



THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA

An Introduction to PITOT3

CfH Talk 2021

Christopher M. James

Centre for Hypersonics, The School of Mechanical and Mining Engineering,
The University of Queensland, Brisbane, QLD, 4072, Australia

8th July 2021



Acknowledgement of Country

- ▶ Before I begin, I would like to acknowledge that we are meeting on custodial land of the oldest living civilisation in the world. This is a contested space, so I pay my respects to both the Jagera and the Turrbul people and their Elders, past, present and emerging, for they hold the hopes, dreams, and traditions of Aboriginal Australia.



Figure: Balun by Shaun Daniel Allen.

- ▶ Today I am going to talk about PITOT3, which is a new basic impulse facility simulation code which I have been writing this year.
- ▶ It is essentially a rebuild from the ground up of PITOT, a similar code which I started as a Summer Research Scholar in the CfH in 2011/2012 (almost a decade ago now!), using the Python to D library in Peter and Rowan's eilmer4 gas library, Python3, and hopefully the improved coding skills which I have gained over the last decade.
- ▶ The main reasons for doing this were the official retirement of Python2 at the end of last year (it is even hard to install Python2 on the latest version of Ubuntu now, as I have come to realise) and the movement of the old Python gas dynamics library which existed with Eilmer3 to the new Eilmer4 Python/D libraries.
- ▶ I had also wanted a chance to have another crack at writing the code itself, as I wasn't that happy with how PITOT was set out.

- As we're going to be talking about facilities and how to simulate them today, I thought it might be good to start here:

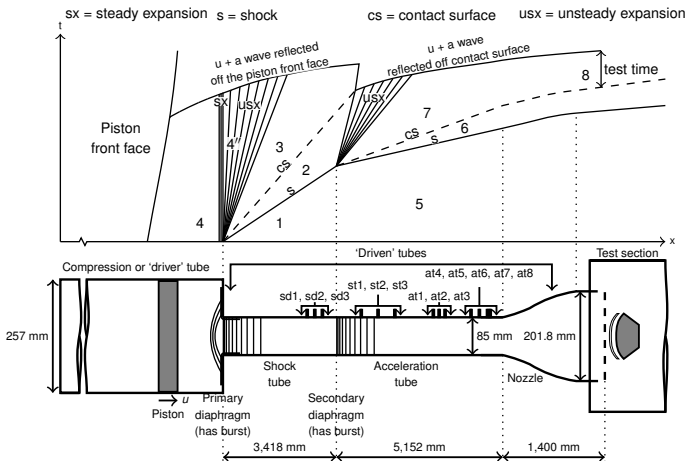


Figure: Schematic and position-time ('x-t') diagram of the X2 expansion tube.

- ▶ Obviously the needs change based on the level of fidelity of the given simulation tool, but from a facility simulation code we want the ability to predict the test flow which would be generated for a given experiment (i.e. a given driver condition, test gas, fill pressures etc.).
- ▶ We would also probably like the ability to predict other variables which are used to verify how well the experiment has been simulated by the code (in our case that is probably shock speeds in each tube, wall pressures, pitot pressure, and maybe things like temperature and species concentrations if our experiments are quite fancy).
- ▶ We also probably want to be able to tailor our simulation code to the given experimental data too to get a 'semi-empirical' or 'inferred' test flow.
- ▶ We might even want to use the code to get things like uncertainties too.

- ▶ There is a tradeoff to be had between complexity and solution time.
- ▶ And there is a place for everything from analytical codes to reacting, axisymmetric CFD codes at different parts of the test condition design and quantification process.

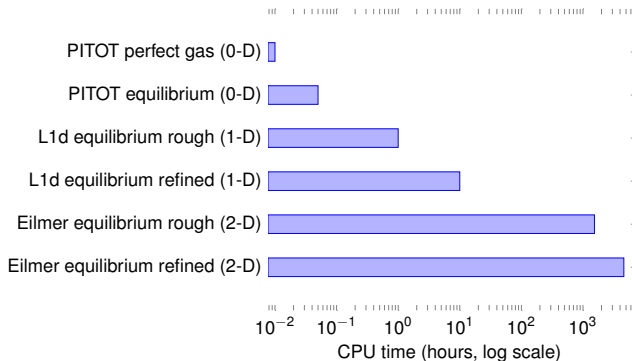
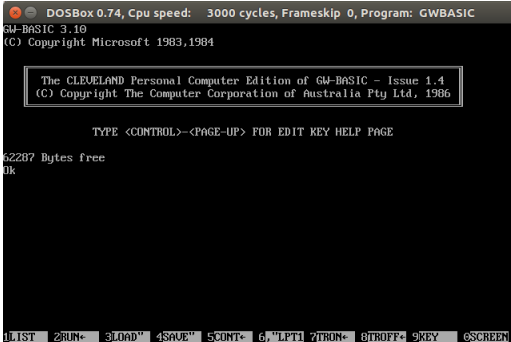


Figure: Rough simulation times for high enthalpy expansion tube conditions using 0-D, 1-D, and 2-D methods.

- ▶ PITOT, as a code, actually started long before I was involved.
- ▶ A basic expansion tube facility simulation code called PITOT was actually written by Richard in the 1990s (I am guessing?) in GWBASIC.
- ▶ I actually used this code in my undergraduate thesis in 2011 to design X2 test conditions to study giant planet entry.



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: GWBASIC
GW-BASIC 3.10
(C) Copyright Microsoft 1983,1984

The CLEVELAND Personal Computer Edition of GW-BASIC - Issue 1.4
(C) Copyright The Computer Corporation of Australia Pty Ltd, 1986

TYPE <CONTROL>-<PAGE-UP> FOR EDIT KEY HELP PAGE

62287 Bytes free
Ok

1|LIST 2|RUN+ 3|LOAD" 4|SAVE" 5|CONT+ 6, "LPT1 7|TRON+ 8|TROFF+ 9|KEY @|SCREEN
```

Figure: Richard's version of GW Basic.

- ▶ PITOT was fairly rudimentary, and it only had perfect gases, but it allowed the driver gas, secondary driver gas (if used), test gas, and accelerator gas to be selected by setting the γ and R of the gases.
- ▶ The driver condition was set empirically using its rupture pressure and temperature (p_4 and T_4).
- ▶ The code then exploited a perfect gas way to analytically solve for the fill pressures in each tube throughout the facility, if the shock speeds were guessed, to allow the fill pressures in each tube to be calculated.
- ▶ This meant a bit of manual iteration to find test conditions, but as the code ran in a second, that wasn't too bad.
- ▶ Being a perfect gas code, it did often overestimate the shock speeds a bit for given fill pressures.

- ▶ The output provided a table of all of the relevant facility states:

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: GWBASIC
Ok
RUN
Us1= 3765 m/s,Ms1= 10.84 ,Us2= 8300 m/s,Ms2= 23.9 ,Us3= 10000 m/s,Ms3= 28.80278
Driver g/R= 1.667 742.9 ,Int 1.4 287 ,test gas 1.4 287 ,accel 1.4 287
Region P T a u M rho pitot stgn
Pa K m/s m/s kg/s kPa MPa
1 14251.5 300 347 0 0 .1655 0 0
2 1952902 7142 1694 3110 1.83 .9526 9447 11.8
3 1952902 931 1074 3110 2.89 2.8215 24950 54.8
4 2.79E+07 2700 1828 0 0 0 0 0
5 45.8 300 347 0 0 .0005 0 0
6 30544.7 33621 3675 6904 1.87 .0031 153 .1
7 30544.7 2177 935 6904 7.38 .0488 2157 177.4
8 17405.6 28630 3391 8323 2.45 .0021 143 .2
9 17405.6 48676 4422 8323 1.88 .0012 87 .1
10 17.9 300 347 0 0 .0002 0 0
11 1.358987E+07 2024 1583 1583 1 9.034701 27900 27.9
12 1.495421E+07 15912 2528 0 0 3.2744 0 0
13 0 0 0 0 0 0 0 0
14 1952902 7142 1694 3110 1.83 .9526 9447 11.8
Pressure behind standing shock in region 2= 7356 kPa
Reflected shock speed= 1276.343 m/s, ref= 0
Ok
1LIST 2RUN< 3LOAD" 4SAVE" 5CONT< 6,"LPT1 7TRON< 8TROFF+ 9KEY 0SCREEN
  
```

Figure: Richard's original Pitot output.

- ▶ X2 is normally ran with a hypersonic nozzle, so during my undergraduate thesis, I would run PITOT to get conditions, then I had a little Python script where I inputted the unsteadily expanded test gas condition and then outputted the nozzle exit / freestream parameters.
- ▶ PITOT didn't do any chemistry either, so I would then manually do a normal shock using NASA's CEA program to get the equilibrium post-shock conditions over the test model.
- ▶ It is actually quite painful in retrospect!

- ▶ When I started a Summer Research Internship in our lab the following summer (2011/2012) Fabs suggested that I convert the program to Python.
- ▶ What I started with was a line-by-line conversion of PITOT into Python which I then made a bit more user friendly by adding the ability to set the test gas and driver gas by name instead of by γ and R, and the ability to do the nozzle expansion and a final CEA calculation for the post-normal shock state over the model.

- I called this 'pitot classic' (which you can still find in the old cfcfd repository as pitot_classic.py).

```

chris@chris-OptiPlex-990: ~/Dropbox/project/coding/pitot_classic
File Edit View Search Terminal Help
chris@chris-OptiPlex-990:~/Dropbox/project/coding/pitot_classic$ python pitot_classic.py
Driver condition
Plston configuration? ['x2_lwp','x2_lwp']
Percentage of Helium in the primary driver? ['90','100','80'] 80
Is the secondary driver being used (y/n)? y
Test gas? ['N2','60H240Ne','N2','85H215Ne','titan','85H215He','air','90H210Ne','mars','15H285Ne','90H210He','venus',
,'He'] air
filename? air_test
Enter shock speeds (in m/s) below to find solution.
Vsd? 3765
Vs1? 8300
Vs2? 10000

Pitot Classic Version: 29-Nov-2012
sd1 is secondary driver fill.
state 1 is shock tube fill. state 5 is acceleration tube fill.
state 7 is expanded test gas entering the nozzle.
state 8 is test gas exiting the nozzle (using area ratio of 2.5).
state 10f is frozen shocked test gas flowing over the model.
state 10e is equilibrium shocked test gas flowing over the model.
Test gas is air (gamma = 1.4, R = 286.0, ['Air': 1.0]).
Driver gas has 80%He (gamma = 1.667, R = 742.9)
Vsd = 3765.00 m/s, Msd = 3.69
Vs1 = 8300.00 m/s, Ms1 = 23.95 ,Vs2 = 10000.00 m/s, Ms2 = 28.85
state P      T      a      V      M      rho      pitot      stgn
      Pa      K      m/s    m/s    M      n^3/kg  kPa      MPa
s4  2.79e+07  2700.0  1829  0.0  0.00  0.0000  0  0.0
s3s 1.358987e+072024.7  1584  1583.5  1.00  9.0347  27900  27.9
sd1 237171.2  300.0  1019  0.0  0.00  0.3806  0  0.0
sd2 3986827  1538.3  2308  2616.5  1.13  1.2478  9702  9.7
sd3 3986827  1239.6  1239  2616.5  2.11  4.3294  27981  38.9
s1  47.50887  300.0  347  0.0  0.00  0.0006  0  0.0
s2  31779.48  33738.0  3675  6904.6  1.88  0.0633  160  0.2
s3  31779.48  222.5  878  6904.6  7.87  0.0688  2903  69.0
s5  18.64842  300.0  347  0.0  0.00  0.0002  0  0.0
s6  18109.19  48845.8  4422  8323.3  1.88  0.0013  92  0.1
s7  18109.19  28729.9  3392  8323.3  2.45  0.0022  149  0.3
s8  4217.847  18946.0  2754  9426.8  3.42  0.0008  66  0.3
s10f 12840.13  28996.7  7109  4687.8  0.66  0.0004  18  0.0
s10e 12722.0  29069.3  7169  4623.8  0.65  0.0004  17  0.0
The stagnation enthalpy (Ht) at the end of the acceleration tube 63.397 MJ/kg.
The flight equivalent speed is 11260 m/s.
Species in the shock layer at equilibrium:
['e-': 0.48995, 'c': 0.831e-05, 'N+': 0.39216, 'O+': 0.18536, 'O': 2.5877e-05, 'N': 7.3349e-05, 'Ar+': 0.002352]
What do you want to change? ('Vsd', 'Vs1', 'Vs2', 'quit')
    
```

Figure: Input and output of my first Python version of PITOT.

- ▶ That didn't solve the issue of really want the whole calculation to be equilibrium though, so PITOT was born.
- ▶ Which I should probably call 'PITOT2', now that I have PITOT3.
- ▶ (I originally thought of PITOT3 for Python3, but it works chronologically too.)

- ▶ Peter Jacobs, Rowan Gollan, Fabian Zander (and potentially others? I am sorry if I missed anyone, but it was mostly before my time) had made a Python library which scripted NASA's CEA program for equilibrium gas properties and featured a set of gas dynamics functions for things like shocks, expansions, total conditions etc [1].
- ▶ A lot of these objects were used to make ESTCj (Equilibrium Shock Tube Conditions, junior) for calculating basic reflected shock tunnel freestream conditions (i.e. without a real nozzle expansion like NENZFr does) or as an input for a supersonic nozzle calculation in a code like NENZFr using equilibrium gas properties.
- ▶ There were some basic expansion tube examples lying around too, and I repurposed all of this stuff to make PITOT2.

- ▶ One of my goals was to make PITOT2 much simpler for the user than Richard's PITOT code was.
- ▶ I retained the ability to force shock speeds and calculate the related fill pressures, but the main mode made the user specify the fill pressures and the code would then calculate the shock speeds.
- ▶ Specifying the fill pressures makes it easier for users to use PITOT2 as a 'virtual experiment' where they can change various input parameters (driver condition, fill pressures, test gas, nozzle area ratio etc.) and see the effect on the generated test flow.

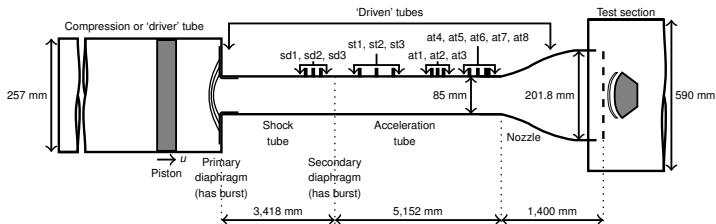


Figure: Schematic of the X2 expansion tube with the x-axis correctly scaled.

- ▶ Facilities, driver conditions, and test gases were specified by name instead of manually inputting the γ and R of different gases. (We also needed the species for the eq chemistry.)
- ▶ While it didn't exist in the original version, eventually I added custom drivers which could either be specified by rupture conditions or fill conditions (with some info about the compression) and custom test gases which could be specified by their fill composition.

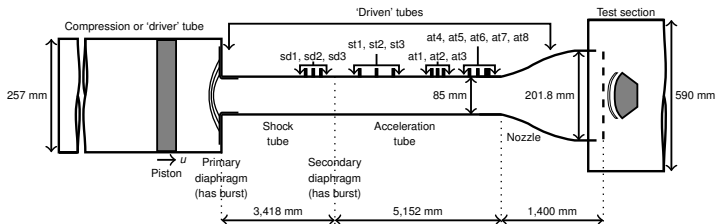


Figure: Schematic of the X2 expansion tube with the x-axis correctly scaled.

- ▶ The use of equilibrium gases instead of perfect gases also meant that the code was much better at predicting the shock speeds of real conditions, which is important.
- ▶ A lot of the calculations become iterative with the equilibrium gases though, which meant thousands of calls to CEA for each run, and PITOT2 generally took a minute or two to run compared to Richard's code's less than a second.
- ▶ Another nice quirk was that the gas dynamics library was independent of the gas model, so with the change of one variable, PITOT2 could be ran in perfect gas mode and it too would run in less than a second.
- ▶ (Things like test gases were still inputted by names or species and then converted to perfect gas objects for the perfect gas calculations so the simplicity of setting up the calculation worked with perfect gases as well.)

- ▶ The fact that CEA has every species under the sun is a real godsend, but working with a code that had to call CEA thousands of times each run, with often low pressure conditions from room temperature up to 20,000 K was a real nightmare sometimes and I had to wrangle the gas dynamics library and PITOT2 a lot over the years to keep CEA behaving.
- ▶ I even fixed a bug in the output of CEA once which was causing numbers to not be outputted due to a rounding issue! So our repository's CEA version has that unique fix.

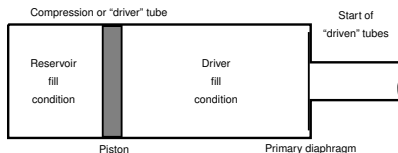
- ▶ One of my big fixes was finally getting CO₂ working after a while.
- ▶ Anyone who has used CEA a lot knows that CO₂ really doesn't behave well at low temperatures, especially in mixtures.
- ▶ My first implementation of it initially set the CO₂ containing gas as a perfect gas state, however, that caused issues with enthalpy calculations as there were no reference states for the subsequent equilibrium states.
- ▶ My current implementation initially only allows the fill gases (CO₂ or CO₂ and N₂ etc.) to be used in the initial CEA calculation to stop the calculation failing.
- ▶ The species restriction is then turned off when the gas temperature is higher and CEA is a bit more stable with CO₂ in the mixture.

- ▶ To make PITOT2 run reliably I had to add a lot of try / catch statements to try to catch any issues and keep the code running. It automatically re-runs low temperature calculations without ions if they fail (a common CEA issue) and if the equilibrium normal shock function had an initial perfect gas temperature guess which is too high it limits it to the more realistic limit of 20,000 K.
- ▶ And there are many others!
- ▶ I even wrapped the normal shock function call in PITOT2 in a function called 'normal_shock_wrapper' which tries to capture failed calculations and fix them (sometimes just changing the shock speed by 0.1 m/s was enough to make a calculation work...)
- ▶ I was eventually able to get PITOT2 running fairly reliably for X2 conditions above 20 km/s towards the end of my PhD.

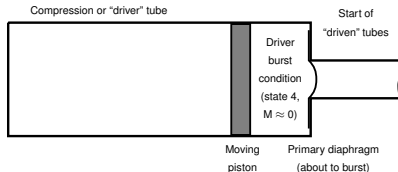
- ▶ PITOT2 is equivalent to similar basic facility simulation codes which have been written overseas such as CHEETAH (the CUBRC equivalent code), a code at Caltech potentially called 'Expansion Tube Calculator' which was benchmarked against PITOT2 when it was written, and I'm sure that there are many others floating around.
- ▶ From reading old papers from our group too, I know that there were similar codes to PITOT floating around in the 90s as well. I'm not sure what was around after that though.
- ▶ David Gildfind had a similar matlab code too.
- ▶ So PITOT2 is definitely not revolutionary, but I think having these codes around are quite helpful, so most groups seem to either have their own or have access to them.
- ▶ I think my support for PITOT2, and a willingness to add the features which different people needed helped it catch on well at UQ and elsewhere.

- ▶ PITOT2 is essentially a '0-D' code as it does not take into account physical geometry to do its calculations.
- ▶ (However it can do things like calculate test time, 'x-t' diagrams etc. using knowledge of the geometry.)
- ▶ We have often called it a 'state-to-state' code as it calculates each gas state as it steps through the facility.
- ▶ There is a journal article about how it works:
- ▶ James, C., Gildfind, D., Lewis, S., Morgan, R., and Zander, F., "Implementation of a state-to-state analytical framework for the calculation of expansion tube flow properties," *Shock Waves*, Vol. 28, No. 2, 2018, pp. 349–377.
- ▶ but I thought that it might be worth explaining how it does a few things as the new code is very similar too and I have got some questions recently about how the code works.

- ▶ As was stated earlier, driver conditions are either specified from empirical rupture conditions or can be calculated from an isentropic compression from the fill state (with a reduced γ if required):



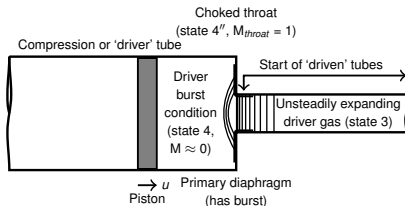
(a) Driver at fill condition



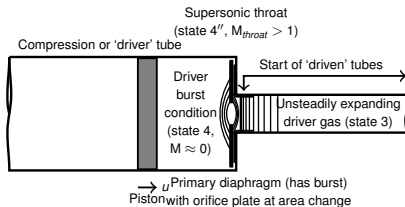
(b) Driver at stagnated rupture condition

Figure: Driver during compression representation. (Not to scale.)

- ▶ The use of a facility area change or orifice plate can be taken into account by inputting the Mach number at the throat:



(a) Without orifice plate



(b) With orifice plate

Figure: Driver after rupture representation. (Not to scale.)

- Shock speeds are found iteratively from the tube inlet condition and the fill state by noting that across the interface behind the shock $p_3 = p_2$ and $V_3 = V_2$, and then using a secant solver and shocking the gas to find p_2 and V_2 , and then unsteadily expanding the inlet condition to the found p_2 and stopping when $V_3 = V_2$ to a given tolerance.

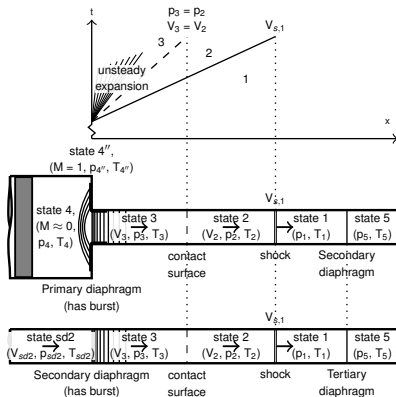


Figure: Expansion tube shock tube representation. (Not to scale.)

- ▶ A similar procedure can be used to guess the fill pressure for a specified shock speed too, if one chooses.

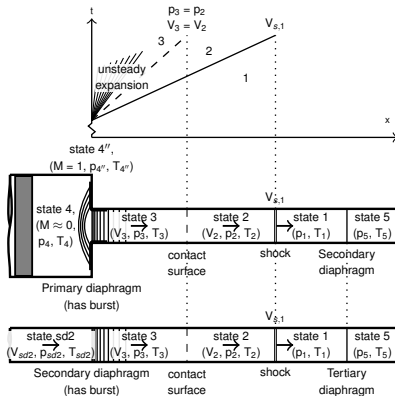


Figure: Expansion tube shock tube representation. (Not to scale.)

- ▶ In the high speed and low pressure acceleration tube the unsteadily expanding test gas is generally expanded to the shock speed after the iterative calculation is performed to simulate Mirels low density shock tube effects.
- ▶ Here is the initial result below:

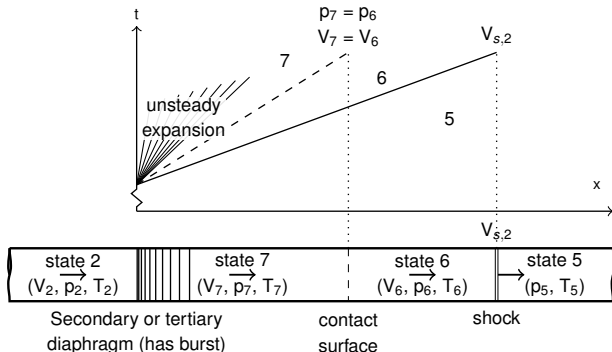


Figure: Expansion tube acceleration tube representation (without over-expansion). (Not to scale.)

PITOT2 - How it Works

- ▶ And then the final result after the over-expansion has been performed:
- ▶ (In the actual PITOT2 code this generally results in p_7 actually being lower than p_6 and V_7 being faster than V_6 though as the over-expansion is done *after* the shock speed has been found.)

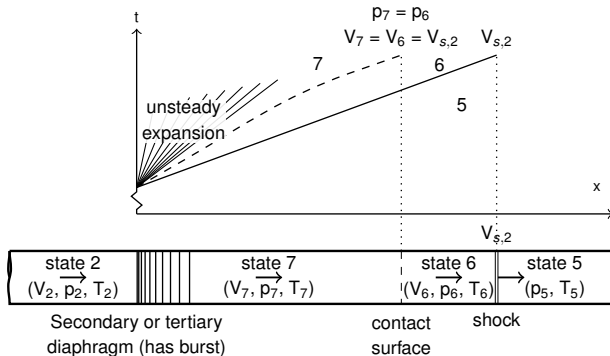
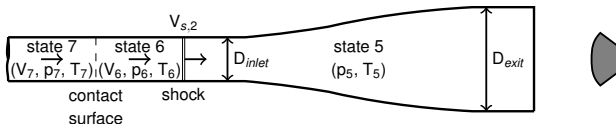


Figure: Expansion tube acceleration tube representation (with over-expansion).
(Not to scale.)

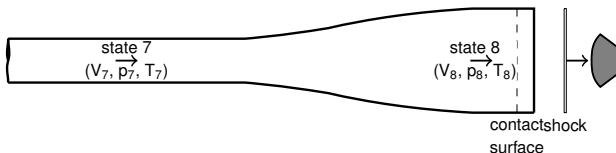
- ▶ This over-expansion has a non-trivial effect on the test condition, so we're starting to get into places where modelling decisions have to be made:
- ▶ (This is for a calculation using Elise Fahy's Hayabusa entry condition [2]. More info about the table can be found in the PITOT paper [3].)

	State 2 expanded to V_6 (9,384 m/s)	State 2 expanded to $V_{s,2}$ (10,010 m/s)	Percentage change (%)
State 7 (nozzle entry condition)			
Static pressure (p_7 , Pa)	18,426	8,721	-52.7
Static temperature (T_7 , K)	2,901	2,659	-8.34
Density (ρ_7 , kg/m ³)	2.13×10^{-2}	1.12×10^{-2}	-47.6
Velocity (V_7 , m/s)	9,384	10,010	6.67
Mach number (M_7)	9.39	10.5	11.9
Stagnation enthalpy (H_t , MJ/kg)	47.9	53.4	11.4
State 8 (nozzle exit condition, using an area ratio of 5.64)			
Static pressure (p_8 , Pa)	2,370	1,069	-54.9
Static temperature (T_8 , K)	2,213	1,904	-13.9
Density (ρ_8 , kg/m ³)	3.72×10^{-3}	1.95×10^{-3}	-47.4
Velocity (V_8 , m/s)	9,547	10,149	6.31
Mach number (M_8)	10.9	12.2	12.4
Stagnation enthalpy (H_t , MJ/kg)	47.9	53.4	11.4

- ▶ The nozzle area ratio is simulated by isentropically expanding the flow through a given area ratio.



(a) Nozzle entry representation



(b) Nozzle exit representation

Figure: X2 nozzle representation, with the shock wave entering and exiting the nozzle. (Not to scale.)

- ▶ The issue is then choosing which area ratio to use (due to boundary layer growth on the acceleration tube wall and in the nozzle):

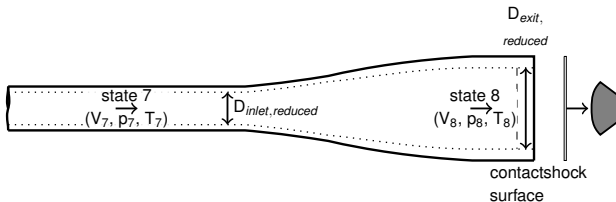


Figure: Nozzle exit representation showing an example of the boundary layer.
(Not to scale.)

- ▶ This is another case which has a large effect on some variables in the final test flow:
- ▶ (Once again for Elise's Hayabusa condition [2].)

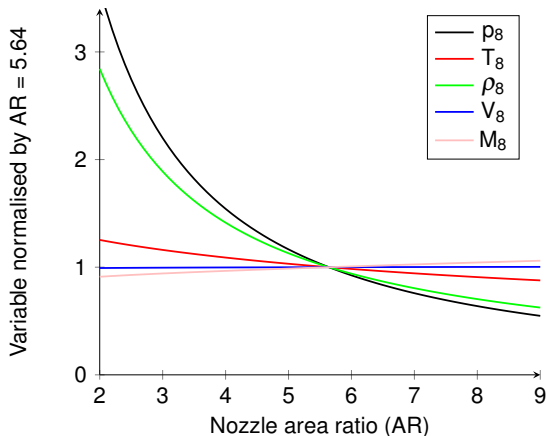


Figure: Effect of changing nozzle area ratio on flow variables at the nozzle exit (state 8) of X2 for Elise's Hayabusa condition [2].

- ▶ When the flow gets to the test section, PITOT2 can do both frozen (to give the value directly behind the shock) and equilibrium normal shocks, conical shocks using the Taylor-Maccoll equations for a given cone half-angle, and oblique shocks for a given wedge angle to do calculations over the test model.
- ▶ The code also calculates pitot pressure, total pressure, stagnation enthalpy, flight equivalent velocity, unit Reynolds number etc.

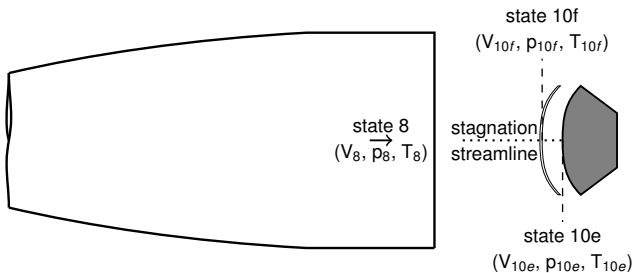


Figure: Representation of flow over a blunt body test model. (Not to scale.)

- ▶ To do experimental calculations, PITOT2 can be ran with the shock speeds fixed (from an experiment).
- ▶ Variables such as the reflected shock at the secondary diaphragm (or not), how far the gas expands in the acceleration tube, and the nozzle effective area ratio can be used to match the measured wall pressure in the acceleration tube and the measured impact pressure in the test section.
- ▶ This is discussed further in the PITOT paper.

- ▶ The output was modelled off Richard's old PITOT code:

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: GWBASIC
Ok
RUN
Us1= 3765 m/s,Ms1= 10.84 ,Us2= 8300 m/s,Ms2= 23.9 ,Us3= 10000 m/s,Ms3= 28.80278
Driver g/R= 1.667 742.9 ,Int 1.4 287 ,test gas 1.4 287 ,accel 1.4 287
Region P          T          a          u          M          rho          pitot stgn
      Pa          K          m/s       m/s       kg/s       kPa          MPa
1  14251.5        300        347        0          0          .1655        0          0
2  1952902        7142       1694       3110       1.83       .9526       9447       11.8
3  1952902        931        1074       3110       2.89       2.8215      24950      54.8
4  2.79E+07       2700       1828       0          0          0          0          0
5  45.8           300        347        0          0          .0005       0          0
6  30544.7        33621      3675       6904       1.87       .0031       153        .1
7  30544.7        2177       935        6904       7.38       .0488       2157       177.4
8  17405.6        28630      3391       8323       2.45       .0021       143        .2
9  17405.6        48676      4422       8323       1.88       .0012       87         .1
10 17.9           300        347        0          0          .0002       0          0
11 1.358987E+07  2024       1583       1583       1          9.034701   27900     27.9
12 1.495421E+07  15912      2528       0          0          3.2744     0          0
13 0              0          0          0          0          0          0          0
14 1952902        7142       1694       3110       1.83       .9526       9447       11.8
Pressure behind standing shock in region 2= 7356 kPa
Reflected shock speed= 1276.343 m/s, ref= 0
Ok
1LIST 2RUN+ 3LOAD" 4SAVE" 5CONT+ 6,"LPT1 7TRON+ 8TROFF+ 9KEY 0SCREEN
    
```

Figure: Richard's original Pitot output.

► With some new things added:

```
Pitot Version: 14-May-2021 doing an expansion tube calculation
state 4 is the driver condition.
state 1 is shock tube fill. state 5 is acceleration tube fill.
state 2 is the shocked test gas.
state 7 is expanded test gas entering the nozzle.
state 8 is test gas exiting the nozzle (using area ratio of 5.64).
state 10f is frozen shocked test gas flowing over the model.
state 10e is equilibrium shocked test gas flowing over the model.
state 10c is conditions over 15.0 degree conehead in the test section.
Solver used is equilibrium.
Test is 'fulltheory-pressure'
Facility is x2.
Driver gas is ('He': 1.0) (by moles).
Test gas (state 1) is air (gamma = 1.4, R = 287.036078025, {'Air': 1.0} by moles).
Accelerator gas (state 5) is Air.
Vs1 = 5641.06 m/s, Ms1 = 16.29, Vs2 = 11237.87 m/s, Ms2 = 32.46
state P      T      a      V      M      rho      pitot  stgn
Pa          K          m/s    m/s    M      kg/m^3   kPa     MPa
s4  2.790e+07  2700.0  3057  0.0    0.00  4.97450  27900.0  27.900
s3s 2.711e+06  1063.0  1918  4123.4 2.15  1.22780  19642.0  27.847
s1  3000.0    298.2   346   0.0    0.00  0.03505  3.0      0.003
s2  1.020e+06  6802.4  1722  5145.3 2.99  0.39883  10697.0  49.401
s3  1.019e+06  719.1   1578  5145.3 3.26  0.68241  16366.0  44.842
s5  10.0       298.2   346   0.0    0.00  1.17e-04  0.0      0.000
s6  13876.0    10941.0 2874  10560.3 3.67  1.94e-03  215.6   2.477
s7  6978.7     3450.2  1167  11237.9 9.63  6.07e-03  741.9   10543.990
s8  935.4911   2808.1  1000  11416.0 11.41 1.06e-03  133.9   10530.360
s10f 129356.2  26836.9 3092  789.0  0.26  0.01532  130.5   0.130
s10e 129240.0  12789.3 3169  799.3  0.25  0.01512  133.9   0.134
s10cf 10700.3  4695.4  1293  10964.6 8.48  7.24e-03  848.8   34673.221
s10c 10773.0  4717.0  1396  10956.2 7.85  6.48e-03  755.2   3151.160
The total enthalpy (Ht) leaving the nozzle is 69.93 MJ/kg (H8 - h1).
The total temperature (Tt) leaving the nozzle is 24724 K.
The freestream enthalpy (h) leaving the nozzle is 4.7681 MJ/kg (h8 - h1).
The flight equivalent velocity (Ue) is 11826 m/s.
The Pitot pressure leaving the nozzle (state 8) is 133.89 kPa
Basic test time = 53.28 microseconds
Using a freestream (s8) dynamic viscosity (mu) of 9.1256e-05 Pa.s.
Freestream (s8) unit Reynolds number is 132429.19/m.
Using a test section post normal shock eq (s10e) dynamic viscosity (mu) of 2.5008e-04 Pa.s.
Test section post normal shock eq (s10e) unit Reynolds number is 48324.37/m.
Species in the freestream state (s8) at equilibrium (by moles):
('CO2': 4.5855e-05, 'CO': 0.00024511, 'NO': 0.023202, 'O': 0.17552, 'N': 3.2117e-05, 'Ar': 0.0085418, 'N2': 0.70059, 'O2': 0.091824, 'NO2': 1.0013e-06)
Species in the shock layer at equilibrium (s10e) (by moles):
('e-': 0.14515, 'C+': 7.8658e-05, 'C': 5.8319e-05, 'N+': 0.12332, 'O+': 0.020952, 'N-': 1.3005e-05, 'NO': 1.1199e-05, 'NO+': 2.693e-05, 'O-': 5.5528e-06, 'O': 0.1592, 'N': 0.547, 'N2+': 3.4276e-05, 'Ar': 0.0032677, 'N2': 0.00012311, 'Ar+': 0.00075426)
Removing temporary files and leaving the program.
```

Figure: PITOT2 output.

- ▶ PITOT2 can also do a lot of other things as well, some of which is a bit hidden in the code, annoyingly (but that is my fault! features just kept building up over the years).
- ▶ It can run in experimental mode where both fill pressures and shock speeds are specified, and semi-experimental modes where one shock speed is specified.
- ▶ Generally the unsteadily expanding test gas in the acceleration tube is expanded to the shock speed to simulate Mirels effects in the low density acceleration tube (as we discussed before).
- ▶ It can drop the post-shock velocity to simulate total pressure loss, can do normal shocks at diaphragms, can freeze the unsteady expansion or the nozzle expansion, and probably more things that I have forgotten!
- ▶ It can also simulate reflected shock tunnels and non-reflected shock tubes.

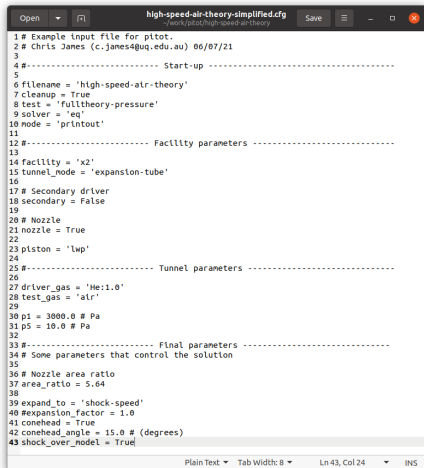
- ▶ I also wrote several related codes over the years:
 - ▶ **Pitot Condition Builder** which allows the code to be scripted to create performance maps. It can restart if it fails, will skip individual failed runs, and creates a large summary at the end.
 - ▶ **Pitot Area Ratio Check** which will try a lot of different nozzle area ratios at the end of one calculation.
 - ▶ **Pitot State 2 Reflected Shock Analysis** which changes the strength of a reflected shock at the end of the shock tube between given limits.
 - ▶ **Pitot Bootstrapper** which allows large amounts of runs to be ran with different uncertainties given to different inputs to get statistical averages of conditions.
 - ▶ A Pitot compression ratio scripting code and a code to try different amounts of helium or neon diluent for giant planet entry work in my PhD.
 - ▶ I made our experimental Shot analysis code run an experimental version of Pitot to give an initial estimate of test conditions.
 - ▶ I wrote a code that would run Pitot with all of the combinations of the shock speed uncertainties to get the nominal condition and related uncertainties for any output parameters.

- ▶ In terms of how the code itself worked, PITOT2 could either be ran from a terminal with a config file which had to conform to Python syntax, and was then loaded into Python as a dictionary, or it could be run in a Python script by importing the `run_pitot` function from `pitot.py` and then giving it a configuration dictionary in the script.

PITOT2 - Running the Code

- ▶ An example config file can be seen below which can be run by running:

```
$ pitot.py --config_file high-speed-air-theory-simpl
```



```
high-speed-air-theory-simplified.cfg
1 # Example input file for pitot.
2 # Chris James (c.james4@uq.edu.au) 06/07/21
3
4 #----- Start-up -----
5
6 filename = 'high-speed-air-theory'
7 cleanup = True
8 test = 'fulltheory-pressure'
9 solver = 'eq'
10 node = 'printout'
11
12 #----- Facility parameters -----
13
14 facility = 'x2'
15 tunnel_node = 'expansion-tube'
16
17 # Secondary driver
18 secondary = False
19
20 # Nozzle
21 nozzle = True
22
23 piston = 'lwp'
24
25 #----- Tunnel parameters -----
26
27 driver_gas = 'He:1.0'
28 test_gas = 'air'
29
30 p1 = 3000.0 # Pa
31 p5 = 10.0 # Pa
32
33 #----- Final parameters -----
34 # Some parameters that control the solution
35
36 # Nozzle area ratio
37 area_ratio = 5.64
38
39 expand_to = 'shock-speed'
40 #expansion_factor = 1.0
41 conehead = True
42 conehead_angle = 15.0 # (degrees)
43 shock_over_model = True
```

Figure: PITOT2 config file.

- ▶ The same simulation can be ran using the Python script below by running: `$ python2 pitot_scripting_example.py`

```
#!/usr/bin/env python2
"""
pitot_scripting_example.py
This example shows how the same PITOT configuration shown in high-speed-air-theory.cfg
could be ran from inside Python by using importing run_pitot and giving it the
configuration dictionary directly from inside Python.

Chris James (c.james4@uq.edu.au) - 28/02/16
"""

import sys, os
sys.path.append(os.path.expandvars("$HOME/e3bin")) # installation directory
sys.path.append("") # so that we can find user's scripts in current directory

from pitot import run_pitot

cfg = {'filename': 'high-speed-air-theory', 'cleanup': True,
       'test': 'fulltheory-pressure', 'solver': 'eq',
       'mode': 'printout', 'facility': 'x2', 'tunnel_mode': 'expansion-tube',
       'nozzle': True, 'secondary': False, 'piston': 'lwp',
       'driver_gas': 'He:1.0', 'test_gas': 'air', 'p1': 3000.0, 'p5': 10.0,
       'area_ratio': 5.64, 'expand_to': 'shock-speed', 'expansion_factor': 1.0,
       'conehead': True, 'conehead_angle': 15.0, 'shock_over_model': True}

run_pitot(cfg=cfg)
```

- ▶ Behind the scenes, PITOT2 then ran an input check function, a set up function which created all of the required gas states, and then a series of functions (depending on the simulation being performed) to do the calculation and output the results.
- ▶ There were a lot of functions, but no classes, and each function had both its input and its output as four dictionaries:
 - ▶ cfg (config data)
 - ▶ states (gas states)
 - ▶ V (velocities for the gas states)
 - ▶ M (the Mach number for the gas states)
- ▶ Overall, I think that the organisation of the code suffered from the fact that a lot of features were bolted on over the time, the fact that I didn't know how to use classes when I started it, and a lot of the configuration like default values, named facilities and driver conditions, named test gases etc. were hidden away in the input function where users couldn't see them without rifling through the source code.

PITOT2 - What it has Been Used For

- ▶ PITOT2 has been a very productive and useful code.
- ▶ In my own PhD, I used it generate performance maps for simulating Uranus entry in X2, and I worked out that I thought that we could *just* get to the required 22 km/s for Uranus entry:

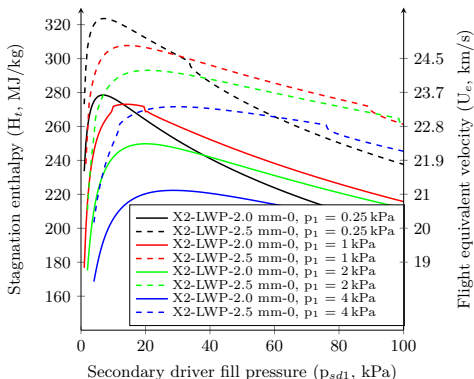


Figure: Effect of secondary driver fill pressure (p_{sd1}) on performance for different Uranus entry test conditions with a set p_5 value of 0.5 Pa.

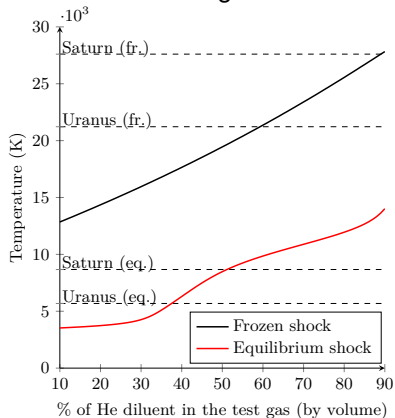
- ▶ I also then used the code to estimate the uncertainties in the flow conditions which we were eventually able to generate:

Table: Computed test section freestream and post-shock state ranges for experiment x2s3249.

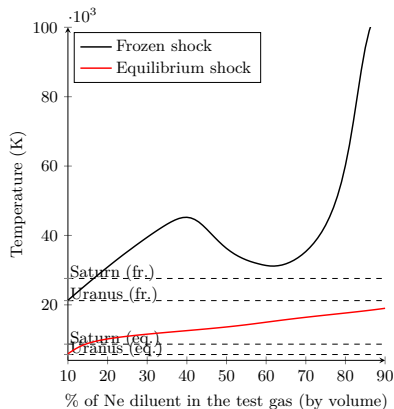
	Nominal	Solution bounds	Uncertainties
State 8 (nozzle exit condition, using an effective area ratio of 3.5)			
Static pressure (p_8 , kPa)	0.407	0.187 – 0.860	-54% / +110%
Static temperature (T_8 , K)	1,390	1,020 – 1,830	-26% / +31%
Density (ρ_8 , kg/m ³)	8.13×10^{-5}	$5.09 \times 10^{-5} - 1.31 \times 10^{-4}$	-37% / +61%
Velocity (V_8 , m/s)	21,400	20,500 – 22,200	-4.1% / +3.6%
Mach number (M_8)	8.14	7.09 – 9.70	-13% / +19%
Stagnation enthalpy (H_t , MJ/kg)	242	222 – 261	-8.2% / +7.8%
Flight equivalent velocity (U_e , m/s)	22,000	21,100 – 22,900	-4.2% / +3.8%
Pitot pressure (p_{pitot} , kPa)	34.5	22.8 – 52.5	-34% / +52%
Stagnation pressure (p_t , MPa)	151	98.4 – 225	-35% / +48%
State 10e (equilibrium post normal shock pressure in the test section)			
Static pressure (p_{10e} , kPa)	34.6	22.8 – 51.9	-34% / +50%
Static temperature (T_{10e} , K)	5,240	4,500 – 6,260	-14% / +19%
Density (ρ_{10e} , kg/m ³)	9.96×10^{-4}	$5.99 \times 10^{-4} - 1.63 \times 10^{-3}$	-40% / +64%

PITOT2 - What it has Been Used For

- ▶ When we couldn't simulate Saturn entry directly, it then allowed me to further investigate use of a test gas substitution to do that:



(a) helium diluent



(b) neon diluent

Figure: Effect of diluent fraction on the frozen and equilibrium post-normal shock temperatures in the stagnation region over the test model (state 10)

PITOT2 - What it has Been Used For

- ▶ When I had a quick look at xlabs PhDs just then, (to my knowledge) the Phds of Sam Stennett, Andreas, Rory, myself, Sangdi, Steven Lewis, Gueric, Elise, and Han, all made use of PITOT, which is almost everyone who started a PhD after I wrote the code. As well as a lot of people who are still students.
- ▶ The code has also been used a lot at Oxford, by colleagues at EPFL and IRS in Stuttgart who were analysing experiments that they did at UQ, the student from Caltech who used it to benchmark a similar code, Sangdi in Germany when he was making his paper about facilities, and the most out there reference is researchers from Turkey who in fact have an expansion tube.
- ▶ That is pretty good, I think!

- ▶ Now that Peter and Rowan have moved to eilmer4 and they have made a new related Python library, and Python2 is no longer officially supported, it seemed like the right time to make PITOT3.
- ▶ I dawlded a bit last year but then got started over the Christmas / new year break last year and have been doing more over the last couple of months.
- ▶ My goal was to do a lot more planning so I would (hopefully) end up with much more well written and modular code.
- ▶ I also wanted to bring the configuration of the code out into the open to make it easier for people to make their own modifications in terms of creating driver conditions, test gases, facilities etc.

- ▶ To start at the end I guess, I tried to keep the output very similar to PITOT2, but I squeezed everything up a bit to get another few columns in. Here is the PITOT2 output from before:

```
Pitot Version: 14-May-2021 doing an expansion tube calculation
state 4 is the driver condition.
state 1 is shock tube fill, state 5 is acceleration tube fill.
state 2 is the shocked test gas.
state 7 is expanded test gas entering the nozzle.
state 8 is test gas exiting the nozzle (using area ratio of 5.64).
state 10f is frozen shocked test gas flowing over the model.
state 10e is equilibrium shocked test gas flowing over the model.
state 10c is conditions over 15.0 degree conehead in the test section.
solver used is equilibrium.
Test is 'fulltheory-pressure'
Facility is x2.
Driver gas is ('He': 1.0) (by moles).
Test gas (state 1) is air (gamma = 1.4, R = 287.036678025, ('Air': 1.0) by moles).
Accelerator gas (state 5) is Air.
Vs1 = 5641.06 m/s, Ms1 = 16.29, Vs2 = 11237.87 m/s, Ms2 = 32.46
state P      T      a      V      M      rho      pitot      stgn
      Pa      K      m/s    m/s    N      kg/m3    kPa      MPa
s4  2.790e+07  2700.0  3057  0.0  0.00  4.97458  27900.0  27.900
s5s 2.711e+06  1603.0  1918  4123.4  2.15  1.22780  19642.0  27.847
s1  3000.0     298.2   346  0.0  0.00  0.03505  3.0  0.003
s2  1.020e+06  6002.4  1722  5145.3  2.99  0.39883  10697.0  49.041
s3  1.019e+06  719.1   1578  5145.3  3.26  0.08241  16366.0  44.842
s5  39.0       298.2   346  0.0  0.00  1.17e-04  0.0  0.000
s6  13876.0    18941.0 2074  18560.3  3.67  1.04e-03  215.6  2.477
s7  6978.7     3458.2  1167  11237.9  9.63  6.07e-03  741.9  10543.990
s8  935.4911   2808.1  1000  11416.0  11.41 1.06e-03  133.9  10530.360
s10f 129350.2   26836.9 3092  789.0  0.26  0.01532  130.5  0.130
s10e 129240.0   12789.3 2169  789.3  0.25  0.01512  133.9  0.134
s10cf 10780.3  4695.4  1293  18964.6  8.48  7.24e-03  848.8  34673.221
s10c 10773.0   4717.0  1396  18956.2  7.85  6.48e-03  755.2  3151.160
The total enthalpy (Ht) leaving the nozzle is 69.93 MJ/kg (H8 - h1).
The total temperature (Tt) leaving the nozzle is 24724 K.
The freestream enthalpy (h) leaving the nozzle is 4.7081 MJ/kg (h8 - h1).
The flight equivalent velocity (Ue) is 11826 m/s.
The Pitot pressure leaving the nozzle (state 0) is 133.89 kPa
Basic test time = 53.28 microseconds
Using a freestream (s8) dynamic viscosity (mu) of 9.1256e-05 Pa.s.
freestream (s8) unit Reynolds number is 134249.19/m.
Using a test section post normal shock eq (s10e) dynamic viscosity (mu) of 2.5800e-04 Pa.s.
test section post normal shock eq (s10e) unit Reynolds number is 48324.37/m.
Species in the freestream state (s8) at equilibrium (by moles):
('CO2': 4.5855e-05, 'CO': 0.00024511, 'NO': 0.023202, 'O': 0.17552, 'N': 3.2117e-05, 'Ar': 0.0085418, 'H2': 0.70059, 'O2': 0.091824, 'NO2': 1.0013e-06)
Species in the shock layer at equilibrium (s10e) (by moles):
('e-': 0.14515, 'C+': 7.8658e-05, 'C': 5.8319e-05, 'N+': 0.12332, 'O+': 0.020952, 'N': 1.3005e-05, 'NO': 1.1199e-05, 'NO+': 2.693e-05, 'O+': 5.5528e-06, 'O': 0.1592, 'N': 0.547, 'N2+': 3.4276e-05, 'Ar+': 0.0032677, 'N2': 0.00012311, 'Ar+': 0.00075426)
Removing temporary files and leaving the program.
```

Figure: PITOT2 output.

► And the current PITOT3 output:

```

PITOT3 Version 14-Jun-2021 doing an expansion_tube calculation.
Calculation mode is 'fully_theoretical'.
Facility is 'x2_nozzle'.
Driver condition is 'x2-lwp-2.0mm-100He-0'. Driver gas model is CEAGas.
Driver gas composition is {'He': 1.0} (by mass) (gam = 1.67, R = 2077.07 J/kg K).
Nozzle area ratio is 5.64.
Test gas (s1) is air13species. Test gas gas model is CEAGas.
Test gas composition is {'N2': 0.75566, 'Ar': 0.01283, 'O2': 0.23151} by mass (gam = 1.40, R = 287.11 J/kg K)
Accelerator gas (s5) is air13species. Accelerator gas model is CEAGas.
Accelerator gas composition is {'N2': 0.75566, 'Ar': 0.01283, 'O2': 0.23151} by mass (gam = 1.40, R = 287.11 J/kg K).
vs1 = 5757.02 m/s, Ms1 = 16.63, vs2 = 11469.08 m/s, Ms2 = 33.13
state p          T          a          v          M          rho          pitot_p  stgn_p  Ht          h
      Pa          K          m/s       m/s          M          kg/m^3       kPa      MPa      MJ/kg      MJ/kg
s4      2.740e+07  2903.0    3170     0.0          0.00    4.54370    27400    27.40    13.53    13.53
s3s     2.589e+06    1129.7    1978    4291.6       2.17    1.10310    19109    27.40    13.53    4.32
s1      3000.0       298.15    346     0.0          0.00    3.50e-02    3.0      0.00    0.00    0.00
s2      1.064e+06    6915.8    1745    5258.2       3.01    0.40447    11337    53.60    30.27    16.45
s3      1.064e+06    791.60    1656    5258.2       3.18    0.64699    16247    42.30    16.39    2.56
s5      10.0         298.15    346     0.0          0.00    1.17e-04    0.0      0.00    0.00    0.00
s6      14455.0      11136    2924    10782       3.69    1.95e-03    226.3    2.62    123.7    65.53
s7      6890.698    3525.4    1191    11469       9.63    5.81e-03    742.3    10879    72.82    7.05
s8      914.5083    2839.9    1011    11649      11.53    1.01e-03    133.9    10879    72.82    4.97
s10e   129020.0     13000    3224    810.8        0.25    1.46e-02    133.9    0.13    72.82    72.49
s10c   10591.0      4815.9    1409    11178       7.94    6.20e-03    753.8    3305.3    72.82    10.35
s10w   62268.0      7786.2    2231    8430.6       3.78    1.48e-02    1036.4    10.35    72.82    37.28
The freestream (s8) total temperature (Tt) is 25650.34 K.
The freestream (s8) flight equivalent velocity (Ue) is 12067.85 m/s.
Wedge angle was 40.00 degrees. Wedge shock angle (beta) was found to be 43.77 degrees.
Freestream (s8) unit Reynolds number is 128106.05 /m (related mu is 9.23e-05 Pa.s).
Post normal shock equilibrium (s10e) unit Reynolds number is 48109.77 /m (related mu is 2.46e-04 Pa.s).
Species in the freestream state (s8) at equilibrium (by mass):
{'N2': 0.74338, 'N': 2.1794e-05, 'Ar': 0.01283, 'O': 0.1164, 'O2': 0.10112, 'NO': 0.026256}
Species in the shock layer at equilibrium (s10e) (by mass):
{'e-': 7.2156e-06, 'O+': 0.030549, 'N2+': 0.00021814, 'N+': 0.59887, 'N2+': 7.3096e-05, 'Ar+': 0.010068, 'O': 0.20092, 'N
O': 2.2654e-05, 'Ar+': 0.0027624, 'N+': 0.15645, 'NO+': 5.884e-05}
    
```

Figure: The current PITOT3 output.

- ▶ PITOT3 forms part of the Centre's Gas Dynamics Toolkit (GDTk) which can be found at <http://cfcfd.mechmining.uq.edu.au/>.
- ▶ (Internally we might know that as the 'Eilmer 4 repository'.)
- ▶ To use PITOT3, you need to pull down the repository at that link using git, install the dependencies, and then download CEA and put it in the folder `dgd/extern/cea2`.
- ▶ Then if one navigates to `dgd/src/pitot3` and runs the makefile (i.e. `make install`) it will install the GDTk gas library, gas dynamics library, CEA, and then the PITOT3 program.
- ▶ The standard additions to one's `.bashrc` file for use of the GDTk gas libraries are required, as well as the following additions:
 - ▶ `export PITOT3_DATA=$DGD/share/pitot3_data`
 - ▶ `export PYTHONPATH=$PYTHONPATH:$DGD/lib:$DGD/bin`
- ▶ The top line is a new variable for PITOT3 and the addition of the `$DGD/bin` folder to the `PYTHONPATH` lets Python3 find `pitot3.py` so PITOT3 can be scripted.

PITOT3 - Setting up the Code

- ▶ In terms of compatible operating systems, I have only ever used PITOT3 on Ubuntu, but I'm sure that it works on other Linux distributions too.
- ▶ I know that people have used it through the Ubuntu terminal which is now available in Windows 10 and people can probably get it working on Mac OS too, but I haven't tried personally.
- ▶ If you are running the code for the first time, I would recommend just setting up an Ubuntu virtual machine using Virtualbox and going from there. I think that that is probably the most straight forward way and PITOT3 isn't too system intensive anyway.
- ▶ The Windows 10 Ubuntu terminal may be a good solution too as PITOT3 isn't graphical at all, but I don't know how hard that is to get working.

PITOT3 - Setting up the Code

- ▶ Just so everyone knows where things are installed:
- ▶ The default pitot3.py installation location is dgdinst/bin.
- ▶ The default install location for the other supporting functions (currently pitot3_classes.py but I make might one for functions too, as it currently includes classes and functions) is dgdinst/lib/pitot3_utils
- ▶ The default configuration file, facility data, and any pre-defined gas models are installed in dgdinst/share/pitot3_data.

- ▶ Like PITOT2 the simulation config data can either be given to `pitot3.py` as a file or by running the function `run_pitot3` in a Python script with the config defined by a Python dictionary.
- ▶ In PITOT2, the config file was pure Python, which is a no no these days as the function in Python2 would execute whatever was in the config file, which is good for simplicity (i.e. you can change a list into an array in the config file etc.) but not good for security, so it doesn't seem to be a done thing anymore.
- ▶ For this reason, PITOT3 uses YAML for all the config files.

- ▶ Thankfully, YAML syntax is quite similar to Python syntax and supports the same comment character, strings, lists, and dictionaries, just with a `:` instead of an `=` when declaring variables.
- ▶ Here is an example PITOT3 config file which can be ran as:

```
pitot3.py --config_file PITOT3_input_config.yaml
```

```
# PITOT3_input_config.yaml: input config for PITOT3
# Just a simple config file for my talk
# Chris James (c.james4@uq.edu.au) - 07/07/21

# initial code set up
mode : 'fully_theoretical'
# facility set up
facility : 'x2_nozzle'
driver_condition : 'x2-lwp-2.0mm-0'
# shock tube
test_gas_gas_model : 'CEAGas'
test_gas_name : 'n2-o2-with-ions'
p1 : 3000.0
# acceleration tube
p5 : 10.0
acceleration_tube_expand_to : 'shock_speed'
# nozzle area ratio
area_ratio : 5.64
# test section stuff
cone_half_angle_degrees : 15.0 # degrees
wedge_angle_degrees : 45.0 # degrees
```

- ▶ And an equivalent Python3 scripted example which can be ran as:

```
$ python3 pitot_3_scripting_example.py
```

```
"""
An example to script PITOT3 for my talk.

Chris James (c.james4@uq.edu.au) - 07/07/21
"""

from pitot3 import run_pitot3

config_dict = {'mode':'fully_theoretical','facility':'x2_nozzle',
               'driver_condition':'x2-lwp-2.0mm-0',
               'test_gas_gas_model':'CEAGas', 'test_gas_name':'n2-o2-with-ions',
               'p1':3000.0, 'p5':10.0,
               'area_ratio':5.64,
               'cone_half_angle_degrees':15.0, 'wedge_angle_degrees':45.0}

config_data, gas_path, object_dict = run_pitot3(config_dict = config_dict)
```

- ▶ To open everything up to the user, all of the PITOT3 configuration information is now stored in a default configuration YAML file in the folder `dgdinst/share/pitot3_data` called `PITOT3_default_config.yaml`.
- ▶ This file is where the default output filename, facilities and preset gas model folder locations, state names (as some groups define these differently), standard temperature and pressure, tube names, secant solver initial guesses, limits, and tolerances, and any other variables which the program needs a default value for.
- ▶ The user is completely free to make their own default configuration file instead of my one (and store it elsewhere so it won't get overridden when the code is next installed).
- ▶ (I do need to get this final functionality working as there is no way to specify your own default configuration file location at the moment...)

- ▶ When PITOT3 runs, the default config is read in first from the default configuration and then values are overridden with values in the user defined configuration file, followed by any configuration Python3 dictionary which is parsed to the `run_pitot3` function.
- ▶ This means that if the user wants to override any default value for their current run script they can just add that variable to either their configuration file or their Python3 dictionary and it will override the default value.
- ▶ (They can also get extra fancy and load in a value from a configuration file and then later override it by another value in a Python dictionary. I can't particularly think of a use for that currently, but figured that someone may so I wrote it that way.)

- ▶ Facility configuration data is now stored externally as well in the folder `dgdinst/share/pitot3_data/facilities` in their own YAML config files.
- ▶ I currently only added X2 with a nozzle and with and without a secondary driver and the Drummond Tunnel with its Mach 7 nozzle myself.
- ▶ One of my undergraduate students, Lewis Barltrop, has converted over some of the X3 facility configurations but we haven't put it in the repository yet.
- ▶ We do plan to very soon though!
- ▶ (Just working through the million X3 geometry configurations...)

- ▶ An example facility configuration file for X2 with its nozzle can be seen below:

```
# x2_nozzle.yaml
# Configuration file for X2 facility with nozzle in Pitot 3
#
# Must conform to Yaml syntax, which is pretty forgiving, thankfully...
# Chris James (c.james4@uq.edu.au) - 26/12/20
# Lengths and sensor locations from Gildfind et al. (2014)
# Production of High-Mach-Number Scramjet Flow Conditions in an Expansion Tube
# AIAA Journal, Vol 52, No. 1, pp 162 - 177
# Updated 07/07/21 to correct incorrect acceleration tube diameter - CMJ.

facility_name : 'X2'
facility_type : 'expansion_tube'
secondary_driver : False
nozzle : True

shock_tube_length : 3.418 # m
shock_tube_diameter : 0.085 # m
shock_tube_sensors : ['sd1', 'sd2', 'sd3']

acceleration_tube_length : 5.167 #m
acceleration_tube_diameter : 0.085 # m
acceleration_tube_sensors : ['st1', 'st2', 'st3', 'at1', 'at2', 'at3', 'at4', 'at5', 'at6', 'at7', 'at8']

nozzle_geometric_area_ratio : 5.64

sensor_locations : {'sd1':2.577, 'sd2':2.810, 'sd3':3.043, 'st1':4.231, 'st2':4.746, 'st3':5.260,
                    'at1':6.437, 'at2':6.615, 'at3':6.796, 'at4':7.590, 'at5':7.846, 'at6':8.096, 'at7':8.342, 'at8':8.588}

driver_conditions_folder : 'x2_driver_conditions' #inside the facilities folder...
```

PITOT3 - Specifying Facility Information

- ▶ Only the test time stuff in PITOT3 uses the facility geometry currently, but I wanted to have it all there for the future, if needed.

```
# x2_nozzle.yaml
# Configuration file for X2 facility with nozzle in Pitot 3
#
# Must conform to Yaml syntax, which is pretty forgiving, thankfully...
# Chris James (c.james4@uq.edu.au) - 26/12/20
# Lengths and sensor locations from Gildfind et al. (2014)
# Production of High-Mach-Number Scramjet Flow Conditions in an Expansion Tube
# AIAA Journal, Vol 52, No. 1, pp 162 - 177
# Updated 07/07/21 to correct incorrect acceleration tube diameter - CMJ.

facility_name : 'X2'
facility_type : 'expansion_tube'
secondary_driver : False
nozzle : True

shock_tube_length : 3.418 # m
shock_tube_diameter : 0.085 # m
shock_tube_sensors : ['sd1', 'sd2', 'sd3']

acceleration_tube_length : 5.167 #m
acceleration_tube_diameter : 0.085 # m
acceleration_tube_sensors : ['st1', 'st2', 'st3', 'at1', 'at2', 'at3', 'at4', 'at5', 'at6', 'at7', 'at8']

nozzle_geometric_area_ratio : 5.64

sensor_locations : {'sd1':2.577, 'sd2':2.810, 'sd3':3.043, 'st1':4.231, 'st2':4.746, 'st3':5.260,
                   'at1':6.437, 'at2':6.615, 'at3':6.796, 'at4':7.590, 'at5':7.846, 'at6':8.096, 'at7':8.342, 'at8':8.588}

driver_conditions_folder : 'x2_driver_conditions' #inside the facilities folder...
```

- ▶ The user can also choose to not need a related facility file by specifying the facility as 'None' and then input the `facility_type`, `secondary_driver` and `nozzle` variables in their script.
- ▶ This does not appear to be working currently, but I will fix this...
- ▶ I can also probably allow other facility information to be added to the simulation input config file too.

- ▶ As we often have multiple different facility configurations, I decided to add a driver conditions folder for each facility, which is specified in the facility configuration files.
- ▶ Driver conditions are then specified externally too in their own config file either as empirical rupture conditions or as an isentropic driver with the isentropic compression to the rupture condition done by PITOT3 at the start of the simulation.
- ▶ I have currently added every X2 piston driven driver condition, however, the one cold driven driver condition currently specified does not work yet as I need to specify it as a perfect gas driver, which I don't have working yet.
- ▶ The user can also put specify their own custom driver condition with the simulation by setting the `driver_condition` input variable to 'custom' and then using the variable 'driver_condition_filename' to specify the location of their driver configuration file.

- ▶ Here is an example of how to set up an empirical driver condition, for X2's 2.5 mm diaphragm driver condition:

```
# X2's 80%He 2.5 mm driver condition with an empirical T4 value.
# Values from Figure 6.8 in Gildfind (2012)
# Development of High Total Pressure Scramjet Flow Conditions using the X2 Expansion Tube

driver_condition_name : 'x2-lwp-2.5mm-0-empirical'
driver_condition_type : 'empirical'

driver_gas_model : 'CEAGas'
driver_fill_composition : {'He':0.8,'Ar':0.2}
driver_speciesList : ['He','Ar']
driver_inputUnits : 'moles'
driver_withIons : False

p4 : 35.7e6 # Pa
T4 : 3077.0 # K

M_throat : 1.0
```

- ▶ And the related isentropic compression (without any losses) input for the same driver condition:

```
# X2's 80%He 2.5 mm driver condition with isentropic compression
# Values from Table 3 of Gildfind et al. (2011)
# Free-piston driver optimisation for simulation of high Mach number scramjet flow conditions

driver_condition_name : 'x2-lwp-2.5mm-0-isentropic'
driver_condition_type : 'isentropic-compression-p4'

driver_gas_model : 'CEAGas'
driver_fill_composition : {'He':0.8,'Ar':0.2}
driver_speciesList : ['He','Ar']
driver_inputUnits : 'moles'
driver_withIons : False

driver_p : 77.2e3 # driver fill pressure, Pa
driver_T : 298.15 # driver fill temperature, K

p4 : 35.7e6 # Pa

M_throat : 1.0
```


- ▶ I decided to put the input for the driver gas inside the driver condition input file so that everything is self contained.
- ▶ PITOT3 then makes a CEA gas model file for the driver gas called PITOT3_cea_driver_condition.lua at run time which PITOT3 uses for the calculation.

```
# X2's 80%He 2.5 mm driver condition with isentropic compression
# Values from Table 3 of Gildfind et al. (2011)
# Free-piston driver optimisation for simulation of high Mach number scramjet flow conditions

driver_condition_name : 'x2-lwp-2.5mm-0-isentropic'
driver_condition_type : 'isentropic-compression-p4'

driver_gas_model : 'CEAGas'
driver_fill_composition : {'He':0.8,'Ar':0.2}
driver_speciesList : ['He','Ar']
driver_inputUnits : 'moles'
driver_withIons : False

driver_p : 77.2e3 # driver fill pressure, Pa
driver_T : 298.15 # driver fill temperature, K

p4 : 35.7e6 # Pa

M_throat : 1.0
```

- ▶ PITOT2 had a function for everything, i.e. a secondary driver calculation function, shock tube calculation function etc.
- ▶ For PITOT3 I wanted to make a set of classes which would cover all that was necessary. These classes are:
 - ▶ **Facility:** Loads in the facility configuration file directly and then can return various inputs when required.
 - ▶ **Driver:** Loads in the driver configuration file directly, then can calculate the driver rupture state and throat state if required.
 - ▶ **Diaphragm:** In the ideal case, lets the gas pass. In other modes, can simulate non-ideal effects which occur at the diaphragm.
 - ▶ **Tube:** Simulates what occurs in a shock tube. Has a fill state and an entrance (driver) state and calculates the related shock speed.
 - ▶ **Nozzle:** Performs a steady expansion through a specified area ratio to simulate a facility's nozzle.
 - ▶ **Test_Section:** Calculates everything needed for the test section state as well as being able to do calculations for basic models.
 - ▶ **Facility_State:** This replaces the V and M dictionaries from PITOT. It is a gas state with a velocity attached, like the 'facility states' seen in x-t diagrams (hence the name).

- ▶ To make PITOT3 as generic as possible, the simulation is built up, object-by-object as it runs, forming a 'gaspath' like L1d does.
- ▶ For this to work, every physical class (Driver, Diaphragm, Tube, Nozzle, Test Section) has what is currently called an 'entrance_state' and an 'exit_state' (maybe I should have called them 'inlet' and 'outlet' states?) which are Facility_State objects of the state which enters and exits each objects.
- ▶ These classes also have related functions called 'get_entrance_state' and 'get_exit_state' which allow the next object along in the facility to grab the exit_state of the object before and set that to its own entrance_state.
- ▶ The particular objects created depend on the facility type and configuration which the user has selected i.e. an expansion tube has a driver, primary diaphragm, shock tube, secondary diaphragm, acceleration tube, maybe a nozzle, and a test section.
- ▶ A reflected shock tunnel has a driver, primary diaphragm, shock tube, secondary diaphragm, a nozzle, and a test section.

- ▶ How the code is structured now has more than halved the size of it compared to PITOT2:
- ▶ PITOT2:
 - ▶ pitot.py: 725 lines
 - ▶ pitot_input_utils.py: 1741 lines
 - ▶ pitot_flow_functions.py: 2678 lines
 - ▶ pitot_output_utils.py: 1383 lines
 - ▶ total : 6527 lines
- ▶ PITOT3:
 - ▶ pitot3.py: 594 lines
 - ▶ pitot3_classes.py: 2418 lines:
 - ▶ total : 3012 lines.

- ▶ I also thought that I should expand further on the most important classes.
- ▶ In the ideal case, the **Diaphragm** class does nothing. It just takes the `exit_state` of the object before it, sets that to its `entrance_state`, and then sets that `entrance_state` to be its `exit_state`.
- ▶ It can also calculate a reflected shock at the diaphragm, either a fully reflected shock or a reflected shock of a user specified Mach number.
- ▶ To simulate total pressure loss at the diaphragm, it can also reduce the velocity of the state exiting the diaphragm by a user specified factor.
- ▶ I also hope to one day implement an inertial diaphragm model here.
- ▶ It currently takes a Python3 dictionary as an input, as I coded it straight after the Facility and Driver classes, which are basically just loaded from files, but the Diaphragm class isn't really like that.

- ▶ The **Tube** class replaces a lot of the bulk of the original PITOT2 program as it takes the fill state and entrance state of a tube on a shock tube and then calculates the shock speed, and shocked and unsteadily expanded states.
- ▶ It also stores the length and diameter of the tube if that information was in the facility configuration file.
- ▶ If the tube is the shock tube of a reflected shock tunnel, it can also calculate the stagnated condition at the end of the shock tube and the stagnated driver gas condition.
- ▶ It can also be instructed to unsteadily expanded the unsteadily expanded entrance state to the shock speed or the shock speed or contact surface velocity multiplied by a factor.
- ▶ I plan to implement a Mirels solver as well.

- ▶ The **Test_Section** class mainly exists to calculate any post-shock states which can be calculated with basic gas dynamic relations.
- ▶ It can do a normal shock to simulate the stagnation limit over a test model.
- ▶ It can do a wedge shock for a given wedge angle and a Taylor-Maccoll conical shock for a given cone half angle.

- ▶ The **Facility_State** is a very important class.
- ▶ Like I said earlier, it replaces what was carried around for each facility state in PITOT2 in the states, V, and M dictionaries.
- ▶ Its main role is to store a gas state and its related velocity together.
- ▶ It can also store a reference gas state (usually a room temperature fill state of a related state) for enthalpy calculations.
- ▶ From there, it has enough information to calculate the pitot and total condition for the facility state which can be used to return the pitot pressure, total temperature, stagnation enthalpy and the flight equivalent velocity (and it has functions to return all of these things).
- ▶ It can also return the sensible enthalpy and the unit Reynolds number.

- ▶ At the moment, I have still written PITOT3 to be mainly backed by CEA's equilibrium gas model.
- ▶ I plan to try Nick Gibbons' new equilibrium gas model when I have some time.
- ▶ One large difference between use of CEA in the old Python2 cfpilib and through the D library now is that it can no longer run CEA without the list of species being specified.
- ▶ This isn't a big deal for most common cases, but for example, we used to be able to run PITOT2 with a small amount of aluminium, carbon, or iron etc. contamination to see what it would form at the different conditions seen in the facility, which we now can't do.
- ▶ While one can input a gas state into CEA in mole fractions, one can only currently return mass fractions, so PITOT3 currently only outputs mass fractions.
- ▶ Currently I haven't yet been able to create a perfect gas object on the fly for frozen shock calculations (like PITOT2's 'state 10f' to give the immediate post-shock condition) or perfect gas driver conditions.

- ▶ Currently in `dgdinst/share/pitot3_data/preset_gas_models` I have added a lot of common gas models which are used on X2.
- ▶ I filled out the species list for each gas from calculations without the species specified in PITOT2, looking at the species in the CEA calculations, and some intuition.
- ▶ The current gases are:
 - ▶ argon with ions ('ar-with-ions')
 - ▶ CO₂ with ions ('co2-with-ions')
 - ▶ giant planet with 20%H₂/80%He with ions ('giant-planet-h2-80-he')
 - ▶ giant planet with 20%H₂/80%Ne with ions ('giant-planet-h2-80-ne')
 - ▶ helium with ions ('he-with-ions')
 - ▶ 'bottle air' (N₂/O₂) with ions ('n2-o2-with-ions')
 - ▶ pure nitrogen ('n2-with-ions')
 - ▶ Uranus ('uranus')
- ▶ I have added Mars compositions (CO₂ and N₂) both with and without argon, but they don't work currently.
- ▶ As well as the `air5species`, `air7species`, `air11species`, and `air13species` CEA gas models from the repository.


- ▶ Currently in `dgdinst/share/pitot3_data/preset_gas_models` I have added a lot of common gas models which are used on X2.
- ▶ I filled out the species list for each gas from calculations without the species specified in PITOT2, looking at the species in the CEA calculations, and some intuition.
- ▶ The current gases are:
 - ▶ argon with ions ('ar-with-ions')
 - ▶ CO₂ with ions ('co2-with-ions')
 - ▶ giant planet with 20%H₂/80%He with ions ('giant-planet-h2-80-he')
 - ▶ giant planet with 20%H₂/80%Ne with ions ('giant-planet-h2-80-ne')
 - ▶ helium with ions ('he-with-ions')
 - ▶ 'bottle air' (N₂/O₂) with ions ('n2-o2-with-ions')
 - ▶ pure nitrogen ('n2-with-ions')
 - ▶ Uranus ('uranus')
- ▶ I have added Mars compositions (CO₂ and N₂) both with and without argon, but they don't work currently.
- ▶ As well as the `air5species`, `air7species`, `air11species`, and `air13species` CEA gas models from the repository.

- ▶ Custom test gases, which should easily give the user access to any of the gas models in the GDtK repository, can be set by setting the `test_gas_model` variable in the input to 'custom' and then setting the variable `test_gas_filename` to the name of the gas model file.

- ▶ There is still a lot to do, which includes:
 - ▶ Getting the condition builder working for performance maps.
 - ▶ Adding more different facilities and driver conditions.
 - ▶ Getting experiment mode working as I realised that it wasn't working yesterday.
 - ▶ Get the Mars and Venus test gases working eventually.
 - ▶ Get perfect gas driver conditions working.
 - ▶ Trying out Nick Gibbons' equilibrium solver which will hopefully allow us to get away from CEA for most cases.
 - ▶ Be able to output in mole fractions and be able to re-add simple calculation of frozen states.
 - ▶ Add the ability to specify a completely independent default configuration.
 - ▶ Change how the code is set up slightly so it can output all of the fill states at the start of the calculation like PITOT2 could. (Currently PITOT3 doesn't make any objects until it performs the calculations so this can't be done easily.)
 - ▶ Get simulations with no facility specified running again.
 - ▶ Add examples to the repository and write documentation.
- ▶ I'm sure I forgot many things which still need to be added!

- ▶ Is there anything which people are itching to do in PITOT3?
- ▶ If you are and I don't have it working yet, please do tell me and I will add it to the code ASAP! As I am trying to prioritise things which people will actually use for now.

- ▶ If there is anyone out there who would like to help me work on the code, that would be great. Please be in touch!
- ▶ Any feedback about the code is appreciated too, either in terms of what it does or how it is written.
- ▶ Thank you to one of my PhD students, Kieren Curtis, who did a lot of the early PITOT3 testing and Lewis Barltrop an undergraduate student of mine who has currently been messing around with it too.



**“ MY FELLOW CITIZENS,
ASK NOT WHAT YOUR PITOT3 CAN DO FOR YOU,
ASK WHAT YOU CAN DO FOR YOUR PITOT3. ”**
- JOHN F. KENNEDY

MyTimelineCoverz.com

Figure: A little motivational poster.

Any questions?

- ▶ Thankyou for listening! Any questions?



Figure: Sunflowers by Vincent van Gogh from the collection of The National Gallery in London.

- [1] Jacobs, P., Gollan, R., Potter, D., Zander, F., Gildfind, D., Blyton, P., Chan, W., and Doherty, L., *Estimation of high-enthalpy flow conditions for simple shock and expansion processes using the ESTCj program and library. Mechanical Engineering Report 2011/02*, Department of Mechanical Engineering, University of Queensland, Australia, 2011.
- [2] Fahy, E., *Superorbital Re-entry Shock Layers: Flight and Laboratory Comparisons*, Ph.D. thesis, the University of Queensland, St. Lucia, Australia, 2017.
- [3] James, C., Gildfind, D., Lewis, S., Morgan, R., and Zander, F., “Implementation of a state-to-state analytical framework for the calculation of expansion tube flow properties,” *Shock Waves*, Vol. 28, No. 2, 2018, pp. 349–377.