shock_angle[0])
>>> mach_initial_normal
1.3096891104605881
>>> mach_surface_3_normal = gd.shock_mach(mach_initial_normal)
>>> mach_surface_3_normal
0.7810840690884703
>>> mach_surface_3 = mach_surface_3_normal / sind(shock_angle[0] - 12)
>>> mach_surface_3
1.3725418664628979
```

We conclude with the final Prandtl-Meyer expansion from surface 3 to 4.

```
>>> angle_surface_3 = gd.prandtl_meyer_angle_from_mach(mach_surface_3)
>>> angle_surface_4 = angle_surface_3 + 10
>>> mach_4 = gd.prandtl_meyer_mach_from_angle(angle_surface_4)
>>> mach_4
1.7132972931071317
>>>
```

gas_dynamics.prandtl_meyer.prandtl_meyer.**prandtl_meyer_angle_from_mach**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the angle through which a flow has turned to reach a Mach number

Notes

Given a Mach number and ratio of specific heats, calculate the angle of a turn through which a flow has traversed to reach the Mach number given, from a Mach number of 1. Also known as the Prandtl-Meyer function. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The angle in degrees through which the flow has turned

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 2
>>> delta = gd.prandtl_meyer_turn(M=2)
26.379760813416475
>>>
```

gas_dynamics.prandtl_meyer.prandtl_meyer.**prandtl_meyer_mach_from_angle**(*angle: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Returns the Mach number given an angle through which the flow has turned from a starting Mach of one

Notes

Given a smooth turn through which a flow has turned and the ratio of specific heats, return the Mach number after the turn.

Parameters

- **angle** (*float*) – The turn angle in degrees
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> angle = 26.37
>>> M = gd.prandtl_meyer_mach(angle=angle)
>>> M
1.9996459342662083
>>>
```

gas_dynamics.prandtl_meyer.prandtl_meyer.**mach_wave_angle**(*mach: float*)

Return the angle of the Mach wave given the Mach number after a turn

Notes

Parameters
mach (*float*) – The mach number after a turn

Returns
The angle of the mach wave in degrees

Return type
float

Examples

CHAPTER

SEVEN

FANNO FLOW

7.1 Equation Map - Fanno

stagnation_enthalpy

$$h_t = h + \frac{G^2}{\rho^2 2}$$

fanno_temperature_ratio

$$\frac{T_2}{T_1} = \frac{1 + [(\gamma - 1)/2]\, M_1^2}{1 + [(\gamma - 1)/2]\, M_2^2}$$

fanno_pressure_ratio

$$\frac{p_2}{p_1} = \frac{M_1}{M_2} \left(\frac{1 + [(\gamma - 1)/2]\, M_1^2}{1 + [(\gamma - 1)/2]\, M_2^2} \right)^{1/2}$$

fanno_temperature_ratio

$$\frac{\rho_2}{\rho_1} = \frac{M_1}{M_2} \left(\frac{1 + [(\gamma - 1)/2]\, M_2^2}{1 + [(\gamma - 1)/2]\, M_1^2} \right)^{1/2}$$

fanno_stagnation_pressure_ratio

$$\frac{p_{t2}}{p_{t1}} = \frac{M_1}{M_2} \left(\frac{1 + [(\gamma - 1)/2]\, M_2^2}{1 + [(\gamma - 1)/2]\, M_1^2} \right)^{\frac{\gamma+1}{2(\gamma-1)}}$$

fanno_temperature_star_ratio

$$\frac{T}{T^*} = \frac{(\gamma + 1)/2}{1 + [(\gamma - 1)/2]\, M^2}$$

fanno_pressure_star_ratio

$$\frac{p}{p^*} = \frac{1}{M} \left(\frac{(\gamma + 1)/2}{1 + [(\gamma - 1)/2]\, M^2} \right)^{1/2}$$

fanno_density_star_ratio

$$\frac{\rho}{\rho^*} = \frac{1}{M} \left(\frac{1 + [(\gamma - 1)/2]\, M^2}{(\gamma + 1)/2} \right)^{1/2}$$

fanno_velocity_star_ratio

$$\frac{V}{V^*} = \frac{M}{1} \left(\frac{(\gamma + 1)/2}{1 + [(\gamma - 1)/2]\, M^2} \right)^{1/2}$$

fanno_parameter

$$\frac{f(x_2 - x_1)}{D_e} = \frac{\gamma+1}{2\gamma}\ln\frac{1+[(\gamma-1)/2]\,M_2^2}{1+[(\gamma-1)/2]\,M_1^2} - \frac{1}{\gamma}\left(\frac{1}{M_2^2} - \frac{1}{M_1^2}\right) - \frac{\gamma+1}{2\gamma}\ln\frac{M_2^2}{M_1^2}$$

fanno_parameter_max

$$\frac{f(x - x^*)}{D_e} = \frac{\gamma+1}{2\gamma}\ln\frac{1+[(\gamma-1)/2]\,M^2}{(\gamma+1)/2} - \frac{1}{\gamma}\left(\frac{1}{M^2} - 1\right) - \frac{\gamma+1}{2\gamma}\ln M^2$$

mach_from_fanno uses an equation solver to determine the mach number.

7.2 Worked Example

In development.

gas_dynamics.fanno.fanno.**stagnation_enthalpy**(*enthalpy: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the stagnation enthalpy

Notes

Given the fluid state and a given enthalpy, return its stagnation enthalpy

Parameters

- **enthalpy** (*float*) – The enthalpy of the fluid
- **gas** (*fluid*) – A user defined fluid object

Returns
Stagnation enthalpy

Return type
float

Examples

gas_dynamics.fanno.fanno.**fanno_temperature_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the temperature ratio for a fanno flow given two Mach numbers

Notes

Given the two Mach numbers of a constant area adiabatic duct under the influence of friction alone, return the temperature ratio of region two over region one. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The mach number at region 1
- **mach_final** (*float*) – The mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
the fanno Temperature ratio T2 / T1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 1.2, 1
>>> T2_T1 = gd.fanno_temperature_ratio(mach_final,mach_initial)
>>> T2_T1
0.9316770186335405
>>>
```

gas_dynamics.fanno.fanno.**fanno_pressure_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the pressure ratio for a fanno flow given two Mach numbers

Notes

Given the two Mach numbers of a constant area adiabatic duct under the influence of friction alone, return the pressure ratio of region two over region one. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The mach number at region 1
- **mach_final** (*float*) – The mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
the fanno pressure ratio p2 / p1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 1.2, 1
>>> p2_p1 = gd.fanno_pressure_ratio(mach_initial,mach_final)
>>> p2_p1
1.243221621433604
>>>
```

gas_dynamics.fanno.fanno.**fanno_density_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the density ratio for a fanno flow given two Mach numbers

Notes

Given the two Mach numbers of a constant area adiabatic duct under the influence of friction alone, return the density ratio of region two over region one. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The mach number at region 1
- **mach_final** (*float*) – The mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The fanno density ratio rho2 / rho1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 1.2, 1
>>> rho2_rho1 = gd.fanno_density_ratio(mach_final,mach_initial)
>>> rho2_rho1
0.8633483482177806
>>>
```

gas_dynamics.fanno.fanno.**fanno_stagnation_pressure_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the stagnation pressure ratio pt2/pt1 for a fanno flow given two mach numbers

Notes

Given the two Mach numbers of a constant area adiabatic duct under the influence of friction alone, return the stagnation pressure ratio of region two over region one. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The mach number at region 1
- **mach_final** (*float*) – The mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The fanno stagnation pressure ratio pt2 / pt1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 1.2, 1
>>> pt2_pt1 = gd.fanno_stagnation_pressure_ratio(mach_final,mach_initial)
>>> pt2_pt1
1.0304397530864196
>>>
```

gas_dynamics.fanno.fanno.**fanno_temperature_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of temperature over temperature where Mach equals one

Notes

Given a Mach number of a constant area adiabatic duct under the influence of friction alone, return the temperature ratio of region two over region one where Mach in region one equals one. Default fluid is air.

Parameters

- **mach** (*float*) – The mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The fanno temperature ratio T / T*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 1.2
>>> gd.fanno_temperature_star_ratio(M)
0.9316770186335405
>>>
```

gas_dynamics.fanno.fanno.**fanno_pressure_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of pressure over pressure where Mach equals one

Notes

Given a Mach number of a constant area adiabatic duct under the influence of friction alone, return the pressure ratio of region two over region one where Mach in region one equals one. Default fluid is air.

Parameters

- **mach** (*float*) – The mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The fanno pressure ratio p / p*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 1.2
>>> gd.fanno_pressure_star_ratio(M)
0.8043618151097336
>>>
```

gas_dynamics.fanno.fanno.**fanno_density_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of density over density where Mach equals one

Notes

Given a Mach number of a constant area adiabatic duct under the influence of friction alone, return the density ratio of region two over region one where Mach in region one equals one. Default fluid is air.

Parameters

- **mach** (*float*) – The mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The fanno density ratio rho / rho*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 1.2
>>> gd.fanno_density_star_ratio(M)
0.8633483482177806
>>>
```

gas_dynamics.fanno.fanno.**fanno_velocity_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of velocity over velocity where Mach equals one

Notes

Given a Mach number of a constant area adiabatic duct under the influence of friction alone, return the velocity ratio of region two over region one where Mach in region two equals one. Default fluid is air.

Parameters

- **mach** (*float*) – The mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The fanno velocity ratio V / V*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 1.2
>>> v_vstar = gd.fanno_velocity_star_ratio(M)
>>> v_vstar
1.1582810137580164
>>>
```

gas_dynamics.fanno.fanno.**fanno_parameter**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the product of friction factor and length divided by diameter for two Mach numbers

Notes

Given the two Mach numbers of a constant area adiabatic duct under the influence of friction alone, return the fanno parameter that describes that system where fanno parameter is the product of friction factor and length over diameter. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The mach number at region 1
- **mach_final** (*float*) – The mach number at region 2
- **gas** (*fluid*) – The user defined fluid object

Returns
The fanno parameter f(x2-x1)/D

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 3, 2
>>> fanno = gd.fanno_parameter(mach_initial, mach_final)
>>> fanno
0.21716290559704166
>>>
```

gas_dynamics.fanno.fanno.**fanno_parameter_max**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the maximum product of friction factor and length divided by diameter

Notes

Given a Mach number of a constant area adiabatic duct under the influence of friction alone, determine the maximum length to diameter ratio for a fluid to reach a Mach number of 1. Default fluid is air.

Parameters

- **M** (*float*) – The starting Mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The maximum fanno parameter f(x*-x)/D

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 2
>>> fanno_max = gd.fanno_parameter_max(M)
>>> fanno_max
0.3049965025814798
>>>
```

gas_dynamics.fanno.fanno.**mach_from_fanno**(*fanno: float*, *mach_initial: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number that would result from the fanno parameter and initial mach number

Notes

Given the Mach number and fanno parameter that describes that system, return the resulting mach number. Default fluid is air.

Parameters

- **fanno** (*float*) – The fanno parameter for the system
- **mach_initial** (*float*) – The starting Mach number
- **gas** (*fluid*) – The user defined fluid object

Returns
The resulting Mach number

Return type
Float

Examples

```
>>> import gas_dynamics as gd
>>> fanno, mach_initial = .3, 2.64
>>> mach_final = gd.mach_from_fanno(fanno=fanno, mach_initial=mach_initial)
>>> mach_final
1.567008305615555
>>>
```

CHAPTER

EIGHT

RAYLEIGH FLOW

8.1 Equation Map - Rayleigh

rayleigh_pressure_ratio

$$\frac{p_2}{p_1} = \frac{1+\gamma M_1^2}{1+\gamma M_2^2}$$

rayleigh_temperature_ratio

$$\frac{T_2}{T_1} = \left(\frac{1+\gamma M_1^2}{1+\gamma M_2^2}\right)^2 \frac{M_2^2}{M_1^2}$$

rayleigh_density_ratio

$$\frac{\rho_2}{\rho_1} = \frac{M_1^2}{M_2^2}\left(\frac{1+\gamma M_2^2}{1+\gamma M_1^2}\right)$$

rayleigh_stagnation_temperature_ratio

$$\frac{T_{t2}}{T_{t1}} = \left(\frac{1+\gamma M_1^2}{1+\gamma M_2^2}\right)^2 \frac{M_2^2}{M_1^2}\left(\frac{1+[(\gamma-1)/2]\,M_2^2}{1+[(\gamma-1)/2]\,M_1^2}\right)$$

rayleigh_stagnation_pressure_ratio

$$\frac{p_{t2}}{p_{t1}} = \frac{1+\gamma M_1^2}{1+\gamma M_2^2}\left(\frac{1+[(\gamma-1)/2]\,M_2^2}{1+[(\gamma-1)/2]\,M_1^2}\right)^{\gamma/(\gamma-1)}$$

rayleigh_mach_from_pressure_ratio

$$M_2 = \left[\left(\frac{p_2}{p_1}\left(1+\gamma M_1^2\right)-1\right)\frac{1}{\gamma}\right]^{1/2}$$

rayleigh_mach_from_temperature_ratio , *rayleigh_mach_from_stagnation_temperature_ratio* , *rayleigh_mach_from_stagnation_pressure_ratio* all use equation solvers to get the desired result.

rayleigh_pressure_star_ratio

$$\frac{p}{p^*} = \frac{1+\gamma}{1+\gamma M^2}$$

rayleigh_temperature_star_ratio

$$\frac{T}{T^*} = \frac{M^2(1+\gamma)^2}{(1+\gamma M^2)^2}$$

rayleigh_density_star_ratio

$$\frac{\rho}{\rho^*} = \frac{1+\gamma M^2}{(1+\gamma)M^2}$$

rayleigh_stagnation_temperature_star_ratio

$$\frac{T_t}{T_t^*} = \frac{2(1+\gamma)^2 M^2}{(1+\gamma M^2)^2}\left(1+\frac{\gamma-1}{2}M^2\right)$$

rayleigh_stagnation_pressure_star_ratio

$$\frac{p_t}{p_t^*} = \frac{1+\gamma}{1+\gamma M^2}\left(\frac{1+[(\gamma-1)/2]\,M^2}{(\gamma+1)/2}\right)^{\frac{\gamma}{\gamma-1}}$$

rayleigh_heat_flux

$$q = c_p(T_{t2} - T_{t1})$$

8.2 Worked Example

In development

gas_dynamics.rayleigh.rayleigh.**rayleigh_pressure_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the pressure ratio pressure_final / pressure_initial given the two Mach numbers

Notes

Given two Mach numbers, determine the resulting pressure ratio pressure two over pressure one in the non-adiabatic constant area frictionless flow. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The Mach number at region 1
- **mach_final** (*float*) – The Mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The rayleigh pressure ratio pressure_final / pressure_initial

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = 3, 2
>>> p2_p1 = gd.rayleigh_pressure_ratio(mach_initial, mach_final)
>>> p2_p1
2.0606060606060606
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_temperature_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the temperature ratio T2/T1 given the two Mach numbers

Notes

Given two Mach numbers, determine the resulting Temperature ratio temperature two over temperature one in the non-adiabatic constant area frictionless flow. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The Mach number at region 1
- **mach_final** (*float*) – The Mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The rayleigh temperature ratio T2 / T1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = .8, .3
>>> T2_T1 = gd.rayleigh_temperature_ratio(mach_initial, mach_final)
>>> T2_T1
0.39871485855083627
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_density_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the density ratio rho2/rho1 given the two Mach numbers

Notes

Given two Mach numbers, determine the resulting density ratio density two over density one in the non-adiabatic constant area frictionless flow. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The Mach number at region 1
- **mach_final** (*float*) – The Mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The rayleigh density ratio rho2 / rho1

Return type
float

Examples

gas_dynamics.rayleigh.rayleigh.**rayleigh_stagnation_temperature_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the stagnation temperature ratio Tt2/Tt1 given the two Mach numbers

Notes

Given two Mach numbers, determine the resulting stagnation temperature ratio stagnation temperature two over stagnation temperature one in the non-adiabatic constant area frictionless flow. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The Mach number at region 1
- **mach_final** (*float*) – The Mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The rayleigh stagnation temperature ratio pt2 / pt1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = .8, .3
>>> Tt2_Tt1 = gd.rayleigh_stagnation_temperature_ratio(mach_initial, mach_final)
>>> Tt2_Tt1
0.35983309042974404
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_stagnation_pressure_ratio**(*mach_initial: float*, *mach_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the stagnation pressure ratio pt2/pt1 given the two Mach numbers

Notes

Given two Mach numbers, determine the resulting stagnation pressure ratio stagnation pressure two over stagnation pressure one in the non-adiabatic constant area frictionless flow. Default fluid is air.

Parameters

- **mach_initial** (*flaot*) – The Mach number at region 1
- **mach_final** (*float*) – The Mach number at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The rayleigh stagnation pressure ratio pt2 / pt1

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, mach_final = .8, .3
>>> pt2_pt1 = gd.rayleigh_stagnation_pressure_ratio(mach_initial, mach_final)
>>> pt2_pt1
1.1758050380938454
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_mach_from_pressure_ratio**(*mach_initial: float*, *pressure_initial: float*, *pressure_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the mach number given the Mach number and two pressures

Notes

Given the initial Mach number, initial pressure, and final pressure, determine the resulting Mach number in the non-adiabatic constant area frictionless flow with heat transfer. Default fluid is air.

Parameters

- **M** (*float*) – The Mach number
- **pressure_initial** (*flaot*) – pressure 1
- **pressure_final** (*float*) – pressure 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The resulting Mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_final = gd.rayleigh_mach_from_pressure_ratio(mach_initial=.8, pressure_
↪initial=1.5, pressure_final=2.5)
>>> mach_final
0.31350552512789054
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_mach_from_temperature_ratio**(*mach_initial: float*, *temperature_initial: float*, *temperature_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number given the Mach number and two temperatures

Notes

Given the initial Mach number, initial temperature, and final temperature, determine the resulting Mach number in the non-adiabatic constant area frictionless flow with heat transfer. Default fluid is air.

Parameters

- **mach** (*float*) – The Mach number
- **temperature_initial** (*float*) – Temperature 1
- **temperature_final** (*float*) – Temperature 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The resulting Mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, T1, T2 = .8, 250, 100
>>> mach_final = gd.rayleigh_mach_from_temperature_ratio(mach_initial, T1, T2)
>>> mach_final
0.3006228581671002
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_mach_from_stagnation_temperature_ratio**(*mach_initial: float*, *stagnation_temperature_initial: float*, *stagnation_temperature_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number given the Mach number and two stagnation temperatures

Notes

Given the initial Mach number, initial stagnation temperature, and final stagnation temperature, determine the resulting Mach number in the non-adiabatic constant area frictionless flow with heat transfer. Default fluid is air.

Parameters

- **mach_initial** (*float*) – The Mach number
- **stagnation_temperature_initial** (*float*) – Stagnation temperature 1
- **stagnation_temperature_final** (*float*) – Stagnation temperature 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The resulting Mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, Tt1, Tt2 = .8, 250, 105
>>> mach_final = gd.rayleigh_mach_from_stagnation_temperature_ratio(mach_initial,␣
↪Tt1, Tt2)
>>> mach_final
0.33147520792270446
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_mach_from_stagnation_pressure_ratio**(*mach_initial: float*, *stagnation_pressure_initial: float*, *stagnation_pressure_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the Mach number given the Mach number and two stagnation pressures

Notes

Given the initial Mach number, initial stagnation pressure, and final stagnation pressure, determine the resulting Mach number in the non-adiabatic constant area frictionless flow with heat transfer. Default fluid is air.

Parameters

- **M** (*float*) – The Mach number
- **stagnation_pressure_initial** (*float*) – Stagnation pressure 1
- **stagnation_pressure_final** (*float*) – Stagnation pressure 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The resulting Mach number

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> mach_initial, pt1, pt2 = .8, 2.3, 2.6
>>> mach_final = gd.rayleigh_mach_from_stagnation_pressure_ratio(mach_initial, pt1,␣
↪pt2)
>>> mach_final
0.40992385119887326
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_pressure_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of pressure over pressure where Mach is equal to one

Notes

Given a mach number, return the ratio of the pressure of the mach number over the pressure over where the mach number is equal to one for a non-adiabatic fricionless constant area system. Default fluid is air.

Parameters

- **M** (*flaot*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of p / p*

Return type
float

Examples

```
>>> gas_dynamics as gd
>>> M = 2
>>> p_pstar = gd.rayleigh_pressure_star_ratio(M)
>>> p_pstar
0.36363636363636365
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_temperature_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of temperature over temperature where Mach is equal to one

Notes

Given a mach number, return the ratio of the temperature of the mach number over the temperature over where mach number is equal to one for a non-adiabatic frictionless constant area system. Default fluid is air.

Parameters

- **mach** (*flaot*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of T / T*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 2
>>> T_Tstar = gd.rayleigh_temperature_star_ratio(M)
>>> T_Tstar
0.5289256198347108
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_density_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of density over density where Mach is equal to one

Notes

Given a mach number, return the ratio of the density of the mach number over the density over where mach number is equal to one for a non-adiabatic constant area frictionless system. Default fluid is air.

Parameters

- **mach** (*flaot*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of rho / rho*

Return type
float

Examples

gas_dynamics.rayleigh.rayleigh.**rayleigh_stagnation_pressure_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of stagnation pressure over stagnation pressure where Mach is equal to one

Notes

Given a mach number, return the ratio of the stagnation pressure of the mach number over the stagnation pressure where mach number is equal to one for a non-adiabatic frictionless system. Default fluid is air.

Parameters

- **mach** (*flaot*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of pt / pt*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 2
>>> pt_ptstar = gd.rayleigh_stagnation_pressure_star_ratio(M)
>>> pt_ptstar
1.5030959785260414
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_stagnation_temperature_star_ratio**(*mach: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the ratio of stagnation temperature over stagnation temperature where Mach is equal to one

Notes

Given a mach number, return the ratio of the stagnation temperature of the mach number over the stagnation temperature where mach number is equal to one for a non-adiabatic frictionless constant area system. Default fluid is air.

Parameters

- **mach** (*flaot*) – The Mach number
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The ratio of Tt / Tt*

Return type
float

Examples

```
>>> import gas_dynamics as gd
>>> M = 2
>>> Tt_Ttstar = gd.rayleigh_stagnation_temperature_star_ratio(M)
>>> Tt_Ttstar
0.793388429752066
>>>
```

gas_dynamics.rayleigh.rayleigh.**rayleigh_heat_flux**(*stagnation_temperature_initial: float*, *stagnation_temperature_final: float*, *gas=<gas_dynamics.fluids.fluid object>*) → float

Return the heat per unit mass in our out given the two stagnation temperatures and the fluid

Notes

Given the initial and final stagnation temperatures, determine the specific heat flux to satisfy the constant area frictionless flow system. Default fluid is air.

Parameters

- **Tt1** (*float*) – The stagnation temperature at region 1
- **Tt2** (*float*) – The stagnation temperature at region 2
- **gas** (*fluid*) – A user defined fluid object. Default is air

Returns
The change in specific heat.

Return type
float

Examples

```
>>> from gas_dynamics.fluids import air
>>> air.cp = 1000
>>> Tt1, Tt2 = 280, 107.9
>>> gd.rayleigh_heat_flux(Tt1=Tt1, Tt2=Tt2, ,air)
-172100.0
>>>
```

CHAPTER

NINE

NOTES ABOUT THE FLUID CLASS

The fluid class is used to hold all the information about a fluid. Importing air from the fluids module, we can see some of the built in properties.

```
>>> from gas_dynamics.fluids import air, air_us
>>> for attr in air.__dict__:
...     print(attr, ':', air.__dict__[attr])
...
name : Air
gamma : 1.4
cp : 1000
cv : 716
R : 286.9
gc : 1
units : J / kg-K
rho : None
temperature : None
velocity : None
a : None
mach : None
mass_veolcity : None
>>>
>>> for attr in air_us.__dict__:
...     print(attr, ':', air_us.__dict__[attr])
...
name : Air
gamma : 1.4
cp : 0.24
cv : 0.171
R : 53.3
gc : 32.174
units : Btu / lbm-R
rho : None
temperature : None
velocity : None
a : None
mach : None
mass_veolcity : None
>>>
```

When creating your own fluid, the properties that must be set when initiated are the fluid name, ratio of specific heats, gas constant, and a string of the units being used. Currently the string is meant to serve as a reminder to the user as to

what units are being used, and no unit checking is done by the module in any way.

```
>>> foobar = fluid(name='foobar', gamma=1.4, R=53.3, units='Btu / lbm-R')
```

If we run out of the gates and immediately try and calculate the speed of sound of this fluid (which is remarkably similar to air) at standard temperature 491.67 Rankine, we notice the answer is considerably off from what we expect, ~~1100 ft/s.

```
>>> a = gd.sonic_velocity(gas=foobar, T=491.67)
>>> a
191.54220266040588
>>>
```

The reason for this is we have yet to set the proportionality factor for our fluid which uses the US standard system. Currently it is set to 1 as a default, as for the metric system the conversion is unity.

```
>>> foobar.gc = 32.174
>>> a = gd.sonic_velocity(gas=foobar, T=491.67)
>>> a
1086.4681666204492
>>>
```

Currently supported fluids in metric and standard are

- Air
- Argon
- Carbon Dioxide
- Carbon Monoxide
- Helium
- Hydrogen
- Methane
- Nitrogen
- Oxygen
- Water Vapor

```
>>> from gas_dynamics import nitrogen, nitrogen_us
```

A class to represent the fluid and its properties

gas_dynamics.fluid.**gamma**

The ratio of specific heats

Type
float

gas_dynamics.fluid.**R**

The gas constant for the fluid

Type
float

gas_dynamics.fluid.**units**

The unit system defining the gas constant

Type

str

No methods at this time

Examples

```
>>> import gas_dynamics as gd
>>> methane = gd.fluid('methane', 1.3, 518.2)
>>> methane.gamma
1.3
>>> methane.R
518.2
>>> methane.units
'metric'
Conversely we can set the units
>>> methane = fluid('methane', 1.3, 0.1238, units = 'btu / lbm-R')
>>> methane.gamma
1.3
>>> methane.R
0.1238
>>> methane.units
'btu / lbm-R'
```

CHAPTER

TEN

EXTRA

gas_dynamics.extra.**degrees**(*theta: float*) → float

Convert from radians to degrees

Parameters

theta (*float*) – The angle in radians

Examples

```
>>> import gas_dynamics as gd
>>> rad = gd.degrees(theta=3.14159)
>>> rad
179.99984796050427
>>>
```

gas_dynamics.extra.**radians**(*theta: float*) → float

Convert from degrees to radians

Parameters

theta (*float*) – The angle in degrees

Examples

```
>>> import gas_dynamics as gd
>>> deg = gd.radians(theta=180)
>>> deg
3.141592653589793
>>>
```

gas_dynamics.extra.**sind**(*theta: float*) → float

Sine given degrees

Parameters

theta (*float*) – The angle in degrees

Examples

```
>>> import gas_dynamics as gd
>>> sine = gd.sind(theta=45)
>>> sine
0.7071067811865476
>>>
```

gas_dynamics.extra.**arcsind**(*sin: float*) → float

Return the arcsine in degrees

Parameters
sin (*float*) – The sine of theta

Examples

```
>>> import gas_dynamics as gd
>>> theta = gd.arcsind(sin = .7071)
>>> theta
44.99945053347443
>>>
```

gas_dynamics.extra.**cosd**(*theta: float*) → float

Cosine given degrees

Parameters
theta (*float*) – The angle in degrees

Examples

```
>>> import gas_dynamics as gd
>>> cosine = gd.cosd(theta=45)
>>> cosine
0.7071067811865476
>>>
```

gas_dynamics.extra.**arccosd**(*cos: float*) → float

Return the arccosine in degrees

Parameters
cos (*float*) – The cosine of theta

Examples

```
>>> import gas_dynamics as gd
>>> theta = gd.arccosd(.7071)
>>> theta
45.00054946652557
>>>
```

gas_dynamics.extra.**tand**(*theta: float*) → float

Tangent given degrees

Parameters
theta (*float*) – The angle in degrees

Examples

```
>>> tan = gd.tand(theta=45)
>>> tan
0.9999999999999999
>>>
```

gas_dynamics.extra.**arctand**(*tan: float*) → float

Return the arctangent in degrees

Parameters
tan (*float*) – The tangent of theta

Examples

```
>>> theta = gd.arctand(1)
>>> theta
45.0
>>>
```

gas_dynamics.extra.**area_dia**(*dia=[]*, *area=[]*)

Notes

Given area or diameter, return the unknown

Parameters

- **dia** (*Diameter*) –
- **area** (*Area*) –

Examples

```
>>> import gas_dynamics as gd
>>> area = gd.area_dia(dia=1)
>>> area
0.7853981633974483
>>> dia = gd.area_dia(area=area)
>>> dia
1.0
>>>
```

gas_dynamics.extra.**lin_interpolate**(*x*, *x0*, *x1*, *y0*, *y1*)

Linear interpolation formula

Notes

Given two x values and their corresponding y values, interpolate for an unknown y value at x

Parameters

- **x** (*float*) – The x value at which the y value is unknown
- **x0** (*float*) – The x value of the first known pair
- **x1** (*float*) – The x value of the second known pair
- **y0** (*float*) – The y value of the first known pair
- **y1** (*float*) – The y value of the second known pair

Examples

```
>>> import gas_dynamics as gd
>>> y = gd.lin_interpolate(x=5, x0=4, x1=6, y0=25, y1=33)
>>> y
29.0
>>>
```

CHAPTER

ELEVEN

ABOUT ME

I'm Fernando de la Fuente, a mechanical engineer deeply invested in the aerospace sector and an enthusiast of writing code and tools to help me solve problems.

If you would like to make feature suggestions or comments for improvement, please reach out at FernandoAdelafuente@gmail.com

Fig. 1: Credits: NASA Images

CHAPTER

TWELVE

LICENSE

MIT License

Copyright (c) 2020 Fernando Alberto de la Fuente

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

THIRTEEN

CREDITS

I would like to thank the developers of Numpy, Scipy, and Matplotlib for their invaluable contributions to scientific computing in Python.

Most functions were made from equations in Robert D. Zucker and Oscar Biblarz's text "Gas Dynamics". Equations relating to Hypersonic flow are from John Anderson's text "Hypersonic and High-Temperature Gas Dynamics".

CHAPTER

FOURTEEN

INDEX

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

P

R

S

T

U