

Hobyah User Guide



May 2025, Guide v1.3

Contents

1 Overview	17
1.1 Software licence	17
1.2 Introduction	18
1.3 Other similar software	19
1.4 Program name and pronunciation	20
1.5 Calculation method	20
1.6 Who is the program for?	21
1.7 Who is the program not for?	22
1.8 Wot no GUI?	23
1.9 Hobyah and SES	24
1.10 Hobyah as a “dumb plotter”	25
1.11 Installing and using	26
1.11.1 Introduction	26
1.11.2 Installing Gnuplot	27
1.11.3 Using Hobyah and SESconv without installing Python . . .	31
1.11.4 Installing Python on Windows	31
1.11.5 Installing Numpy, Pandas and SciPy	33
1.12 The calculation process	34
1.13 A simple tunnel example	37
1.14 Sectypes in Tunnels	38
1.15 Segments	39

1.16 Input file formats	42
1.17 Mandatory settings	43
1.18 Portals	44
1.19 Nodes	44
1.20 Joins	45
1.21 Dampers	45
1.22 Fans	46
1.23 Routes	46
1.24 Jet fans	46
1.25 Converting Hobyah input into SES input	47
1.26 Road traffic	48
1.27 Traffic use example	49
1.28 Verification and validation	52
1.29 Glossary	52
1.30 Concluding remarks	57
2 Into the Detail	59
2.1 Blocks of input	59
2.2 Sample files	60
2.3 Conventions	62
2.3.1 Chainages, back end and forward end	62
2.3.2 Up and down	62
2.4 Rules for processing input files	63
2.5 How lines are read	65
2.6 How blocks are read	66
2.7 Text editors	67
2.8 Required software	67
2.9 Pressure loss factors	69
2.10 Constants and units conversion	72

CONTENTS	5
2.11 Error messages	73
3 Input Specification	75
3.1 Introduction	75
3.2 Comments at the top of the file	75
3.3 The <code>settings</code> block	76
3.3.1 <code>Version</code> setting	77
3.3.2 <code>runtype</code> setting	77
3.3.3 <code>frictiontype</code> setting	78
3.3.4 <code>frictionapprox</code> setting	79
3.3.5 <code>units</code> setting	80
3.3.6 <code>QA</code> settings	80
3.3.7 <code>P_atm</code> setting	81
3.3.8 <code>rho_atm</code> setting	81
3.3.9 <code>aero_step</code> setting	82
3.3.10 <code>aero_time</code> setting	82
3.3.11 <code>footer</code> setting	82
3.3.12 <code>header</code> setting	83
3.3.13 <code>plotnames</code> setting	83
3.3.14 <code>images</code> setting	84
3.3.15 <code>keytexts</code> setting	84
3.3.16 <code>autokeys</code> setting	84
3.3.17 <code>solver</code> setting	85
3.3.18 <code>gamma</code> setting	85
3.3.19 <code>rise_time</code> setting	86
3.3.20 <code>max_vel</code> setting	87
3.3.21 <code>time_accuracy</code> setting	87
3.4 Optional entries	87
3.4.1 Basics of optional entries	87

3.4.2	How to see all valid optional entries	88
3.5	The <code>constants</code> block	89
3.6	Sentences	90
3.7	Phrases	91
3.7.1	Lists	91
3.7.2	Ranges (list generators)	93
3.7.3	Combining ranges and lists	95
3.8	Defining fans	96
3.9	The <code>files</code> block	96
3.10	The <code>csv</code> block	98
3.11	The <code>data</code> block	99
3.12	The <code>plotcontrol</code> block	101
3.13	The <code>sectypes</code> blocks	102
3.14	The <code>tunnel</code> block	103
3.14.1	Features at tunnel ends	104
3.14.2	Features within tunnels	104
3.14.3	The <code>back</code> keyword	105
3.14.4	The <code>fwd</code> keyword	108
3.14.5	The <code>change</code> keyword	108
3.14.6	The <code>loss1</code> keyword	110
3.14.7	The <code>loss2</code> keyword	110
3.14.8	The <code>damper1</code> keyword	110
3.14.9	The <code>damper2</code> keyword	112
3.14.10	The <code>join</code> keyword	112
3.14.11	The <code>joins</code> keyword	113
3.14.12	Fans in tunnels	114
3.14.13	Jet fans in tunnels	114
3.14.14	SES pragmats	115
3.15	The <code>tunnelclones</code> block	116

3.15.1	Introduction	116
3.15.2	The numbering entry	116
3.15.3	The sectypes entry	117
3.15.4	Inner workings	117
3.16	The route block	118
3.16.1	Introduction	118
3.16.2	The origin keyword	118
3.16.3	The portal keyword	118
3.16.4	The tunnels sub-block	119
3.16.5	The gradients sub-block	120
3.16.6	The elevations sub-block	121
3.16.7	The lanes sub-block	122
3.16.8	The speedlimits sub-block	123
3.16.9	The schedule sub-block	124
3.16.10	The radii sub-block	127
3.16.11	The sectors sub-block	127
3.16.12	The coasting sub-block	128
3.16.13	The regenfraction sub-block	129
3.17	Fans	129
3.17.1	Introduction	130
3.17.2	The fanchar block	130
3.17.3	The fan1 keyword	133
3.17.4	The fan2 keyword	134
3.18	Jet fans	135
3.18.1	Introduction	135
3.18.2	The jetfantypes block	135
3.18.3	The jetfans1 keyword	137
3.19	Traffic types	137
3.19.1	The calculate keyword	138

3.19.2 The <code>vehicle</code> keyword	138
3.19.3 The <code>speedgradpairs</code> keyword	139
3.19.4 Traffic pollution	139
3.20 Putting traffic into routes	140
3.20.1 The <code>routes</code> keyword	140
3.20.2 The <code>moving</code> keyword	141
3.20.3 The <code>standstill</code> keyword	141
3.20.4 Setting traffic flows	142
3.21 The <code>SESdata</code> block	144
3.21.1 The <code>target</code> keyword	145
3.21.2 The <code>filename</code> keyword	146
3.21.3 The <code>form1B</code> keyword	146
3.21.4 The <code>form1F</code> keyword	146
3.21.5 The <code>form1G</code> keyword	147
3.21.6 The <code>3temperatures</code> keyword	147
3.21.7 The <code>form3F</code> keyword	148
3.21.8 The <code>4A_locn</code> keyword	148
3.21.9 The <code>4B_heat</code> keyword	148
3.21.10 The <code>4C_flames</code> keyword	149
3.21.11 The <code>form7AB</code> keyword	149
3.21.12 The <code>form7C1</code> keyword	149
3.21.13 The <code>form7C2</code> keyword	150
3.21.14 The <code>form8</code> keyword	151
3.21.15 Other SES forms	152
3.22 The <code>plots</code> block	155
3.22.1 The plot settings	155
3.23 The <code>page</code> block	159
3.23.1 The <code>pageunits</code> setting	159
3.23.2 Example <code>page</code> block	159

3.24 The <code>graph</code> block	161
3.24.1 Setting the graph labels	161
3.24.2 Controlling the limits and intervals of axes	163
3.24.3 Setting the size and location of graphs	164
3.24.4 The <code>begin verbatim...end verbatim</code> block	165
3.24.5 The <code>verbatim</code> command	167
3.25 Curve definitions	167
3.25.1 <code>Userdata</code> curves	169
3.25.2 <code>Transient</code> curves	169
3.25.3 <code>Profile</code> curves	171
3.25.4 <code>Fadata</code> curves	171
3.25.5 <code>Icons</code> curves	172
3.25.6 Curve optional entries	176
3.26 The <code>image</code> block	179
3.26.1 Location and sizing	179
3.26.2 The <code>width</code> and <code>height</code> keywords	180
3.27 The <code>timeloop</code> block	181
3.28 The <code>filesloop</code> block	182
3.29 Behind the scenes in the <code>plot</code> block	184
3.29.1 Introduction	184
3.29.2 Looking inside <code>.plt</code> files	185
3.30 Curve data files	185
4 Interpreting Hobyah output	187
4.1 Printed output from the airflow calculation	187
4.2 Printed output for traffic	188
5 classSES.py	191
5.1 Using it in Python sessions	191
5.2 <code>Describe()</code>	192

5.3	PrettyClassPrint()	192
5.4	WriteInputFile()	194
6	Plot properties	195
6.1	Plotting	195
6.2	Plots of Hobyah transient properties in tunnels	195
6.2.1	The <code>velocity</code> and <code>-velocity</code> plot types	197
6.2.2	The <code>celerity</code> plot type	198
6.2.3	The <code>density</code> plot type	199
6.2.4	The <code>pdiff</code> plot type	200
6.2.5	The <code>pstat</code> and <code>-pstat</code> plot types	201
6.2.6	The <code>ptot</code> and <code>-ptot</code> plot types	202
6.2.7	The <code>pstatabs</code> plot type	203
6.2.8	The <code>ptotabs</code> plot type	204
6.2.9	The <code>volflow</code> and <code>-volflow</code> plot types	205
6.2.10	The <code>massflow</code> and <code>-massflow</code> plot types	206
6.3	Plots of fixed properties along routes/tunnels	207
6.3.1	The <code>elevations</code> plot type	209
6.3.2	The <code>stacks</code> plot type	210
6.3.3	The <code>gradients</code> plot type	211
6.3.4	The <code>stackgrads</code> plot type	212
6.3.5	The <code>speedlimits</code> plot type	213
6.3.6	The <code>lanes</code> plot type	214
6.3.7	The <code>radius</code> plot type	215
6.3.8	The <code>sectors</code> plot type	216
6.3.9	The <code>coasting</code> plot type	217
6.3.10	The <code>svsregen2</code> plot type	218
6.3.11	The <code>sections</code> plot type	219
6.3.12	The <code>segments</code> plot type	220

6.3.13	The subsegs plot type	221
6.3.14	The sublength plot type	222
6.3.15	The fireseg plot type	223
6.3.16	The area plot type	224
6.3.17	The perimeter plot type	225
6.3.18	The roughness plot type	226
6.3.19	The Darcy plot type	228
6.3.20	The Fanning plot type	229
6.3.21	The wetted plot type	230
6.4	Plots of Hobyah fan properties	231
6.4.1	The fanchar plot type	232
6.4.2	The system plot type	233
6.4.3	The fanchar-cursed plot type	234
6.4.4	The system-cursed plot type	235
6.5	Plots of Hobyah damper properties	236
6.5.1	The damper area_d plot type	236
6.5.2	The damper zeta_bf k-factor plot type	237
6.5.3	The damper zeta_fb k-factor plot type	238
6.5.4	The damper R_bf resistance plot type	239
6.5.5	The damper R_fb resistance plot type	240
6.6	Plots of icons	240
6.6.1	Trains , fires and jetfans icons	240
6.7	Plots of Hobyah traffic properties	242
6.7.1	The < trafficname >_flow plot type	243
6.7.2	The < trafficname >_dens plot type	244
6.8	Plots of SES transient properties in tunnels	245
6.8.1	The qSES and -qSES plot types	249
6.8.2	The qcold and -qcold plot types	250
6.8.3	The qtrains plot type	251

6.8.4	The qwarm and -qwarm plot types	252
6.8.5	The vSES and -vSES plot types	253
6.8.6	The vcold and -vcold plot types	254
6.8.7	The vwarm and -vwarm plot types	255
6.8.8	An SES volume flow/air velocity recap	255
6.8.9	The dp and -dp plot types	258
6.8.10	The dp_indiv plot type	259
6.8.11	The DB plot type	260
6.8.12	The wall_temp plot type	262
6.8.13	The W plot type	263
6.8.14	The sens and sens_pul plot types	264
6.8.15	The lat and lat_pul plot types	266
6.8.16	The annulus plot type	267
6.9	Plots of SES train properties	268
6.9.1	The accel plot type	269
6.9.2	The speed plot type	270
6.9.3	The route plot type	271
6.9.4	The type plot type	272
6.9.5	The chainage and time plot types	273
6.9.6	The aerodrag plot type	274
6.9.7	The cd plot type	275
6.9.8	The TE plot type	276
6.9.9	The motoramps plot type	277
6.9.10	The lineamps plot type	278
6.9.11	The flywh_rpm plot type	279
6.9.12	The acceltemp plot type	280
6.9.13	The deceltemp plot type	281
6.9.14	The pwr_all plot type	282
6.9.15	The heat2air plot type	283

6.9.16	The <code>mode</code> plot type	284
6.9.17	The <code>pwr_aux</code> plot type	285
6.9.18	The <code>pwr_prop</code> plot type	286
6.9.19	The <code>pwr_regen</code> plot type	287
6.9.20	The <code>pwr_flywh</code> plot type	288
6.9.21	The <code>pwr_accel</code> plot type	289
6.9.22	The <code>pwr_decel</code> plot type	290
6.9.23	The <code>pwr_mech</code> plot type	291
6.9.24	The <code>heat_AM</code> plot type	292
6.9.25	The <code>heat_sens</code> plot type	293
6.9.26	The <code>heat_lat</code> plot type	294
6.9.27	The <code>effic1</code> plot type	295
6.9.28	The <code>SVSeffic</code> plot type	297
6.9.29	The <code>SVSregen1</code> plot type	298
6.10	Plots of SES Environmental Control Zone data	299
6.10.1	The <code>meanDB_AM</code> plot type	299
6.10.2	The <code>meanDB_PM</code> plot type	300
6.10.3	The <code>meanDB_off</code> plot type	301
6.10.4	The <code>meanW_AM</code> plot type	302
6.10.5	The <code>meanW_PM</code> plot type	303
6.10.6	The <code>meanW_off</code> plot type	304
6.10.7	The <code>wallT_used</code> plot type	305
6.10.8	The <code>wallT_AM</code> plot type	306
6.10.9	The <code>wallT_PM</code> plot type	307
6.10.10	The <code>misc_sens</code> plot type	308
6.10.11	The <code>misc_lat</code> plot type	309
6.10.12	The <code>steady_sens</code> plot type	310
6.10.13	The <code>steady_lat</code> plot type	311
6.10.14	The <code>ground_sens</code> plot type	312

6.10.15 The <code>airex_sens</code> plot type	313
6.10.16 The <code>airex_lat</code> plot type	314
6.10.17 The <code>HVAC_sens</code> plot type	315
6.10.18 The <code>HVAC_lat</code> plot type	316
6.10.19 The <code>HVAC_total</code> plot type	317
6.11 Plots of SVS properties	318
6.11.1 The <code>pipetemp</code> plot type	318
6.11.2 The <code>pipe_ht</code> plot type	319
7 Miscellany	321
7.1 What's a Hobyah?	321
7.2 Minimum cell size in characteristic calculations	322
7.3 Convergence problems	324
7.4 How to submit bug reports	324
7.5 Stability of calculations	325
7.6 Limits on use cases	325
7.6.1 Variable properties	325
7.6.2 Explicit terms in jet fan thrust and traffic drag	326
7.6.3 Micro-pressure waves	328
7.7 SES train numbering	328
7.8 Limits	329
7.8.1 Maximum number of air paths at junctions	329
7.8.2 Limits on the size of numbers	330
7.8.3 Limits on time accuracy	331
7.8.4 Deciding when to merge boundaries	331
7.8.5 Why are route profiles/gradients extended?	331
7.9 A Gnuplot problem with image files	332
7.10 Gnuplot number limits	333
7.11 Customising the Gnuplot terminal	333

CONTENTS	15
7.12 Roughness heights	334
7.13 Why show three types of friction factor?	335
7.14 A <code>damper2</code> problem	335
7.15 Ignored input in the <code>plots</code> block	335
7.16 Command line options	336
7.16.1 Command line option <code>-debug1</code>	336
7.16.2 Command line option <code>-nocalc</code>	337
7.16.3 Command line option <code>-nofortran</code>	337
7.16.4 Command line option <code>-serial</code>	337
7.16.5 Command line option <code>-showerrors</code>	338
7.16.6 Command line option <code>-dudbins</code>	338
A Using the Fortran routines	339
A.1 Compiling the Fortran	339
A.2 Placing the executable	341
B References	345
C Error message transcripts	347
Index	455

Chapter 1

Overview

1.1 Software licence

This software is released under the BSD 2-clause licence (SPDX identifier: BSD-2-Clause).

Copyright (c) 2020–2025, Ewan Bennett (ewanbennett@fastmail.com).

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2 Introduction

Hobyah is an open source program for calculating airflow in tunnels by the method of characteristics (MoC). It plots output from its calculations.

It also plots output from Subway Environment Simulation (SES¹) calculations and can be told to generate the basis of an SES input file from the tunnel geometry defined in Hobyah.

Most early users use it for its ability to quickly build the skeletons of SES input files and to plot output from SES files rather than running calculations in Hobyah directly.

This third issue of the program's code, test files and documentation takes the feedback I have had from users of the second issue of the software (October 2024) and tries to address the things they told me that they liked and disliked. This third version also supports processing SVS² output files and plotting their results in the same way that SES output files are plotted.

My main aims in writing the software were:

1. to make it easier for me and my colleagues to design tunnel ventilation systems.
2. to write an open source preprocessor and postprocessor for the SES program.
3. to write an open source method of characteristics program for the tunnel ventilation profession. MoC is perfect for certain types of tunnel ventilation calculation but has (as far as I'm aware) only ever been written as closed-source software.
4. to document the mathematics that underpin the method of characteristics in detail. Even if this program ends up sinking without a trace, that knowledge is now out there and available to luckier programmers.
5. to make it easier to catch mistakes in tunnel ventilation designs by writing good error messages.

The program is aimed at road and rail tunnels, because those are what I have spent most of my career working on. The first release (2024) calculates airflow only. The program, test files, verification/validation files and documentation can be found at <https://github.com/ECB2020/Hobyah>.

The program can handle input and output in SI and US customary units, mostly because I had to write an SI-US converter to handle SES output.

¹SES is a tunnel ventilation program funded by the US Department of Transportation (US DoT) in the 1970s and is still much used in the industry despite its age.

²SVS (Subway Ventilation System) is a fork of SES written by one of the developers of SES (Parsons Brinckerhoff, now part of WSP) with additional features.

The internal calculations and the binary output files are in SI units, but you can define an input file in either system of units and tell it to plot output in SI units US units or a mix of the two.

The processing and plotting of SES files is included for two reasons:

- I've now written an SES converter and plotter while not under contract to an employer. This means I am free to release them under an open source licence for all engineers to use (I expect to be using SES until I retire).
- It adds the ability to plot Hobyah calculations alongside the output of SES. This makes it easier to cross one of the biggest barriers to the use of new software in any engineering field: the question "what has it been validated against?" from a category 3 checker, independent verifier or railway inspector.

The remainder of this section is an abbreviated description of the program and some advice for engineers who might be considering using the software in fee-earning work as consulting engineers.

1.3 Other similar software

Tunnel ventilation is a specialised field. Many consulting engineering firms have written tunnel ventilation software that is used in-house, though the programs may be little-known outside the firm apart from a few technical papers in tunnel ventilation conference proceedings or in engineering journals.

I know of three tunnel ventilation design programs for which licences can be purchased:

- Thermotun (Dundee Tunnel Research, Scotland).
- Camatt (centre d'Études des Tunnels, France).
- IDA (Equa, Sweden).

There are also three publicly-available versions of the US DoT's SES program:

- SES versions 1 to 4.1, given out freely by the US DoT from 1975 until about 2008 (a scan of the complete source code listing of SES v1 is available online in the US government's National Technical Reports Library).
- OpenSES (<https://github.com/Open-SES/OpenSES>), which is a set of open-source files built on top of the SES v4.1 source code (the SES v4.1 source is still owned by the US government). Disclaimer: I have contributed code to the OpenSES project.

- SVS v6.6.2, an in-house extension written by Parsons Brinckerhoff (now WSP), one of the firms who developed SES for the US DoT. They were given permission by US DoT to release SVS in executable form in late 2019.

There are others. Many firms have their own in-house codes or variants of SES 4.1; I wrote a large number of extensions for one former employer (called **Aurecon SES**). I also have a private variant of SES 4.1 that I use as a testbed (called **offline-SES**). It is occasionally mentioned in this manual and in the verification documentation.

Many other tunnel ventilation programs have been written in connection with final year university degrees or PhDs (such as TunnelTrain and NUMSTA) but these are either not commercially available or are used in limited circumstances.

Finally, there are some codes have been described in detail in the literature but appear to have vanished off the face of the earth, such as the non-homentropic method of characteristics developed by the University of Liverpool in conjunction with British Rail (Woods & Gawthorpe 1972, Woods & Pope 1980).

1.4 Program name and pronunciation

Hob-yah. Pronounce “hob” as in “hobby” and “yah” as in “yak”. The stress should be on the first syllable: HOB-yah rather than hob-YAH. Think of how Brits stress the first syllable of the word “python” (PY-thon) compared to how New Yorkers stress the second syllable (py-THON) and channel your inner Brit.

To find out what a Hobyah is, see Section 7.1. The “7.1” on this line is a clickable link that should work in most pdf readers.

1.5 Calculation method

Hobyah solves one-dimensional, compressible flow with friction and other body forces by the method of characteristics with two variables (speed of sound and velocity). It is suitable for flows of low Mach number M (let’s say $M < 0.3$, so train speeds and airspeeds less than 350 km/h or 220 mph).

This type of flow (termed “homentropic flow” in the engineering literature) is compressible and isentropic. Homentropic flow has irreversible losses (friction and discrete pressure losses) but it has no concept of temperature or heat transfer, so the energy dissipated by friction and by discrete pressure losses is removed from the system rather than being turned into a temperature rise at gridpoints. As a result, Hobyah is unsuitable for fire modelling and calculations that involve heat transfer.

On the plus side, the homentropic method of characteristics is a fast and stable way of calculating airflow at speeds under approximately 100 m/s. It

handles—with aplomb—fans being pushed into stall by other prime movers, gravity-powered nonreturn flap dampers being slammed shut by rapid pressure changes and powered dampers slamming shut in short periods of time. The homentropic method of characteristics was originally developed for waterhammer calculations (which it is perfect for) and gradually made its way into the tunnel ventilation field in the 1960s and 1970s.

The theory that underpins the solver and the choices I made when implementing it are given in the file `MoC.pdf` in the `Hobyah Documentation` folder. If you take the time to read that file and think you've found a mistake in it, please let me know.

1.6 Who is the program for?

Anyone who writes software usually has a set of intended users in mind.

Some programs are designed to be used by as many people as possible (e.g. Excel) and the programmers spend a lot of effort making the Graphical User Interface (GUI) as intuitive as possible.

At the other end of the spectrum are programs like OpenFoam, an extremely capable, open source CFD solver. Users of OpenFoam must understand the physics of what they want to model, know how to use a text editor and know how to use a command-line interface before they can do any useful work with OpenFoam.

The writers of OpenFoam make no secret that their software is intended for people who are willing and able to spend time and effort climbing OpenFoam's steep learning curve.

I hope that Hobyah's approach is two-thirds OpenFoam, one-third Excel. Hobyah does not tell you everything you need to know to use the software properly. I wrote Hobyah and its documentation for me, for engineers with similar experience to me and for the engineers I expect to train to use it.

Prospective users who understand a lot about tunnel or mine ventilation and whose only experience is point-and-click in the GUIs of programs like IDA or Ventsim will struggle at first. Some may never get the knack of using it.

Most of this manual is couched in terms that engineers not familiar with how things are done in the tunnel ventilation field may know nothing of.

In this manual, ways of doing things in Hobyah are compared to how they are done in SES, Aero and other proprietary tunnel ventilation software. Road traffic calculations conform to the recommendations in PIARC's documents. If you get lost as you read through this manual because I am not explaining the specialised stuff adequately, it may just mean that you aren't yet in the program's target audience. Which—just to reiterate—is people who know the specialised stuff already or who are being trained by someone who knows the

specialised stuff.

On a less doom-and-gloom note: mine ventilation engineers may also find Hobyah useful. If you are a mine ventilation engineer, you may find the method of characteristics in this software useful if you are trying to model airflow in the aftermath of mine roof collapses. The tunnel ventilation software “as is” will probably not be of much use to you when modelling mine ventilation systems, as it ignores heat transfer and humidity and doesn’t have a GUI.

But the theoretical background documents might be, and I can see how to adapt some of the routines into something that the mine ventilation field might find useful. The program accepts SI input in terms of Atkinson resistance and Atkinson friction factor, to make the learning curve a bit easier to climb.

1.7 Who is the program not for?

Hobyah is not intended as a way for beginners to get into tunnel ventilation design work.

If you are an engineer looking to learn how to design tunnel ventilation systems solely by reading this manual: I strongly advise you not to, for the following reasons:

- Firstly, your career designing tunnel ventilation systems is likely to be nasty, brutish and short unless you are extremely lucky, extremely smart and the company you work for has the kind of Professional Indemnity insurance that a rich, indulgent grandparent would buy for their adorable infant grandchild (if the adorable infant grandchild was somehow also a consulting engineer).
- Secondly, there are many tidbits of knowledge about tunnel ventilation that only come from experience or by being taught by a senior colleague. Few of those are given in my documentation, as there are far too many of them.
- Thirdly, if you make one mistake, your contractor client may well take your company to the cleaners the moment you stuff up a design.

Far better to go and work for one of the many consulting engineering firms who have experienced tunnel ventilation engineers and learn from them.

If you ignore the advice in this section and try to design a project based on the contents of this manual and end up getting sued by your client, don’t get in touch with me in the hope that I’ll swoop in and bail you out. I will remind you of the disclaimers in Section 1.1, send you and the people suing you the text in this section, then stop responding to you.

1.8 Wot no GUI?

There is no Graphical User Interface (GUI) in Hobyah. Hobyah input files are plain text files with a block structure. Quite a few people I've shown Hobyah's user interface to don't like that the program has no GUI. Others were OK with it.

I'm happy to not have a GUI, and a lot of colleagues I respect share that sentiment. I have often seen in my career that when a GUI for an engineering program is available, competent engineers with tight deadlines tend to not use the GUI if they can use scripts, spreadsheets or text editors to build their input files instead.

In recent detail designs I have spent weeks coding scripts to generate hundreds of input files that would have taken me months of clicking to generate through a GUI.

This "let's use scripts instead of the GUI" is most noticeable during tender designs for tunnels with programmes³ that put the designers under extreme time pressure. We use text editors, spreadsheets and scripting languages to generate our input files instead, because those methods are so much more flexible than point and click in a GUI.

Also I doubt I would make a good job of writing a GUI; my programming experience isn't suited to writing GUIs. So Hobyah is a straight-up "input files are plain text files" program.

The quickest way to get started is to write the files in a text editor, but there is nothing to prevent engineers writing scripts or spreadsheets to automate the process of generating the text files. I write scripts all the time when I need dozens of test input files that differ slightly.

While I don't feel a need to write programs to let me set input in a GUI, I am 100% in favour of good, automated graphical output.

Good graphical output helps us all catch mistakes before a project's third party checkers or independent verifiers point out a mistake that we've made but failed to catch during checking. Good graphical output can also be used to explain complicated concepts to non-specialists. For example:

- Have you set up a run with a bunch of trains moving through the system? Plotting train icons in Hobyah output lets you check whether trains are stopping at your stations (right) or stopping in the middle of your tunnels (wrong).
- Do you want to explain to someone how a fan behaves when an outside influence forces the fan into the reverse flow region or freewheeling region of the fan characteristic? Run the example file `ok-031-fans-in-series.txt` and flip through its output file, `ok-031-fans-in-series-1p1.pdf`.

³"Schedules" for those of you in North America.

- Do you need to explain to a non-technical person how axial fans with terrible performance characteristics can be trapped in the stall region when run in parallel? Run the example file `ok-032-fans-in-parallel.txt` and flip through the .pdf file it creates, `ok-032-fans-in-parallel-lp1.pdf`.
- Want to check the operation of the jet fans when you try to pressurise a non-incident tunnel? Plot the jet fan icons and the pressure profiles.

One final point, intended for anyone thinking of forking Hobyah, improving it by writing a GUI/making it calculate non-homentropic flow/<insert your proposed improvement here> but are concerned that I might be annoyed by you stepping on my toes. Y'all are welcome to fork the code. You can improve it in whatever way you want, then try to sell your fork or publish it under an open source licence.

As long as you meet the conditions of the BSD 2-clause licence that I released my code under, I won't object. I would love to see people fork the program and add different solvers, for example.

1.9 Hobyah and SES

The programs in the Hobyah suite can

- Generate binary files (.sbn files) that have the SES input data and runtime data in SI units from SES and SVS output,
- Process SES output (v4.1 and OpenSES) into SI units,
- Plot graphs of SES properties from .sbn files
- Generate new SES input files from .sbn files
- Generate skeleton SES input files from the geometry and routes defined for Hobyah calculations.

The Hobyah suite has methods that allow it to plot output from SES. This is mostly because I expect to be using SES in one form or another for the rest of my career, so it is useful to have an open source code that I can use to process SES input and output instead of writing a new SES processor whenever I switch employers.

There are three Python scripts for handling SES files.

The first is `SESconv.py`, which converts SES output files (.PRN files, .TMP files and .OUT files) to SI units, calculates annulus air velocities and density corrections and creates a binary file (a .sbn file) that Hobyah can plot from. `SESconv.py` can process output from SES v4.1, SVS, OpenSES and a private version of SES that I develop for my own purposes (called `offline-SES`).

The second script for handling SES files is `classSES.py`, which reads the `.sbn` files that `SESconv.py` writes. The class has methods that return transient and profile data for plotting and other methods that return details of the locations and state of trains, jet fans and fires at instants in time. It can also be told to generate SES input files (this is useful when writing Python scripts that generate multiple SES input files from one “known to be good” SES output file).

The third is `Hobyah.py`, which can generate an SES input file from Hobyah tunnel and route definitions. These methods generate an SES input file that has section definitions in forms 2A and 2B, segment definitions in forms 3 and 5, route definitions in form 8 (and a really inefficient train type in form 9).

One of the best things from my perspective as a long-time SES user is that the gradients in Hobyah’s route definitions can be used to set the stack heights of SES segments as well as the gradients in routes in SES input files. See Sections 1.25 and 3.21 of this manual.

1.10 Hobyah as a “dumb plotter”

Hobyah can be used as a dumb plotter. “Dumb plotter” Hobyah input files do not carry out any calculations, they just plot data from internal `data` blocks, external `.csv` files, Hobyah `.hbn` files and SES `.sbn` files.

I mostly use the dumb plotter function to validate data that I’ve digitized from technical papers that have full-scale test results. Figure 1.1 shows a scan of one of the classic test cases in the tunnel ventilation field: the change in static pressure 100 m from the entry portal of the Patchway tunnel as a train enters (Figure 3c in Gawthorpe & Pope 1976).

The upper image in Figure 1.1 shows a scan of the image from the paper. The lower image shows the same image with a graph of the digitized data overlaid on top of it. Gnuplot has such fine control over its graph placement that the values on the X and Y axes can be lined up and the correctness of the digitization checked.

While your attention is on Figure 1.1, I’d like to draw your attention to the two lines of small text below the lower graph. These are quality assurance (QA) data for the Figure. The QA data gives the names of the files that hold curve data, names of images, the name of the file that created the Figure, the date & time it was created, and the username of the engineer who created it. There is also room for a project number and a project description, which may be useful in project work. See Sections 3.3.11 & 3.3.12 for details of how to control/hide these lines.

I also used the dumb plotter function to generate Figures for this manual: Figure 2.1 is one such. All the data plotted on that Figure is contained in its Hobyah input file. Another example can be found in the input file used to generate Figure 1.1 (it is called `docfig-BHR-1976-G+P-3c.txt`).

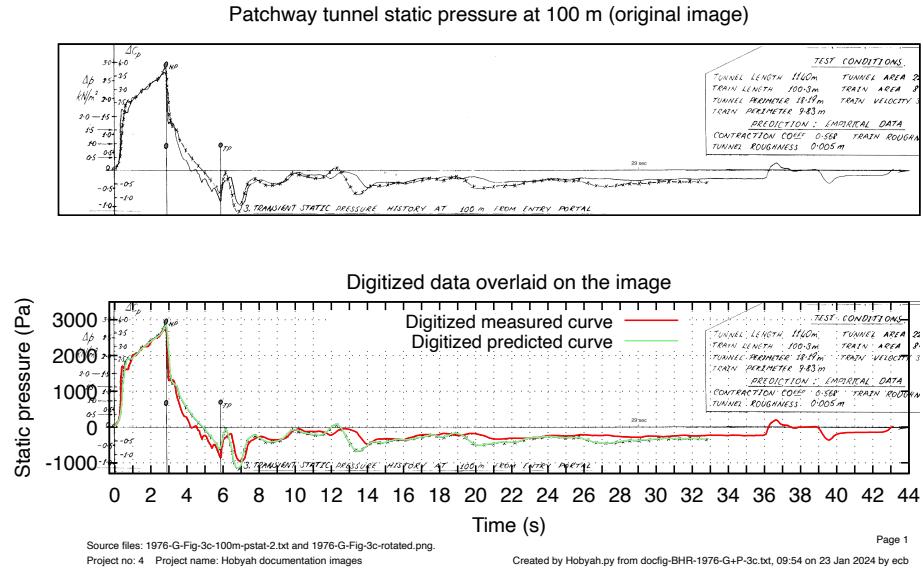


Figure 1.1: Dumb plotting to validate digitized data

1.11 Installing and using

1.11.1 Introduction

There are two ways of using Hobyah and SESconv:

- Install Gnuplot, Python, Numpy, Pandas and SciPy on your computer. The scripts `Hobyah.py` and `SESconv.py` should then run using the Python installed on your computer. See Sections 1.11.2, 1.11.4 and 1.11.5.
- Install Gnuplot on your computer. Use the files `Hobyah.exe` and `SESconv.exe` (this option is available on Windows only). This is recommended for those with little experience of installing Python packages and no access to a Python expert. See Sections 1.11.2 and 1.11.3.

Option 1 is the preferred option if you have some experience of Python, as it is the most flexible.

Option 2 is better if you have limited experience of Python or just want to try the software out.

1.11.2 Installing Gnuplot

This section has guidance on installing the Gnuplot plotting program on Windows and macOS.

Installing Gnuplot on Windows

A few engineering firms may have an approved version of Gnuplot in their software repository. If that's the case, get it from there. If not, ask your IT department to install it for you. Point them to this section so that when they install it, they tell the installer to update the Windows PATH environment variable (don't worry if that means nothing to you, PATH should be a familiar term to IT folks).

If you don't have an IT department, go to <https://sourceforge.net/projects/gnuplot/> and click on the large "Download" button. Ignore all the windows that pop up—sourceforge used to be a safe place to store software, but is now awash with scammers urging you to download malware.

After a few minutes, a file starting with `gp` and ending in `.exe` (on Windows) will have been downloaded into your "Downloads" folder.⁴

At the time of writing, the "Download" button offers `gp601-win64-mingw.exe` on Windows, which is Gnuplot version 6.0.1 (the current stable version of the software). The file you download may have higher numbers.

Also, if you have turned off Windows filename extensions (this is not recommended) the file will be just named `gp601-win64-mingw` as the `.exe` will be hidden.

Double-click on the downloaded file. Windows' User Account Control will ask you "Do you want to allow this app from an unknown publisher to make changes to your device?" At this point, you need to make a decision that only you can make. If you trust the gnuplot project to offer software free of malware, click yes. If you don't trust the gnuplot project, click no.

For what it is worth, I have been using gnuplot since about 2008 and have had no problems downloading gnuplot from Sourceforge.

If you click yes, the Gnuplot installer will then run, asking you to respond to a series of questions as follows:

1. Select setup language (default is English).
2. Read the licence agreement and accept its terms.
3. Read some important information about the program.
4. Set the folder to install gnuplot into (default is `C:/Program Files/gnuplot`).
I recommend that you use the default.

⁴If you have changed the folder that your browser saves downloads to, the executable will be in whichever folder you changed it to.

5. Select which components to install. All except “Japanese language support” are installed by default. I recommend that you use the default.
6. Set the Start Menu folder to install gnuplot into (default is `gnuplot`).
7. Set the Additional Tasks you want the installer to perform. *This is the one dialog box that you must change something in!*

An example of the top entries in the dialog box are given in Figure 1.2.

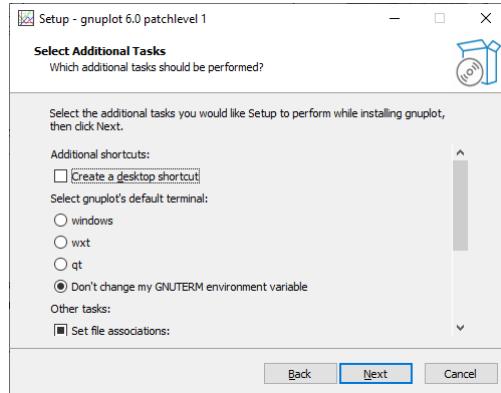


Figure 1.2: Gnuplot installer dialog box example 1

The default you need to change is at the end of the entries in the dialog box. Pull the scroll bar at the right-hand side of the dialog box to the bottom to show the last few entries in the dialog box (see Figure 1.3).

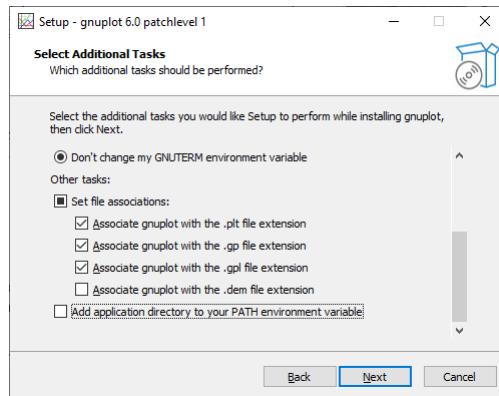


Figure 1.3: Gnuplot installer dialog box example 2

Put a tick in the tick box next to “Add application directory to your PATH environment variable”, as per Figure 1.4.

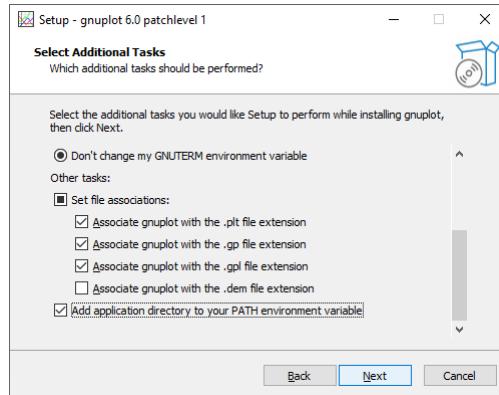


Figure 1.4: Gnuplot installer dialog box example 3

If you don't tick this box, Hobyah will not be able to call Gnuplot until you manually edit your PATH environment variable to add it (or get a knowledgeable user of Windows to edit your PATH environment variable for you).

8. The last dialog box just shows you a summary of the options you have accepted. Scroll to the bottom of the list using the scroll bar and check that the last entry in the list reads “Add application directory to your PATH environment variable”, as in Figure 1.5.
If it does, click “Install”. If it doesn’t, click the “Back” button and repeat steps 7 and 8.
9. The installer will then unzip the program into a folder on your hard drive and show a list of program changes. These may be of interest to advanced users of gnuplot, everyone else should just click “Next” without reading.
10. Once the program is installed, there is a dialog box to let advanced users

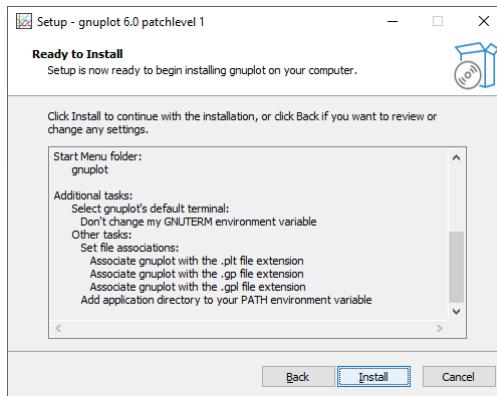


Figure 1.5: Gnuplot installer dialog box example 4

of gnuplot read some release notes and/or start the program. Just click “Finish”.

11. Finally, turn your machine off and restart it so that the change to the PATH environment variable are applied to all your applications.

Installing Gnuplot on macOS

I generally find it easiest to keep Gnuplot updated via Homebrew, a package manager that I use to keep a number of other programs on my mac up to date (such as gfortran and ImageMagick). If you use macOS, have not installed Homebrew and do not use another package manager then search online for the term “macos homebrew” in a browser to get the current best guidance for installing Homebrew.

After I installed Homebrew, I installed the three programs with the following Terminal commands:

```
% brew install gnuplot
% brew install gfortran
% brew install imagemagick
```

Every now and again I go online and run the following commands in the macOS Terminal window:

```
% brew upgrade gnuplot
% brew upgrade gfortran
% brew upgrade imagemagick
```

to ensure that they are all up to date.

If you are a macOS user who does not want to use Homebrew you may be able to install the packages directly or by another package manager, but I can't give you any detailed advice—you're on your own.

1.11.3 Using Hobyah and SESconv without installing Python

You can use Hobyah and SESconv without explicitly installing Python on your Windows PC.

The two files `Hobyah.exe` and `SESconv.exe` are versions of the programs that have a full installation of Python—with all the required libraries—in a subfolder named `_winexelib`. When either `.exe` file runs, it uses the files in `_winexelib` rather than a version installed on the PC. These executables were generated by using the `PyInstaller` module, which is a simple way of building a Python script into an executable.

If you want to move or copy `Hobyah.exe` and `SESconv.exe` to some other location, remember to move or copy the `_winexelib` folder too. If you don't remember, the `.exe` files in the new location will immediately crash with a message like the following:

```
[PYI-9796-ERROR] Failed to load the Python DLL 'C:\Users\tester\Desktop\Hobyah\_winexelib\python313.dll'
```

The path may differ, but the last folder in the path will always be `_winexelib`. Just a reminder that you should copy the `_winexelib` folder into the folder you copied the executables into.

1.11.4 Installing Python on Windows

Many engineering firms have a vetted version of Python 3 in their software repository. If that's the case, get it from there. If not, ask your IT department to install it. Ask them to install Numpy, Pandas and SciPy while they're at it.

If you don't have an IT department, download the current stable version of Python from <https://python.org/downloads> and install it. At the time of writing the file downloaded is `python-3.13.0-amd64.exe` meaning that the stable version is Python 3.13.3.

Navigate to the folder that your browser stores downloads in, this is usually your `Downloads` folder, but may be different if you have changed it.

Run `python-3.13.0-amd64.exe` (or whatever numbers the file has). A dialog box similar to that in Figure 1.6 should appear.

If you do not have admin privileges on your PC (or do not know what admin privileges are) untick the box next to the text “Use admin privileges when installing py.exe”.

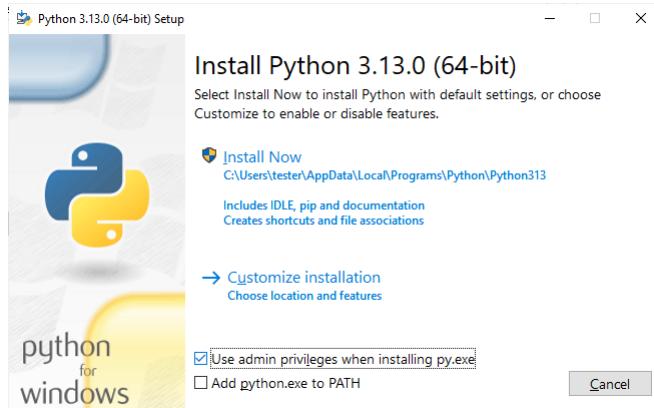


Figure 1.6: Python installer dialog box example 1

Make sure to tick the box next to the text “Add python.exe to PATH” before you click on the text “Install Now” in the middle of the dialog box, as in Figure 1.7.

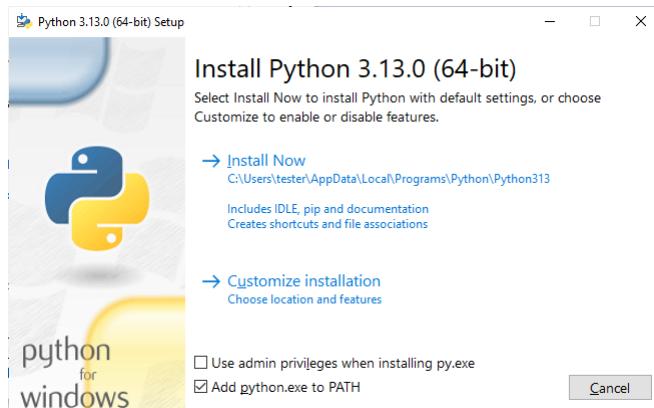


Figure 1.7: Python installer dialog box example 2

If you don’t tick that box, you will have to edit the PATH yourself or get a knowledgeable Windows user to edit it for you.

The next dialog box that appears should be titled “Setup was successful”, as in Figure 1.8.

If it is, click “Close” and restart your machine so that the change to the PATH environment variable is applied.

If it is something else, start searching for solutions to whatever problem occurred online. Good luck!

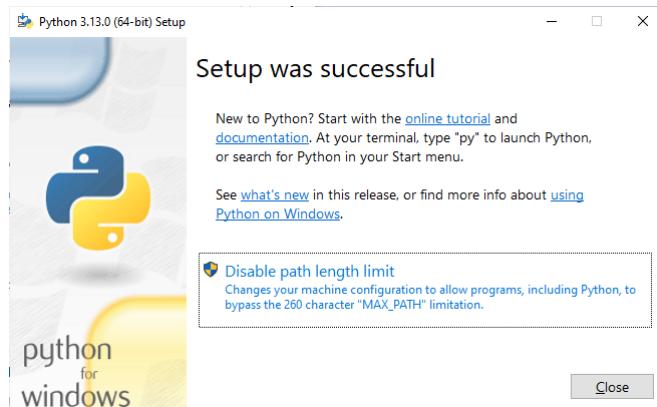


Figure 1.8: Python installer dialog box example 3

1.11.5 Installing Numpy, Pandas and SciPy

This section assumes that you have a working python installation on your machine. Modern versions of python come with a package manager named pip and it is easiest to use this. The same commands work on both macOS and Windows:

1. Open an interactive terminal session (`cmd.exe` on Windows, `Terminal` on macOS).
2. Type

```
python3 -m pip install numpy pandas scipy
```

and press the Enter key.

On some systems the command may need to be

```
python -m pip install numpy pandas scipy
```

or

```
py -m pip install numpy pandas scipy
```

Pip is a pretty reliable package manager on both Windows and macOS. If it throws up errors your best bet is to start doing online search with parts of the error message.

1.12 The calculation process

Figure 1.9 depicts the program operations and the locations of files and sub-folders generated/read when an input file for Hobyah (let's call it `foo.txt`) is fed to `Hobyah.py` for processing.

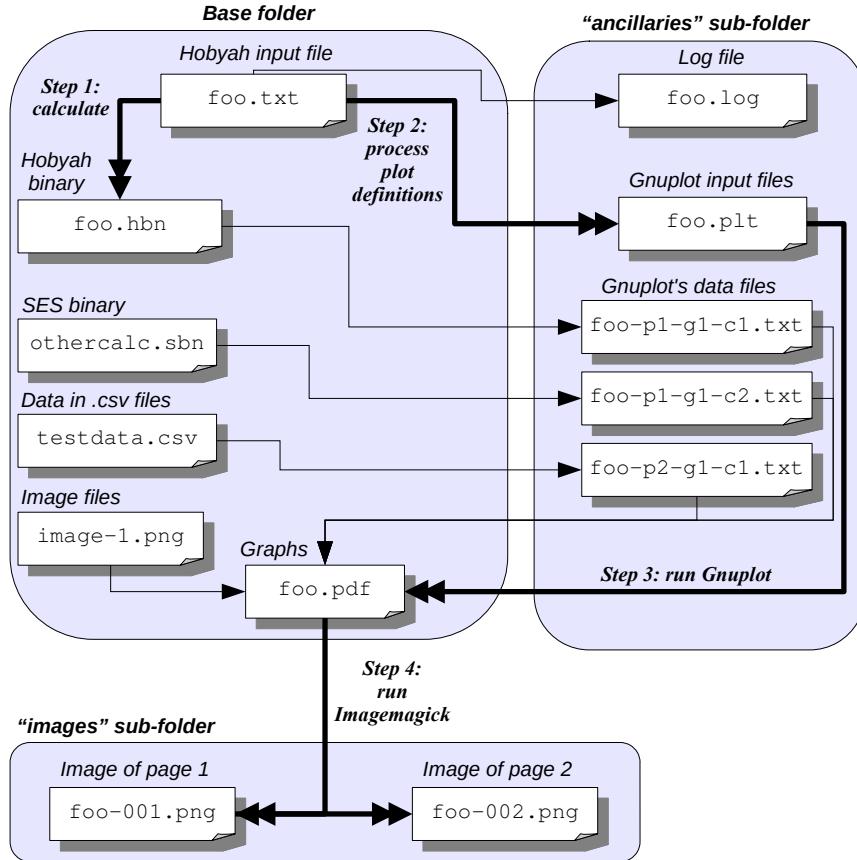


Figure 1.9: Program flow, files and folder structure

The large rounded rectangles in Figure 1.9 represent folders. There is one base folder, which holds the input file (`foo.txt` at the top left of the Figure). There are two subfolders within the base folder. One holds files that are either only of interest during debugging (the `ancillaries` folder), the other contains image files that would otherwise clutter up the base folder (the `images` folder).

The thick lines with double-headed arrows in Figure 1.9 represent a program running and generating new files. The thinner lines represent transfers of data from one program to another via files.

Hobyah runs its “calculate” function first and produces a new `.hbn` file full of calculation data (in this case, `foo.hbn` in the Figure). After finishing the calculation, it runs its “plot” function to generate files that can be read by

Gnuplot. It then calls Gnuplot to generate one or more new `.pdf` files full of graphs. In most cases the process will end there (run a calculation, create one or more `.pdf` files of output). But it can be ordered to call ImageMagick to convert the pages of those `.pdf` to `.png` images (step 4 in Figure 1.9).

The program can be instructed to skip step 1 (calculation) and just run the “plot” functions. Skipping the calculation is a timesaver when all you want to do is adjust the graph definitions instead of rerunning a calculation that hasn’t changed.

Some notes on the files and folders shown in Figure 1.9:

- `foo.log`: this file contains text describing what Hobyah does as it processes the input file. It is used for debugging, and is the only output file that is always written. It is in a subfolder called `ancillaries`, because when everything is going well, the `log` file is of no interest.
- `foo.hbn`: a binary file of results from a Hobyah calculation. It is only written if the input file was told to run a calculation. It is in the same folder as the input file, mostly to make it easy to check that it exists. `.hbn` files are Python pickle files.
- `othercalc.sbn`: a binary file of results from running `SESconv.py` on an SES output file. `.sbn` files are also Python pickle files. Data can be taken from any number of `.hbn` or `.sbn` files for plotting.
- `foo.pdf`: this holds one or more pages of graphs/images generated by the `plot` function. These graphs may be of calculation results in `foo.hbn`, from calculation results in other binary files, (like `othercalc.sbn` in Figure 1.9), from `.csv` files (like `testdata.csv`) and may include images (like `image-1.png`). More than one `.pdf` file may be created.
- `foo.plt`: this holds the Gnuplot commands used to generate `foo.pdf`. More than one `.plt` file may be created, depending on the plot specification (there is one `.plt` file for each `.pdf` file). When things are going well the `.plt` files are of no interest, so they are in the `ancillaries` subfolder.
- `foo-p1-g1-c1.txt`: this is a text file that holds the data used to plot the first curve in the first graph on the first page (the entries `-c1`, `-g1` and `-p1` in the file name signify `curve 1`, `graph 1` and `page 1` respectively). Similarly-named files exist for every curve on every graph on every page. A curve data file might be generated from the input file’s binary file, other binary files, `.csv` files or blocks of data in the input file.
The files are in `.csv` format to let them be loaded into spreadsheets. They are held in the `ancillaries` subfolder because there may be thousands of them.
- `foo-001.png`: this is an image generated from the first page of `foo.pdf` by ImageMagick’s `convert` function. If there are other pages in the `.pdf` file, `convert` will generate a `.png` file for each one. The `.png` files are all stored in a subfolder named `images` so that they don’t clutter the base folder.

It is worth reiterating that Hobyah input files can be used purely for calculating (step 1 in Figure 1.9) or purely for plotting (steps 2 and 3 in Figure 1.9). It is useful to be able to run a calculation once then run without recalculating as you adjust the graph extents or add annotations.

1.13 A simple tunnel example

Input files in Hobyah use the keywords `sectype`, `tunnel`, `join`, `joins` and `node` to construct a tunnel's geometry. A brief explanation:

- A `sectype` specifies the properties of a one-dimensional air path: area, perimeter and either roughness height or fixed friction factor. Each `sectype` is assigned a nickname rather than a number (although you are free to use numbers for the nicknames if you want to). See Section 3.13 for details.
- A `tunnel` is an air path that starts and ends at a portal or a node. A tunnel starts with one `sectype` and can have changes of `sectype`, fixed losses, fans, jet fans, dampers and joins along its length. See Section 3.14.
- A `node` is a point where the ends of between two and six tunnels meet. See Sections 3.14.3 and 3.14.4 (at the back end and forward end respectively) for more details.
- A `join` is a place at which a tunnel has the ends of up to four other tunnels socketed into the side of the tunnel (the `join` acts as a `node` for the ends of the other tunnels). See Section 3.14.10 for more details.
- A `joins` command specifies multiple `join` commands in a compact manner, and is mostly used for defining dozens of escape cross-passages between two tunnels. See Section 3.14.11 for more details.

The geometry of a simple road tunnel is given in Figure 1.10. This has one simple traffic tube and a more complex one that has an off-slip tunnel (in Australia we drive on the left). The traffic tubes are formed by four tunnels, five portals and one three-way node.

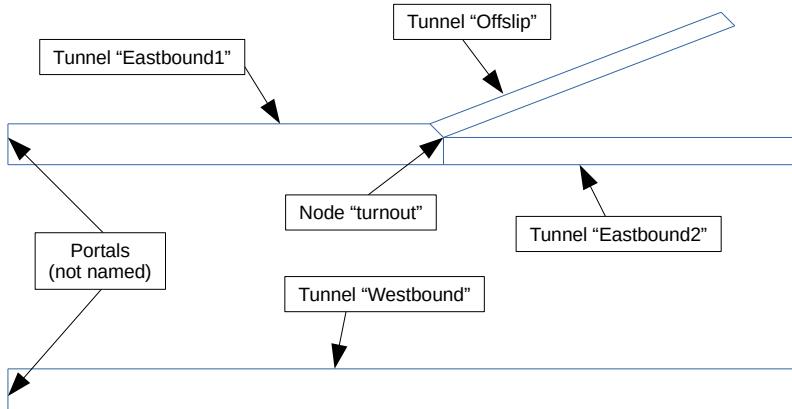


Figure 1.10: Example road tunnel geometry

One tube (Westbound, at the bottom of Figure 1.10) is modelled by a tunnel that starts and ends at portals to the open air. The other tube is modelled by three tunnels that have a portal at one end and a node at the other.

The tunnel system in Figure 1.10 can be made more complex by having multiple `join` or `joins` keywords along the length of tunnels. `Join` and `joins` commands make it easy to attach tunnels that model cross-passages and ventilation shafts along the length of a running tunnel.

Figure 1.11 shows a modified version of Figure 1.10 with 14 new joins and seven new tunnels along the length of the traffic tubes to represent escape cross-passages.

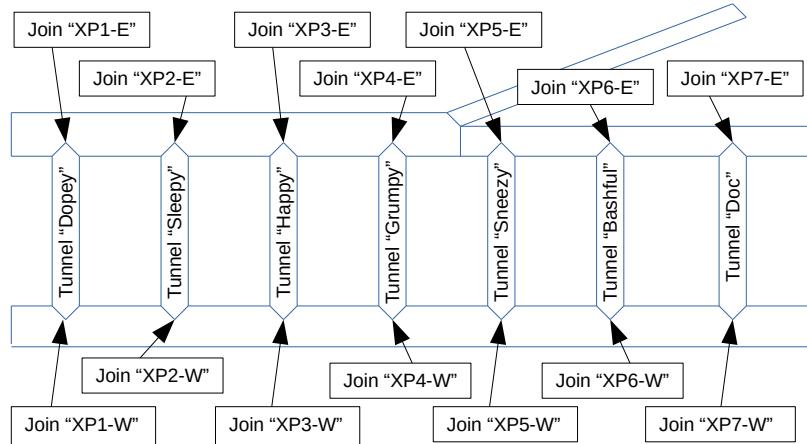


Figure 1.11: Illustrations of joins and adits in tunnels

The names of the seven escape cross-passage tunnels are a cheap joke that I couldn't resist making, but oddly enough there is a real-life equivalent⁵.

1.14 Sectypes in Tunnels

Tunnels have a back end (left end) and a forward end (right end). The definition of the back end includes a sectype in its input. This sectype persists until it is changed or until it reaches the forward end of the tunnel. Changes can be made in two ways: a direct change of sectype or a change of sectype merged with a join.

Figure 1.12 illustrates this. It shows one tunnel with three sectypes along its length. One sectype is defined at the back end, one is set at a change of sectype

⁵In the early 2000s Mersey Tunnels built three new escape cross-passages to connect the two tubes of the Kingsway (road) Tunnel. Someone floated the idea of naming them Tom, Dick and Harry in honour of the three tunnels portrayed in “The Great Escape”, and the names stuck. I owe a tip of the hat to Peter Bishop of MT for this.

(using the `change` keyword, see Section 3.14.5). One change of sectype happens to be close to a cross-passage socketed into the side of the tunnel, so it made sense to combine the two into one entity and use a `join` that is also a change of sectype (see Section 3.14.10).

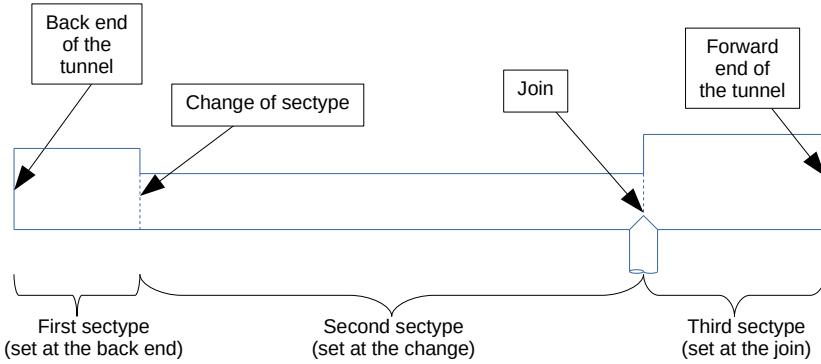


Figure 1.12: Illustration of changes of sectype in a tunnel

1.15 Segments

After a tunnel geometry has been defined, the program divides it up into lengths of tunnel of constant area with loss factors at both ends and with body forces caused by traffic drag and jet fan thrust. These are known as segments (because they are similar to segments in SES).

Each segment has constant area, constant perimeter and either constant roughness height or constant friction factor. Each segment starts at a feature (portal, area change, join, damper, fan etc.) and ends at another feature. Segments have no features along their length: each segment has a mass of air, a friction term, traffic drag terms and jet fan thrust terms. Most of the interesting stuff like fans, dampers, nodes, portal doors, etc. happen at the ends of segments and are treated as boundary conditions applied at two or more gridpoints.

The process of dividing a model into segments is best explained in pictures. Figures 1.13, 1.14 and 1.15 show the segments derived from Figures 1.10, 1.11 and 1.12 respectively.

The numbering scheme of the segments is set by the order the `tunnels` blocks appear in the input file. Segments are a purely internal feature, so the order in which tunnels are defined does not matter unless you are debugging something.

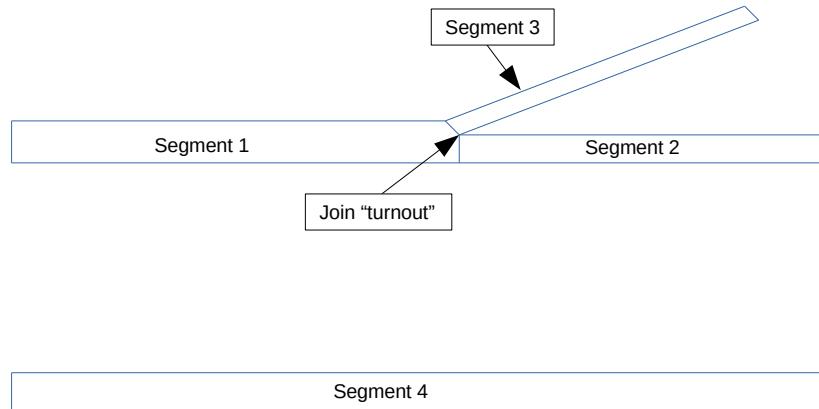


Figure 1.13: Segments generated by Hobyah.py from Figure 1.10

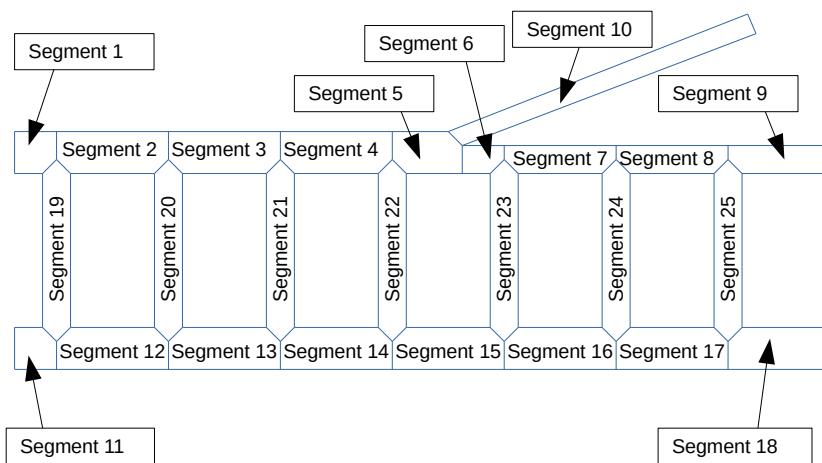


Figure 1.14: Segments generated by Hobyah.py from Figure 1.11

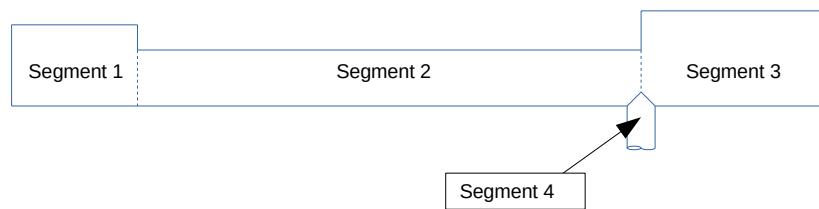


Figure 1.15: Segments generated by Hobyah.py from Figure 1.12

The `log` file holds a list of all the properties of the segments, in a form that I hope makes sense. An extract of one (segment 6 in Figure 1.14) is given below.

```
Seg_ID 6, 1st segment of tunnel "eastbound2":
    Back end: 12100.0 m (tunnel end at a node named "Turnout")
    Forward end: 12160.0 m (adit with a node named "xp5-e")
    Segment length: 60.0 m
    Segment area: 103.0 m^2
    Perimeter: 55.0 m
    Hydraulic diameter: 7.4909 m
    Roughness height: 0.07 m
    Relative roughness: 0.0093447
        >>> Fanning c_f at 5 m/s: 0.009269 Colebrook's 1939 approximation (used)
        >>> Fanning c_f at 5 m/s: 0.009277 Colebrook-White (exact)
        >>> Fanning c_f at 5 m/s: 0.009242 Moody's 1947 approximation
        >>> Fanning c_f at 5 m/s: 0.00928 SES's 1974 approximation
        >>> Darcy lambda at 5 m/s: 0.037075 Colebrook's 1939 approximation (used)
        >>> Atkinson k at 5 m/s: 0.0055613 kg/m^3
    Back inflow zeta: 0.3      (zeta_bf)
    Back outflow zeta: 0.12    (zeta_fb)
    Forward outflow zeta: 0.0   (zeta_bf)
    Forward inflow zeta: 0.0   (zeta_fb)
    Count of cells: 3
    Cell length: 20.0 m
    Count of gridpoints: 4
```

A few notes about jargon in this extract:

- Fanning friction factor `c_f`, Darcy friction factor `lambda` and Atkinson friction factor `k` are three ways of looking at the same tunnel friction factor (see Section 3.3.3).
- Four calculations of friction factor at an airspeed of 5 m/s are included. This is mostly to give engineers a feel for how accurate each approximation is (as the relative roughness goes up, some of the approximations we currently use for friction factor in the tunnel vent field become woefully inaccurate).
- Zeta and ζ are the name and symbol used for dimensionless pressure loss factors (`k`-factors, see Section 2.9).

The `log` file also lists the properties of segments attached at joins/nodes. An extract of one entry (for the node `turnout` in Figures 1.14 and 1.11) is given below. It starts with a list of the internal segment identifiers (`seg_IDs`: compare the list [1, -2, -3] to the numbering in Figure 1.13).

```
Seg_IDs connected at join "turnout": [1, -2, -3]
* Forward end of seg_ID 1 attached, details:
```

```

        Area: 103.0 m^2
        D_h: 7.4909 m
        Roughness height/-c_f: 0.035
            Zeta towards: 0.1      (fb)
            Zeta away: 0.25     (bf)
        Stagnation term towards: 0.18
        Stagnation term away: 0.25
    * Back end of seg_ID 2 attached, details:
        Area: 103.0 m^2
        D_h: 7.4909 m
        Roughness height/-c_f: 0.035
            Zeta towards: 0.12     (bf)
            Zeta away: 0.3       (fb)
        Stagnation term towards: 0.176
        Stagnation term away: 0.26
    * Back end of seg_ID 3 attached, details:
        Area: 103.0 m^2
        D_h: 7.4909 m
        Roughness height/-c_f: 0.035
            Zeta towards: 0.49     (bf)
            Zeta away: 0.22     (fb)
        Stagnation term towards: 0.102
        Stagnation term away: 0.244

```

Most of the entries here are the same as in the entries for the segments. There are two additional entries for stagnation pressure change coefficients, which are used in the routines that calculate flow splits at the joins in the method of characteristics calculation. The stagnation terms are pressure loss/gain factors divided by a constant factor ψ (see MoC.pdf in the documentation folder for the definition of ψ).

1.16 Input file formats

Let's turn to a different aspect of things. SES input files are fixed format (one number in each field 10 characters wide, eight fields per line). That has caused a lot of heartache over the years. IDA files are all set in a GUI, something that has its good points and bad points. The Mott MacDonald suite of programs use free format input, as does the fork of SES I wrote for Aurecon between 2012 and 2020. How does Hobyah do it?

Hobyah input files are free format and use a block structure. Any number of blank lines are permitted. There are few counters in Hobyah input files.

Input is in blocks delimited by `begin <block type>...end <block type>`.

Two types of block are mandatory: A `begin settings...end settings` block and a `begin plots...end plots` block.

The line with `begin settings` marks the start of the input.

The line with `end plots` marks the end of the input.

Everything before `begin settings` is ignored, as is everything after `end plots`. A short example input file is as follows:

```
This is a line of comment before the start of input.

begin settings #           **Input starts at this line**
end settings

# This is a comment inside the input.

begin plots
end plots #           **Input ends at this line**

This is a line of comment after the end of input.
```

The example illustrates how comments work:

- Between `begin settings` and `end plots` the blocks of input comments must be preceded by `#` (exactly like comments in the Python programming language).
- On lines before `begin settings` and lines after `end plots` the comments do not need to be preceded by `#`.

The `plots` block can be empty, but the `settings` block has three mandatory entries. These are described in the next section.

1.17 Mandatory settings

There are three mandatory entries in the `settings` block: `version`, `frictiontype` and `runtype`.

```
begin settings
    version 1
    runtype calc
    frictiontype Darcy
end settings
```

The first mandatory entry (`version`) is a version of the input file (not the version of the Hobyah program). It is used to set how the syntax of the input file is processed. In the event that a new feature makes it attractive enough to break the syntax rules of older files, the input file version will be increased so

that files with old syntax and files with new syntax can both be processed. See Section 3.3.1.

The second mandatory entry (`runtypes`) defines what the run is to do: calculate then plot, or just plot. See Section 3.3.2.

The third mandatory entry (`frictiontype`) defines what type of friction factor to use in the calculation input, `Fanning`, `Darcy` and `Atkinson`. These are explained in more detail in Section 3.3.3.

There is no default type of friction factor, because I've seen far too many engineers mistake Fanning friction factor for Darcy friction factor over the years and I want to stamp that out wherever possible.

If you don't know whether Fanning friction factor, Darcy friction factor or Atkinson friction factor is the friction factor you are most used to using, the best way to start understanding the difference is to read Section 3.3.3 and the document that it points to.

1.18 Portals

Tunnels that have an end open to atmosphere can have several types of boundary condition:

- constant (gauge) pressure outside,
- constant velocity,
- constant volume flow,

See Section 3.14 for more details of boundary conditions at portals.

1.19 Nodes

Nodes can have between two and six tunnel ends attached at them. Nodes have no pressure loss factors at them, so pressure losses at nodes must be set at the tunnel ends attached to the nodes. You can check the pressure loss factors at nodes by looking in the `log` file (see Section 1.15).

Traffic and trains can pass through nodes at the ends of tunnels.

Nodes are explained in more detail in Section 3.14.1 and the Sections it references.

1.20 Joins

Joins are placed partway along a tunnel and can have up to four tunnel ends attached at them. Joins may be accompanied by a change of area in the tunnel they are partway along: if there is a change of area, default pressure loss factors are set for expansion and contraction in the smaller of the two sectypes. These defaults can be over-ridden by optional input.

One join can be created by the `join` command (see Section 3.14.10). Multiple joins can be created in one tunnel by the `joins` command (see Section 3.14.11). The `joins` command takes a mutable name and a list of distances at which to place the joins. This may not sound like much, but it is incredibly useful for tunnels with scores or hundreds of escape cross-passages or pressure relief ducts.

Traffic and trains cannot pass through joins partway along tunnels into other tunnels. This is the main difference between `joins` and `nodes`.

1.21 Dampers

Dampers are entities whose resistance to airflow varies with runtime, following a predefined pattern that varies with time. Dampers are placed at a particular distance in a tunnel.

In dampers of type `damper1`, three things vary with time:

- damper area,
- `zeta_bf` (k-factor applied to the velocity through the damper area for flow from back end to forward end) and
- `zeta_fb` (k-factor for flow from forward end to back end).

All four columns (time, area, `zeta_bf`, `zeta_fb`) must be in the same datasource (a `begin data` block or a `.csv` file).

Dampers can open, close or move to a partially open position any number of times.

Dampers can also be defined with a variation of two Atkinson resistances with time (`damper2`).

See Sections 3.14.8 and 3.14.9 for more details.

1.22 Fans

Fans are defined in a two-step process: define the performance characteristics of a fan in a `fanchar` block, then put fans in a `tunnel` block with a `fan1` or `fan2` keyword.

Simple fans (keyword `fan1`) have one speed, one start time and one stop time. They are provided so that Hobyah has a type of fan that maps directly to the fan operations in SES v4.1 (fans in SES form 5C have forward and reverse characteristics, a start time, a stop time and a run-up time).

Complex fans (using keyword `fan2`) take the variation of fan speed with time from a datasource. These fans can start, stop, reverse and change speed multiple times, all controlled by two columns (time and speed) in the datasource. There is no limit to the amount of changes of fan speed/direction in the `fan2` keyword.

Future fan types (`fan3`, `fan4` etc.) will be similar to the `fan2` type but may be controlled by events like a train passing a given point in a route or by values of pollution concentration at a point in a tunnel.

All fan characteristics in the input are fan total pressure, and the recommended fan characteristics are fan total pressure. Fan static pressure and system static pressure can be plotted if you specify the fan diameter in the `fanchar` block but cannot be input, as I want to steer younger engineers away from fan static pressure.

See Sections 3.14, 3.17.2, 3.17.3 and 3.17.4 for more details.

1.23 Routes

Routes are a list of tunnels for plotting profiles, setting road traffic and setting rail traffic. Routes can define gradients and road traffic lane counts for emissions calculations. Routes are explained in more detail in Section 3.16.

When Hobyah builds an SES file from the Hobyah geometry, the route gradients in Hobyah are used to set segment stack heights for segments (vent or line) that have nodes that are in routes. This makes generating SES input files easier than it otherwise would be.

1.24 Jet fans

Like main fans, jet fans are defined in a two-step process. The `jetfan` block defines performance for unidirectional and reversible jet fans. See Section 3.18.2.

Jet fans can currently be placed in tunnels and will eventually be able to be placed in routes. The `jetfan1` keyword in the `tunnel` block puts a bank of jet

fans in a tunnel and defines the count of jet fans, their speed, start time and stop time.

See Section 3.18 for details of how to specify jet fan types with the `jetfantypes` keyword and how to place jet fans in tunnels using the `jetfans1` keyword.

1.25 Converting Hobyah input into SES input

Hobyah divides its tunnels into segments of constant area, has nodes at which up to six segments come together and has routes with gradients and speed limits that pass through tunnels.

It follows that a popular use of Hobyah might be to use it to build SES input files. Features that may be of particular interest to experienced users of SES are:

- Hobyah can calculate segment stack heights for all the tunnels that pass through routes and for segments (e.g. escape cross-passages) that have both nodes in routes.
- Hobyah allows tunnels to be socketed into the side of other tunnels, so you can define a few tunnels in a Hobyah input file and generate an SES file with a large number of sections, segments and nodes. The test file `ok-049-write-SESfile.txt` generates SES input files has five Hobyah `tunnel` blocks and one `tunnelclones` block. When run, it generates 23 Hobyah cross-passages. It writes an SES input file with 74 sections. This is something I intend to use next time I have to write an SES file for a 3 km twin-tube road tunnel with escape cross-passages every 120 m.

It is simple enough to get Hobyah to build the bare bones of an SES input file: the counters in form 1D and forms 2, 3, 5, 6, and 8.

The `SESdata` block controls how the SES input file is written. It has entries that set entries in SES forms that have no equivalent in Hobyah, such as forms 1B, 1F, 1G, 3G, 4, 12 and 13.

It has entries that tell the program how to transform entities in Hobyah into their equivalents in SES, such as axial fans (SES forms 7A and 7B), jet fans (form 7C) and routes (SES forms 8A to 8F).

It can't always manage to wrangle the entities in Hobyah into equivalent entities in SES, alas. But the function can do about two-thirds of the heavy lifting for you. See Section 3.21 for more details.

Input files can be generated for five versions of SES that I am familiar with:

- SES v4.1, published by the US Department of Transportation in 2002.
- OpenSES v4.3, the current work-in-progress version on github.

- Aurecon SES v107.0, a private fork of v4.1 developed for the Australian consulting engineering firm Aurecon.
- offline-SES v204.6, a private fork of v4.1 that I started developing in May 2020.
- SVS v6.6.2, a fork of v4.1 developed by PB/WSP that you may be able to get a yearly licence for if you ask the right people in WSP.

Input files for v4.1, v4.3 and v204.5 are in US customary units. Input files for v107.0 and v6.6.2 are in SI units.

The feature is activated by the presence of one or more **SEsdata** blocks in the input file. See Section 3.21 for details of that block.

Control over the numbering of the sections/segments and the type of segment (line or vent) is provided by pragmatics in the **tunnel** blocks (see Section 3.14.14).

1.26 Road traffic

Setting road traffic is a two-step process. You define the properties of traffic in a **traffictypes** blocks and define blocks of traffic (moving and stationary traffic) in **trafficsteady** blocks.

You define the properties of road vehicles (drag properties, PCU values, HGV mass, speeds on upgrades) in a **traffictypes** block (see Section 3.19).

Traffic is then put into routes using one or more **trafficsteady** blocks (see Section 3.20).

If you have stationary traffic in a route, a count of lanes must be defined in the route definition.

The **trafficsteady** blocks apply two types of speed restriction: speed limits set in the route definition and speed limits set by HGVs needing to slow on steep upgrades. All these restrictions are user-settable.

Trafficsteady blocks are also used to let traffic with different origins and destinations share a common set of lanes in some tunnels. This is useful for complex road tunnels with multiple entry and exit portals. It is way over the top for simple road tunnels (road tunnels with one entry portal and one exit portal). But I reckon that having the program do the hard work of figuring out the vehicle mix of half a dozen mixing streams of traffic is much better than getting engineers to do it in spreadsheets. See Section 1.27 for a worked example.

1.27 Traffic use example

Figure 1.16 represents one bore of a road tunnel with five portals. The southbound bore has three entry portals and two exit portals. The adjacent northbound bore (not shown) has two entry portals and three exit portals. The portal



Figure 1.16: Road tunnel with named portals

names are taken from five parallel streets right next to each other not far from my home.⁶

Traffic modellers work on a point to point basis. A typical specification for a road project like this would have a table of traffic flows like Table 1.1, giving the flowrates of passenger cars from point to point.

	Allambee	Bringa	Carramar	Doonkuna	Elaroo
Allambee	—	572	801	302	245
Bringa	294	—	104	705	422
Carramar	53	83	—	601	1400
Doonkuna	370	215	900	—	261
Elaroo	532	481	1540	81	—

Table 1.1: Typical traffic flows (cars, veh/hr)

There is a diagonal line of blanks for point to point travel to the same place, as you might expect. The traffic modellers have 104 cars/hour going from Bringa to Carramar: this happens on surface roads and is of no interest in tunnel ventilation design. If we strip out the traffic flows on surface roads we end up with Table 1.2.

The block of numbers in the top right of Table 1.2 represents vehicle flows in the southbound tube of the road tunnel. The block in the bottom left represents vehicle flows in the northbound tube of the road tunnel. For completeness, we'll invent traffic flows in the tunnels for two other traffic types: light commercial

⁶I reckon that back in the 1910s, Melbourne's town planners decided to have an informal contest, in which each planner strove to have the longest alphabetic sequence of adjacent streets arranged on the map like the rungs of a ladder.

	Allambee	Bringa	Carramar	Doonkuna	Elaroo
Allambee	—	—	—	302	145
Bringa	—	—	—	705	422
Carramar	—	—	—	601	1400
Doonkuna	370	215	900	—	—
Elaroo	532	481	1540	—	—

Table 1.2: Traffic flows in tunnels (cars, veh/hr)

vehicles (LCVs) and heavy goods vehicles (HGVs). Tables 1.3 and 1.4 show the LCV and HGV traffic flows.

	Allambee	Bringa	Carramar	Doonkuna	Elaroo
Allambee	—	—	—	33	21
Bringa	—	—	—	108	94
Carramar	—	—	—	132	97
Doonkuna	48	67	81	—	—
Elaroo	34	73	253	—	—

Table 1.3: Traffic flows in tunnels (LCVs, veh/hr)

	Allambee	Bringa	Carramar	Doonkuna	Elaroo
Allambee	—	—	—	82	49
Bringa	—	—	—	181	172
Carramar	—	—	—	264	381
Doonkuna	75	94	101	—	—
Elaroo	51	103	372	—	—

Table 1.4: Traffic flows in tunnels (HGVs, veh/hr)

These are now in a state that makes it easy to turn them into traffic entries in a Hobyah input file. We'll model stationary traffic in the southbound tunnel and moving traffic in the northbound to illustrate how the data from the traffic modellers appears in an input file for both these cases.

The first step is to define twelve routes for traffic; six in the southbound and six in the northbound. Route names like Allambee-to-Doonkuna are a bit unwieldy. We'll abbreviate them: A2D, A2E, B2D, B2E, B2D and B2E in the southbound and D2A, D2B, D2C, E2A, E2B and E2C in the northbound.

The first route in the southbound would likely be:

```
begin route A2D  # Allambee to Doonkuna
  origin 0
  portal 45201
  begin tunnels
    SB1 SB3 SB4 SB5
  end tunnels
  begin lanes
    1 46801 2 47102 3 49435 2 50120
```

```
    end lanes
end route
```

The other 11 routes would be defined in similar blocks.

Next, generate a **trafficsteady** block with six routes and three vehicle types. The structure of the **trafficsteady** block sets the flows in a form that can be easily cross-checked against the data in the traffic modellers' tables.

```
begin trafficsteady Southbound
  routes   A2D    A2E    B2D    B2E    C2D    C2E  # route names
  cars     302    145    705    422    601    1400 # used to get vehicle densities at rest
  LCVs     33     21     108    94     132    97
  HGVs     82     49     181    172    264    381
  standstill A2D  165 PCU/lane-km      up_ptl  down_ptl
  standstill B2D  165 PCU/lane-km      up_ptl  down_ptl
  standstill C2D  165 PCU/lane-km      up_ptl  down_ptl
  standstill A2E  165 PCU/lane-km      up_ptl  down_ptl
  standstill B2E  165 PCU/lane-km      up_ptl  down_ptl
  standstill C2E  165 PCU/lane-km      up_ptl  down_ptl
end trafficsteady
```

In the example above, there are six **standstill** entries for the six routes in the file.

There is a shortcut term that means “all the routes in this block”; it is **allroutes** as the second entry on the line instead of a route name. This is used in the example below for the other tunnel, which would otherwise have six lines of entry, each with a route name on it like the example above.

```
begin trafficsteady Northbound
  routes   D2A    D2B    D2C    E2A    E2B    E2C  # route names
  cars     370    215    900    532    481    1540 # vehicles/hour
  LCVs     48     67     81     34     73     253
  HGVs     75     94     101    51     103    372
  moving   allroutes 50    # km/h
end trafficsteady
```

This sets traffic at 50 km/h in all six routes in the Southbound. Hobyah handles the calculations of traffic densities as the flows of traffic in the routes merge and split.

The model from which this example is taken is in a Hobyah example input file called `ok-034-Hartwell-tunnel.txt`.

1.28 Verification and validation

All the evidence for the verification of the programs' calculations and the validation of their requirements is contained in named `Verification+validation.pdf` in the Hobyah Documentation folder.

1.29 Glossary

This glossary is intended to be your first port of call when you encounter an unfamiliar term. If you look for a term here and can't find it, please raise a bug report on github and ask for the term to be added to the glossary.

Autokeys

Autokeys are the default text used in curve keys if the user did not set a curve's key text or if `Autokeys on` is active in the `settings` block.

The autokey texts are usually boring, descriptive string like 'Total pressure along route "eastbound" at 300 sec', or 'Sensible heat from the ground in zone 1'. The texts are boring and descriptive because they are intended to make it easy for engineers to catch mistakes. See Section 3.3.16 for some background on why every curve has autokey text, why autokeys are useful and how to use autokeys properly.

Back

The `back` keyword defines the location of the back end of a tunnel and what happens at the back end (portal to atmosphere, fixed flow, node etc).

It also sets the first sectype of the tunnel, which persists along the length of the tunnel until the sectype is changed or the forward end of the tunnel is reached.

Boundary

A location at which there are two or more gridpoints. Boundaries with two gridpoints occur at changes of sectype, fans, dampers and some junctions. Boundaries with more than two gridpoints occur at nodes, joins and train ends when they cross other boundaries.

Cell

A cell fills the distance between two gridpoints (cells end at gridpoints). Cells have properties that represent friction, traffic drag and jet fan thrust along their length. These are used to calculate the conditions at gridpoints in the next timestep.

Chainage

Chainage is a civil engineering term named after a 17th century surveying instrument, Gunter's chain.

In Hobyah "chainage" is used to identify locations in routes. Locations in tunnels are referred to as "distances".

Entities (fans, dampers etc.) are at a given chainage in a route. The same entity

may be at a given distance along a tunnel. The difference between distances and chainages can be confusing, so keep the difference in mind.

Change

The `change` keyword creates a change of sectype at a given distance along a tunnel.

Damper1

The `damper1` keyword creates a variable resistance in a tunnel. The damper is controlled by a datasource with four columns: time, area, and two pressure loss factors (one for +ve airflow and one for -ve airflow). This allows dampers to open, close or stay at a partially open position an unlimited number of times.

Damper2

The `damper2` keyword creates a variable resistance in a tunnel. The damper is controlled by a datasource with three columns: time and two Atkinson resistances (one for +ve airflow and one for -ve airflow). If you don't know what Atkinson resistance is, stick to using the `damper1` keyword.

Datasource

A datasource can be one of two things: numbers in a `.csv` file or numbers in a `begin data` block in the Hobyah input file. Datasources can be used to define the operation of fans, dampers etc. and can be plotted, making them useful for test data.

Distance

A way of distinguishing between locations in tunnels and locations in routes.

Entities (fans, dampers etc.) are at a given distance in a tunnel. The same entity may be at a given chainage in a route.

The difference between distances and chainages can be confusing, so keep the difference in mind.

Entity

A catch-all term for something of interest that the program can plot at. Fans are entities. So are changes of sectype and dampers. The up ends and down ends of trains are entities that move through routes.

Fan1

The `fan1` keyword places a fan of a given type in a tunnel. The fan has one speed, a start time and a stop time. Its operation resembles the operations in form 5C in SES.

Fan2

The `fan2` keyword places a fan of a given type in a tunnel. The fan is controlled by a datasource that gives pairs of time and fan speed. This allows fans to start, stop, reverse and change speed an unlimited number of times.

The `fan2` keyword is preferred, because it has finer control over fan behaviour. If you need to replicate fan behaviour in SES, use the `fan1` keyword.

Fanchar

The `fanchar` block generates a fan type, defining the flow-total pressure characteristics in forwards and reverse mode at a given base density.

The block gives the fan type a name, which can be used to place fans in tunnels using the `fan1` and `fan2` keywords in `tunnel` blocks.

Filesloop

The `filesloop` block appears inside the `plots` block and contains the definition of one page block that can be plotted for multiple files. It generates a `.pdf` file that is useful for showing the results of multiple SES fire runs with fires at different points in a tunnel system.

Forward

The `forward` keyword defines the location of the forward end of a tunnel and what happens there (portal to atmosphere, fixed flow, node etc). Similar to the `back` keyword, but doesn't need the `sectype` defined.

Friction factor

If you don't already know what a friction factor is, you probably shouldn't be using this program.

One of three types of friction factor must be selected in each `settings` block of an input file: Fanning friction factor c_f , Darcy friction factor λ or Atkinson friction factor k . See Section 3.3.3 for a fuller explanation of friction and an explanation of why none of these friction factors have the symbol f .

Graph

The `graph` block appears inside `page`, `filesloop` and `timeloop` blocks. It defines the extents of a graph, an arbitrary amount of gnuplot commands inside `verbatim` blocks and the curves to plot on the graph.

Gridpoint

A location in a segment at which values of celerity and velocity are calculated. Celerity and velocity are calculated in each successive timestep by the method of characteristics.

Image

The `image` block appears inside `page`, `filesloop` and `timeloop` blocks. It defines the size and location of the image on the page and the filename of the image that is to be displayed.

Join

A named location placed part-way along a tunnel. The ends of up to four tunnels can be socketed into the side of the tunnel at the join. See also `node` and `joins`.

Joins

A way of generating scores or hundreds of joins in a tunnel with one line of input. Intended for defining points at which escape cross-passages are socketed into the side of a running tunnel.

See also the `tunnelclones` keyword, which is intended to work with the `joins` keyword.

Method of characteristics

A calculation procedure suitable for solving hyperbolic partial differential equations. Hobyah uses the method of characteristics to solve the one-dimensional mass continuity and momentum equations in air with the restriction that changes of state must be isentropic. For a tedious, long-winded explanation of this, see the file `MoC.pdf` in the documentation folder.

Node

A named location at which the ends of between two and six tunnels meet. See also `join`.

Page

The `page` block appears inside the `plots` block and contains the definition of multiple `graph` blocks and `image` blocks that all appear on one page of output.

Plots

The `plots` block is the last block in the input file. It defines three types of output:

- a page with multiple graphs and multiple images on it (a `page` block)
 - one page that is plotted at multiple timesteps (a `timeloop` block)
 - one page that is plotted for the output of multiple files (a `filesloop` block)
- The `plots` block may contain multiple `page`, `timeloop` and `filesloop` blocks.

Route

A path that goes through multiple tunnels joined together at `nodes`. Routes can have speed limits, gradients, counts of road traffic lanes, and launch groups of trains in either direction.

Note that routes cannot pass from tunnel to tunnel via `joins`, they can only pass between tunnels at `nodes`.

Same

`Same` is a reserved name for a sectype, meaning “the same sectype as the current one”. It is used in the `change` keyword to place a gridpoint at a specific location, usually to ensure that there is gridpoint at the location of a sensor in the tunnel in a full-scale test.

Schedule

A train schedule. Takes a train type, a list of train launch times, a location at which to spawn the trains at and the train speed specification.

Multiple `schedule` sub-blocks can appear in each `route` block.

Sectype

The definition of a tunnel cross-section. A sectype needs a name, an area, a perimeter and either a fixed friction factor or a roughness height.

Segment

An internal concept: the program divides every tunnel into multiple segments.

Each segment has constant sectype (constant area). They start and end at features like portals, nodes, joins, fans, etc. The properties of every segment are printed to the `.log` file in the `ancillaries` subfolder.

SESdata

A block of instructions that allows Hobyah input to be converted into the outline of an SES input file. See Sections 1.25 and 3.21 for details.

SESpragmat

An entry in a `tunnel` block that tells Hobyah how to convert its input into an SES input file. For example, the pragmat “`sespragmat 901 vent`” in a `tunnel` block means that when the geometry is converted to SES format, the segments in the tunnel should be numbered in the range 901 to 999 and be converted to vent segments. See Section 3.14.14.

Settings

The `settings` block is the first block in the input file. It defines the main properties of the run (outside air conditions, runtime, timestep etc.).

Timeloop

The `timeloop` block appears inside the `plots` block and contains the definition of one `page` that can be plotted at multiple times. It generates a `.pdf` file that can be converted into an animation showing how the state of the system evolves over time. These animations are surprisingly helpful when you need to explain stuff: see `ok-031-fans-in-series.pdf`, `ok-031-fans-in-series.gif`, `ok-032-fans-in-parallel.pdf` and `ok-032-fans-in-parallel.gif`

Traintype

The properties of a train: length, area, perimeter, roughness/friction factor and rules for handling losses at the train ends.

Tunnel

The only type of air path in the program. Tunnels have a back end and a forward end and any number of entities (changes of sectype, fans, dampers) along their length.

Tunnelclones

Not an “invasion of the body snatchers”-style horror, but a way of defining multiple tunnels (in `tunnelclones` blocks). `Tunnelclones` make it easy to define scores or hundreds of tunnels that represent escape cross-passages between two running tunnels.

See also the `joins` keyword, which is intended to work with the `tunnelclones` keyword.

Verbatim

The `verbatim` block appears inside `graph` and `image` blocks. All text inside the block is passed to gnuplot mostly verbatim.

The only change made is to replace instances of the British English word “centre” to the North American equivalent, “center”, which gnuplot requires.

Zeta (ζ)

The Greek symbol used to represent dimensionless pressure loss factors. You may be used to seeing the symbol k , or call it a k-factor. k is not used for pressure loss factors in Hobyah (because k is used to represent Atkinson friction factor).

1.30 Concluding remarks

Hopefully the preceding sections give you enough information about how to get it running and give you a feel for what the software does.

The full details are in the following chapters.

Chapter 2

Into the Detail

2.1 Blocks of input

The input files are UTF-8 plain text files that use a block structure to define entities.

There are no counters for the user to keep track of.

The following `begin tunnel...end tunnel` blocks are a typical example:

```
begin tunnel Eastbound
    <Lines of input that define a tunnel go here>
end tunnel

begin tunnel Westbound
    <Lines of input that define a tunnel go here>
end tunnel
```

When a `tunnel` block is named `Eastbound`, the data defining the tunnel are assigned to the name `eastbound` (converted to lower case) in the tunnels dictionary and the (internal) count of tunnels goes up by one. The same structure serves all the other entities (for example there is no counter of tunnels in a route, just a list of tunnel names in a sub-block).

This behaviour may seem odd to those used to using tunnel ventilation programs that define the input with counters and a rigid structure for the input files. That approach is no longer necessary now that languages like Python have advanced list handling and string slicing capabilities. I would never have used this block-based approach if I used Fortran to process the input files: it would have been a nightmare to program.

The block-based approach means that every `begin` keyword must be matched by an equivalent `end` keyword at the same level of `begin...end` nesting. For

example:

```
begin plots
  begin page
    begin graph
      <definition of the first graph on the page>
    end graph
    begin graph
      <definition of the second graph on the page>
    end graph
  end page
end plots
```

2.2 Sample files

The following is the shortest valid input file (no comments at the top, one `settings` block with all three mandatory inputs and one empty `plots` block). This file runs, but it does nothing.

```
begin settings
  version 1
  frictiontype Darcy
  runtype plot
end settings

begin plots
end plots
```

The following block gives the outline of a moderately complex file. It defines three tunnels, two routes and some plots.

< Lines of description at the top of the file.>

```
begin settings
  version 1
  frictiontype fanning
  runtype calc
  <other lines of run settings>
end settings

begin sectypes
  <lines of sectype definitions>
end sectypes

begin tunnel main
  <lines of tunnel definitions>
```

```

end tunnel

begin tunnel through
  <lines of tunnel definitions>
end tunnel

begin tunnel off-slip
  <lines of tunnel definitions>
end tunnel

begin route mainline
  begin tunnels
    <list of tunnels in this route>
  end tunnels
  <lines of other route definitions>
end section

begin route branch
  begin tunnels
    <list of tunnels in this route>
  end tunnels
  <lines of other route definitions>
end section

begin plots
  <plot settings>
  begin page # ignore # 1
    <page settings>
    begin graph
      <definitions of graphical results>
    end graph
    begin graph
      <definitions of graphical results>
    end graph
  end page
  begin page # ignore # 2
    <page settings>
    begin image
      <definitions of image to display>
    end image
    begin graph
      <definitions of graphical results>
    end graph
  end page
end plots

```

It doesn't matter what order the **sectypes**, **tunnel** and **route** blocks are in. As long as they are placed between the **settings** and **plots** blocks they will be processed.

2.3 Conventions

2.3.1 Chainages, back end and forward end

The program refers to distances along routes as chainages, regardless of whether the distances along the routes are in feet or in metres. Chainages in Hobyah are either metres (if you are using SI units) or feet (if you are using US customary units).

Don't confuse them with true chainages; the chain is a 17th century surveyor's unit. 1 chain = 66 feet and there are 80 chains in a mile. Miles and chains were used in the 18th century to lay out canals and in the 19th century to lay out railways.

Directions in tunnels use back end and forward end, to make it familiar to those used to the back and forward ends in SES segments. Users of IDA and Motts Aero can think of back end as left end and forward end as right end.

Pressure loss factors (ζ) may differ depending on the direction of flow. When flow is from back end to forward end, ζ_{bf} is used. When flow is from forward end to back end, ζ_{fb} is used instead.

2.3.2 Up and down

Directions in routes take the form of down and up. This distinction is useful because tunnels can appear in routes in reverse orientation: the back end of a tunnel in a route could be either the down end in the route or the up end in the route (think of how a rail crossover tunnel could be oriented in different directions in the two routes that pass through it).

Here are the rules for distinguishing between up end and down end in routes. Let's say that you are standing at a particular chainage in a route:

- everything up from you is at lower chainage in the route,
- everything down from you is at higher chainage in the route,
- the up end of a train or station is at low chainage in the route and
- the down end of a train or station is at high chainage in the route.

These are the conventions I've been taught on every railway track safety course I've attended. Most railways are laid out with the lowest chainage at a station in a prominent city. As you move away from that station the chainage increases. The track safety course instructors taught us to think of directions as going **up to the big city** and **down to the country** when we were trackside.

If you are a mine ventilation engineer, think of the down direction in routes as inbye and the up direction in routes as outbye.

Descriptions of train ends do not use nose and tail (nose and tail switch locations when trains reverse). Trains have an up end (at low chainage in a route) and a down end (at high chainage in a route).¹

Route definitions (see Section 3.16) start at an origin chainage (not always zero) and go down the route to higher chainages.

2.4 Rules for processing input files

Input files have the filename extension `.txt`.

Nothing in the input files is case-sensitive.

Lines of input can be of any length.

Any number of blank lines can be included in an input file.

Each input file is split into three parts:

- Comments at the start of the file, before the start of the first block.
- Blocks of valid input in the middle of the file.
- Text that is ignored at the end of the file, after the end of the last block.

The first block of valid input starts when a line beginning with the phrase “`begin settings`” is encountered.

The end of the last block of valid input occurs when a line beginning with the phrase “`end plots`” is encountered.

All text before the start of the first block of valid input and after the end of the last block of valid input are ignored.

Any number of lines of run description (of any length) can be included at the start of the file.

As long as there is at least one space between words, the count of spaces between words is ignored. `Begin SETTINGS` is treated as if it were `begin settings`².

Everything after the line with the phrase “`end plots`” is ignored. This makes the end of the file a good place to store blocks of unused input that you think you might need later but don’t want to delete.

Blank lines will be ignored except in one case: blank lines within `data` blocks will cause a break in the line of the curve if the data is plotted.

¹This is inconsistent with rail terminology, which usually refers to the down ends of trains and platforms as the “country end” and names the up ends either after a city (“London end”) or a main station (“Flinders St end”). But “up end” and “down end” make more sense in the context of this program.

²For those familiar with Python, the code that parses the input uses Python’s `.split()` and `.lower()` functions a lot.

Comments may be put inside the block of valid input (between `begin settings` and `end plots`). Those comments must start with a `#` character.

You can have a comment after valid input, as in the following line:

```
frictiontype Darcy      # Same friction factor as SES and IDA
```

In the line above, “`frictiontype Darcy`” was the valid input. Everything after the `#` is ignored by the program. You can have multiple lines of comments starting with `#`:

```
# See calc 34 under project 82196CO for the variation of
# air resistance with time in these platform screens.
```

The valid lines of input consist of a series of nested `begin...end` blocks. Each `begin` must be matched by an `end` at the same level of nesting.

I recommend using spaces to indent sub-blocks, but it is not required. In my input files I indent each sub-block by two spaces:

```
begin plots
  begin page
    begin graph
      <definition of the first graph>
    end graph
  end page
end plots
```

Each input file must have a `begin settings...end settings` block and a `begin plots...end plots` block. All other blocks are optional. See section 2.2 for some sample files.

Some characters and combinations of characters are reserved, which means they cannot be used in the names of anything or in descriptive strings. They are:

- The `#` symbol, which starts an explanatory comment. The program throws away everything on the line after the first `#`.
- The `:=` character sequence, which defines an optional input on a line. Optional inputs are described in more detail in Section 3.4.

Instances of `#` are parsed before instances of `:=`, so if you have `:=` to the right of `#` the `:=` will be treated as part of the comment and ignored. For example, the line

```
back 10000  portal 0  #  zeta_in := 0.6  Explanatory comment
```

is a line of input in which “back 10000 portal 0” defines the back end of a tunnel. The words “# zeta_in := 0.6 Explanatory comment” don’t affect the calculation.

The line

```
back 10000 portal 0 zeta_in := 0.6 # Explanatory comment
```

is a line of input in which “back 10000 portal 0 zeta_in := 0.6” defines the back end of a tunnel with the optional entry `zeta_in` set to 0.6. The words “# Explanatory comment” don’t affect the calculation.

It is up to you to make sure that the optional entries you want to be active are to the left of the first # on a line. The program will not warn you about optional entries to the right of a #.

File names in macOS and Linux are case-sensitive. File names in Windows are not.

File names must not contain the symbol #, the two-letter phrase := (see above) or whitespace.

Blocks that can only appear once are: `settings`, `files`, `plotcontrol`, `csv`, `traffictypes` and `plots`. Other types of block can appear more than once.

2.5 How lines are read

A line is read in and a check is made for the presence of # on the line. Everything after the first # is discarded.

It then runs through the text to the left of the first # checking for the presence of := from left to right (:= marks an optional argument).

When it finds an instance of := it removes the word to the left of the := and the phrase to the right of it and stores them in a dictionary of optional entries.

The phrase to the right of the := may be one of the following:

- An integer (300 in the optional entry `dpi :=300`)
- A real number (1.1 in the optional entry `zeta_out:= 1.1`)
- A word (`open` in the optional entry `state := open`)
- A string delimited by single quotes (the string `shaft open` in the optional entry `state1 := 'shaft open'`)
- A string delimited by double quotes (the string `d'Alembert shaft open` in the optional entry `state2 := "d'Alembert shaft open"`). This is

only really here for strings that have to have single quotes within them, like the word `d'Alembert` in this example.

- A simple Python list definition without spaces, e.g. `jf_active := [1,1,1,0,0,0,0,-2,-2,1]`
- A simple Python list definition with spaces and enclosed in single or double quotes, e.g. `jf_active := "[1, 1, 1, 0, 0, 0, 0, -2, -2, 1]"`
- A complex Python list definition, e.g. `jf_active := '[1,] * 3 + [0]*4 + [-2, -2, 1]'`. Users of Python may spot that this list definition is the same as in the previous two examples.

2.6 How blocks are read

Blocks are not processed in the sequence they appear in the input file. This allows you to cluster groups of input close to one another in a big model. For example, you can have a block of sectypes for vent shafts next to the tunnel definitions for those shafts, then a different block of sectypes for running tunnels next to the definitions of the running tunnels.

The program figures out where each block starts and ends, then reads them in the following sequence:

- the `settings` block
- all the `constants` blocks
- all the `data` blocks
- the `csv` block
- all the `sectypes` blocks
- all the `fanchar` blocks
- all the `jetfantypes` blocks
- all the `tunnel` blocks
- all the `tunnelclones` blocks
- all the `route` blocks
- the `traffictypes` block
- all the `trafficsteady` blocks
- the `plotcontrol` block
- all the `files` blocks
- all the `SESdata` blocks
- the `plots` block

The aim of this is to figure out the basics first (is the input in SI or US units? Does the engineer want to set input as Darcy friction factor λ or Fanning friction factor c_f ? and so on).

Next in line are the blocks defining arbitrary values (**constants**) and blocks of data (**data** and **csv**). These are used in many later blocks: for example, you can define the points in a fan characteristic in a **data** block or a **.csv** file and reference it in a fan definition. The same is true for other list-based data like the timing of when dampers open and close.

Then it reads all the **sectypes** blocks to get the names of all the cross-sections. It must do that before reading the **tunnel** blocks, because every **tunnel** block refers to the cross-sections by the names declared in the **sectypes** blocks. And so on: every **route** has a list of names of tunnels, so the **tunnel** blocks must all be read before the **route** blocks.

2.7 Text editors

Engineers who want to use Hobyah should get hold of a decent programmer’s text editor. The default text editor on Windows (**Notepad**) is not suitable, as most of the error messages in Hobyah tell you the line number(s) of the input file that caused the error and **Notepad** does not have the ability to jump to a given line number. If you get an error message from Hobyah that tells you that the bad input is on line 221 of the input file, please do not open the file in **Notepad** and press the DownArrow key 220 times—you’re just wasting your own time, folks. Instead, get hold of a programmer’s text editor and figure out what keyboard shortcut allows you to jump to line 221.

The default text editor on macOS (**TextEdit**) is excellent. **TextEdit** also has built-in versioning, which comes in handy.

If you use Windows and do not already have a favourite plain text editor, it may be worth trying out **Notepad++**. It’s free, open source, very capable and is regularly updated.

For what it is worth, I use **Sublime Text** (I bought a licence for it), **PFE** (an old editor on Windows with better keyboard macros than **Sublime Text**) and **ne**, which is a nice editor to have when accessing servers over a plain text interface like **puTTY** (**ne** implements menus over terminal emulators).

2.8 Required software

Hobyah can be run purely in Python 3 but runs faster if you use a mixture of Python 3 and Fortran. The characteristics calculations are best run in Fortran because the Fortran characteristics calculations are significantly faster than the Python characteristics calculations. I avoid custom language extensions so it should compile relatively easily, although getting it into the form of a Python

module can be difficult on Windows. See Appendix A.1 for a description of how I compiled it and turned it into a Python module.

If you can't get the Fortran code to compile, don't worry about it. Hobyah will switch to internal Python routines that do the same calculations (but slower). Please, don't waste a week doggedly figuring out how to compile Fortran on your machine if you are just trying out the software.

Python 3.7 or higher is required.

Three Python libraries (and their dependencies) are not provided in the base Python 3 installation and must be installed separately; **Numpy**, **Pandas** and **SciPy**. All are popular packages in the scientific computing world, so even if you are working at a firm that has strict rules about what can be installed, all three are likely to be permitted.

Python is notorious for its wide range of poor ways to handle optional packages (so notorious that it even has its own **xkcd**, <https://xkcd.com/1987>). Be careful which Python environment you choose: I once installed Anaconda on a Windows 10 machine and it broke something in the registry so badly that I can no longer drag-and-drop files onto Python scripts (even after uninstalling Anaconda).

I develop code in macOS and only switch to Windows 10 and Linux (mint) to run the test suite of files before uploading. I've found **Homebrew** to work well as my source of updated versions of Python and **pip** to work well as the Python package manager.

Hobyah uses **Gnuplot** for plotting. **Gnuplot** is a stable, freely-available and very capable plotting package that has been around since the 1980s. I chose it partly because a lot of my younger colleagues who had just finished PhDs were already acquainted with it, but mostly because it uses a plain text file to store its commands. I can get Hobyah to write a plain text command file for Gnuplot (a **.plt** file) then call **Gnuplot** to process it. When things go wrong, having a **.plt** file full of plotting commands that I can read makes troubleshooting far easier than it otherwise would be.

Gnuplot is not part of any Python distribution and has to be installed separately. I had no issues when installing **Gnuplot** via **Homebrew** on macOS, via **apt-get** on Linux and the installer from the **Gnuplot** website on Windows.

By default, Hobyah uses **Gnuplot**'s **pdfcairo** terminal to generate **.pdf** files containing many pages of printed output. If **Gnuplot** is not available, Hobyah will not be able to generate a **.pdf** file of the results. However, it will generate all the **.csv** files that hold the curve data and the command file for **Gnuplot** and you may be able to copy those to another machine or load them into a spreadsheet.

There are two optional programs.

The first is ImageMagick, an open source image conversion program. If it is installed, Hobyah can be told to call ImageMagick to convert the pages of pdf output into .png files suitable for including in reports, presentations and similar documents.

I included this capability because I have worked with senior engineers who genuinely thought that the best way to get figures into reports is to make their graduate engineers take screenshots of plots from an engineering program, paste the screenshots into Microsoft Paint, crop them, then paste the resulting images into technical reports.

That is a soul-destroying waste of graduate engineers' time that ought to be avoided at all costs.

By installing ImageMagick, you can automate two of these steps: the conversion of pages into .png files and the cropping of the images.

If ImageMagick is not available on a particular machine Hobyah will still work, but you won't be able to generate a set of .png files from the .pdf files.

ImageMagick depends on a second program to process .pdf files, ghostscript. Ghostscript is another open source image conversion program. I have had ghostscript installed on so many machines for so many years that I don't even know if needs to be installed separately or if it is installed under the aegis of ImageMagick.

2.9 Pressure loss factors

Pressure loss factors (k-factors) have names based on the Greek letter zeta (ζ).

Using ζ is my tribute to a book that I use a lot in my work, I. E. Idelchik's *Handbook of Hydraulic Resistance*. It's a classic, despite its many typesetting problems. Also, I can't use the symbol k for pressure loss factors because I need to use k to represent Atkinson friction factor.

In every place where a pressure loss occurs, the convention used in Hobyah is to define one pressure loss factor for airflow in the back-to-forward direction and another for airflow in the forward-to-back direction (ζ_{bf} and ζ_{fb} respectively). The subscripts `bf` and `fb` appear all the time in the log file; they represent the back-to-forward direction and forward-to-back direction. Keep an eye out for ζ_{bf} and ζ_{fb} (`zeta_bf` and `zeta_fb` in your input files).

At portals, the default factors for inflow and outflow are set to 0.5 dynamic heads and 1.0 dynamic heads respectively, applied to the area of the tunnel at the portal. They can be changed by the optional arguments `zeta_in` and `zeta_out`, `zeta_bf` and `zeta_fb` in the definition of the back end and forward end of tunnels.

At area changes, the default k-factors are calculated as if the area changes are abrupt expansions and abrupt contractions.

The Borda-Carnot formula is used for flow in abrupt expansions (Idelchik equation 4–1 in all editions),

$$\zeta_{\text{expansion}} = \frac{\Delta P}{\frac{1}{2}\rho U_1^2} = \left(1 - \frac{A_1}{A_2}\right)^2 \quad (2.1)$$

where

- ΔP = pressure drop (Pa),
- ρ = air density (kg/m^3),
- U_1 = air velocity in the smaller of the two tunnels (m/s),
- A_1 = area in the smaller of the two tunnels (m^2),
- A_2 = area in the larger of the two tunnels (m^2).

The default factors for contraction at area changes taken from Figure 14.14 in D. S. Miller's *Internal Flow Systems*, 2nd edition. The Figure has five curves, each representing loss factors for various radii at the contraction. The curve used is for a sharp-edged contraction (in the book's parlance, “ $r/d = 0$ ”).

I could have used Idelchik's expression³ instead:

$$\zeta_{\text{contraction}} = \frac{\Delta P}{\frac{1}{2}\rho U_1^2} = 0.5 \left(1 - \frac{A_1}{A_2}\right)^{0.75}. \quad (2.2)$$

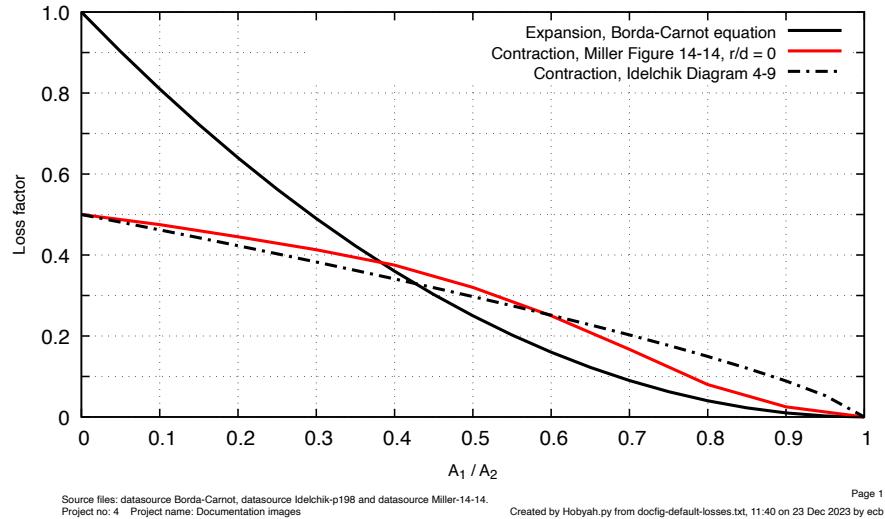


Figure 2.1: Loss factor ζ at sharp-edged abrupt expansions and contractions

³This expression can be found at the start of Idelchik's Diagram 4–9.

Figure 2.1 shows the curve of contraction loss from Miller and the curve of contraction loss from Idelchik. After I plotted Figure 2.1, I couldn't decide whether to use Miller's or Idelchik's as the default for abrupt contractions. I ended up flipping a coin; Miller's curve won.

Users are free to set their own pressure loss factors by using the optional arguments `zeta_bf` and `zeta_fb` in the definition of area changes. In order to avoid misunderstandings, if you set optional pressure loss factors you must state which of the two areas to use (smaller, larger, before or after) in another optional argument, e.g. `area:=smaller`. See Section 3.14.5 for more details.

Resistances can be set directly with the `loss1` keyword (see Section 3.14.6). This entity sets an arbitrary area, a pressure loss factor for positive airflow (back end to forward end) and a pressure loss factor for negative airflow (forward end to back end): think of them as an orifice plate in the tunnel with values of ζ_{bf} and ζ_{fb} . This is intended for ventilation shafts and draught relief shafts: you can set the shaft resistance by area and ζ in the same place. The arbitrary area of the resistance need not be smaller than the tunnel it is in (unlike an orifice plate in a real pipe).

Resistances that vary with time (dampers) can be set with the `damper1` keyword (see Section 3.14.8). This is similar to the the `loss1` keyword but instead of one fixed value of area and two fixed values of pressure loss factor, the area and pressure loss factors are defined as time sequences defined in `data` blocks or `.csv` files. I recommend using `data` blocks instead of `.csv` files.

All ways of specifying a resistance discussed so far set two numbers: a pressure loss factor ζ and an area that it applies at. In the case of junctions and tunnel portals the area is set in one place and the pressure loss factor is set elsewhere. This can lead to problems in the modelling.

I've lost count of how many fee-earning projects I've worked on in which we wrote paper calculations that summed up the losses at 10–20 fittings in a ventilation shaft and lumped them all into one value of ζ appropriate to the velocity of an airflow in a given cross-sectional area. Then we built a 1D model and inadvertently applied our calculated value of ζ to a part of the 1D model that had a different cross-sectional area (and thus a different air velocity). So our air resistances in those shafts were wrong. This is a common occurrence (my colleagues and I have made mistakes like this in SES, Aero, IDA and bespoke software that I've written).

To help avoid this, I've included a concept from the field of mine ventilation, the gaul⁴. Mine ventilation engineers have a neat way of defining resistances to airflow. Instead of setting an area and a k-factor applied to the air velocity in that area, they set a flow resistance that applies to a volume flow. Their resistance has the symbol R and units of $N \cdot s^2 / m^8$ (equivalent to $Pa \cdot s^2 / m^6$).

If you use it, the program will multiply R by the square of volume flow (m^6 / s^2)

⁴At this point almost every tunnel ventilation engineer that has ever worked with me will groan. I've been harping on about how great gauls are since 2005. Hear me out!

to get the pressure drop in N/m² (Pa):

$$\Delta P = RQ^2 \quad (2.3)$$

where

$$\begin{aligned} \Delta P &= \text{pressure drop (Pa),} \\ R &= \text{resistance (N-s}^2/\text{m}^8 \text{ at standard air density),} \\ Q &= \text{volume flow of air at the pressure loss (m}^3/\text{s).} \end{aligned}$$

R is known as Atkinson resistance, in tribute to J J Atkinson, a 19th century English inspector of mines who wrote one of the earliest comprehensive explanations of mine ventilation circuits (1854).

Atkinson resistance is a concept that really ought to be more widely used in the tunnel ventilation field. It removes a whole class of errors that can occur every time one engineer calculates the pressure loss factor ζ applied to the air velocity in an adit of (say) area 20 m² and a second engineer builds a 1D model and applies that ζ to the air velocity in an adit of area 30 m².

2.10 Constants and units conversion

The program sets the following hardwired constants:

- g , acceleration of gravity = 9.80665 m/s² \approx 32.174 ft/s²
- Absolute zero temperature = $-273.15^\circ\text{C} \equiv -459.67^\circ\text{F}$
- ν , kinematic viscosity of air = $1.5 \times 10^{-5} \text{ m}^2/\text{s} \approx 1.394 \times 10^{-6} \text{ ft}^2/\text{s}$
- Standard air density = 1.2 kg/m³
- Density of mercury = 13595.26 kg/m³
- Ratio of specific heats for air $c_p/c_v = \gamma = 1.4$

Some of these can be modified by entries in the `settings` block, in the unlikely case that you need to twist the program to a weird use-case, like a low pressure pipe network or gases that are not standard air.

The program calculates in SI units but can convert to and from US customary units. A few US customary units have ambiguous values: the BTU has many definitions and giving pressures in inches of mercury requires that a density of mercury be chosen.

In most cases I've chosen to use the conversions in the SES v4.1 source code. The only one that differs is the BTU, which is taken to be the International Tables BTU (1 IT BTU \approx 1055.056 J).

The exact arithmetic used is given in the Python file `UScustomary.py`, along with a lot of comments explaining the logic behind it. Those wanting to look more closely into the conversion factors are encouraged to read the comments in that file.

`UScustomary.py` includes a conversion test routine that prints a transcript of the conversions. Edited transcripts for SI→US and US→SI conversions are given in `verification+validation.pdf`.

When run normally, Hobyah does the conversion silently. However, if the command line option `-debug1` is used then it will write detailed information about the units conversion to the logfile. A typical example for the conversion of the values in a distance is given below.

```
converted 3334.323024 m to 10939.38 FT by dividing by 0.3048.
```

That level of detail in the log file is usually only useful when debugging something complicated.

2.11 Error messages

I tend to put a lot of informative error messages in my software, because good error messages make everyone's lives easier. All of Hobyah's error messages are numbered and most of them have one or more files to test them.

Many error messages have variant text. There are routines specifically written to pretty-print lists for error messages. These write text like
“`that keyword was not recognised, here is a list of the available keywords that can be used in this context`”
or
“`the name of that sectype was not recognised, here is a list of the valid sectype names`”.

Not all error messages have test files, because some errors can only be raised during software development. There is a QA script (`_error-statements.py`) that checks that every error raised in the scripts either has a test file or appears in a list of errors that can't have test files. Best example of an error message that can't have a test file: error number 2003, “the input file doesn't exist”.

A transcript of error messages generated by test files is given in Appendix C.

Chapter 3

Input Specification

3.1 Introduction

This chapter describes in detail the lines of input.

First, a warning: this chapter is poorly structured.

It is poorly structured because it has been written, edited, re-edited, added to, had sections deleted from it and whatnot. It repeats stuff in earlier chapters. Parts were written years ago and haven't been reviewed, parts were written last week. In short, it is very much a work in progress and not at all what a team of technical writers would have written if they were paid to write a software manual.

If you spot something that is misleading or have a suggestion that would improve it I would like to hear from you.

It is written with the assumption that you are writing an input file in SI units. Every now and again it mentions US customary units, but only where I think that stressing which US unit to use is of interest. For example:

- base atmospheric pressure is set in Pascals in SI units (not kPa) and in inches of mercury in US units.
- wind pressures at portals are set in Pascals in SI units (consistent) and in inches of water gauge in US units (inconsistent with the inches of mercury used for atmospheric pressure, so worth pointing out).

3.2 Comments at the top of the file

Most programs allow the user to put some lines of comments at the start of an input file to note what the file is for. In a Hobyah input file any number of lines

of comment (of any length) can be written at the top of the file. Blank lines are allowed. The comments end when the first two words on a line (separated by one or more spaces) are `begin` and `settings`. Those two words at the start of a line signal to the program that the line is the start of the `settings` block and the start of input.

3.3 The settings block

The `settings` block contains the main run controls and sets the values of things like outside air pressure and air density.

Only one `settings` block can be defined. It is a special block, in that the line starting with `begin settings` marks the end of the comments at the top of the file.

If you accidentally put other types of blocks before the `begin settings` block they will be treated as lines of comment at the top of the file and ignored. The program will not warn you about this, although it does write the comments to the log file so you may spot a suspiciously long intro there.

The settings block has three mandatory entries and some optional entries that change default behaviours. The mandatory entries are `version`, `runtype` and `frictiontype`.

The entries can be in any order.

The comments in the example settings block below indicate which subsection deals with each setting.

```
begin settings
    version # subsection 3.3.1 (mandatory)
    runtype      # 3.3.2      (mandatory)
    frictiontype # 3.3.3      (mandatory)
    frictionapprox # 3.3.4
    units         # 3.3.5
    QA1, QA2, QA3 # 3.3.6
    P_atm        # 3.3.7
    rho_atm      # 3.3.8
    aero_step     # 3.3.9
    aero_time     # 3.3.10
    footer        # 3.3.11
    header        # 3.3.12
    plotnames     # 3.3.13
    images         # 3.3.14
    keytextscales # 3.3.15
    autokeys      # 3.3.16
    solver         # 3.3.17
    gamma          # 3.3.18
    rise_time     # 3.3.19
```

```
max_vel      # 3.3.20
time_accuracy # 3.3.21
end settings
```

3.3.1 Version setting

The version setting sets the version of the input file, which defines what syntax the program should expect to read. The version setting is accompanied by a number:

```
begin settings
  version 1
end settings
```

The version number will be increased whenever a new feature is added that breaks backwards compatibility with the syntax of older input files.

The only valid version is currently 1. Breaking backwards compatibility in input files will be avoided where possible. But every now and again a new idea might turn up that makes it attractive to allow version 2 input files so it makes sense to have an entry for the input file version.

3.3.2 runtype setting

Two **runtype** setting are allowed:

```
begin settings
  runtype plot
  runtype calc
end settings
```

The setting exists because Hobyah can run in two ways.

The **runtype plot** setting allows the program to be used to plot the results of existing Hobyah runs, SES runs, data from .csv files and data in **data** blocks in the input file. No calculations are carried out; any blocks that define calculation data are skipped.

The **runtype calc** setting tells the program to run a calculation and then try to plot.

The aim of these two options is to allow Hobyah to be used as a tunnel ventilation plotter as well as a tunnel ventilation calculator.

The intent is to run a calculation once with **runtype calc** then adjusting the plotted output without rerunning by setting **runtype plot** and changing the plot settings. This may or may not be a good idea (time will tell). If it turns

out to not be a good idea then the calculation and plotting programs may be split in two.

If you want to run a calculation and not plot anything, use an empty plots block:

```
begin plots
end plots
```

You can still use the results of the calculation in another input file (see Section 3.9).

If you tend to use a command line interface (Terminal in macOS and Linux, PowerShell in Windows) you can force Hobyah to run in `plot` mode by using the command line option `-nocalc` even if the file contains the line `runtype calc`.

This is a convenient shortcut that I use all the time. I often type something like

```
% ./Hobyah.py foo.txt
```

into a macOS Terminal session. The computer runs my calculation and generates the plots. Then I look at the plots and realise that they are not what I want, so I edit the graph definitions in the `plots` block in the file and run

```
% ./Hobyah.py -nocalc foo.txt
```

and repeat the “edit the graphs, run with the `-nocalc` option” loop until the plots are what I want. Rerunning the calculation every time the plot definitions change is unnecessary.

3.3.3 frictiontype setting

```
begin settings
    frictiontype fanning
    frictiontype darcy
    frictiontype atkinson
end settings
```

This sets the type of friction factor to use. There are three choices:

- Fanning friction factor c_f
- Darcy friction factor λ ($= 4c_f$) and
- Atkinson friction factor k ($= \frac{1}{2}\rho c_f$)

Users must choose one of these three. This choice is to force users who are mechanical or aeronautical engineers to understand the difference between λ and c_f . None of the friction factors in the program have the symbol f .

I've included the ability to set Atkinson friction factor k in case any mine ventilation engineers find a use for the program.

If `frictiontype Darcy` is used then any friction factors set in the input file are assumed to be Darcy friction factor λ in

$$\Delta P = \frac{\lambda L}{D_h} \frac{1}{2} \rho v^2. \quad (3.1)$$

If `frictiontype Fanning` is used then any friction factors set in the input file are assumed to be Fanning friction factor c_f in

$$\Delta P = \frac{c_f L S}{A} \frac{1}{2} \rho v^2 = \frac{4 c_f L}{D_h} \frac{1}{2} \rho v^2 \quad (3.2)$$

If `frictiontype atkinson` is used then any friction factors set in the input file are assumed to be Atkinson friction factor k in

$$\Delta P = \frac{k L S}{A} v^2 \quad (3.3)$$

instead. Note the absence of $1/2\rho$ here: the $1/2\rho$ is included in the k .

Users who can't tell if the f they already know about is c_f or λ should read “`friction-rant.pdf`” in the Hobyah documentation folder. If you still aren't sure which friction factor you want after reading that document, I recommend that you stop using Hobyah. Instead, find someone who is competent to explain what friction factors are to you.

3.3.4 frictionapprox setting

```
begin settings
    frictionapprox Colebrook
    frictionapprox Colebrook-White
    frictionapprox Moody
    frictionapprox SES
end settings
```

This option sets the equation used to approximate friction factors from roughness heights.

Four options are available:

- **Colebrook**: roughness heights are converted to friction factor using the explicit approximation in Colebrook's 1939 paper.
- **Colebrook-White**: roughness heights are converted to friction factor by making up to fifty iterations of the Colebrook-White function, stopping when the difference between successive estimates is less than 0.1%.

- Moody: roughness heights are converted to friction factor using the explicit approximation in Moody’s 1947 paper.
- SES: roughness heights are converted to friction factor using the explicit approximation in the SES source code in the annulus around trains.

For more details, the equations and a comparison of them can be found in section 4 of the file “`friction-rant.pdf`” in the Hobyah documentation folder.

The default is `Colebrook`. After plotting heat maps of the differences between the various friction factor approximations I reckon that Colebrook’s 1939 approximation is better than the other approximations commonly used in the tunnel ventilation field (it is more accurate than Moody’s, more accurate than the one in SES and I don’t see a need to iterate the Colebrook-White function in every cell at every timestep—lots of our other inputs generate greater inaccuracies than friction does).

Each time a Hobyah file is run, it prints the Fanning friction factors of all four options at 5 m/s to the log file and tags the one being used. This is a way of comparing the differences between the estimates, which can be large at high relative roughness.

3.3.5 units setting

```
begin settings
    units SI
    units US
end settings
```

The `units` setting defines whether to use US customary units or SI units in the input.

There are two valid entries: `SI` and `US`.

The default is `SI`. Internally the program uses SI units.

See Section 2.10 for details of the conversion between SI units and US customary units.

The same units system will be used for the output unless it is overridden in the `plots` block (see section 3.22.1).

3.3.6 QA settings

Most commercial projects require some form of quality assurance (QA). Three strings can be set to help with this: `QA1`, `QA2` and `QA3`.

- `QA1` is a project number.

- QA2 is a project name.
- QA3 is a project description or comment.

QA1 and QA2 appear on plotted output by default and all three will appear in curve data files and Gnuplot files. Although QA1 is intended to be a project number it can be any string. For example:

```
begin settings
  QA1 23942-A2W          # Project number and stage
  QA2 Albury to Wodonga HSR tunnel    # Project name
  QA3 A high speed rail tunnel from Albury to Wodonga?
end settings
```

If a QA entry is not set the defaults are used. QA1 will be No project number, QA2 will be No project name and QA3 will be No project description.

An example of the first two QA entries on plotted output can be seen in the header and footer text on Figure 2.1.

3.3.7 P_atm setting

```
begin settings
  p_atm 101200
end settings
```

This sets the base air pressure outside the tunnel at datum level (defined as an elevation of 0 m). If it is present in a file that is used purely for plotting it will be ignored.

In SI input files, the units of pressure are Pascals. In files using US units, the units of pressure are inches of mercury (in. Hg), assuming that the density of mercury is 13595.26 kg/m³.

If you do not include an entry for this, then the default air pressure will be 101,325 Pa (\approx 29.921 in. Hg). It is recommended that you account for the elevation of your project. I have worked on a few tunnel projects in places at elevations between 500 m and 2000 m and it is surprising how much money such projects can save by buying fans that can run with 315 kW motors on site, even if they need 355 kW motors when being tested in a factory at sea level.

If the P_atm setting appears on a line before the units setting, all is well; the program is smart enough to process the units setting before processing any of the settings that define physical constants.

3.3.8 rho_atm setting

```
begin settings
```

```
rho_atm 1.2
end settings
```

This sets the base air density (ρ_{atm}) at datum level outside the tunnel. It is only required in files that run a calculation. If it is present in a file that is used purely for plotting it will be ignored.

In SI input files, the units of density are kg/m³. In files using US units, the units are lb/ft³.

The default air density is 1.2 kg/m³(≈ 0.0749 lb/ft³).

If you set the units to US after setting the air pressure it does not matter; the program will interpret the density as lb/ft³.

3.3.9 aero_step setting

```
begin settings
    Aero_step 0.1
end settings
```

This sets the timestep (Δt) for the aerodynamic calculation (like the first number in form 13 in SES). It is in seconds.

It is only required in files that run a calculation. If it is present in a file that is used purely for plotting it will be ignored.

3.3.10 aero_time setting

```
begin settings
    aero_time 700.
end settings
```

This sets the time (seconds) that an aerodynamic calculation will run for. If `aero_time` is not an integer multiple of `aero_step` then `aero_time` will be rounded up until it is.

It is only required in files that run a calculation. If it is present in a file that is used purely for plotting it will be ignored.

3.3.11 footer setting

```
begin settings
    footer on
    footer off
end settings
```

Each page of plotted output can include a footer that shows a list of the source files used to plot the graphs and images on the page (e.g. Hobyah files, .csv files, SES files) on the left hand side and a page number at the right-hand side. If the setting `footer off` is used the footer will not be printed.

The default is `footer on`.

An example of the footer can be seen in Figure 3.1.

3.3.12 header setting

```
begin settings
  header on
  header off
  header underfoot
  header CC1
end settings
```

Each page of plotted output can include a header that shows the text in the QA1 and QA2 settings at the left-hand side (project name and project number) and some traceability data on the right hand side (name of the .pdf file, date it was plotted and the user's login name). If the setting `header off` is used the header will not be printed.

If the setting `header underfoot` is used, the header will be moved below the footer. This is useful when generating pages intended for presentations, flip-books and videos, where a line of small text at the top of the page can be visually intrusive.

If the setting `header cc1` is used, the header is moved below the footer and the project name and number are replaced with text that marks the document as being released under the Creative Commons Attribution/ShareAlike licence (v4.0).

The default is `header on`.

An example of the header can be seen in Figure 2.1.

3.3.13 plotnames setting

```
begin settings
  plotnames  .txt
  plotnames  .csv
end settings
```

The data for every curve plotted are held in files in a subfolder called `ancillaries` (see Section 1.12). Each file is a text file in `csv` format. Some people like the files to have the filename extension `.txt`, others may prefer `.csv`.

The `plotnames .csv` setting is for those users who prefer the files to open automatically in a spreadsheet instead of opening in a text editor.

The default is `plotnames .txt`.

3.3.14 images setting

```
begin settings
    images optional
    images required
    images hidden
end settings
```

Hobyah can include images on the pages (using the `begin image` block). These are typically `.png` or `.jpg` files, such as scans of graphs from technical papers or company logos. The `images` setting tells the program what to do when an image is to be plotted.

- `images optional` means that if an image file cannot be read, a placeholder (rectangle with a cross inside it) will be shown in place of the image.
- `images required` means that if an image file cannot be read, the program will stop with an error message.
- `images hidden` means that all images will be replaced with placeholders. This is convenient when modifying pages in which the processing of images causes Gnuplot to run slowly.

The default is `images optional`.

3.3.15 keytextscale setting

```
begin settings
    keytextscale 0.8
end settings
```

The `keytextscale` setting is a multiplier on the size of the text used for the automatically generated key on graphs. The default is 0.65, which means 65% of the text size set in the Gnuplot terminal command. Use this setting if you find the key text to be too small or too large (I tend to use it in files that generate the figures in this document).

3.3.16 autokeys setting

```
begin settings
```

```
autokeys on
autokeys off
end settings
```

The `autokeys` setting tells the program to use the automatically-generated description for each curve in the graph's key, even if the user has entered their own key text for the description in the graph's key. The default is `off`.

This option is useful for checking if the program is plotting something other than what you expect it to be plotting, and it is recommended that you use it before publishing output in reports and papers. It's surprising how easy it is to believe you are plotting one thing whilst actually plotting something else or plotting the correct property at the wrong location. Plotting your pages of curves with the `autokeys on` setting before finalising your report is a good way of catching mistakes.

I usually have a line with `# autokeys on` in the settings block, so I can remove the `#` to activate it.

3.3.17 solver setting

```
begin settings
    solver moc2
end settings
```

This is a setting intended for future use.

Hobyah input files can currently be solved by one method, `moc2`.

The MoC2 solver is a method of characteristics that solves homentropic flow (compressible, isentropic flow with losses) using two variables: speed of sound in air (c) and air velocity (u).

In future, other solvers (like MoC3 for non-homentropic flow or SES for incompressible flow) may be implemented.

The only allowable setting is `solver moc2`. Don't bother using it, it is redundant.

3.3.18 gamma setting

```
begin settings
    gamma 1.4
end settings
```

This sets the ratio of specific heat at constant pressure to specific heat at constant volume ($\gamma = c_p/c_v$) for the air in the tunnel. This is treated as a constant

in the MoC2 solver. γ for air varies with temperature, but not by much: it changes from 1.401 to 1.398 over the range $-25\text{ }^{\circ}\text{C}$ to $75\text{ }^{\circ}\text{C}$ (-13 to $167\text{ }^{\circ}\text{F}$).

The default is $\gamma = 1.4$. I can't see why anyone would want to change it, but the option is there if you have a reason.

3.3.19 rise_time setting

```
begin settings
    rise_time 2.0
end settings
```

This sets a time duration (seconds) over which fixed velocities and fixed volume flow boundary conditions at portals rise to their specified values in the MoC2 calculation. Ramping up fixed velocities/flowrates over a few seconds instead of instantaneously helps dampen down the wilder fluctuations that the method of characteristics can cause. The default value is to ramp up in 2 seconds. Figure 3.1 shows a comparison of the pressure and flowrate fluctuations with a two-second rise time and an instantaneous rise time, to illustrate why an instantaneous rise time is not a good idea.

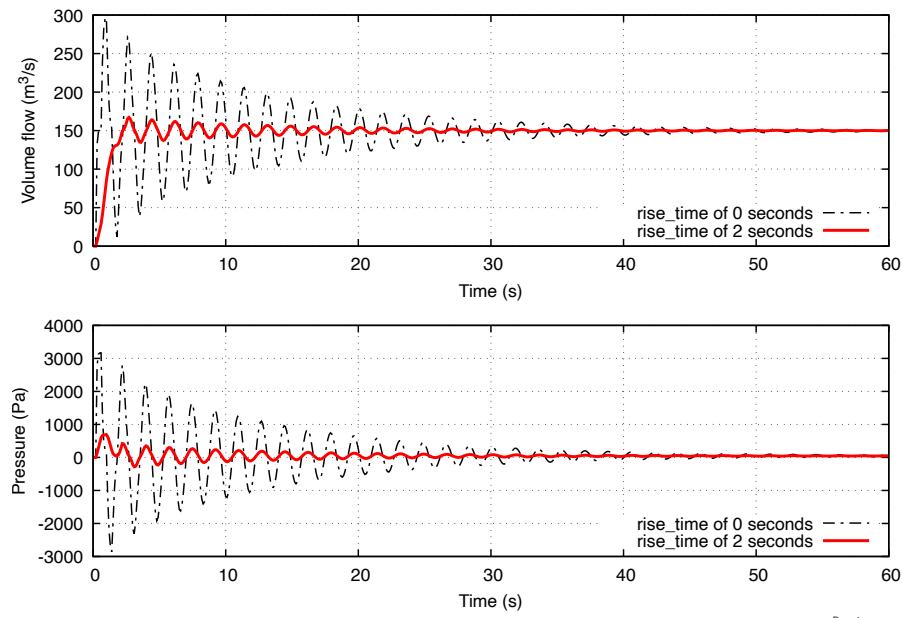


Figure 3.1: Effect of ramping up fixed velocities/flowrates

3.3.20 max_vel setting

```
begin settings
    max_vel 20
end settings
```

This sets a velocity to use in the MoC2 solver when setting the minimum cell size to use in the calculation. The default is 20 m/s, which should cover most use cases. If you find that a method of characteristics calculation is overflowing, try setting `max_vel` to a higher value than 20 m/s (more than 3,937 fpm if you are using US units). Higher values force the MoC2 solver to use longer cells between gridpoints, which might cure stability problems.

Section 7.2 has a more detailed explanation of the effect of this setting.

3.3.21 time_accuracy setting

```
begin settings
    time_accuracy 8
end settings
```

A setting used during debugging. It sets a count of decimal places to round times to before using them in the calculation. Rounding times ensures that comparisons of times don't fail due to slight differences between floating-point numbers that ought to be equal but actually differ due to having been generated in different ways. The default is to round times to 8 decimal places.

3.4 Optional entries

3.4.1 Basics of optional entries

Some types of input have optional entries as well as mandatory entries. These are best explained by example.

A good example is in the `sectypes` block (which defines tunnel cross-sections and is explained in more detail in Section 3.13). Each entry defines the properties of a tunnel section. You must set at least four entries for each:

```
begin sectypes
    Patchway 22.61 18.19 0.005
end sectypes
```

The first mandatory entry is a nickname that this sectype can be referred to by, the second is an area, the third is a perimeter and the fourth is a rough-

ness height. Fairly straightforward: in the example above the sectype called `patchway` has area 22.61 m², perimeter 18.19 m and roughness height 0.005 m.

You can set two types of optional entries. The first type of optional entry is a description. Optional descriptions consume everything on the line after the mandatory entries:

```
begin sectypes
  Patchway 22.61    18.19      0.005    Patchway Old, Gawthorpe & Pope 1976
end sectypes
```

In this case, the optional description is `Patchway Old, Gawthorpe & Pope 1976`. It identifies the source of these numbers (an excellent paper on full-scale measurements of transient pressures in tunnels). The optional description isn't used in the calculation; it's just convenient to have it in the file for future reference.

I chose not to put this optional description after a `#` character so that the string is stored as a description in the binary file and can be preserved as a comment in input files generated from the binary file.

The second type of optional entry assigns a phrase to a one-word key. These set optional values that `do` affect the calculation. Again, this is best explained by example.

The `tunnel` block (see Section 3.14) has a keyword (`join`) that allows the ends of other tunnels to connect part-way along the length of a tunnel. The `join` keyword has an optional argument that allows it to change the sectype in the tunnel being connected to:

```
begin tunnel <name>
  <lines defining the tunnel>
  join 10090 XP-01
  join 10230 XP-02 sectype:=bored
  <further lines defining the tunnel>
end tunnel
```

In the example above, the first `join` command at chainage 10090 just creates a join that other tunnels can attach to gives it a name (`XP-01`) that other tunnels can refer to it by. The second `join` command has an optional entry in it: `sectype:=bored`. When the program sees the `:=` on the line it knows that the words on either side of the `:=` are an optional setting. In this case the optional entry causes the sectype in the tunnel to become `bored` after the join. An example of such a change of sectype at a join can be seen in Figure 1.12.

3.4.2 How to see all valid optional entries

The sections in this manual that cover each type of keyword give details of the optional entries that each keyword has, but you have to search through this

manual. A better way to see which optional entries are valid has been put into Hobyah's error messages. To get a list of the valid optional entries for a given line of input, put in a fake optional entry and read the error message that is printed.

If the keyword has optional entries, the names of the valid optional entries will be listed in the error message.

Take the `leftbase` keyword in the `image` block (`image` blocks have not been explained yet but they make a good example):

```
leftbase 0.89 0.87      chinchillas:=4.2
```

As there is no optional argument named `chinchillas` in the `leftbase` command, Hobyah prints the valid optional arguments in its error message:

```
> *Error* type 2108
> Came across an invalid optional entry in
> "ok-035-train-types.txt".
> The keyword "leftbase" cannot use the
> optional entry "chinchillas", the only valid
> optional entries for this keyword are:
> ratio, border, namecheck and rotate.
> Faulty line of input (162nd) is
>     leftbase 0.89 0.87      chinchillas:=4.2
```

The seventh line of the error message lists the valid optional entries for the keyword. I find that defining nonsense optional entries like `chinchillas:=` or `slartibartfast:=` is easier and faster than reading this manual or searching through the source code.

3.5 The constants block

The `constants` block assigns one-word names to an integer, a floating-point value, a sentence, or an expression that generates a list of numbers.

Constants assigned to integers or floats can then be used wherever that type of value is required. It is akin to assigning names in Gnuplot (or in a programming language), although there is no equals sign.

```
begin constants
    min_v    -2
    max_v   13.5
end constants
```

The most useful place to set a constant is when setting the extents of graphs. Do you have 20 graphs all with the same maximum on the Y axis? You could

set the same numerical value in 20 different graph definitions. But if you want to change that numerical value you must edit the graph extents in 20 different places in your input file.

Much better to set a constant in your input file's **constants** block and use the name of the constant in the Y axis definitions. If you decide you need to change axis extent, all you have to do is change the definition in the **constants** block once.

Constants are not just for setting graph extents. They can also be used as the result of optional arguments.

Constants cannot be used in the **settings** block, because the **settings** block is read before the **constants** block.

The **constants** block can be anywhere in the file between **end settings** and **begin plots**. I've found that the best place is just after the **settings** block; it makes it easier to find.

Each entry in the **constants** block has a one-word name (no spaces) followed by a number or a phrase that generates a list of numbers (to understand phrases, see Section 3.7). There are no optional entries.

As with every other input, the one-word name can't contain the reserved phrases (# or :=). But the names of constants have three extra limitations. First, they must not contain an asterisk (*) because that clashes with the syntax used for axis extents in graphs. Second, they must not contain commas, because that would foul up the syntax in the **data** blocks. Third, they must not contain the @ character because that would foul up the plotting syntax.

The names are not case-sensitive. If you define a name as **MAX_v** you can refer to it by **max_v** or **MAX_V**.

It follows that you can't name two constants **min_v** and **MIN_V**. The fact that they are the same when converted to lower case constitutes a name clash.

Alas, you can't do arithmetic on constants (as you can in Gnuplot). It is too complicated to implement, and you can use all the functions of Gnuplot by using **begin verbatim...end verbatim** blocks in graphs (see Section 3.24.4).

3.6 Sentences

You can assign an optional argument to be a string enclosed in double quotes or single quotes. To be honest I can't see a use for these yet, but I need the code to process list generators, so I might as well use it for sentences.

```
begin tunnel eastbound
    back    0 portal 0 C+C1   description := "in the west cut and cover"
    change   302 TBM    description := 'in the TBM tunnel'
    fwd     1325 node ventshaft   description:= "d'Alembert shaft"
```

```
end tunnel
```

The optional argument `description` can be picked up by other entities (such as train ends) during the run to provide a description of their location that is a bit more informative than `chainage 905 in route "main"`.

Single or double quotes may be used as the enclosing character in case the sentence contains one or the other, as in “d’Alembert shaft”. If you want to use both, you’re out of luck.

If the program finds an opening quote after the `:=` without a matching closing quote farther to the right on the same line, it will raise an error.

3.7 Phrases

A phrase is a group of Python-like statements that can be processed into a list of numbers.

They can be assigned to the names of constants (see Section 3.5) or to optional arguments (see Section 3.4) or used in the `plotcontrol` and `timeloop` blocks to set plot times.

When phrases that contain spaces are assigned to optional arguments, you must enclose them in double or single quotes:

```
begin adits
    first eastbound1 where := "[103, 223, 343, 463]" # Has spaces, so needs quotes
    second westbound where := [103,223,343,463]          # Has no spaces, needs no quotes
    second westbound where := '[103,223,343, 463]'      # Single quotes work too
end adits
```

The quotes are used to pick out the start and end of the value returned by the optional argument that contains spaces. If the quotes were absent the program would split on the space after “103”, return `[103,` and end up raising an error.

When phrases are assigned in the `constants` block (see Section 3.5) they do not need to be enclosed in quotes even if the expression has spaces.

Once the text of a phrase has been picked out, the program tries to turn it into a list of numbers. There are several ways of defining the entries: lists, ranges, or a mixture of the two. These are discussed below.

3.7.1 Lists

Phrases may be defined as straight-up lists, in the same sense as a list in the Python programming language (numbers separated from one another by commas and enclosed in square brackets):

```
begin constants
    jetfancounts [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, -2, -2, -1]
end constants
```

Lists are used where a number of similar features exist and their properties need to be set. The example above may be used to set the count of operational jet fans in 14 banks of jet fans along a route. In this example the jet fans along a route are set to create a low backwards airflow and raise the pressure in a non-incident tunnel (three jet fans blowing forward are opposed by five jet fans blowing backward).

List definitions can include list multiplication and list addition (same syntax as Python's). In the example below a one-element list is copied three times to produce a three-element list. Then a two-element list is copied four times and added to the first list. Finally, a three-element list is added at the end.

```
begin constants
    jetfans_on [1, ] * 3 + [0, 0] * 4 + [-2,-2,-1]
end constants
```

The result of this list is identical to the previous example, just expressed in a different way. Both return the following:

```
[1, 1, 1, 0, 0, 0, 0, 0, 0, -2, -2, -1]
```

The first example sets it directly. The second expands each of the three elements in turn and adds them together. The process goes as follows:

```
[1, ] * 3 + [0, 0] * 4 + [-2,-2,1]
[1, 1, 1] + [0, 0] * 4 + [-2,-2,1]
[1, 1, 1] + [0, 0, 0, 0, 0, 0, 0, 0] + [-2,-2,-1]
[1, 1, 1, 0, 0, 0, 0, 0, 0, -2, -2, -1]
```

Each element in a list must be a number or the name of a constant that is assigned a number. Say you were defining the status of 12 escape cross-passages in a fire case. You could define a constant for the area when closed (leakage only) and a second constant for the area when both doors are open. Then use them in a third constant that defines a list of which cross-passages are open:

```
begin constants
    open 2.4      # Cross-passage area when open
    closed 0.1    # When closed
    XP_areas [closed] * 6 + [open] * 2 + [closed,] *4
end constants
```

This is equivalent to

```
begin constants
    XP_areas [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 2.4, 2.4, 0.1, 0.1, 0.1, 0.1]
end constants
```

You may be wondering whether being able to define lists by summing and multiplying shorter lists is overkill. It's just a 12-element list, why not define it directly as one entry? One reason is that chaining lists together is handy when you are working on long tunnels.

In long tunnels there may be scores or hundreds of regularly-spaced features. A few years ago I worked on a road tunnel where we had to develop fire cases that set the operation of about 60 banks of jet fans. Most of them were off. Initially we tried setting the operations manually by clicking through dialog boxes to tunnel properties. Then we tried setting up 60 number entry boxes in the program's GUI, but got bogged down when we had to adapt it to alternate geometries and broke some of the links between the boxes in the GUI and the jet fans we thought they controlled. We quickly realized that we would never get them right the first time, the second time or even the twentieth time.

We ended up writing Python scripts to generate input files for the calculation program (those scripts took an input file and a .csv file of jet fan operations, then wrote a modified input file with those jet fan operations in it). If the program had had the ability to take a Python list as input to define the operation of jet fans along the routes, we'd have jumped at it.

Another example: the Channel Tunnel stretches 50 km between pas-de-Calais in France and Kent in England. It has two running tunnels carrying trains, a pressurised service/escape tunnel, two crossovers, 270 escape cross-passages, 190 pressure relief ducts and a few other air paths for ventilation shafts. Defining every air path in the Channel Tunnel individually in an input file would be a nightmare task. Lists are the way I've decided to make it less nightmarish.¹

I owe a tip of the hat to Dr. David Henson, who solved a similar problem in the early 1970s when he was commissioned by British Rail to write the program (now the Mott MacDonald Aero program) that ended up being used to design the Channel Tunnel's ventilation system. Dr. Henson devised an elegant way of specifying a large number of air paths using one line of input: he set up his input file syntax so that one line could define many identical air paths spaced at regular intervals, i.e. "create air paths numbered 301–378, put air path 301 at ch. 26500 and space the other 77 air paths 250 m apart".

3.7.2 Ranges (list generators)

Lists may also be created by range functions. These are similar to the `range()` function in Python. There are three forms:

- `range(<start value>, <value to stop before>, <step value>)`

¹You can see an example of how lists are used to define multiple air paths in the example file `ok-037-Channel-Tunnel.txt`.

- `startstopcount(<start value>, <value to stop at>, <integer count>)`
- `startstepcount(<start value>, <step value>, <integer count>)`

`range()` calls Numpy's `arange()` function. You give it a value to start at, a value to stop before and a step value to increment (or decrement) by. Three examples:

```
range(1, 20, 5) = [1, 6, 11, 16]
range(10400.0, 10792.3, 120.1) = [10400.0, 10520.1, 10640.2, 10760.3]
range(20, 1, -5) = [20, 15, 10, 5]
```

Numpy's `arange()` function accepts floating point numbers as arguments, so Hobyah's `range()` function does too.

One thing I'd like to stress: if the entries in a `range()` function are such that one of the entries could be the stop value, then **the stop value is not in the list**. For example:

```
range(0, 10, 2) = [0, 2, 4, 6, 8]
```

You might expect the function to return `[0, 2, 4, 6, 8, 10]`. It doesn't. I don't like this rule. I find it particularly annoying when using a `range` in a `tunnelclones` block. But I take the view that it is best to be consistent with how the `range()` function is implemented in Numpy and Python.

If you want a value at 10, use a stop value that is slightly above 10, such as `range(0, 10.1, 2)`.

The `startstopcount()` function takes a value to start at, a value to stop at and a count of entries. The first entry is at the start value, the last entry is at the stop value and the other values are placed at equally-spaced intervals:

```
startstopcount(5, 10, 6) = [5, 6, 7, 8, 9, 10]
startstopcount(1, 20, 4) = [1, 7.333, 14.667, 20]
```

In a `startstopcount()` range, the stop value is **always** in the list. If that is confusing when you compare it to `range()`—where the stop value is never in the list—then you have my sympathy, but that's just how things are.

The `startstepcount()` function takes a value to start at, a step value and a count of entries. Of the three range functions it is probably the least open to misinterpretation. Some examples:

```
startstepcount(5, 10, 6) = [5, 15, 25, 35, 45, 55]
startstepcount(1.1, 2.5, 7) = [1.1, 3.6, 6.1, 8.6, 11.1, 13.6, 16.1]
```

There are a few things to be aware of when using the three functions:

- The start, stop and step values can be floating point numbers or integers.
- The count of entries must be an integer that is 1 or above.
- If the start value is below the stop value, the step value must be positive.
- If the start value is above the stop value, the step value must be negative.
- Start and stop values can't be too close together (the test is Python's `math.isclose()` function with an absolute tolerance of 10^{-9}).
- Step values cannot be zero.
- Entries may be the names of constants that return one number.

Also, ranges need not go up, they can go down as well.

```
range(10, 0, -2) = [10, 8, 6, 4, 2]
range(10, -0.1, -2.1) = [10, 8, 6, 4, 2, 0]
startstopcount(25.2, 10.2, 4) = [25.2, 20.2, 15.2, 10.2]
startstepcount(55, -10, 6) = [55, 45, 35, 25, 15, 5]
```

3.7.3 Combining ranges and lists

Ranges and lists can be chained together as if the ranges were lists. Every range function is boiled down into a list, ranges and lists are processed recursively in the same procedure, so there is no reason to forbid input like

```
begin constants
  mixture [1, ] * 3 + startstepcount(1, 1, 12) + [12]*2
end constants
```

This is identical to

```
begin constants
  mixture [1, 1, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 12, 12]
end constants
```

The first application for chained ranges/lists was to set variable plot time steps. You can get the program to save the state of the run at short time intervals when interesting things are happening and at longer time intervals when they are not (similar to how form 12 in SES is used).

An example of the use of chained ranges can be found in the example file `ok-031-fans-in-series.txt`: it generates a flipbook-style `.pdf` that shows three fans interacting with one another (one fan is forced into the reverse flow

part of its characteristic, another is forced into its freewheeling region). Between 114 seconds and 116 seconds a fan negotiates the stall hump in its flow-pressure characteristic. During that two-second period the calculation struggles to converge. By using chained lists to define what timesteps to plot out, the pages can be set to show the state of the calculation every 0.05 seconds while the duty point of the fan hunts and then change to 5 seconds after the calculation settles down. This may not sound like much, but having fine control over the print interval makes it a lot easier to explain things like fan characteristics to graduate engineers.

3.8 Defining fans

Fans can be put into the tunnels to generate airflow. On many tunnel projects only a few types of fans are used (mostly to allow for commonality of spare parts and to make factory acceptance tests cheaper to carry out). As a result, the way Hobyah defines its fans is a three-step process that makes easy to define a common fan characteristic and put instances of it in many different places in the model. The three steps are:

1. Lists of pairs of volume flow and fan total pressure rise are put into a **data** block (see Section 3.11).
2. A fan characteristic is defined in a **fanchar** block (see Section 3.17.2). This specifies which datasource to use for the characteristic and which columns in the datasource are the volume flow and the fan total pressure rise.
3. Finally, a line is added to place a fan in a tunnel (in the **tunnel** block (see Section 3.14)). The fan definition refers to a fan characteristic and sets the timing of how the fan operates (see Sections 3.17.3 and 3.17.4).

Fans can be defined by four pairs of total pressure rise similar to how fans are defined in SES form 7B. See Section 3.17.2. These can be written into SES input files (see Sections 1.25 and 3.21).

There will eventually be a process in place to define a fan curve as a cubic polynomial derived from the four points. When this is done, fans will be able to be defined in Hobyah in the same way they are defined in SES, but that feature has not been implemented yet.

3.9 The **files** block

Hobyah is set up so that the results of a calculation can be compared to the results of calculations in other files. The **files** block defines what files those other calculations come from. It gives each file a one-word nickname that entries in the **plots** block reference them by.

The other calculations must be `.hbn` files generated by `Hobyah.py` or `.sbn` files generated by `SESconv.py`. Both types are Python pickle files containing binary data.

Nicknames can be generated in one of three ways:

- `begin files numbered`
- `begin files nicknames`
- `begin files 2syllables`

The first (`numbered`) is just a list of input file names. Each file is given a nickname according to its position in the list: `file1`, `file2`, `file3` and so on. For example:

```
begin files numbered
Songkhla-001.sbn    # .sbn extensions means an output file from an SES calculation
Songkhla-001.hbn    # .hbn extensions means an output file from a Hobyah calculation
end files
```

This gives the nickname `file1` to the binary file generated by `SESconv.py` from SES output file `Songkhla-001.PRN` and the nickname `file2` to the output file generated by `Hobyah` file `Songkhla-001.txt`.

The second (`nicknames`) is to give a specific one-word nickname to each file. The nickname is taken as the first word on the line and the rest of the line is treated as a file name.

```
begin files nicknames
orig NWRL-013-inc-1329ab.sbn
new NWRL-014-inc-1329ab.sbn
end files
```

This gives the nickname `orig` to the output of `NWRL-013-inc-1329ab.ses` and the nickname `new` to the output file generated by `NWRL-014-inc-1329ab.ses`.

The third (`2syllables`) is for my own convenience. It is specific to how I tend to structure the names of input files on projects. I avoid spaces in file names (because I do a lot of work on the command line).

I tend to make the first syllable of a file name an acronym of the project name (`NWRL` above stood for “North West Rail Link”, a project I did a lot of work on in 2013). The second syllable is generally a version number for the geometry—each time it changes significantly the number was increased, so `NWRL-014` would signify the 14th iteration of geometry. The last syllable is generally a code identifying the specifics of a run, such as a number signifying the chainage of

the incident train, how many others trains there are present, which direction the smoke is being blown. I can't recall now what `1329ab` meant, but it would have made sense to me at the time. So the third way of choosing a nickname suits my personal file naming scheme: it builds the nickname from the second and last syllables (where “syllables” are separated by a `-`).

```
begin files twosyllables
    NWRL-013-inc-1329ab.sbn
    NWRL-014-inc-1329ab.sbn
end files
```

This gives the nickname `013-1329ab` to `NWRL-013-inc-1329ab.sbn` and the nickname `014-1329ab` to `NWRL-014-inc-1329ab.sbn`. It works for me.

3.10 The csv block

The `csv` block is a way to get external data into Hobyah, either for use in calculations or for use in plots. Each entry in the `.csv` block gives a one-word nickname that other keywords can reference the `.csv` file's contents by. For example, say we had a `.csv` file that holds a fan characteristic: a column of volume flows and a column of fan total pressure rises. If we want to use that data in a calculation, we create a `.csv` block and assign a nickname to the `.csv` file:

```
begin csv
    fanchar      fan_curve5.csv
end csv
```

This gives the nickname `fanchar` to the `.csv` file `fan_curve5.csv`. The nickname is one word and is used to refer to it in other blocks.

Any number of `.csv` files can be given in the block. The `.csv` block is based heavily on the `files nicknames` block (see Section 3.9) and shares a lot of its features.

In terms of processing the contents of the `.csv` files, the following apply:

- Any text after a `#` character is ignored,
- All items of data must be separated by commas,
- The amount of whitespace is ignored,
- All entries must be numbers, not the names of constants and
- Blank entries in columns of data are permitted, and are created by having commas separated by nothing but whitespace.

When you want to use the contents of a `.csv` file, you generally give its nickname and the numbers of the columns you want to use. Examples can be found in the Sections on dampers, fans and plots (Sections 3.14.9, 3.17.4 and 3.25.1 respectively).

The nicknames given to `.csv` files must be unique, and must not be the same as a nickname given to a `data` block (see Section 3.11).

Elsewhere in this manual the term `datasource` is used interchangeably to refer to the contents of a `.csv` file or the contents of a `data` block. This is because once their contents have been processed they are stored in the same dictionary.

3.11 The data block

The `data` block stores a rectangular block of data of arbitrary size $m \times n$ in `.csv` form (entries separated by commas and optional whitespace). The example below defines a block of data with three columns and six rows.

```
begin data random1
    0,      37.97,   165.08
    10,     14.39,   110.55
    20,     68.83,   81.46
    30,    116.26,   168.06
    40,     45.36,   167.04
    50,      23,     160.64
end data
```

The block of data is referenced by its nickname (in this example, `random1`) and each column of data can be referenced by the column number: 1, 2 or 3 (column numbers start at 1, not at zero).

In the above example the line after `begin data random1` consisted of numbers separated by commas and spaces. The use of numbers on that first line means that none of the columns have names. The example block below is a `data` block in which the columns have been given names: they are `time`, `area`, `zeta_bf` and `zeta_fb`.

```
begin data PSDs
    time,     area ,  zeta_bf ,  zeta_fb
    0,       0.5 ,   2.4 ,   2.4
    1.5 ,   6.3 ,  12 ,  12
    38.5,   6.3 ,  12 ,  12
    40,     0.5 ,   2.4 ,   2.4
end data
```

The block of data is still referenced by its nickname (`PSDs`). Each column can be referenced either by the column name in the first line or by the column number. Column names:

- must not contain commas, asterisks or `:=`,
- must not be numbers, names of constants or anything that could be interpreted as a number by Python’s `float()` function (no naming your column `inf` or `NaN`, folks).

The count of columns is set by how many words there are on the first line. The count of rows is defined by the `end data` entry. Any number of rows and columns are permitted.

The `data` block can be used for two purposes, either defining the variation of things for a calculation (such as the variation of area and resistance in the platform screens example above) or entering test results that can be plotted as curves on graphs.

For the latter purpose, it is useful to be able to skip entries to represent gaps in test data. For this, the entry can be left as whitespace between commas. When the plot routines process the file, Hobyah will pass two blank lines to Gnuplot, which will cause a break the curve.

```
begin data testresults
  time,      vel1 ,  vel2 ,   press1
    0,       1.2,   2.4,    45.2
   10,      5.5,   3.5,    36.1
   20,      9.2,   8.7,   19.9
   30,      9.6,   8.4,   18.2
   40,      9.8,   8.6,   19.9
   50,      6.1,   8.7,   20.1
   60,      ,      7.5,   19.9
   70,      2.3,   4.3,   18.2
   80,      4.5,   2.7,   17.5
   90,      1.2,   4.5,   19.5
end data
```

If you plotted `vel1` against `time` there would be a break in one of the lines between 50 and 70 seconds, as in Figure 3.2.

Another way to get Gnuplot to break its plotted line is to include one or more blank lines.

```
begin data testresults
  time,      vel1
  10,      5.5
  20,      9.2

  30,      9.8
  40,      6.1

  50,      2.3
```

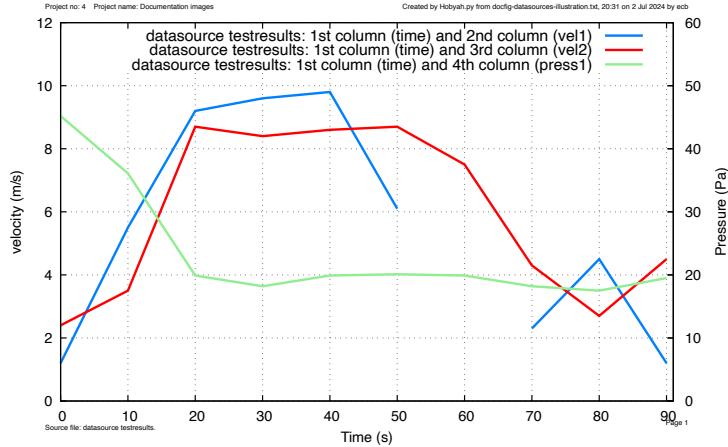


Figure 3.2: Breaks in curves 9 in datasources

```
60,      4.5
end data
```

A line plotted from this would have breaks between 20 and 30 seconds and between 40 and 50 seconds.

I've found it useful to put very large datasets into .csv files that I want to plot and **data** blocks for smaller datasets that hold part of a calculation. But either type can be used. **Data** blocks in the file probably have the edge, because their columns can be given names, can use the names of constants in the data and the data can't get separated from the plotting commands. Data in .csv files can't have column names, can't refer to the names of constants and can get lost if the input file is moved or copied to a different folder and the associated .csv file is left in the original location.

Elsewhere in this manual the term **datasource** is used interchangeably to refer to the contents of a **data** block and the contents of a .csv file.

3.12 The plotcontrol block

The **plotcontrol** block specifies when to save data to the output files (the .hbn and log files).

It is an optional block. If it is not present, data will be saved once per second.

It has one entry: the word **aero** followed by a list of numbers, e.g.

```
begin plotcontrol
  aero  range(0, 100, 0.1)
```

```
end plotcontrol
```

This means “store output data every 0.1 seconds for 100 seconds”.

The constant name `duration` can be used to pick up the run time set in the `settings` block.

```
begin plotcontrol
    aero  startstepstop(0, 0.5, duration)
end plotcontrol
```

This means “store output data every 0.5 seconds for the length of the run”.

The plot time steps can have chained lists so that the program plot infrequently when not much of interest is happening and plot very frequently when something interesting occurs.

```
begin plotcontrol
    aero  startstepstop(0, 1, duration) + range(100, 125, 0.1)
end plotcontrol
```

This plots every second for most of the run. Between 100 and 125 seconds it plots every 0.1 seconds. Note that this definition has duplicate numbers in it—duplicates are removed after generating the list.

A good example can be found in the test file `ok-032-fans-in-parallel.txt`.

3.13 The sectypes blocks

When doing tunnel ventilation calculations, we need to tell the program what the geometry looks like.

There are a few ways of doing this. I’ve decided to define tunnel sections by assigning a nickname to a set of properties (area, perimeter etc.) and refer to a section by its nickname whenever I want to assign those properties to a length of tunnel. The nickname need not be a number, it can be almost any one-word string (the only characters not allowed are whitespace, # and :=).

Sectypes have four mandatory inputs and a few optional ones. The following block shows the mandatory inputs and describes what they are in a comment:

```
begin sectypes
# nickname      area (m^2)    perimeter(m)  roughness(m)
    West_C+C     60.3          32            0.005
end sectypes
```

The first three are respectively a one-word nickname, an area (m^2) and a perimeter (m).

If the fourth entry is zero or positive it is treated as a roughness height (in metres or feet), to be used for calculating friction factor. If the fourth entry is less than zero its absolute value is taken and is treated as a fixed friction factor.

```
begin sectypes
  West_C+C 60.3    32    0.005 # Roughness height 0.005 m
  TBM        40.2    28    0      # Smooth walls, roughness = 0 m
  East_C+C 65.7    34   -0.032 # Fixed friction factor +0.032
end sectypes
```

Note that if the fourth entry is -0 , it is treated as if it is $+0$ (smooth walls, not zero fixed friction factor)². If you want to set negligible friction factor, use something like -0.0000001 (but be aware that this can cause stability problems in the method of characteristics).

Those who prefer numbers are free to use them for the nicknames:

```
begin sectypes
  101     60.3    32    0.005
  102     40.2    28    0
  103     65.7    34   -0.032
end sectypes
```

A description (of any length and including spaces) may be added at the end to give more explanation:

```
begin sectypes
  TBM     40.2    28   -0.032  8.1 m ID TBM tunnel
end sectypes
```

The names of sectypes cannot be the word “**same**” or the word “**sectypes**”. “**Same**” is reserved for situations where you want to put a gridpoint at a specific point in a tunnel, such as a pressure sensor or velocity sensor and “**sectypes**” is used internally when building cloned tunnels.

3.14 The tunnel block

Each **tunnel** block defines a tunnel. A tunnel can start and end at:

- a portal to the open air with a specified gauge pressure,
- a portal to the open air with a specified velocity,
- a portal to the open air with a specified volume flow,

²The IEEE-754 standard for floating-point arithmetic specifies that $-0 = +0$.

- a node where two or more tunnels meet, or
- a join at which the end of the tunnel is socketed into the side of another tunnel.

The names of tunnels must not start with a dash (-) or an asterisk (*), as these characters have special meanings in some circumstances.

The names of tunnels must not contain #, := or @ for the same reason.

Tunnels can have features along their lengths. These should not be too close to one another, or it invalidates the assumptions made in the method of characteristics and raises error type 2087 (see Appendix C). Error 2087 gives advice about what to do in that circumstance. The rule of thumb I use is to set the aero timestep to 0.05 seconds and place entities no closer than 20 m. Section 7.2 goes into the reasoning behind this in more detail.

The only exception to this restriction is jet fans, which are treated as a body force in the characteristics. Their only restriction is that the jet fan outlets cannot coincide with other features like changes of area or joins.

3.14.1 Features at tunnel ends

Each `tunnel` block has two mandatory lines of entry, `back` and `fwd`. These define what happens at the back end (left end) and forward end (right end) of tunnels.

The first entry after the `back` and `fwd` keywords is the distance that the back end or forward end is located at. For example:

```
begin tunnel my_tunnel
  back 10000  portal 0  C+C1
  fwd 10900  Portal 3
end tunnel
```

This defines a 900 m long tunnel whose back end is at distance 10000 and whose forward end is at distance 10900.

The other entries on the line describe what the boundary condition at the end of the tunnel is (fixed pressure, fixed velocity, fixed volume flow or a `node` and its node name. The `back` keyword also sets the initial `sectype` in the tunnel. These are described in Sections 3.14.3 and 3.14.4 in more detail.

3.14.2 Features within tunnels

A tunnel can have changes of `sectype` along its length (see Section 3.14.5).

Tunnels can have fixed pressure loss factors in them. There are two keywords that define losses: `loss1` and `loss2`. Sections 3.14.6 and 3.14.7 give more details.

A tunnel can have N-way junctions along its length by using the `join` keyword. Up to four tunnels representing ventilation shafts, adits, escape cross-passages, intervention stairs or whatever else may socketed into the side of a tunnel at each join. See Section 3.14.10 for more details.

Multiple joins at specified spacings can be created by using the `joins` keyword. See Section 3.14.11 for more details.

Fans (axial/centrifugal fans) can be placed in tunnels. There are two keywords for fan definitions.

The first is `fan1` and takes the name of a fan definition and a fan speed. It is used for a fan that runs at constant speed, switches on once and switches off once.

The second fan keyword (`fan2`) takes a fan definition, the nickname of a data-source (name of a `datablock` or the nickname of a `.csv` file) and names or indices of two columns in the block of data. It is used for fans that can change speed, reverse, and turn on and off multiple times.

For more details of fans and fan characteristics, see Section 3.17.

Dampers can be placed in tunnels and opened and closed multiple times. See Section 3.14.8.

Jet fans can be placed in tunnels. Jet fans differ slightly from the other features, in that they do not create a boundary (a pair of gridpoints), but exert a body force term in the cells that the plume of high speed air from the jet fan occupies. See Section 3.18.3.

3.14.3 The `back` keyword

A line of entry starting with the `back` keyword sets three things: the location of the start of the tunnel, what happens at the back end of the tunnel (it could be open to atmosphere or connected to a node) and the `sectype` that the tunnel starts off with.

Two examples:

<i>keyword</i>	<i>location</i>	<i>end treatment</i>	<i>sectype</i>
<code>back</code>	10000	<code>portal 0</code>	C+C1
<code>back</code>	10000	<code>node Xover1up</code>	C+C1

Each example starts with the keyword, `back`. The number that follows (10000) sets the location at which the tunnel starts (10,000 metres or 10,000 feet, depending on the system of units used in the `settings` block).

The next two entries (`portal 0`) identifies that the end of the tunnel is a portal to the open air with a gauge pressure of zero pascals (or zero inches of water gauge if US units are used).

The last entry (`C+C1`) is the name of a sectype that will be used from the back portal up to the first change of sectype. If there are no changes of sectype along the length of the tunnel, then sectype `C+C1` will persist all the way to the forward end of the tunnel.

The only difference between the first and second examples is that in the second example the end is connected to a node named `xover1up`. This name alludes to the fact that this tunnel starts at the up end of the first rail crossover cavern.

The ends of tunnels may also be given preset velocities and volume flows by using the end treatments `v_inflow`, `v_outflow`, `q_inflow` and `q_outflow`:

<i>keyword</i>	<i>location</i>	<i>end treatment</i>	<i>sectype</i>
back	10000	<code>v_inflow 4.5</code>	<code>C+C1</code>
back	10000	<code>v_outflow 6.2</code>	<code>C+C1</code>
back	10000	<code>q_inflow 415</code>	<code>C+C1</code>
back	10000	<code>q_outflow 0</code>	<code>C+C1</code>

`V_inflow` and `v_outflow` set constant velocity. If the input is in SI units the velocities are in m/s. If the input is in US customary units then velocities are in feet per minute (fpm). `Q_inflow` and `q_outflow` set constant volume flow. Volume flows are in m³/s or in cubic feet per minute.

If you want to close the end of a tunnel, use a zero value for the velocity or flow. The method of characteristics calculation doesn't have a problem with dead ends and setting a zero value is quicker than adding a damper to a vent shaft.

Negative values of velocity and volume flow can be used, but are not recommended as they just cause confusion. It is better to change `_inflow` to `_outflow`. But there is nothing to stop you using `v_outflow -4.5` or `q_inflow -212` (if you want to be pointlessly cruel to your colleagues, who am I to spoil the fun?).

Fixed velocities and volume flows can cause stability problems in the calculation.

If they are set as a boundary condition the pressure fluctuations in first few seconds of the calculation could cause pressure waves to propagate that cause the method of characteristics calculation to fail.

To avoid these problems, fixed velocities and volume flows increase from zero to the specified value over a specified rise time. The default value of rise time is two seconds, which seems to avoid most of the problems. See Section 3.3.19 for more details.

The `back` keyword has five optional entries: `area`, `zeta_in`, `zeta_out`, `zeta_bf` and `zeta_fb`.

If the end of a tunnel is a portal to atmosphere it has a default outflow loss coefficient of 1.0 and a default inflow loss coefficient of 0.5. These can be changed by the use of optional arguments `zeta_in` and `zeta_out`.

`zeta_in` sets the inflow loss factor and `zeta_out` sets the outflow loss factor.

At back ends `zeta_in` can be replaced with `zeta_bf` and `zeta_out` can be replaced with `zeta_fb`. This is mostly to help out those SES users who are used to figuring out which coefficients to set in SES form 3C. You don't need to remember these: if your input file has optional entries that clash, error message 2441 will catch them.

By default the pressure loss factors are applied to the air velocity in whatever area is present in the sectype at the portal. However it is not uncommon for the area of tunnel portals to be different to the tunnel they lead into.

Smaller areas at portals are often caused by architectural facades, overheight vehicle roof removal systems or concrete downstands supporting the edges of roof slabs. Larger areas may occur for a variety of reasons. Figure 3.3 shows the portals of two tunnels on the same railway and designed with the same tunnel cross-section. The image on the left has a portal the same size and shape as the tunnel interior, the image on the right has a short length of wider civil works to accommodate a set of points that starts inside the tunnel.



Figure 3.3: Different size portals

It is usually uneconomic to model very short lengths of tunnels of different area, so the program has an optional argument that allows users to set a specific area for the portal that is used with the inflow and outflow pressure loss factors (regardless of whether they are custom or default), such as `area := 45.5`. The pressure loss is applied to the velocity through the area in the optional argument instead of the area of the sectype at the portal.

3.14.4 The fwd keyword

The `fwd` keyword is similar to the `back` keyword, and all the discussion in Section 3.14.3 applies to the `fwd` keyword, with three differences:

- No sectype is needed (the forward end of a tunnel has whichever sectype was set at the last change of sectype).
- The `zeta_in` optional entry can be replaced with `zeta_fb`.
- The `zeta_out` optional entry can be replaced with `zeta_bf`.

Some examples:

<i>keyword</i>	<i>location</i>	<i>end treatment</i>	
fwd	12000	portal -15	# open with -15 Pa gauge pressure
fwd	12000	node Xover1up	
fwd	12000	v_inflow 0	# closed end
fwd	12000	v_outflow 12	# m/s in SI files, fpm in US files
fwd	12000	q_inflow 145	# m^3/s in SI files, cfm in US files
fwd	12000	q_outflow 209	

Each line starts with the keyword, `fwd`. The first number sets the location at which the tunnel ends (12,000 metres or 12,000 feet, depending on the system of units being used). Note that the location of the forward end must be higher than the location of the back end. The next two entries either set a portal and its pressure, give the name of a node/join or set a velocity or volume flow.

The five optional arguments that are valid in lines defining back ends are also valid in lines defining forward ends: `zeta_in`, `zeta_fb`, `zeta_out`, `zeta_bf` and `area`. They behave as they do in the `back` keyword.

3.14.5 The change keyword

A line of entry starting with the `change` keyword changes the sectype in the tunnel. There must be the keyword, the chainage and the name of the new sectype:

<i>keyword</i>	<i>location</i>	<i>new sectype</i>
change	10543	West_C+C2

If the area of the new sectype is different to the current one, default pressure loss factors will be applied at the area change. The default pressure loss factors are for abrupt contractions and expansions (see Section 2.9 for details of how they are calculated).

There is a special sectype name, `same`. It is used to place a boundary at a specific place in a tunnel without changing the sectype (this is useful when you are comparing calculations to test data with sensors at specific points in a tunnel).

Figure 3.4 shows two changes, one to a smaller area and the other to a larger area. It shows the default pressure loss factors for those particular area ratios ($15 \text{ m}^2/22 \text{ m}^2$ and $52 \text{ m}^2/82 \text{ m}^2$). The default figures are applied in the segment with the smaller of the two areas.

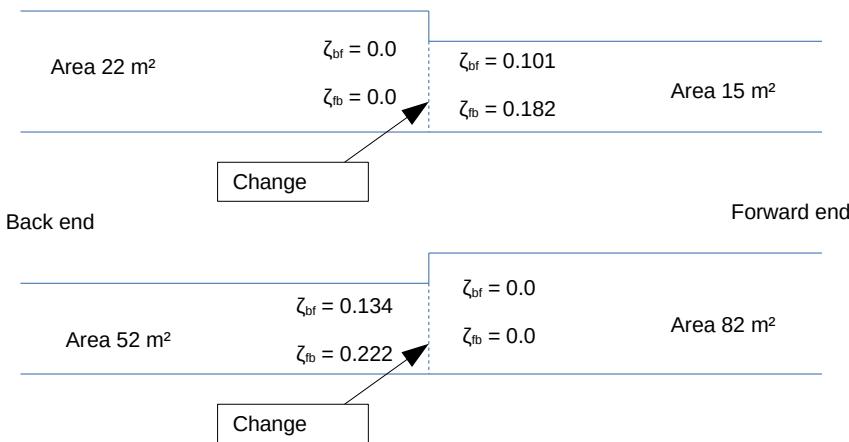


Figure 3.4: changes and default pressure loss factors

The default pressure loss factors can be changed by optional arguments. There are three: `zeta_bf`, `zeta_fb` and `area`.

If you set non-zero custom pressure loss values (`zeta_bf` and/or `zeta_fb`) you must specify which area it/they apply to in an optional entry:

- the smaller area (`area := smaller`),
- the larger area (`area := larger`),
- the area before the change (`area := back`) or
- the area after the change (`area := fwd`)

The area to use must be specified so that engineers can't assume which area it applies to. I generally use `area := smaller`.

If you use the optional arguments to set both pressure loss factors to zero, the `area` optional argument is not needed.

3.14.6 The loss1 keyword

A `loss1` keyword sets pressure losses in a tunnel. There must be the keyword, the location in the tunnel, an area that the losses apply to and two pressure loss factors (the first is for flow in the back-to-forward direction and the second is for flow in the forward-to-back direction):

<i>keyword</i>	<i>location</i>	<i>area</i>	<i>zeta_bf</i>	<i>zeta_fb</i>
<code>loss1</code>	10543	35.2	16.52	12.14

The area of the tunnel's sectype has no influence on the pressure losses at `loss1` keywords.

This may cause some angst for users who are familiar with setting pressure loss factors in programs like IDA and Aero. However, it makes sense to do this because for thirty years I've worked on fee-earning projects in which idiot engineers (including me) calculated pressure loss factors that applied to the air velocity through a tunnel of one area, then built simulations in which those factors were accidentally applied to the air velocity in a tunnel of different area.

A note about locations: the losses cannot be set too near the tunnel ends. The absolute minimum distance is 1 metre (3.28 feet) but it will probably need to be increased, depending on the aero timestep. Good figures to use are 20 m if your aero time step is under 0.05 s, 40 m if it is 0.1 s.

3.14.7 The loss2 keyword

A `loss2` keyword also sets pressure losses in a tunnel. There must be the keyword, the chainage and the pressure losses in gauls ($\text{N}\cdot\text{s}^2/\text{m}^8$, or $\text{Pa}\cdot\text{s}^2/\text{m}^6$ at air density 1.2 kg/m^3) for flow back-to-forward and flow forward-to-back:

<i>keyword</i>	<i>location</i>	<i>R_bf</i>	<i>R_fb</i>
<code>loss1</code>	10543	0.008	0.005879

A lot of tunnel ventilation engineers distrust gauls (not-invented-here syndrome, I think) and a lot of others straight-up haven't heard of the concept. If you fall into either of these categories, just use the `loss1` keyword instead and set your pressure loss factors and the area you calculated them at.

3.14.8 The damper1 keyword

The `damper1` keyword creates an entity in a tunnel whose resistance varies with runtime (exactly like a damper in the real world).

The damper definition consists of the keyword **damper1**, the location in the tunnel, the nickname of a datasource and the nicknames/numbers of four columns in the datasource to define the time, area, ζ_{bf} and ζ_{fb} .

<i>keyword</i>	<i>location</i>	<i>nickname of datasource</i>	<i>column for time</i>	<i>column for area</i>	<i>column for zeta_bf</i>	<i>column for zeta_fb</i>
damper1	10924	FID-1	time	area	zeta+	zeta-
damper1	9213	FID-2	1	2	5	6

The first example gives column names (only valid for **data** blocks), the second example gives column numbers (valid for both **data** blocks and **.csv** files).

3.14.9 The damper2 keyword

The **damper2** keyword creates an entity in a tunnel whose resistance varies with runtime but the resistances in the two flow directions are set in gauls for the two flow directions.

The input form is similar to the **damper1** keyword but instead of defining data-source columns for time, area, ζ_{bf} and ζ_{fb} the keyword defines datasource columns for time, R_{bf} and R_{fb} .

The damper definition consists of the keyword **damper2**, the location in the tunnel, the nickname of a datasource and the nicknames/numbers of three columns in the datasource to define the time, R_{bf} and R_{fb} .

<i>keyword</i>	<i>location</i>	<i>nickname of datasource</i>	<i>column for time</i>	<i>column for R_{bf}</i>	<i>column for R_{fb}</i>
damper2	10924	FID-2	time	R+	R-
damper2	10924	FID-2	1	3	4

The first example gives column names, the second example gives column numbers. Column names can only be used with nicknames that refer to **data** blocks in the input file, while column numbers must be used in nicknames that refer to **.csv** files or to **data** blocks.

I generally don't like to use the **damper2** keyword, because the resistances are interpolated linearly between the points defined, which I believe can be less realistic than varying area and zeta in the **damper1** keyword.

3.14.10 The join keyword

The **join** keyword creates a node partway along a tunnel.

The join definition consists of the keyword **join**, a distance along the tunnel and the name of the join:

keyword location join name

```
join      10920      XP1wb
join      11040      XP2wb
join      11160      XP3wb
```

The ends of other tunnels are socketed into the join by using the name of the join in the **back** or **fwd** keywords in the tunnel definition.

If you have to create a lot of similar joins in a tunnel it may be easier to use the **joins** keyword (see Section 3.14.11 below).

There are five optional entries in **join** keyword. One changes the sectype at the join (**sectype:=**).

Two set the pressure loss factors at the left side of the join (**zeta_bf1:=** and **zeta_fb1:=**).

Two set the pressure loss factors at the right side (**zeta_bf2:=** and **zeta_fb2:=**). The nomenclature is similar to the nomenclature for losses at portals (**zeta_bf** and **zeta_fb**) in the **back** and **fwd** keywords.

3.14.11 The **joins** keyword

The **joins** keyword generates multiple nodes at different locations along the length of a tunnel. The locations are set by lists or ranges (see Sections 3.7.1 and 3.7.2) and the naming scheme of the joins must incorporate numbers (to distinguish between them).

The join definition consists of the keyword **joins**, a distance along the tunnel and the namestem of the join:

```
keyword      location (list and/or range)      name stem
joins       [10920,11040,11160]           XP*wb
```

In this case the location specifier is a list of three numbers and the namestem has a * in it. The code that processes the **joins** command will generate three joins named **XP1wb**, **XP2wb** and **XP3wb** (replacing the * with a number).

This line of input generates the same data as the three lines of input in the example in Section 3.14.10.

Ranges can be used in place of the list:

```
keyword      location (list and/or range)      name stem
joins       range(10920,11161,120)          XP*wb
```

This also creates three nodes at 10920, 11040 and 11160. The **startstepcount** and **startstopcount** functions could also be used.

By chaining lists and list generators together, variable spacing can be achieved:

<i>keyword</i>	<i>location (list and/or range)</i>	<i>name stem</i>
<code>joins</code>	<code>"[80] + startstepcount(200,120, 7) + [1010]"</code>	<code>XP*wb</code>

This creates nodes for nine cross-passages, mostly at approximately 120 m intervals but with 90 m between the last two.

It is also notable that the phrase creating the list of locations is enclosed in double quotes, unlike the previous examples. The double quotes allow the list generator to include spaces, which makes it easier to read the phrase. Enclosing single quotes are also allowed. If you write the location without any spaces, quotes are not needed:

```
joins [80]+startstepcount(200,120,7)+[1010] XP*wb
```

would also work.

The ends of up to four other tunnels can be socketed into the side of the tunnel at the join by using the name of the join in the `back` or `fwd` keywords in the tunnel definition. You can also choose to socket no tunnels into the join, just have it in the model for use in a later simulation.

The default is to start numbering the joins at one. An optional entry can be used to change this:

```
joins range(10920,11161,120) XP*wb start:=20
```

This generates three joins named `XP20wb`, `XP21wb` and `XP22wb`.

Files that use the `joins` keyword typically also use the `tunnelclones` keyword (see Section 3.15) to generate the tunnels attached to the joins (which are usually escape cross-passages between two running tunnels). See the test file `ok-037-tunnelclones.txt` for an example of its use.

3.14.12 Fans in tunnels

Fans are defined in a two-step process: the definition of fan performance characteristics (in `fanchar` blocks), and the placing of the fan in a tunnel and specifying how it runs (the `fan1` or `fan2` keywords in `tunnel` blocks).

The process is explained in Section 3.17.

3.14.13 Jet fans in tunnels

Jet fans are defined in a two-step process similar to fans: the definition of fan performance in `jetfantype` blocks and the placing of a bank of jet fans in a

tunnel in `jetfan1` keywords in `tunnel` blocks.

The process is explained in Section 3.18.3.

3.14.14 SES pragmats

In each `tunnel` block (or `tunnelclones` block), there is an optional entry telling the program how to construct an SES input file from the Hobyah geometry. These entries (called pragmats) tell the program things like

- whether to generate SES line segments or SES vent segments from this tunnel,
- which range of segment numbers and node numbers to use in the SES geometry,
- which route to get the stack heights of the segments from.

The entry

```
SESpragmat    901    vent
```

in a `tunnel` block tells the program to use the number range 901–999 for the SES segments generated from this tunnel. It also tells the program to build vent segments rather than line segments.

The number must be in the range 1 to 999 unless you are generating SES input files for Aurecon SES v107.0, where the range is 1 to 9,999.

The second entry can be either `line` or `vent` and tells the program whether to turn the tunnel into SES line segments or SES vent segments.

There is an optional argument, `routename`. It tells the program which route to use to get the stack height of line segments from. It is only needed if more than one route passes through a tunnel; if only one route passes through the tunnel it is not needed.

```
SESpragmat    901    vent    routename := eastbound1
```

Examples of the use of the `SESpragmat` function can be found in the test file `ok-049-write-SES-file.txt`.

They are used to make the segments made from the eastbound tunnels be numbered in the 100 and 200 ranges, the ones in the westbound tunnel be numbered in the 300 range, and to make the escape cross-passages vent segments numbered in the 900 range.

The pragmats are picked up by the `SESdata` block, see Section 3.21.

3.15 The tunnelclones block

3.15.1 Introduction

Each **tunnelclones** block can be used to define multiple tunnels with similar names, such as XP1, XP2, XP3 etc. It is intended for escape cross-passages. All features of tunnels (dampers, fans, jet fans etc.) can be included in the **tunnelclones** definitions.

The syntax includes all the syntax of the **tunnel** block with two additional entries to control the numbering scheme: the **numbering** and **sectypes** lines of entry.

```
begin tunnelclones  XP*
    back 0  node XP*wb sectypes    zeta_bf:=2.4  zeta_fb:=2.4
    fwd 30 node XP*eb

    numbering "range(2, 23.1, 1)"
    sectypes "[closed]*10 + [open]*3 + [closed]*9"
end tunnelclones
```

In this example the name of the block is **XP***. The ***** will be replaced by a different number when each cloned tunnel is generated.

The **back** and **fwd** keywords generate a 30 m long tunnel with loss factors of 2.4 with nodes at each end.

The names of the nodes also include a ***** character; these will also be replaced by numbers, e.g. **XP1wb**, **XP2wb** etc.

In the **back** keyword the name of the sectype can be replaced by a special parameter, the word **sectypes**. This parameter means “take the name of the sectype for each clone from the list of names in the **sectypes** line of entry in this **tunnelclones** block” (see Section 3.15.3).

The other two lines in the example begin with **numbering** and **sectypes**. These are specific to the **tunnelclones** block and are explained below.

3.15.2 The numbering entry

The **numbering** keyword tells the program two things: how many tunnels to clone and what numbers to replace the ***** with in each tunnel and node name.

It consists of the word **numbering** followed by a list of numbers (the list may be given directly, generated by the list generation process described in Section 3.7 or be the name of a constant that returns a list. For example:

```
numbering [1,2,3,4,5,6,7,8,9,10]
```

```

numbering "[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]"
numbering range(1, 10.1, 1)
numbering startstepcount(1, 1, 10)
numbering startstopcount(1, 10, 1)

```

These all generate the same set of cloned tunnels, numbered 1 to 10.

Any number of cloned tunnels can be generated. The test file `ok-039-Channel-Tunnel.txt` has 193 cloned tunnels representing 193 pressure relief ducts linking the two running tunnels. The `tunnelclones` block that generates them takes up ten lines (including comments). `Tunnelclones` blocks are a useful feature that you should take advantage of.

3.15.3 The `sectypes` entry

The `sectypes` keyword tells the program which sectypes to assign to which cloned tunnels.

It consists of the word `sectypes` followed by a list of names of valid sectypes. As with the `sectypes` keyword the list may be given directly, generated by the list generation process described in Section 3.7 or be the name of a constant that returns a list. For example:

```

sectypes "[closed, closed, closed, open, open, closed, closed, closed]"
sectypes "[closed]*4 + [open]*2 + [closed]*3"

```

Both the above examples generate six cross-passages with the `closed` sectype and two with the `open` sectype.

The second of these (chained lists) is preferred where you have a lot of adits like this. It is common to have between one and three cross-passages open upwind of a fire in a fire simulation and chained lists are a good way to set them.

The count of entries in the `sectypes` list must be the same as the count of entries in the `numbering` list.

3.15.4 Inner workings

The `tunnelclones` block works by using the numbers and names in the `numbering` and `sectypes` blocks to generate a set of entries that can be processed in exactly the same way as a `tunnel` block. The `tunnelclones` block calls the routine `ProcessTunnel` multiple times.

Each tunnel definition is written to the `log` file in full, so if you want to customise the cloned tunnels you can copy the input from the `log` file to the input file, delete the `tunnelclones` block and start customising each tunnel definition directly.

3.16 The route block

3.16.1 Introduction

Each `route` block defines a path through a series of tunnels. The definitions are set up in a way that matches my experience of being given alignment data from road alignment engineering software (12d), rail alignment software (12d and Bentley Rail Track) and data from traffic modellers.

Each route may be just a list of tunnels to plot in, but various extras can be defined such as gradients, counts of traffic lanes, speed limits and train schedules.

3.16.2 The origin keyword

The `origin` keyword sets the lowest chainage in a route. This is used as the first chainage for gradients/elevations, speed limits and anything else that might affect train performance.

```
begin route
    origin 5000
end route
```

If you try to place a feature at a chainage lower than the origin, it will raise a fault.

The user can use an optional argument `elevation :=` to set the elevation at the origin. If none is used the elevation at the origin is set to zero.

```
begin route
    origin 10000    elevation := 23.2
end route
```

Each route must have an `origin` keyword.

3.16.3 The portal keyword

The `portal` keyword sets where the tunnels start along the route. This need not be a portal to atmosphere: it could be the dead end wall in an underground reversing siding. The keyword has no optional arguments.

```
begin route
    origin 10000
    portal 11000
end route
```

Each route must have a `portal` keyword, and the portal must be on the down side of the origin. See Section 2.3.2 for a refresher on the meanings of “up” and “down”.

3.16.4 The tunnels sub-block

Every route needs to be told how many tunnels it passes through. This is done with a `tunnels` sub-block inside the `route` block.

```
begin route eastbound-slip
    origin 0
    portal 1000
    begin tunnels
        EB1
        offslip
    end tunnels
end route
```

This tells the program that the route `eastbound-slip` passes through two tunnels named `eb1` and `offslip`. The names of the tunnels may be on the same line:

```
begin tunnels
    EB1    offslip
end tunnels
```

This works too.

The program figures out the tunnel orientations by comparing the names of the nodes at each end of the tunnels.

Routes can only change tunnels at `nodes`. They cannot switch tunnel at `joins`, because it is too much hassle to program the logic to figure out the path of a route through part of a tunnel.

There is one special case in the `tunnels` block: the case where the route is going through only one tunnel. This is the one case in which the program can't use the node connectivity to figure out the orientation of the tunnel in the route.

In this special case, the tunnel will be assumed to have its back end at the up end of the route. If you need to reverse it, put the tunnel name in with a minus sign prepended to it, e.g.

```
begin tunnels
    -westbound
end tunnels
```

This places the forward end of the tunnel `westbound` at the up end in the route.

3.16.5 The gradients sub-block

Gradients in routes can be set in a `gradients` sub-block or in an `elevations` sub-block (see Section 3.16.6).

Some engineers insist that gradients should be set as percentages (range -100 to $+100$), others swear that setting them as fractional values (range -1 to $+1$) is more logical. I lean slightly towards the fractional values camp, mostly because it is less ambiguous.

There may also be users who have only ever used IDA software and want to set gradients by setting height along a route. If you fall into that camp, Section 3.16.6 (the `elevations` sub-block) is what you're looking for.

The `begin gradients` sub-block forces you to state whether you want to use percentages or fractions by means of a third word on the first line. You must start the block with either

```
begin gradients percentages
or
begin gradients fractions
```

A simple example of a `gradients` sub-block is shown below:

```
begin route eastbound-slip
  origin 0
  portal 1000
  begin gradients fractions
    0.0 570 -0.05 1250 -0.01 1829 0.013 2421 0.045 2990
  end gradients
  begin tunnels
    EB1
    offslip
  end tunnels
end route
```

In this example, the gradients are pairs of numbers that define the gradient and the chainage that the gradient applies up to. The gradients are fractions, -1 to $+1$. The route is flat down to ch. 570, then changes to -0.05 (-5%) down to ch. 1250 and so on.

The last chainage in the route is 2990. If the route extends beyond ch. 2990 the associated gradient ($+4.5\%$) will be extended to the end of the route.

The list of entries can be all on one line, as in the above example. But the recommended way is to put in pairs on each line, with a crib in comments to remind you of the rules. See the example below (which uses percentages):

```
begin route eastbound-slip
```

```

origin 0
portal 1000
begin gradients percentages
  # gradient      chainage it
  # (percent)    applies down to
    0.0          9200
    -4.0         9700
    -0.5         10700
    0.5          11400
    3.0          11560
    1.2          13000
end gradients
begin tunnels
  EB1
  offslip
end tunnels
end route

```

Those two lines of comment text make things unambiguous: I use them all the time. Some engineers may prefer to put the chainage first then the gradient. If you fall into that category, you are welcome to use the `switcheroo` optional entry in the `begin gradients` line. This switches the order of the two columns; chainages before gradients. For example:

```

begin gradients percentages      switcheroo:=true
  # chainage to apply      gradient
  # a gradient down to    (percent)
    9200          0
    9700         -4.0
    10700        -0.5
    11400        0.5
    11560        3
    13000        1.2
end gradients

```

If a route definition has a `gradients` sub-block, the route definition cannot have an `elevations` sub-block (because they do the same job, just in different ways).

3.16.6 The elevations sub-block

Gradients in routes can be set in routes in an `elevations` sub-block, which sets values of elevation above a datum at different chainages.

```

begin route eastbound-slip
  origin 0    elevation:=6.5
  portal 1000
  begin elevations

```

```

      570 6.5   1250 -27.5   1829 -33.39   2421 -25.594   2990  0.011
end elevations
begin tunnels
  EB1
  offslip
end tunnels
end route

```

The elevations are pairs of numbers that define a chainage, then the elevation at that chainage. The program calculates the route gradients from the changes in elevation.

If a route definition has an **elevations** sub-block, it cannot have a **gradients** sub-block (because they do the same job, just in different ways).

Like the **gradients** sub-block, the recommended way is to put in pairs on each line, with a crib in comments at the top of the two columns, as below.

```

begin elevations
#  chainage    elevation at
#                  the chainage
  9200          0
  9700         -20.0
  10700        -25.0
  11400        -21.5
  11560        -16.7
  13000         0.58
end elevations

```

The **switcheroo** optional entry does not apply in the **elevations** block, you must have the chainage then the elevation it is at.

3.16.7 The lanes sub-block

Routes can contain stationary traffic. Part of the definition of stationary traffic is to define how many lanes there are along a route, so that stationary traffic can fill one, two, three etc. lanes. The counts of lanes are set in a **lanes** sub-block. Like most of the other sub-blocks, the best way to lay it out is to have two entries on each line with a descriptive header:

```

begin lanes
#  count      chainage it
# of lanes   applies down to
  2           5150
  3           11900
  2           13200
end lanes

```

The `switcheroo` optional argument described in Section 3.16.5 also applies here:

```
begin lanes switcheroo:=true
#   chainage      count of lanes
#                   before that chainage
    5150          2
    11900         3
    12300         2
end lanes
```

Lane counts are used to set the densities of stationary traffic.

3.16.8 The speedlimits sub-block

Routes can contain moving traffic (road or rail), and both types of tunnel may have different speed limits in different locations due to sightlines, turnouts, tight radii or stations. The `speedlimits` sub-block defines these. Like most of the other sub-blocks, the best way to lay it out is to have two entries on each line with a descriptive header:

```
begin speedlimits
#   speed
#   limit      chainage it
#   (km/h)    applies before
    80          7920
    60          9210
    80          14200
end speedlimits
```

The `switcheroo` optional argument that changes the order of the entries also applies here:

```
begin speedlimits switcheroo:=true
#                   speed limit
#   chainage      before the chainage
    7920          80
    9210          60
    14200         80
end speedlimits
```

Where speed limits are applied to trains, the lowest speed limit along the length of the train will be used, as this is how rail speed limits work in the real world.

Users of SES may know that in SES's train speed calculation the speed limits are treated differently: in SES, trains follow the speed limit in whichever sector the nose of the train is in. As a result, there is an optional argument that tells

Hobyah to mimic this behaviour (for compatibility). It is `seslimits:=on`. If this optional argument appears in the `begin speedlimits` line, trains will treat the speed limit at the down end of the train as the speed limit, even if there is a lower speed limit elsewhere along the length of the train.

```
begin speedlimits  switcheroo:=true      SESlimits := on
#                      speed limit
# chainage before that chainage
    7920          80
    9210          60
    14200         80
end speedlimits
```

The default behaviour is `SESlimits:=off`. The optional argument has no effect on the speed of road traffic.

3.16.9 The schedule sub-block

In routes that have trains, `schedule` blocks are used to generate the trains. More than one `schedule` block can be used in each route definition.

Each `schedule` block has the following entries:

- `traintype`, with the nickname of a `traintype` block,
- `downstart`, which specifies the chainage of the down end of the trains when they start,
- `times`, with a train spawn time or a list of spawn times,
- `fixed-speed`, with a fixed train speed,
- `time-speed`, with the nickname of a datasource and the numbers or names of two columns setting times and speeds at those times, and
- `dist-speed`, with the nickname of a datasource and the numbers or names of two columns setting distances and speeds at those distances.

The `traintype` keyword

The `traintype` keyword sets the type of train used by this schedule.

```
traintype      ICE3
```

The word after `traintype` must be the name of a `begin traintype` block at the top level. The name must not contain spaces and is not case-sensitive.

If you want to spawn different types of train, you will need to use one schedule for each train type and co-ordinate the spawn times in the individual schedules.

Each train schedule must have one `train-type` keyword.

The `downstart` keyword

The `downstart` keyword is used to set the location of where trains spawn. If it is not used, trains will start with their down ends at the route origin.

You must specify one (and only one) of `fixed-speed`, `time-speed` and `dist-speed` keywords.

Trains can move in either direction in routes and may stop and reverse unlimited times. Their movement is dictated by their speed profiles and information coded at station stops.

Schedules must be given a unique name (i.e. `begin schedule 24-tph`). The name can be used to figure out which train schedule spawned a given train.

The `times` keyword

The `times` keyword is followed by either one number or a list generator can be turned into a list of numbers. Some examples:

```
times 25.2      # One train launches at 25.2 seconds
times [0, 600, 1200, 1800, 2700]    # Six trains per hour (tph)
times range(0, 3600, 600) + range(3600, 10800, 300)
```

The last example gives one hour of six tph followed by two hours of 12 tph: this probably only makes sense to experienced users of other tunnel ventilation or train performance software. See Section 3.7.2 for a refresher on the `range()` function and how to mix lists and ranges.

If there is more than one train launching at the same time in a route (from different `schedule` blocks) an error message will be raised giving details of the clash.

If the list generators contain spaces, there is no need to enclose them in quotes (quotes are only needed when list generators are used in circumstances where arguments are treated as being separated by spaces, like optional arguments).

The `fixed-speed` keyword

The `fixed-speed` keyword sets constant trainspeed for all the trains in this schedule.

```
fixed-speed    45.9
```

Positive values mean that trains move down the route, negative values mean that trains move up the route (see Section 2.3.2).

In files that use SI units, the default units are km/h. In files that use US units, the default units are mph.

The units can be changed by an optional argument, `units`. In files that use SI units, the valid entries are `m/s`, `km/h`, `kmh` and `kph`. In files that use US units, the valid entries are `fpm` (feet per minute), `fps` (feet per second) and `mph`.

The optional entries `km/h`, `kmh` and `kph` all mean the same thing.

Units of `fpm`, `fps` and `mph` cannot be used in files that are set in SI units. Likewise `m/s`, `km/h`, `kmh` and `kph` cannot be used in files that are set in US units.

The following examples give the same train speed (allowing for rounding errors).

```
fixed-speed    20.0          units:=m/s
fixed-speed    72.0          units:=km/h   # Default in SI files
fixed-speed    44.739        units:=mph   # Default in US files
fixed-speed    65.617        units:=fps
fixed-speed    3937.0        units:=fpm
```

Internally the train speeds are stored as m/s.

The trains with fixed speed ignore instructions to stop at stations (this is because they are intended for software development rather than for engineering design work).

Each train schedule must have one of the `fixed-speed`, `time-speed` or `dist-speed` keywords.

The time-speed1 keyword

These are not implemented yet. They consist of the name of a datasource and the columns to use for time and speed at that time. The trains follow that curve, interpolating for speed at intermediate times.

The dist-speed1 keyword

These are not implemented yet. They consist of the name of a datasource and the columns to use for distance and speed at that distance.

They are converted internally into time-speed curves like those set directly in `time-speed1` entries.

At intermediate times the speeds of the trains are interpolated from the internally-calculated time-speed pairs, not from the distance-speed pairs.

3.16.10 The radii sub-block

When using Hobyah to build SES files it is convenient to have the ability to write data that is not used in Hobyah but is used in SES form 8C. SES form 8C sets track radius for its train performance calculation, so the **routes** block in Hobyah has a **radii** sub-block to define it.

The definition of track radius is similar to all the other sub-blocks:

```
begin radii
  # track      chainage it
  # radius    applies down to
    0          9170
    8000      13200
    3400      18400
end radii
```

Track radii are in metres. A track radius of zero signifies straight track.

The **switcheroo** optional argument described in the sections above also applies here:

```
begin radii  switcheroo:=true
  #   chainage      track radius
  #                   before that chainage
    9170          0
    13200        8000
    18400        3400
end radii
```

These values are not used in Hobyah calculations, just copied over to form 8C when generating SES input files.

3.16.11 The sectors sub-block

When using Hobyah to build SES files it is convenient to have the ability to write data that is not used in Hobyah but is used in SES form 8C. SES form 8C sets train energy sectors for its train performance calculation, so the **routes** block in Hobyah has a **sectors** sub-block to define it.

The definition of energy sectors is similar to all the other sub-blocks:

```
begin sectors
```

```

# energy      chainage it
# sector     applies down to
    1          7260
    3          9840
    0          12900
end sectors

```

The sectors are numbered 0 and above.

The `switcheroo` optional argument described in the sections above also applies here:

```

begin sectors  switcheroo:=true
  # chainage      energy sector
  #                  before that chainage
    7260          1
    9840          3
    12900         0
end sectors

```

These values are not used in Hobyah calculations, just copied over to form 8C when generating SES input files.

3.16.12 The coasting sub-block

When using Hobyah to build SES files it is convenient to have the ability to write data that is not used in Hobyah but is used in SES form 8C. SES form 8C sets a train coasting switch for its train performance calculation, so the `routes` block in Hobyah has a `coasting` sub-block to define it.

The definition of the coasting switch is similar to all the other sub-blocks:

```

begin coasting
  # coasting      chainage it
  # switch     applies down to
    0          1300
    1          2850
    0          7400
end coasting

```

If train coasting is allowed in a track section the coasting switch is 1. If it is not allowed, the coasting switch is zero.

The `switcheroo` optional argument described in the sections above also applies here:

```
begin coasting  switcheroo:=true
```

```

#   chainage    coasting switch
#           before that chainage
  1300          0
  2850          1
  7400          0
end coasting

```

These values are not used in Hobyah calculations, just copied over to form 8C when generating SES input files.

3.16.13 The regenfraction sub-block

When using Hobyah to build SVS files it is convenient to have the ability to write data that is not used in Hobyah but is used in SVS form 8C. SVS form 8C sets a track regenerative braking fraction for its train performance calculation (which over-rides the regen braking percentage set in form 9H), so the **routes** block in Hobyah has a **regenfraction** sub-block to define it.

The definition of the regenerative braking fraction is similar to all the other sub-blocks:

```

begin regenfraction
  # regenerative
  # braking      chainage it
  # fraction    applies down to
    0.243        6350
    0.172        8210
    0            17800
end regenfraction

```

The **switcheroo** optional argument described in the sections above also applies here:

```

begin regenfraction  switcheroo:=true
  # chainage  regenerative braking fraction
  #           before that chainage
    6350        0.243
    8210        0.172
    17800       0
end regenfraction

```

These values are not used in Hobyah calculations, just copied over to form 8C when generating SVS input files. It is not written to SES input files, just to to SVS files.

3.17 Fans

3.17.1 Introduction

Fans are defined in a two-step process: define the fan characteristics in a `fanchar` block, then place the fans at point in a tunnel with rules for how the fans should behave (the `fan1` or `fan2` keywords in a `tunnel` block).

The performance characteristics are defined as curves of fan total pressure against volume flowrate. Fan static pressure characteristics are not allowed as input (I hate fan static pressure).

A fan total pressure characteristic can be defined once, then used in multiple fans in different tunnels.

All fan characteristics are for fans whose characteristics cannot be changed while the fan is spinning.

Fans whose characteristics can be changed while the fan is spinning (such as variable pitch in motion fans or fans with adjustable guidevanes) are not covered. They probably could be, but these types of fan are so scarce in the tunnel ventilation field that it doesn't make sense to spend time adding them until the need arises.

3.17.2 The `fanchar` block

`Fanchar` blocks define the performance characteristic of a fan. Each block must have a unique name. Multiple `fanchar` blocks can be defined at the top level.

The name is used in fan definitions in `tunnel` blocks to pick up the fan's properties and apply it to a fan in the tunnel.

At least one `datablock` line must be provided inside the block. This has the nickname of a data block and the column names/column numbers that give the fan volume flow and fan total pressure rise. If one `datablock` entry is given it will be used for both forwards mode and reverse mode (i.e. a truly reversible fan). If two `datablock` lines are given, one must be tagged with a `direction` optional argument to identify whether it is to be used for forwards mode or reverse mode.

For example:

```
begin fanchar  trulyrev

      nickname      column      column
      of            to use for   to use for
      keyword       datasource   flowrate   total pressure rise
      datasource    fan_curve   Q          P_tot

end fanchar
```

This generates a fan characteristic named `trulyrev`. The fan definition has one line of entry (beginning with the keyword `datasource`). This sets the nickname of the datasource to `fan_curve`, sets the column name for the volume flow to `Q` and sets the column name for the total pressure rise to `P_tot`. This fan characteristic will be used for the forward flow direction and the reverse flow direction.

In files that use SI units, flow is in m^3/s and fan total pressure rise is in Pa. In files that use US units, flow is in cfm and fan total pressure rise is in inches of water gauge.

Another example:

```
begin fanchar my_char
  datasource fan_curve2    1    2  direction:=forward
  datasource fan_curve2    3    4
end fanchar
```

This generates a fan characteristic called `my_char`. The data defining forwards mode is in a datasource named `fan_curve2`: volume flow in column 1 and fan total pressure rise in column 2. The data defining reverse mode is in the same datasource, just with column 3 for flow and column 4 for fan total pressure rise.

There is no need to include the optional argument `direction:=reverse` in the second datasource statement, but it doesn't hurt to have it. If you tag the datasources as both being for the forwards mode or both for the reverse mode, the program will throw up an error.

The entries in the column of volume flows must increase as they go down the column, but the entries in the fan total pressure rise need not. This means that the program can model stall humps, as in Figure 3.5.

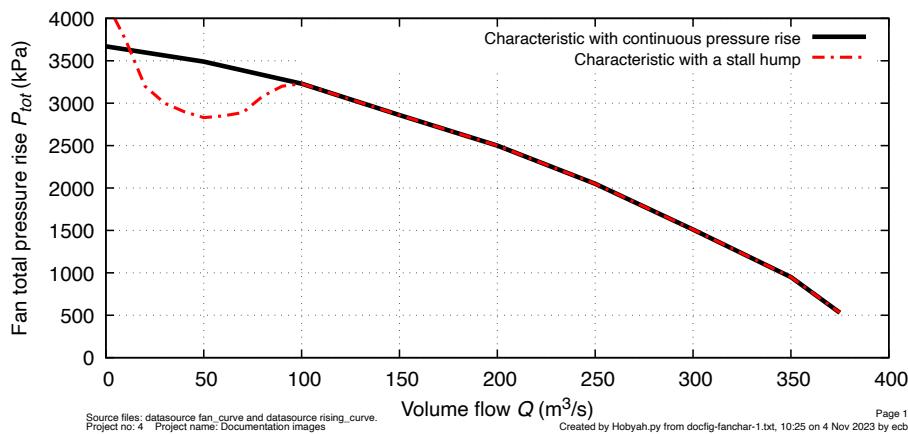


Figure 3.5: Fan characteristics with and without stall humps

There are two optional lines of entry in the `fanchar` block, the `density` keyword and the `diameter` keyword.

The `density` keyword

Most fan manufacturers give the fan performance characteristic at air of density 1.2 kg/m^3 ($\approx 0.0749 \text{ lb/ft}^3$).

But sometimes the data is given at the air density at the project's elevation. The `density` keyword is included to let you tell the program what density the fan characteristic was defined at if it was not 1.2 kg/m^3 .

```
begin fanchar  reversible
  density 1.058    # density at 1500 m above sea level (kg/m^3)
  datasource  fan_curve  Q  P_tot
end fanchar
```

It doesn't matter if the `density` keyword appears before or after the associated `datasource` keyword(s).

Be careful when using this keyword. There have been expensive legal disputes in which designers and fan suppliers claimed that the other misinterpreted their engineering documents. I would only use it on projects for which I have rock-solid evidence that the fan performance characteristic is given at a density other than 1.2 kg/m^3 .

The `diameter` keyword

The `diameter` keyword is provided so that engineers who prefer to work in fan static pressure instead of fan total pressure can see graphs of fan static pressure.

If you don't know the difference between a fan static pressure characteristic and a fan total pressure characteristic, you should not use the `diameter` keyword until you learn what the difference is. The fan diameters are in metres or feet, depending on the system of units being used in the input file.

Example `fanchar` block entries

An example of all the entries in a `fanchar` block is given below.

```
begin fanchar  comprehensive
  datasource  fan_curve  Q1  P_tot1  direction:=reverse
  datasource  fan_curve  1      4      direction:=forward
  density 1.2
  diameter 2.24
end fanchar
```

Fan characteristics can be defined in an alternative manner to the **datasource** keyword by using the **SES_7B** keyword instead. It is followed by four pairs of numbers, each pair giving values of pressure and flow (in that order).

```
begin fanchar  SESchar1
  ses_7B  3000 0    2700     80   2100    160    0    220 direction:=forwards
  ses_7B  1920 0    1730     64   1340    128    0    176
end fanchar
```

This matches the layout of the numbers defining fan characteristics in form 7B of SES.

At the present time, the spline fit that SES uses has not been copied over, so if these characteristics are used in Hobyah runs the calculation will use linear interpolation between points. So it is recommended to only use the **ses_7B** in input files that are generating SES input files.

The **SES_7B** line of entry has an optional entry that allows engineers to set the units of the eight numbers to US units or SI units regardless of the **units** entry in the **settings** block. It is included so that engineers switching between SES files in US units and Hobyah files in SI units don't have to convert the eight numbers themselves.

```
begin fanchar  SESchar2
  ses_7B  3000 0    2700     80   2100    160    0    220 direction:=reverse
  ses_7B  7.708 0   6.937 135610   5.396 271220    0  372920  units:=US
end fanchar
```

This has the same fan performance as the fan characteristic **SESchar1** above (give or take a bit of rounding in the conversion to US units).

3.17.3 The **fan1** keyword

A **fan1** keyword is used in a **tunnel** block to put a prime mover (an axial, centrifugal or mixed-flow fan) in a tunnel. All the flow in the tunnel goes through the fan.

The fan has one speed, a start time and a stop time. This is intended to match how fans are used in SES form 5C.

You must identify the location of the fan in the tunnel, a nickname that you refer to the fan in curve definitions, what fan characteristic you want the fan to use, the nickname of a datasource and the speed the fan should run at. There is an annotated example below:

```
begin tunnel  <>name>>
  <lines of tunnel definition>
```

```

      nickname      nickname
      of            of the fanchar   fan
keyword  location    fan        block to use   speed
fan1     100        GD-exh     WI-28ft      +0.667  start:=50  stop:=900

<more lines of tunnel definition>
end tunnel

```

This places a fan at distance 100 m in the tunnel, gives it the nickname `GD-exh` (for use in the `plots` block), then assigns it the fan characteristic defined in the `fanchar` block called `WI-28ft`.³ The fan is run at 2/3rds of full speed (0.667 is close enough to 2/3rds for engineering work).

If a fractional speed is given, the fan characteristic will be adjusted to the fractional speed by the fan laws, i.e. multiplying the volume flows by the speed and multiplying the fan total pressures by the square of speed.

The start and stop times are optional entries. If they are not present, the fan will start at time zero and continue running until the end of the run.

3.17.4 The `fan2` keyword

A `fan2` keyword is used in a `tunnel` block to put a prime mover (an axial, centrifugal or mixed-flow fan) in a tunnel. All the flow in the tunnel goes through the fan.

In the `fan2` keyword, the fan direction and speed is controlled by numbers in a datasource; the fan can start, stop, reverse and change speed any number of times.

You must identify the location of the fan in the tunnel, a nickname that you refer to the fan in curve definitions, what fan characteristic you want the fan to use, the nickname of a datasource and the columns in the datasource to use for time and fan speed. There is an annotated example below:

```

begin tunnel <><>
  <lines of tunnel definition>

      nickname      nickname      nickname      column      column
      of            of the fanchar   of the       to use      to use
keyword  location    fan        block to use   datasource  for time   for speed
fan2     29         CTA-N     TLT-2500-11  fanops      1          12

<more lines of tunnel definition>
end tunnel

```

³For those engineers who have an interest in history, the nicknames allude to the 28 foot diameter Walker Indestructible centrifugal fans installed in Georges Dock exhaust shaft in the Mersey Queensway tunnel. The Walker fans in Queensway have been running for around 90 years, so maybe they are indestructible.

This places a fan at distance 29 m in the tunnel, gives it the nickname CTA-N (for use in the `plots` block), then assigns it the fan characteristic defined in the `fanchar` block called TLT-2500-11.

The fan follows the speed-time curve given in the datasource named `fanops`. The time values are given in the first column of the datasource and the fractional speed values are given in the twelfth column (+1 = full speed in forwards mode, -1 = full speed in reverse mode).

Where fractional speeds are given in the datasource, the fan characteristic will be adjusted to the relevant speed by the fan laws, i.e. multiplying the volume flows by the speed and multiplying the fan total pressures by the square of speed.

The fan speed is dictated by linear interpolation between the speeds in the datasource. Be careful to not change fan speeds too abruptly, or the method of characteristics calculation may struggle to converge.

There are no optional entries associated with the `fan2` keyword.

There are no ways to place multiple fans in parallel in one tunnel. You will need to set up a tunnel system in which you have a tunnel representing the inlet plenum and one tunnel representing the each fan in parallel. See the test files `ok-032-fans-in-parallel.txt` and `ok-033-nodes-in-parallel.txt` for examples of this.

3.18 Jet fans

3.18.1 Introduction

Jet fans are defined in a two-step process: define the performance of types of jet fans in the `jetfantypes` block, then place banks of jet fans at specific locations in tunnels (the `jetfans1` keyword in a `tunnel` block).

3.18.2 The `jetfantypes` block

The `jetfantypes` block sets the properties of unidirectional and reversible jet fans and assigns a nickname to each jet fan type.

Only one `jetfantypes` block can be defined.

The input for the jet fans is their static thrust when passing air at 1.2 kg/m³, installation efficiency and jet velocity. The thrust and velocity are best taken from jet fan manufacturers' catalogue.

The installation efficiency depends partly on how close the jet fans are to the tunnel wall and partly on how much M&E stuff in the ceiling of the tunnel is placed such that it absorbs jet fan thrust—Unistrut supports, luminaires,

cable trays, pipe brackets, pipes, leaky feeder cables etc. It all depends on the design of the M&E systems. Note that all of the published papers on installation efficiency (that I know of) only deal with proximity to the wall. None assess how much thrust is absorbed by all the M&E equipment intruding into the plume downwind of the jet fan. In most of my recent designs this feature (“excess form drag of services”) has caused significant reductions in the installation efficiency. Most Australian tunnels have deluge systems in the ceiling space; the pipes and their structural supports can absorb a lot of jet fan thrust as form drag irrespective of how close the jet fan is to the tunnel roof or wall.

Each line of input in the `jetfan` block defines a type of jet fan. These may be unidirectional jetfans or reversible jet fans, depending on whether the `unidirectional` or `reversible` keyword is used.

<i>keyword</i>	<i>nickname</i>	<i>forwards static</i>	<i>forwards installation</i>	<i>forwards jet</i>
<code>unidirectional</code>	<code>fwd</code>	1100	0.7	29.9

The `unidirectional` keyword is followed by the name of the jet fan type (used when defining banks of jet fans) and three numbers: static thrust in forwards mode (N or lbf when passing air at 1.2 kg/m³), installation efficiency in forwards mode (as a fraction 0–1, not as a percentage) and the jet velocity in forwards mode (m/s or fpm).

The `reversible` keyword is similar: name of the jet fan type, three values for when the fan runs in forwards mode and three values for when it runs in reverse mode. The air velocity in reverse mode must be a negative number.

<i>keyword</i>	<i>nickname</i>	<i>forwards static</i>	<i>forwards installation</i>	<i>forwards jet</i>	<i>reverse static</i>	<i>reverse installation</i>	<i>reverse jet</i>
<code>reversible</code>	<code>rev1</code>	1080	0.7	29.6	1055	0.7	-29.2

The jet fan types are put into the model by the `jetfan` keyword in `tunnel` blocks, in the same way that fans and dampers are placed in tunnels. See Section 3.18.3.

There are two optional entries that can be given to each `jetfan`: `plumelength` and `fanlength`.

`Plumelength` sets the distance over which the plume of fast air from the jet fan decays and transfers its thrust to the air in the tunnel. Unlike most other tunnel ventilation programs, Hobyah does not model jet fans as a step change in air pressure but as a pressure change spread over a relatively long length of tunnel, similar to how traffic drag is modelled. This feature produces more accurate pressure profiles in the tunnels.

The default value (applied if there is no `plumelength` optional entry) is 80 m.

`Fanlength` sets the length of the jet fan carcass. It is used to adjust where the pressure rise of the jet fan starts. It is intended for faux jet fans: installations in which a set of large axial fans is built with manifolds, attenuators and transition pieces in a fanroom above a tunnel. Such installations draw air from the tunnel and discharge it back into the tunnel at high speed. Such facilities often have 20–40 m between the tunnel connections. They are rare, but they do exist: Epping Services Facility on North West Rail Link is one such, with connections about 30 m apart. Most jet fans are too short

to for it to be worth using the `fanlength` keyword. I wouldn't use it even with the longest jet fans I know of (1.6 m ID jet fans with 3D silencers, ~11 m long). The default value (applied if there is no `fanlength` optional entry) is 0 m.

3.18.3 The `jetfans1` keyword

The `jetfans1` keyword is placed in a `tunnel` block (see Section 3.14). It creates a bank of jet fans in the tunnel.

The keyword gives the location of the jet fans, gives a nickname that can be used to refer to the bank of jet fans in plots, the count of operational jet fans, the name of the `jetfantype` to use and the jet fan speed fraction (0 = off; 1 = full speed in forwards mode; -1 = full speed in reverse mode).

		<i>nickname of the bank</i>	<i>count of jet fans</i>	<i>nickname of the jetfantype</i>	<i>fractional rotational speed</i>
<i>keyword</i>	<i>location</i>	<i>jet fans</i>	<i>operating</i>	<i>to use</i>	<i>(-1 to +1)</i>
<code>jetfans1</code>	10294	EB-JF2	2	uni	+1
<code>jetfans1</code>	9213	EB-JF1	3	rev1	-1

The `jetfans1` keyword has four optional entries, `start:=50`, `stop:=900`, `runup:=5` and `rundown:=10`.

The default values are to start the fan at time zero and continue running until the end of the run. The runup time defaults to 5 seconds and the rundown time defaults to 10 seconds.

3.19 Traffic types

The `traffictypes` block sets the properties of road traffic. This is mostly based on the rules developed by PIARC (also known as the World Roads Association) over the years in their various guidance documents on road tunnel ventilation. This section has been written with the assumption that readers are familiar with those PIARC documents.

Only one `traffictypes` block can be defined.

The `traffictypes` block defines the following:

- drag properties of traffic types,
- whether to include or exclude a blockage correction term in the program's drag calculation,
- the number of PCUs each type of traffic occupies,
- the mass of vehicles and
- speed limits on steep upgrades

These are handled by the `vehicle`, `calculate` and `speedgradpairs` keywords. For example:

```

begin traffictypes
    calculate with blockage correction term
# keyword    name    area   C_d   PCU/veh
    vehicle    car     2.0    0.35    1
    vehicle    LCV     4.5    0.8     1
    vehicle    HGV     6.0    0.9     3    10    2      mass := 22.0
    speedgradpairs HGV    120 0.04    90 0.08
end traffictypes

```

This creates three vehicle types named `car`, `LCV` and `HGV` and sets their properties. The different lines of entry are explained below.

3.19.1 The calculate keyword

The `calculate` key tells the program whether to calculate vehicle drag with a traffic blockage correction term or without it. See Section 8.7 of the verification document for a definition of the blockage correction term. There are two valid lines of entry:

```
calculate with blockage correction term
```

and

```
calculate without blockage correction term
```

I expressed them this way because both are pretty unambiguous instructions to the program.

The reason this is a mandatory entry is because I have often had to resolve differences between calculation programs/spreadsheets written by different firms. Some programs/spreadsheets apply the blockage correction term, some don't. Having a program that forces you to figure out whether you should apply it or not makes us all more informed about this minor (but problematic) feature of traffic drag calculations in road tunnel ventilation design.

3.19.2 The vehicle keyword

The `vehicle` keyword creates a vehicle type. Each line of entry sets a name, cross-sectional area, drag factor and the count of PCUs it occupies at each speed (more on this later).

					<i>PCUs per</i>
<i>keyword</i>	<i>name</i>	<i>area</i>	<i>C_d</i>	<i>vehicle</i>	
vehicle	car	2.0	0.35	1	

The first four entries are single numbers but the last entry can have multiple numbers because for long vehicle types like HGVs, the PCU/vehicle value changes with speed.

If the last entry is one number then that value of PCU/vehicle applies at all speeds. If the last entry is not one number, it is treated as values of PCU/vehicle followed by a speed. The PCU/vehicle value applies up to and including that speed. For example:

<i>PCUs per vehicle and speeds</i>						
<i>keyword</i>	<i>name</i>	<i>area</i>	<i>C_d</i>	<i>PCU/veh</i>	<i>speed</i>	<i>PCU/veh</i>
vehicle	HGV	6	0.9	3	10	2

This means that the HGV vehicle type occupies the space of three cars at and below 10 km/h and the space of two above 10 km/h (this example is from the 2019 PIARC ventilation report). There is no limit on the count of entries.

The speeds are in km/h in SI input files and in mph in US input files.

The PIARC emission calculation tables for HGVs have a multiplier that is set by vehicle mass, while the emission calculation tables for cars does not. There is an optional entry to set vehicle mass:

<i>PCUs per vehicle and speeds</i>						
<i>keyword</i>	<i>name</i>	<i>area</i>	<i>C_d</i>	<i>PCU/veh</i>	<i>speed</i>	<i>PCU/veh</i>
vehicle	HGV	6	0.9	3	10	2
						mass := 22.0

The optional argument **mass** gives the mass in metric tonnes in SI input files and in short tons (2000 lb) in US input files.

3.19.3 The speedgradpairs keyword

The **speedgradpairs** keyword takes as input the name of a vehicle type and pairs of speeds and gradient. It is used to slow heavy vehicles on steep uphill gradients: the code that calculates traffic densities looks at the gradients on routes and if a vehicle type has a **speedgradpairs** keyword, it ensures that vehicles going up the slope never go faster than the corresponding speed in the vehicle type's **speedgradpairs** entry.

		<i>gradient</i>	<i>gradient</i>
<i>vehicle</i>	<i>it applies</i>	<i>vehicle</i>	<i>it applies</i>
keyword	<i>name</i>	<i>speed</i>	<i>up to</i>
speedgradpairs	HGV	120	0.04
		<i>speed</i>	<i>up to</i>
		80	0.06

The above means that when HGV vehicle types are climbing upgrades that are 4% or less, they can move at up to 120 km/h. On gradients above 4% and equal to or less than 6% they cannot move faster than 80 km/h. On gradients above 6% they continue be capped at 80 km/h speed (if you need to slow them further, add more pairs of speeds and gradients).

When the speed limits are calculated, the speed limits of traffic on uphill gradients are combined with the speed limits in routes; whichever speed is lower is used. This corresponds well to the real world, in which a route may have a speed limit for a small radius curve (this is particularly common in older road tunnels) while having no restrictions on gradient.

3.19.4 Traffic pollution

Alas, not implemented yet.

3.20 Putting traffic into routes

The **trafficsteady** block puts stationary and moving road traffic into routes.

Multiple **trafficsteady** blocks can be defined. If you have bidirectional traffic in one of the tunnels, it's best to have one block for traffic on all the routes in one direction (e.g. northbound) and a second block for traffic in the opposite direction.

Each **trafficsteady** block defines the following:

- which routes this block puts traffic into,
- the speed of moving traffic on a route,
- the extent of stationary traffic,
- the density of stationary traffic on a route and
- the flowrates of each vehicle type.

The first four bullet points are handled by the **routes**, **moving** and **stationary** keywords. For example:

```
begin trafficsteady
  routes EBmain EBbranch WB
  moving EBmain 60
  moving EBbranch 60
  stationary WB 165 PCU/lane-km 1000 1300
end trafficsteady
```

The line of entry starting with **routes** is followed by three route names, meaning that this block sets traffic in three routes. In this case, the three routes are called **EBmain**, **EBbranch** and **WB**. The two routes **EBmain** and **EBbranch** have moving traffic and the route **WB** has stationary traffic of density 165 PCU/lane-km that extends from chainage 1000 to chainage 1300 in the route.

Assume that we have three types of traffic called cars, LCVs and HGVs (from the example in Section 3.19).

The last entry in the bullet list above (flowrates of each vehicle type) is handled by lines of entry that start with the name of a vehicle type defined in the **traffictypes** block. This is explained in more detail in Section 3.20.4.

The specifics of each entry are covered in the sections below.

3.20.1 The routes keyword

The **routes** keyword is followed by the names of the routes that this **trafficsteady** block sets traffic in:

<i>keyword</i>	<i>name of 1st route</i>	<i>name of 2nd route</i>	<i>name of 3rd route</i>	<i>...</i>	<i>name of nth route</i>
routes	EBmain	EBbranch	WB	...	<route name>

Each route may only be used in one **trafficsteady** block. If a route appears in more than one **trafficsteady** block, an error is raised.

Any number of routes can be put into each `routes` keyword.

Routes that appear in the same `trafficsteady` block have three restrictions applied to them:

- if moving traffic in more than one route passes through the same tunnel, those routes must have the same base traffic speed in that tunnel (base traffic speed is the speed set in the `moving` keyword),
- stationary traffic in the tunnels of one route must not overlap moving traffic in the same tunnel in another route and
- where stationary traffic in two or more routes overlap, the density of stationary traffic in the routes must be the same.

The reason for these restrictions is to catch mistakes when modelling tunnels with multiple entry and exit portals like Westconnex in Sydney. It's not unusual to have to model a queue of stationary traffic in one part of a long tunnel network while traffic continues to move in other parts. It is amazingly easy to accidentally ask for stationary and moving traffic to overlap one another in the same tunnel.

When modelling traffic in single-tube bidirectional tunnels (like Mont Blanc or Saltash) you can avoid these restrictions by putting traffic into more than one `trafficsteady` block. The program assumes that routes in different `trafficsteady` blocks use different lanes (e.g. a `trafficsteady eastbound` block and a `trafficsteady westbound` block in the case of Saltash tunnel).

3.20.2 The `moving` keyword

The `moving` keyword sets moving traffic in a route. It is followed by the name of a route and a base traffic speed:

<i>keyword</i>	<i>name of route</i>	<i>traffic speed</i>
<code>moving</code>	<code>EBmain</code>	50

The speeds are in km/h in files in SI units and mph in files in US units.

The traffic speed has a limit of 120 km/h (the highest speed in the PIARC emission tables). In the unlikely case that higher speeds are needed, you can use the `hoons:=allowed` optional argument.⁴

```
moving WB 150 hoons := allowed
```

3.20.3 The `standstill` keyword

The `standstill` keyword is followed by the name of a route, the traffic density with its units, and two chainages:

⁴If you don't know what a hoon is, look it up online. `Hoons:=allowed` is the most Aussie keyword in this program.

keyword	name of route	traffic density	units of density	start chainage	stop chainage
standstill	EBmain	165	PCU/lane-km	2000	2450

The valid units for density are PCU/lane-km, veh/lane-km, PCU/lane-mile and veh/lane-mile.
The recommended unit is PCU/lane-km.

The traffic density has an upper limit of 165 PCU/lane-km (because that is what PIARC recommends). If you want to use higher densities for some reason, you can use the **highdensity** optional argument to tell the program not to raise an error:

```
standstill EBmain 170 PCU/lane-km 0 1600 highdensity :=allowed
```

Note that these values are per lane. Routes that carry road traffic must have entries that specify how many lanes of traffic they contain (the **lanes** sub-block, see Section 3.16.7). If one part of a route has two lanes and another part has three lanes, the part with two lanes will hold 330 PCU/km (165×2) of stationary traffic and the part with three lanes will hold 495 PCU/km (165×3).

The two chainages at the end of the line of entry set the extents of the queue of stationary traffic. The chainages must be above the route origin chainage but otherwise there are no restrictions (you are free to put all the traffic in the open air if you want to temporarily turn off the traffic drag in the tunnels).

One or both chainages can be replaced with the names of temporary constants that return the chainage of the up portal and the down portal in the route named in this line of entry. The names of the temporary constants are **up_ptl** and **down_ptl**:

```
standstill EBmain 170 PCU/lane-km up_ptl down_ptl
```

The **Up_ptl** and **down_ptl** constants are only valid during the processing of the **standstill** keyword. They are provided so that you don't have to look up the chainage of the up portal or calculate the chainage of the down portal by adding up the tunnel lengths (it is far better to make the program do that calculation).

When you set stationary traffic the program takes your specified density and turns it into a density for each vehicle type by looking at which routes have stationary traffic in each part of each tunnel. This calculation (which is confusing and easy to foul up) is explained in detail in **verification+validation.pdf**.

3.20.4 Setting traffic flows

The keywords (first word on a line of input) to set traffic flows cannot be known in advance by the program, as the keywords are the names of the vehicle types set in the **traffictypes** block in the input file. The names of the vehicle types are followed by the traffic flows (in vehicles per hour) in each route in the **routes** keyword of this block:

keyword (name of vehicle type)	flowrate in 1st route (veh/hr)	flowrate in 2nd route (veh/hr)	flowrate in 3rd route (veh/hr)	flowrate in nth route (veh/hr)
cars_p	1615	847	2320	...	2620

Two examples:

```
begin trafficsteady
  routes EBmain EBbranch WB
  cars    1300     800   2050
  LCVs    100      50    125
  HGVs    350     210   525
  moving  EBmain  60
  moving  EBbranch 60
  stationary WB 165 PCU/lane-km 1000 1300
end trafficsteady
```

```
begin trafficsteady
  routes EBmain EBbranch WB
  cars_p  900     650   1650
  cars_d  400     150   400
  LCVs    100     50    125
  HGVs    350     210   525
  moving  EBmain  60
  moving  EBbranch 60
  stationary WB 165 PCU/lane-km 1000 1300
end trafficsteady
```

In both examples the route names and the vehicle flowrates in those routes are lined up in columns. This is not required, but writing files in this way makes using the program so much simpler that lining up the columns is highly recommended.⁵

The first example is from a file which has three types of traffic called `cars`, `LCVs` and `HGVs` in its `traffictypes` block.

The second example is from a file which has four types of traffic called `cars_p`, `cars_d`, `LCVs` and `HGVs` in its `traffictypes` block.

Any number of routes can be put into each `routes` keyword.

It is recommended (but not necessary) to have a line of entry for every type of vehicle. If the flowrate of a particular vehicle type is zero in all routes, it may be assigned all zeros for the routes or left out entirely.

⁵The design of the `trafficsteady` block was heavily influenced by work I did between 2017 and 2020 on the tender and detail designs of Westconnex 3A in Sydney. Four of us in the design JV worked on what felt like the most complex spreadsheet on the planet, to convert the traffic modeller's work into something that the tunnel vent software we were using would accept. The `trafficsteady` block gets the program to do all the heavy lifting, by making the input to Hobyah more compatible with the output from the traffic modelling work.

3.21 The SESdata block

The **SESdata** block contains commands to control how a skeleton SES input file is generated from the Hobyah data.

The file generated may or may not run in SES. Even if it does, you will have to do a lot of editing of the SES input file before it does anything useful. Anyone using the **SESdata** block on project work should have at least ten years experience at using SES with nothing but a text editor or be under the control of an SES guru who has at least ten years experience at using SES with nothing but a text editor.

To be clear: the **SESdata** block is not intended to be used by newbie SES users (other than those under the supervision of an expert SES user). If you are a newbie to SES, using the **SESdata** block without expert guidance will likely end up with your company getting sued by a contractor due to your mistakes. And I won't help you out (see Section 1.7 for more details).

The comments at the top of the SES input file give a few pointers about what you will need to do:

OpenSES 4.3 skeleton file from "ok-049-write-SES-file.txt", a Hobyah input file. This file has a geometry (forms 2, 3, 5 & 6), fans (form 7) & routes (form 8) based on the geometry, fans and routes of the above Hobyah run. This SES file won't give correct results because it has dud values that you should adjust:

- * outside air conditions,
- * passenger mass, TES capture rules and fire simulation entries,
- * segment types (all line and vent segments are type 1),
- * stack heights. Line segments in routes have had their stack heights set from gradients in a route, which may or may not be the correct route. Most other segments have stack height zero. Please review all the stack heights.
- * fixed and steady heat gains, wetted wall perimeters & pressure loss factors,
- * wall and ground properties and temperatures,
- * node types (nodes are type 1 or 7: too many type 7s may be poor practice),
- * there are no axial fans in vent segments or jet fans in line segments,
- * train timetables, track radii, energy sectors and coasting parameters,
- * train properties (form 9 has a terribly inefficient default train type),
- * the zones (at the moment all segments are in one uncontrolled zone),
- * print timesteps, ECZ and calculation intervals, timesteps and run time.

Finally, many values permitted in Hobyah runs may be out of range in SES runs.

There are two mandatory entries in the block: **target** and **filename**. These set the type of SES input file to write (see Section 3.21.1) and the name of the SES input file to write (see Section 3.21.2).

The following is the minimum valid input for the **SESdata** block:

```
begin sesdata
  target 4.1
  filename test-SES-file.ses
end SESdata
```

There are twelve optional entries to set values for SES that either have no equivalent in Hobyah or that control how Hobyah's routes are converted into SES routes.

- **form1B** sets the three entries needed in SES form 1B,
- **form1F** sets the eight entries needed in SES form 1F,
- **form1G** sets the eight entries needed in SES form 1G,
- **3temperatures** sets the three initial temperatures in forms 3E and 5B,
- **form3F** sets the seven tunnel and ground properties needed in SES form 3F,
- **4A_locn** sets the location of an unsteady heat load (first line in SES form 4),
- **4B_heat** sets the size and duration of the fire (first half of the second line in SES form 4),
- **4B_flames** sets the radiant heat transfer properties for the flames (second half of the second line in SES form 4),
- **form7AB** sets name of a Hobyah fan definition to copy over to SES forms 7A and 7B,
- **form7C1** sets name of a Hobyah jet fan type definition to copy over to SES forms 7C,
- **form7C2** allows the user to define entries for SES form 7C directly,
- **form8** sets name of a Hobyah route definition to copy over to SES forms 8A to 8F.

Note that if you build a model that is larger than your version of SES allows, Hobyah will not complain about the counter it writes in form 1 being too high. The most likely one to blow out is form 7C (the SES arrays that hold jet fan performance data are sized to hold data for only six jet fan types).

3.21.1 The target keyword

The **target** keyword sets which type of SES input file to generate. There are five valid entries:

- **target 4.1**, meaning generate an input file for SES v4.1.
- **target openses**, meaning generate an input file for OpenSES v4.3.
- **target offline-SES**, meaning generate an input file for offline-SES v204.5.
- **target Aurecon-SES**, meaning generate an input file for Aurecon SES v107.0.
- **target SVS**, meaning generate an input file for SVS v6.6.2.

Input files for v4.1, OpenSES and offline-SES are in US customary units. Input files for Aurecon SES and SVS are in SI units, though some of the units differ. Here are the differences between the units in Aurecon SES input files and SVS input files, as a service to my future self and my former colleagues at Aurecon. Aurecon SES units first, SVS units second.

- Form 1F: atmospheric pressure is given in Pa in Aurecon SES, in kPa in SVS
- Form 3B: roughness heights are given in metres in Aurecon SES, in mm in SVS
- Form 4: heat gains are given in MW in Aurecon SES, in watts in SVS
- Form 7C: static thrusts are given in N at 1.2 kg/m³ in Aurecon SES, in N at whatever density is in the seventh entry in SVS form 7C in SVS

- Form 9D: grid diameters are given in metres in Aurecon SES, in mm in SVS
- Form 9E: static friction coefficients are given in N/kg in Aurecon SES, in N/tonne in SVS
- Form 9E: rolling friction coefficients are given in N/kg per m/s in Aurecon SES, in N/tonne-kph in SVS
- Form 9E: rotating mass resistance is given as % tare mass in Aurecon SES, in kg/car in SVS
- Form 9F: wheel diameters are given in metres in Aurecon SES, in mm in SVS

3.21.2 The filename keyword

This gives the name of the SES file to write to. For example

```
filename ok-049-v41.ses
```

If the name does not have a filename extension in it, the following are used:

- **.ses** for v4.1, Aurecon SES and offline-SES,
- **.INP** for OpenSES and SVS.

3.21.3 The form1B keyword

The **form1B** keyword is followed by three numbers giving the entries in form 1B (as you might expect): design hour, design month and a year that SES reads but does not use. For example:

```
form1B 17    7    2024
```

When the form is written to an OpenSES input file, the current input file version (4.3) is written as the fourth number of the form.

If you don't include a **form1B** entry, it will default to 17, 7 and 2024.

You can only define one **form1B** entry.

3.21.4 The form1F keyword

The **form1F** keyword is followed by eight entries for form 1F, giving temperatures, atmospheric pressure and annual amplitude. For example:

```
form1F 25.  19.  101325  23.5  18.73  28.78  20.17  4.6
```

SVS users (if there are any) should note that if the units of the Hobyah input file are SI, the outside air pressure is input in Pascals, not in kPa like you are used to using.

You can replace the outside air pressure (third number) with the nickname `p_atm`. In that case the nickname will be replaced by the atmospheric pressure set for the Hobyah run in the `settings` block:

```
form1F 25. 19.  P_atm  23.5 18.73  28.78 20.17  4.6
```

If you didn't set it in the `settings` block, the default value for `p_atm` is 101,325 Pa.

If you don't include a `form1F` entry, it will default to hot conditions based on the climate of the city of Darwin, Australia.

Only one `form1F` entry can be defined.

3.21.5 The `form1G` keyword

The `form1G` keyword is followed by eight entries for form 1G, giving passenger mass, TES capture efficiencies and transition speed and fire simulation settings. For example:

```
form1G    75    10.  12.2    14.5 12.8    96.6    1    0.2
```

If you don't include a `form1G` entry, it will default to 90.5 kg for passenger mass (\approx 200 pounds), all TES capture efficiencies set to zero and the fire simulation on:

```
form1G    100.    0.  0.    0.  0.    30.0    1    0.2
```

The passenger mass and TES capture figures are extremely conservative. They are conservative to encourage you to change them to more reasonable figures based on your experience with SES/SVS.

You can only define one `form1G` entry.

3.21.6 The `3temperatures` keyword

The `3temperatures` keyword is followed by three entries for the initial temperatures subsegment set in forms 3E and 5B. For example:

```
3temperatures  21.  25.  19.
```

This entry sets the initial wall temperatures to 21 °C, the initial air dry bulb temperature to 25 °C and the initial air wet bulb temperatures to 21 °C.

If you don't include a `3temperatures` entry, it will use the temperatures in form 1F for Darwin, Australia, which is

```
3temperatures  29.  29.  27.
```

These are extremely hot default temperatures: walls at 29 °C. You should consider changing to figures that reflect the climate of the city you are designing for, based on your experience of using SES.

You can only define one **3temperatures** entry.

3.21.7 The **form3F** keyword

The **form3F** keyword is followed by seven entries for form 3F, giving tunnel wall thickness, tunnel spacing, tunnel wall properties, ground properties and deep sink temperature. For example:

```
form3F    0.25    25.    0.934  4.903E-07   0.242  2.581E-07   25.
```

If you don't include a **form3F** entry, it will default to the values above. The deep sink temperature is for a hot climate, the tunnel wall thermal properties are for dry concrete and the ground thermal properties are for dry clay. You should review these figures based on your experience of using SES.

You can only define one **form3F** entry.

3.21.8 The **4A_locn** keyword

The **4A_locn** keyword is followed by one of the following:

- The name of a route, a chainage in the route and a description or
- The name of a tunnel, a distance in the tunnel and a description.

Hobyah figures out which SES segment and subsegment that location is in and puts the segment, subsegment and description into the SES file's first line of form 4. If the location is exactly on the boundary of a subsegment, the subsegment with the lower chainage/distance is used.

If you don't include a **4A_locn** entry, there will be no fire defined in the SES input file.

If you include a **4A_locn** entry you must have a **4B_heat** entry. If the fire simulation option switch in form 1G is not zero, you must also have a **4C_flames** entry.

You can only define one **4A_locn** entry.

3.21.9 The **4B_heat** keyword

The **4B_heat** keyword is followed by a sensible heat gain, a latent heat gain, a word defining what the units of the heat gains are, a start time and stop time. For example:

```
4B_heat  50  0  MW  20  9999
```

In this case, it means a 50 MW fire. There are three valid units texts: **MW**, **watts** and **BTU/hr**.

If you include a **4B_heat** entry you must have a **4A_locn** entry.

You can only define one **4B_heat** entry.

3.21.10 The **4C_flames** keyword

The **4C_flames** keyword is followed by a flame temperature, a number that flame area is derived from, and an entry that sets the units of the number. For example:

```
4C_flames 1090 1.5 m^2/MW
```

In this case, it means a 50 MW fire. There are four valid units texts: **m²**, **m²/MW**, **ft²**, **ft²/MW**. I have a strong preference for **m²/MW** because it means that the flame area scales whenever I change the fire size.

If you include a **4C_flames** entry you must have a **4A_locn** entry and a **4B_heat** entry.

There is no need to put in a **4C_flames** entry in non-fire runs (as the flame radiant heat calculation is turned off in non-fire runs).

You can only define one **4C_flames** entry.

3.21.11 The **form7AB** keyword

The **form7AB** keyword is followed by the name of a Hobyah fan characteristic. The entries in that fan characteristic are copied across to forms 7A and 7B. For example:

```
form7AB SESfan1
```

The curves in the fan characteristic definition must have been defined by the **SES_7B** keyword, not the more flexible **datasource** keyword. If they are not, an error message will be raised.

The fan description in form 7A includes the name of the Hobyah fan characteristic, e.g. "SESfan1" **fan characteristic**.

Multiple **form7AB** entries can be defined. SES v4.1 has a limit of 75 fan characteristic types in its arrays so you are unlikely to breach the limit.

3.21.12 The **form7C1** keyword

The **form7C1** keyword is followed by the name of a Hobyah jet fan type, the word **forwards** or the word **reverse**, a start time and a stop time to put into the SES jet fan type.

```
form7C1 JFtype2 forwards 15 1800
form7C1 JFtype2 reverse 15 1800
```

This keyword takes the jet fan thrust, installation efficiency and jet speed from the relevant part of the equivalent entry in the `jetfantypes` block. A quick reminder of what is in the `jetfantypes` block:

```
begin jetfantypes
    unidirectional JFtype1 1230 0.75 31.2
    reversible      JFtype2 1200 0.75 30.9      1185 0.75 -30.6
end jetfantypes
```

`Form7C1 JFtype2 forwards 15 1800` would take the properties of `JFtype2` in forwards mode (1200 N, installation efficiency 0.75 and 30.9 m/s) and generate a form 7C with equivalent properties.

If the target program is SES v4.1, offline-SES, Aurecon SES or OpenSES, the static thrust is converted to a volume flow in m³/s or cfm using air density 1.2 kg/m³ and the jet velocity.

If the target program is SVS, it writes the thrust in Newtons; a sixth entry for air density 1.2 kg/m³ is added; and a seventh entry of temperature derating switch is set to 1 (on).

If you try to access for the properties in reverse of a unidirectional `jetfanstype` an error will be raised, because such jet fans have no properties in reverse defined for them.

Any number of `form7C1` entries can be defined. Note that SES v4.1 can have no more than six jet fan types (for line segment types 9 to 15). Hobyah does not check if you have more jet fan types than the target program can handle.

3.21.13 The `form7C2` keyword

The `form7C2` keyword is followed by the definition of a form 7C in SES with one change: there is an extra entry for the units after the first number. Four units are allowed, regardless of the file's units:

```
form7C2 1100 N 0.75 29.2 10 1800
form7C2 31.393 m^3/s 0.75 29.2 10 1800
form7C2 247.29 lbf 0.75 29.2 10 1800
form7C2 66518 cfm 0.75 29.2 10 1800
```

If the target program is SES v4.1, offline-SES, Aurecon SES or OpenSES, the static thrust is converted to a volume flow in m³/s or cfm using air density 1.2 kg/m³ and the jet velocity.

If the target program is SVS, it writes the thrust in Newtons; a sixth entry for air density 1.2 kg/m³ is added; and a seventh entry of temperature derating switch is set to 1 (on).

Any number of `form7C2` entries can be defined. Note that SES v4.1 can have no more than six jet fan types (for line segment types 9 to 15). Hobyah does not check if you have more jet fan types than the target program can handle.

3.21.14 The form8 keyword

The `form8` keyword is deceptively simple. It consists of the keyword and the name of a route defined in the Hobyah input file:

```
form8 EBmain
```

In Hobyah, trains can reverse; in SES, they can't. So it makes sense to let SES routes be defined that are Hobyah routes in reverse by including the `orientation:=reverse` optional argument:

```
form8 WBmain      orientation:=reverse
```

The program then generates forms 8A, 8C, 8C, 8D and 8F from the entries in the Hobyah route definition.

Form 8A has a route description that states which Hobyah route it came from and whether it was reversed. It sets one train off at 9,999 seconds and a coasting speed of 60 km/h. If the SES route is reversed or the Hobyah route starts at negative chainage (unlikely), the origin is set to zero and everything else is shifted up to match.

Form 8C is put together as follows:

- chainages from the Hobyah route definition
- radii from the Hobyah route definition, or straight track if there is no `radii` block
- gradients from the Hobyah route definition for gradients or elevations
- zero for the elevations
- speed limits from the Hobyah route definition, or speed limits of 120 km/h if there was no `speedlimits` block
- coasting from the Hobyah route definition, or coasting turned off if there is no `coasting` block
- energy sector numbers from the Hobyah route definition, or all track in sector 1 if there is no `sectors` block
- regenerative braking fraction from the Hobyah route definition, or blank entries if there is no `regenfraction` block (SVS input files only)

When generating an instance of form 8C from an SES route, Hobyah calculates multipliers and offsets to the SES output that can be applied to the plots of elevations to allow the elevations from SES to be compared to those from Hobyah.

Form 8D sets no stops and one person on the train.

Form 8F figures out the SES segments that pass through the route and their orientations and sets those.

3.21.15 Other SES forms

Form 1C

Form 1C is set to do a run with a temperature and humidity calculation, printing water content. The train performance option is set to 1 and the supplementary print option set to 1. All other entries are zero.

Form 1D

The counters are calculated automatically, the count of junctions is set to 1 so that pressure changes at nodes are calculated. In SVS input files the counter of portals is removed.

Forms 101H and 1H

Aurecon SES form 101H is generated with zero for all counters and no extensions active.

SVS form 1H is set to have no extensions active and default ECZ estimate settings (cook the ground for 30 years and assume that the mean SHTC over the 30 years is half the mean SHTC over the ECZ interval).

Forms 2A and 2B

Forms 2A and 2B are set with one segment per section. The section numbers are the same as the segment numbers. Initial flows are zero.

Forms 3A to 3F

Line segment definitions have descriptions that indicate which Hobyah tunnel they came from. All line segments are of type 1 (no jet fans are set yet).

If the elevations of both nodes are known from route definitions, the stack heights are set to the difference in node elevations. If both elevations are not known, the stack height is set to zero. All lines are fire segments and in SVS they have zero for the fixed segment pressure.

Length, area, perimeter and roughness are copied over from the Hobyah segment.

If a Hobyah segment used a sectype with fixed friction factor, the SES roughness height is back-calculated from Moody's approximation at infinite Reynolds number,

$$\varepsilon = \frac{D_h}{20,000} \left(\frac{\lambda}{0.0055} - 1 \right)^3. \quad (3.4)$$

SES uses Moody's approximation to calculate friction factors in tunnels without trains in them, so that makes sense, I think.

Moody's approximation has minimum friction factors $\lambda = 0.0055$, $c_f = 0.001375$. If you try to convert a fixed friction factor that is below that, `Hobyah.py` will write a negative roughness height to your SES input file and SES will throw an error up.

The four pressure loss factors at the back end and forward end of segments are copied over. Subsegments lengths are set to be as close to 20 m as possible, with no heat gains.⁶

The building of form 3F is described in Section 3.21.7.

Forms 5A to 5D

Vent segment definitions have descriptions that indicate which Hobyah tunnel they came from are of type 1.

Form 5B is set to have no fans, even if there is an axial fan at one end of the Hobyah segment.

Subsegments lengths are set to be as close to 20 m as possible.

Initial temperatures are set as described in Section 3.21.6.

Grate area and discharge velocity limit are set to 25 m^2 and 12 m/s respectively.

If the elevations of both nodes are known from route definitions, the stack heights are set to the difference in node elevations. If both elevations are not known, the stack height is set to zero. All line segments are fire segments and in SVS have zero for the fixed segment pressure.

Area and perimeter are set from the values in the Hobyah segment. SES vent segments have no friction term, so the pressure loss due to friction in a velocity of 5 m/s is calculated and added to the pressure loss factors at the back end of the segment.

Forms 6A to 6H

Nodes are of type 0 (portal) or type 7 (zero pressure loss).

In the definition of type 7 nodes, the SES segment numbers attached at the node are printed after column 81 on the line. This is to make it easy to turn the type 7 nodes into other types of node (which you should definitely consider doing) and make it easier to build and check node diagrams.

Forms 9A to 9I

A specimen train type has been added, with typical properties for a train on the European loading gauge. The trains have high nose loss, high friction, random values for the heat gains, horrendously sweaty passengers and poor traction efficiency (about 89%).

⁶Setting faux heat gains in Hobyah routes and copying them over to form 3D is high on my "to-do" list, as it would make my professional life considerably easier.

When writing SVS input files, zero values are set for train tail loss, grid blower air velocities and the equivalent spinning mass.

User should not use this train type in SES simulations—you should build a less terrible train type in whatever way you usually do when building SES files from scratch. The only reason this train type is written out is because at least one train type is needed in SES files that have routes.

If you use this default train type, you will not win work—you will lose it. Savvier engineers than you—engineers who know how to build SES train types properly—will win the tenders you want to win, all because you didn’t know how to improve this terrible example of SES form 9.

Form 10

No instances of form 10 (trains in the system at time zero) are provided.

Forms 11A and 11B

One uncontrolled zone is provided and contains all the segments.

Form 11B is not read if there is one uncontrolled zone.

Building a list of segment numbers with eight numbers per 80-character line is time-consuming when you have to do it manually, so a list of segment numbers is given after the end of input, eight per line. SES users can use it as a basis for instances of form 11B if they need more than one zone.

Form 12 and form 13

Form 12 sets the run to plot every second and to run for 10 seconds.

3.22 The plots block

The `plots` block defines pages of graphs and looped animations. It is the most complex of all the blocks, with three levels of nesting.

A `plots` block can contain three types of sub-block: `begin page`, `begin timeloop` and `begin filesloop`.

These three types of sub-block can have two types of sub-sub-block: `begin graph` and `begin image`.

The `begin graph` and `begin image` sub-sub-blocks can contain one or more `begin verbatim` sub-sub-sub-blocks.

At each level there may be specific commands to set parameters. The general outline is indicated below.

```
begin plots
  <plot settings>
  begin page
    <page settings>
    begin graph
      <graph settings>
      begin verbatim
        <verbatim Gnuplot commands>
      end verbatim
      <more graph settings>
    end graph
    begin image
      <image settings>
      begin verbatim
        <verbatim Gnuplot commands>
      end verbatim
      <more image settings>
    end image
  end page

  begin timeloop
    <animation settings>
    begin graph
      <definitions of graphical results>
    end graph
  end timeloop
end plots
```

Each type of block below `begin plots` can appear more than once.

3.22.1 The plot settings

There are ten plot settings, as follows:

- `pagesize`—set a standard or custom page size

- **orientation**—set portrait or landscape orientation
- **plotunits**—plot the curves in SI units or US units
- **terminal**—sets the gnuplot terminal (you need to know a lot about gnuplot before you use this).
- **font**—tell gnuplot which font to use
- **fontsize**—tell gnuplot what size of characters to use
- **linewidth**—tell gnuplot what the default linewidth should be
- **basemargins**—change the default graph size on new pages
- **pngtrim**—convert the output to individual .png files (not compatible with **splitpages**)
- **splitpages**—write each page to an individual .pdf file instead of writing all pages to one .pdf file (not compatible with **pngtrim**)

These must be put directly after the **begin plots** command and before the first instance of **begin page**, **begin timeloop** or **begin filesloop**. If they are not, they will be ignored.⁷

The pagesize setting

This sets the size of the pages in the pdf output files. The default is an ISO A4 page in landscape mode.

There are five standard settings: **pagesize A4**, **pagesize A3**, **pagesize letter** (ANSI A), **pagesize ledger** (ANSI B in landscape mode) and **pagesize tabloid** (ANSI B in portrait mode).

A sixth setting (**pagesize custom**) allows the user to set the page width and page height directly, e.g.

```
pagesize    custom    11.7    9.0
```

This would set the page size to be 11.7 cm by 9 cm, because custom sizes are assumed to be in centimetres regardless of the units of the input file.

But you can set a different unit by adding an optional argument **unit**:

```
pagesize    custom    4.6    3.5    units := in
```

Valid entries for the **units** optional arguments are **cm** (centimetres), **in** (inches), **mm** (millimetres) and **pt** (points—there are 72 points in an inch)

A seventh setting of the page size (**pagesize none**) tells Hobyah to not write a **set terminal** command at all. It is provided so that experienced Gnuplot users can set custom terminal properties in **verbatim** blocks. Details of how to use it can be found in Section 7.11.

⁷See Section 7.15 for a long and tedious explanation of the reason for this restriction.

The orientation setting

This sets the page orientation (`landscape` or `portrait`) for the five standard page sizes. It is ignored if the `custom` or `none` page sizes are used.

The default is portrait mode for the `tabloid` page size and landscape for the others.

The plotunits setting

This sets the system of units to use for the plots, `SI` or `US`. The default is whichever set of units was declared in the `settings` block. If the `settings` block did not contain a `units` setting, `SI` units will be used.

The font setting

This sets the typeface to use in Gnuplot's `set terminal` command. Anything can be used, but if you misspell it (or if it is not available on a particular computer) Gnuplot will use its default. Multiple words are allowed e.g., "times new roman". The default font is `Sans`, which may map to Helvetica, Arial or another font, depending on what computer you are using.

The fontsize setting

This sets the text size to use in Gnuplot's `set terminal` command. Any number over zero can be used. The default is 16 point text, because that seems to work best on my system.

The linewidth setting

This sets the linewidth to use in Gnuplot's `set terminal` command. Any non-negative number can be used (including zero). The default is a linewidth of 1.

The basemargins setting

This sets the default location of graph borders on each page. The entry needs four numbers:

```
# left   right   base   top
basemargins 0.12  0.89  0.17  0.88
```

It follows Gnuplot's convention: bottom left corner of the page is (0, 0) and top right corner is (1, 1).

The left edge of the graph frame will be 12% of the way across the page width. The right edge of the graph frame will be 89% of the way across the page width. The base

of the graph frame will be 17% of the way up the page and the top of the graph frame will be 88% of the way up the page.

The `basemargins` setting is just a convenient shorthand for Gnuplot's four `margin` commands. The example above causes Hobyah to write the following lines to Gnuplot's `.plt` file:

```
set lmargin at screen 0.12; set rmargin at screen 0.89 # User's 'basemargins' entries
set bmargin at screen 0.17; set tmargin at screen 0.88
```

Whenever a new page is started, the graph margins are set to these values.

If the `basemargins` command is not present, the margin commands written to the `.plt` file will be

```
set lmargin at screen 0.13; set rmargin at screen 0.885 # Default graph margins
set bmargin at screen 0.17; set tmargin at screen 0.83
```

This is equivalent to having the following entry at the top of the `plots` block:

```
basemargins 0.13 0.885 0.17 0.83
```

The `pngtrim` setting

The `pngtrim` command tells Hobyah to use the ImageMagick program to generate `.png` files of each page of the output files.

If you use the entry `pngtrim false` it will include the white borders around the graphs on the page when generating the `.png` files.

If you use the entry `pngtrim true` it will trim the white borders from the pages before generating the `.png` files. This is useful for cases where `.png` files are to be included in technical reports.

The `pngtrim` command has an optional entry to set the resolution of the image in dots per inch:

```
\texttt{\{pngtrim true dpi:=144\}}
```

This would command the program to generate images with the borders stripped off with a resolution of 144 dots per inch. The default resolution is 300 dpi.

The images are written to a subfolder of the folder the `.pdf` file is in, called `images`. Each `.png` file has the name stem of the `.pdf` file it was generated from with a number appended to it. For example, an output file name `foo.pdf` would be turned into images named `foo-001.png`, `foo-002.png`, `foo-003.png` etc. in the `images` subfolder.

The `splitpages` setting

The `splitpages` setting tells the program not to put all the pages in one `.pdf` file but put the pages into separate, numbered `.pdf` files. It has two settings:

```
splitpages true
splitpage false
```

The default is `splitpage false`. The numbering scheme takes account of how many pages there are. If there are fewer than 1,000 pages the numbering will be of the form `foo-p001.pdf`, `foo-p002.pdf`, `foo-p003.pdf` etc. If there are between 1,000 and 9,999 pages the number field will be widened to four digits. If there are between 10,000 and 99,999 pages the field will be widened to five digits, and so on.

This setting is useful when `.pdf` images have to be included in L^AT_EX documents. Many of the images in this document were generated by using `splitpage true`.

3.23 The page block

Each `page` block holds all the `graph` blocks and `image` blocks on one page. Many pages make up part of a `plots` block, each `timeloop` block must have one `page` block in it and each `filesloop` block must have one `page` block in it. There is no limit to the number of graphs or images that can be plotted on a page.

If the `begin page` command has the word `ignore` after it, the page will not be processed. This is a useful way of temporarily causing a page to be ignored without having to comment out all the lines of entry in the page (using `#`) or cutting the page text and pasting it in the input file after the `end plots` block.

I tend to write `begin page # ignore` at the start of every page, because that makes it easy to activate or deactivate the word `ignore` by removing or retyping the `#` character.

The page block has one setting of its own (`pageunits`), which lasts until the end of the page. It must be set at the top of the page block (i.e. before the first graph or image).

3.23.1 The pageunits setting

This sets the system of units to use for the graphs on this page, `SI` or `US`. The default is whichever set of units is being used by the `plots` block.

The intent of this is to let two pages be defined with the same data on them, just plotted in different units.

3.23.2 Example page block

A typical page block may start with a `pageunits` settings, `begin graph` or `begin graph`:

```
begin page # ignore
pageunits US
begin image
  <Lines of input that define an image go here>
```

```
end image
begin graph
    <Lines of input that define a graph go here>
end graph
begin graph
    <Lines of input that define another graph go here>
end graph
begin image
    <Lines of input that define an image go here>
end image
end page
```

3.24 The graph block

Each `graph` block contains commands that set the details of one graph and plots curves on it.

The lines in the `begin graph` block are divided into five groups as in the example below:

```
begin plots
  begin page # ignore
    begin graph # ignore
      # Group 1
      title "Velocity profiles at 205 seconds"
      xlabel "Distance (m)"
      ylabel "Velocity (m/s)"

      # Group 2
      xrange 2000 7500 1000
      yrange -10 9 *2

      # Group 3
      margins 0.13 0.87 0.15 0.88

      # Group 4
      begin verbatim
        set key top right
        set label "{/*0.7 incident location}" at first 2900, 5 left
        set arrow from 2900, 4.5 to 3200, 3.1 head
      end verbatim

      # Group 5
      profile velocity calc eastbound@205 In Eastbound
      profile velocity calc westbound@205 In Westbound
    end graph
  end page
end plots
```

The first three groups set some titles, the axis extents and the size of the graph frame. Anyone with some experience of Gnuplot or putting graphs in spreadsheets should be able to figure out the gist of the blocks.

The fourth group is a block of lines of that are passed to Gnuplot mostly unchanged (a `verbatim` block).

The fifth group defines what curves to plot.

The following sections go through these groups in turn.

3.24.1 Setting the graph labels

There are five command words to set the graph title and the labels on Gnuplot's four axes (`x1`, `x2`, `y1` and `y2`):

```

begin graph
    title A simple graph title
    xlabel Distance (m)
    ylabel Volume flow (m^3/s)
    x2label "{/*1.1 Time (s)}"
    y2label 'Volume flow (m^3/s)' offset 0, -0.2
end graph

```

The command words `title`, `xlabel`, `ylabel`, `x2label` and `y2label` have two forms: plain and complex, because Gnuplot has some sophisticated L^AT_EX-like text formatting commands.

Plain settings are a string of words after the command word without quotes or double quotes at the beginning and end, e.g.

```
ylabel Volume flow (m^3/s)
```

The program sees that the first word after `ylabel` is not preceded by ' or " and assumes it is a plain setting with no formatting. It turns this line into a Gnuplot `set ylabel` command with double quotes around the title and Gnuplot's `noenhanced` flag set:

```
set ylabel "Volume flow (m^3/s)" noenhanced
```

The `noenhanced` flag turns off Gnuplot's formatting commands, such as superscripts and subscripts. The units text will be printed as "m³/s" rather than formatted with a cubed symbol (m³/s). See the Gnuplot documentation for details of how Gnuplot's text formatting commands work.

The `noenhanced` setting will escape all quote marks unless there is a pair of matching quote marks at the start and end of the string. This is difficult to describe in words; the following eight examples may explain how it works in practice.

<i>Entry in the .txt file</i>	<i>Entry in the .plt file</i>
<code>ylabel "Area (m^2)"</code>	<code>> set ylabel "Area (m^2)" enhanced</code>
<code>ylabel 'Area (m^2)'</code>	<code>> set ylabel 'Area (m^2)', enhanced</code>
<code>ylabel Area ("m^2")</code>	<code>> set ylabel "Area (\\"m^2)" noenhanced</code>
<code>ylabel "Area (m^2)"</code>	<code>> set ylabel "\\"Area (m^2)" noenhanced</code>
<code>ylabel Area (m^2)"</code>	<code>> set ylabel "Area (m^2)\\" noenhanced</code>
<code>ylabel Area ('m^2)</code>	<code>> set ylabel "Area ('m^2)" noenhanced</code>
<code>ylabel 'Area (m^2)</code>	<code>> set ylabel "'Area (m^2)" noenhanced</code>
<code>ylabel Area (m^2)"</code>	<code>> set ylabel "Area (m^2)\\" noenhanced</code>

The guiding rules are that enhanced strings are the ones that start and end with matching quote characters (two " characters or two ' characters) and all non-matched quote characters get preceded by a backslash so that Gnuplot knows they do not mean that the string is ended.

Complex labels are those in which the program finds a single quote or double quote after the `label` word and finds a matching quote later on the line.

When this happens, the program assumes that the user has written a valid set of Gnuplot title/label commands, e.g.

```

ylabel 'Volume flow (m\^3/s)'
ylabel "Volume flow (m\^3/s)" enhanced
y2label '{/*1.1 Volume flow (m\^3/s)}', offset 0, -0.2
y2label '{/*1.1 Volume flow (m\^3/s)}', enhanced offset 0, -0.2

```

The program checks for Gnuplot's `enhanced` or `noenhanced` flag. If neither is present then the `enhanced` flag is added.

If there are mistakes in the Gnuplot string, you will need to use Gnuplot's error messages to figure out what went wrong; Hobyah can't help you. This is likely to be pretty fragile: I can see a lot of Hobyah users who are new to Gnuplot messing up the Gnuplot text formatting commands and starting to seethe with rage as they read Gnuplot's error messages. If it's any consolation, every experienced Gnuplot user has been there, done that and got the T-shirt.

3.24.2 Controlling the limits and intervals of axes

There are four commands to set the range and grid interval of Gnuplot's four axes: `xrange`, `x2range`, `yrange` and `y2range`. The four axes in Gnuplot are the same as in most spreadsheet programs: `xrange` sets the axis at the base of the graph frame, `x2range` sets the axis at the top of the graph frame, `yrange` sets the axis at the left side of the graph frame and `y2range` sets the axis at the right side of the graph frame.

An example of each:

	#	minimum	maximum	interval
<code>xrange</code>		-100	250	50
<code>x2range</code>		0	100	20
<code>yrange</code>		-6	7	1
<code>y2range</code>		-220	130	40

The first number is the minimum value to set the axis scale at, the second is the maximum value and the third is the interval between the gridlines. A good way to remember the arrangement is to make a mental note that this is how graph extents are set in the dialogue boxes in Excel and LibreOffice.

These four lines of entry would cause the following Gnuplot commands to be written to the `.plt` file:

```

set xrange[-100:250]; set xtics 50
set x2range[0:100]; set x2tics 20
set yrange[-6:7]; set ytics 1
set y2range[-220:130]; set y2tics 40

```

Gnuplot's axis-specification commands (`set xrange[<stuff>:<stuff>]; set xtics <stuff>` and their equivalents on the x2, y and y2 axes) have extremely sophisticated autoscaling capabilities. Hobyah allows the bare minimum of access to those capabilities, in that you can precede any combination of these three numbers by * and Hobyah will tell Gnuplot to autoscale the appropriate entries:

```
xrange *-100 250 50
```

```
xrange    -100    *250    50
xrange    -100    250    *50
xrange    *-100   *250    *50
```

These cause the following to be written to the Gnuplot input file:

```
set xrange[:250]; set xtics 50
set xrange[-100:]; set xtics 50
set xrange[-100:250]; set xtics autofreq
set xrange[*:*]; set xtics autofreq
```

Putting `*` in the square brackets of the Gnuplot `xrange` term means “autoscale this entry”. So does `set xtics autofreq`. The aim of prepending the `*` to the number in the Hobyah input file to get that entry autoscaled is useful, in that it means that you can autoscale without having to remove the number you previously had for that entry.

If you want to use Gnuplot’s more sophisticated autoscaling capabilities, you can put them in a `verbatim` block (see Section 3.24.4) or on a line starting with `verbatim` (see Section 3.24.5).

3.24.3 Setting the size and location of graphs

There are seven margin commands. One sets the location of all four margins, two set the location of two margins and four set the location of one margin:

- The `margins` command sets the location of the left side, right side, base and top of the graph (in that order)
- The `lrmargins` command sets the location of the left and right sides of the graph (first left, then right)
- The `btmargins` command sets the location of the base and top of the graph (first base, then top)
- The `lmargin` command sets the location of the left side of the graph
- The `rmargin` command sets the location of the right side of the graph
- The `bmargin` command sets the location of the base of the graph
- The `tmargin` command sets the location of the top of the graph

This sets the default location of graph borders on each page. The `margins` command needs four numbers:

```
# left  right   base   top
margins 0.61  0.91    0.67  0.88
```

The example above causes Hobyah to write the following lines to Gnuplot’s `.plt` file:

```
set lmargin at screen 0.61      # From line 203
set rmargin at screen 0.91
set bmargin at screen 0.67
set tmargin at screen 0.88
```

In this case, the graph frame is in the upper-right corner of the page (Section 3.22.1 has a fuller explanation of how Gnuplot's margins are laid out, and the Gnuplot documentation has even more detail).

The other six commands are similar to the `margins` command, but with fewer entries:

```
# left right
lrmargins 0.61 0.91
# base top
btmargins 0.67 0.88
lmargin 0.61
rmargin 0.91
bmargin 0.67
tmargin 0.88
```

In practice, the last four are rarely used. But `lrmargins` and `btmargins` are useful when you need to stack graphs together vertically or horizontally; examples of both use cases can be found in the test file `ok-032-fans-in-parallel.txt`.

3.24.4 The `begin verbatim...end verbatim` block

A block of Gnuplot commands can be embedded in a `graph` (or `image`) block by using a `verbatim` block. The contents of the `verbatim` block are written to the file mostly unchanged.

```
begin verbatim
plot "--" with lines
-1500  0.2
-500   0.2
e
end verbatim
```

Backslashes (Gnuplot's symbol for continuation lines) can be used in `verbatim` blocks:

```
begin verbatim
plot "--" with lines
-1500  0.2
-500   \
      0.2
e
end verbatim
```

Comments that follow valid Gnuplot entries will be included but comments on lines of their own will not be (they were thrown away at the start of processing):

```
begin verbatim
plot "--" with lines
-1500  0.2
-500   0.2 # This comment is included in the .plt file.
```

```

e
# This comment is not included in the .plt file.
end verbatim

```

The purpose of the `verbatim` block is to let user access all the features of Gnuplot without Hobyah having to support them in the code that processes the `plots` block. I'm happy to write code in Hobyah to turn "xaxis 0 *20 2" into the Gnuplot equivalent (`set xrange[0:]*; set xtics 2`) because it makes my life easier. But I draw the line at anything more complex than that.

The `verbatim` block writes out the contents mostly unchanged. The only change that is made is to the word "centre". If "centre" appears outside of a string it is assumed to be an alignment command and is changed to "center", the North American spelling that Gnuplot accepts. Occurrences of "centre" within strings are left unchanged (I think; I didn't do extensive testing so there may be cases I missed).

In the example below the second and third occurrences of `centre` are changed to `center`:

```

begin verbatim
  set label 4 "{/*0.6 Along the centre line}" at first 1000, 43 centre
  set key top center
end verbatim

```

The first occurrence of `centre` is inside a quoted string, so it is not changed.

The `verbatim` block can also be used to annotate the graph with comments, lines and shapes. You could write an entire Gnuplot definition inside a `verbatim` block and Hobyah would write it to the `.plt` file verbatim (apart from changing some instances of `centre` to `center`).

I generally use `verbatim` blocks for text labels, arrows, lines, rectangles and polygons, as in the example block below.

```

begin verbatim
  set label 4 "{/*0.6 Along the centre line}" at first 1000, 43 left
  set object arrow 101 from first 2500, 1.5 to 2900, 3.6 nohead # line with no arrows
  set object arrow 102 from first 2500, 1.5 to 2900, 3.6 head # arrow at the end
  set object arrow 103 from first 2500, 1.5 to 2900, 3.6 backhead # arrow at the start
  set object arrow 104 from first 2500, 1.5 to 2900, 3.6 both # arrows at both ends
  set object rectangle from screen 0.48, 0.71 to screen 0.88, 0.89 fc blue # blue rectangle
end verbatim

```

The object, arrow and label numbering in your `verbatim` blocks should be kept below 2,000,000,000, because Hobyah uses all the numbers above that for numbering its internal objects, arrows and labels. See Section 7.8.2 for more detail and advice on what to do if you decide that you want to use numbers above two billion in your `verbatim` blocks.

Good examples of the use of these annotations in `verbatim` blocks can be found in test files `ok-031-fans-in-series.txt` and `ok-032-fans-in-parallel.txt`, in which they were used to draw the illustrations of the air ducts and fans. For a full set of

the options, see Gnuplot's documentation (or search on Stack Overflow, it's probably faster these days).

Finally, a warning: Hobyah's interface to `Gnuplot` is so simplified that every knowledgeable Gnuplot user will be tempted to edit the `.plt` file in the `ancillaries` subfolder. Alas, anyone making edits to that `.plt` file runs the risk of running Hobyah again (which overwrites the edited `.plt` file). I've done that more times than I can count and I'm always annoyed with myself when it happens.

3.24.5 The `verbatim` command

The `verbatim` command is a short form of the `begin verbatim...end verbatim` block. It lasts for one line. It can be followed by valid Gnuplot input on the same line (one line only, no continuation characters allowed). The input will be written to the `.plt` file. For example

```
verbatim set xrange [*<10:50*>]
```

would write

```
set xrange [*<10:50*>]
```

to the `.plt` file. This example sets one of many complex forms that the `set xrange` command can take. This one-line `verbatim` command allow users to access Gnuplot's full capabilities from Hobyah without Hobyah having to parse them.

The rules for substituting “center” for “centre” described in Section 3.24.4 apply here too. The command

```
verbatim set label 1 "/*0.6 Along the centre line}" at first 1000, 43 centre
```

would write the following to the `.plt` file:

```
set label 1 "/*0.6 Along the centre line}" at first 1000, 43 center
```

The second instance of “centre” is changed because it is outside the string. The first instance is not changed because it is inside the string.

3.25 Curve definitions

After setting the main features of a graph (where it is on the page, what the axis extents are, the graph title, the axis titles and anything else available under a `verbatim` block), you must generate a set of curves that create a Gnuplot `plot` command to print the graph.

Generating a Gnuplot `plot` command from within Hobyah is done by defining curves. Each curve is translated into a new entry in a Gnuplot `plot` command.

Each line of curve definition contains four mandatory entries. The rest of the text on the line is used as the curve's key text.

Curves from datasources start with the keyword **userdata**, the nickname of the datasource, and which columns in the datasource to use for the X and Y values:

<i>keyword</i>	<i>nickname of datasource</i>	<i>column for X</i>	<i>column for Y</i>
userdata	P_stat-100m	1	6

See Section 3.25.1 for more detail.

Anything else on the line will be treated as text for the curve in the graph's key.

<i>keyword</i>	<i>nickname of datasource</i>	<i>column for X</i>	<i>column for Y</i>	<i>Graph key text</i>
userdata	P_stat-100m	1	6	Pressure at 100 m

Curves from calculation programs start with a keyword describing the type of curve, followed by the property to plot, the nickname of the source, and where/when to plot it. For example:

<i>keyword</i>	<i>property to plot</i>	<i>nickname of source file</i>	<i>location and distance</i>	<i>Graph key text</i>
transient	velocity	file5	upline@7432	upline headwall at XYZ station

The first entry (descriptive keyword) defines what type of curve it is. This could be:

1. **transient**: A property plotted against time (or against distance on a moving train),
2. **profile**: A property plotted against distance along a route,
3. **icons**: The location/state of something (trains, fires, jet fans, adits) plotted against distance along a route at an instant in time,
4. **fandata**: Fan characteristics and system characteristics plotted on the flow-pressure plane,

The second entry is the property to plot, (**velocity** in the example above).

The third entry is the nickname of the source file (a Hobyah .hbn file or an SES .sbn file in the **files** block).

The fourth entry specifies where to take data from and (in some cases) at what time to take the data.

All curves have optional entries. These are used to select which axis to plot on, to scale & offset the X and Y values and clip the values on the X axis: **axes**, **xmult**, **xdiv**, **xoffset**, **ymult**, **ydiv**, **yoffset**, **xstart** and **xstop**. See Section 3.25.6 for more details.

The different curve types are described in the sections below.

3.25.1 Userdata curves

Curves from **.csv** files or **datablocks** start with a descriptive keyword, followed by the nickname of the source and the two columns to take X values and Y values from:

<i>keyword</i>	<i>nickname of datasource</i>	<i>column for X</i>	<i>column for Y</i>
userdata	my_csv_source	3	6

The program looks up what the nickname **my_csv_source** is assigned to and finds that it is a **.csv** file. It opens the **.csv** file assigned to the nickname **my_csv_source** and reads in its columns. It uses the contents of the 3rd column as the X-values and the contents of the 6th column as the Y-values.

<i>keyword</i>	<i>nickname of datasource</i>	<i>column for X</i>	<i>column for Y</i>	<i>Graph key text</i>
userdata	no_headers	2	4	A userdata curve

The program looks up what the nickname **no_headers** is assigned to and sees that it is a **data** block. It uses the contents of the 2nd column as the X-values and the contents of the 4th column as the Y-values.

The rest of the text on the line is used as the entry in the graph key.

<i>keyword</i>	<i>nickname of datasource</i>	<i>column for X</i>	<i>column for Y</i>
userdata	with_headers	time	area

The program looks up what the nickname **with_headers** is assigned to and sees that it is a **data** block in the input file. It looks for the column names **time** and **area** and converts them into their associated column numbers. It uses the contents of the column named **time** as the X-values and the contents of the column named **area** as the Y-values.

You can refer to column numbers directly even if the **data** block had column names. If you knew that the column named **time** was the sixth column, then **userdata with_headers 6 area** would give the same result.

.Csv files do not have column names in them. When you refer to a **.csv** file only column numbers will be accepted in the third and fourth entries.

The column numbers start at 1, not at zero.

3.25.2 Transient curves

Transient curves may be plotted at a fixed point in the geometry or at a point at a fixed distance from a train's up end or down end.

<i>keyword</i>	<i>property to plot</i>	<i>nickname of source file</i>	<i>tunnel name and distance</i>
transient	velocity	Patchway-1980	Old@300

This would plot the air velocity against time at whichever gridpoint is closest to 300 m in the tunnel named **Old** in whichever file in the **files** block was given the nickname **Patchway-1980**.

The first entry is just the keyword telling the program to plot against time.

The second entry sets the property to plot. The available properties depend on the program being plotted from. In Hobyah the properties are typically celerity, velocity, pressure, density, and volume flow. In SES they are typically a velocity, volume flow, temperature and train speed. The full list of Hobyah plottable properties is given in the text of error 7041 and a similar list for SES is given in the text of error 5041, so a good shortcut to find out what can be plotted from a given file is to give a nonsense word for the property and look at the list of correct words in the error message.

The third entry sets the nickname of the source of the data. This could be the nickname of an SES file or Hobyah file (the nicknames are set in the `files` block, see Section 3.9) or the nickname `calc`, which means “use the data in the calculation run by this Hobyah input file”.

The fourth entry sets where to plot. In Hobyah, fixed locations can be defined in a number of ways:

- At a chainage along a route e.g., `eastbound@100`. In this example `eastbound` is the name of the route and 100 is the chainage on the route.
- At a distance along a tunnel e.g., `mainline1@100`. `Mainline1` is the name of the tunnel and 100 is the distance. (I don’t want to call this a “chainage”, to avoid confusion with routes but they are essentially the same thing).
- At a named entity, like a node, join or fan (just use the name of the entity, like `sup-fan-12`).
- A distance from the up end of a train e.g., `train14@up+5`.
- A distance from the down end of a train e.g., `train20@down-2`.

In SES, fixed locations may be defined slightly differently:

- At a chainage in a route e.g., `route4@1400`.
- In a given section e.g., `sec504`
- In a given segment e.g., `302`
- In a given subsegment e.g., `302-5`
- In a given subpoint⁸ e.g., `302-5b`
- A distance from the up end of a train e.g., `train14@up+5`.
- A distance from the down end of a train e.g., `train20@down-2`.

In both programs, distances along trains may be measured from the down end or the up end. Positive offsets go down the route and negative ones go up the route. The offset must start with “up” or “down” to identify which end of the train to use and be followed by a number that includes a positive or negative sign. For example:

- `down+1` means 1 metre (or foot) in front of the down end of the train, in the open tunnel
- `up-1` means 1 metre (or foot) behind the up end of the train, in the annulus
- `down-1` means 1 metre (or foot) in front of the up end of the train, in the open tunnel

⁸A `subpoint` is the term used to describe the back end, midpoint or forward end of a subsegment. For more details, see Section 6.8.

- `up+4` means 4 metres (or feet) in front of the up end of the train, in the annulus

I would much prefer to be able to use the terms “nose” and “tail” but (as explained earlier), “nose” and “tail” cannot be used to define train ends because they switch places when trains reverse in Hobyah files. As an aside, when SES prints the locations of trains, it prints the location of the down end of the train.

3.25.3 Profile curves

Profile curves are plotted along a route (Hobyah and SES), a tunnel (Hobyah only) or a section or segment (SES only).

<i>property</i>	<i>nickname</i>	<i>tunnel name</i>
<i>keyword</i>	<i>to plot</i>	<i>of source file</i>
profile	vcold	fire_run2
		route5@1800

The second entry sets the property to plot. The available properties depend on the program being plotted from. As described in Section 3.25.2, the best way to see what properties are available is to give a nonsense word for the property and read the text of error 7041 or 5041.

The third entry sets the nickname of the source of the data. This could be the nickname of an SES file or Hobyah file (the nicknames are set in the `files` block, see Section 3.9) or the nickname `calc`, which means “use the data in the calculation run by this Hobyah input file”.

The fourth entry sets which entity to plot along and what time to plot at. This applies to both Hobyah and SES.

- Along a route at a time e.g., `mainline1@100` (Hobyah) or `route5@1800` (SES). In this example `mainline1` is the name of the route and 100 is the time (seconds) to plot at. The first example has the same text as the example in Section 3.25.2 but the number is a time instead of a distance. When plotting properties that don’t vary with time (like gradients) the time is ignored, although the @ and a number need to be included.
- Along a tunnel at a time e.g., `mainline1@100`. The rules that apply to routes also apply to tunnels.

3.25.4 Fandata curves

Fandata curves are plotted along in the flow-pressure plane (volume flow on the X-axis and pressure on the Y axis). The layout of the curve definition is the same as most of the other curve definitions: four entries (keyword, what to plot, source file, fan identifier@time).

<i>property</i>	<i>nickname</i>	<i>fan name</i>
<i>keyword</i>	<i>to plot</i>	<i>of source file</i>
fandata	fanchar	a2w-calc4
fandata	fanchar	file12
		CTA-N@125 Hobyah calculation
		921@125 SES calculation

In Hobyah, fans have names (like `CTA-N` in the example above). In SES fans are in vent segments, so the vent segment number is given (like 921 in the example above).

Fanchar is a property that plots fan total pressure rise against volume flow. The source file has the nickname **a2w-calc4** and the fan characteristic is plotted at a fan named CTA-N at 125 seconds. The program takes the characteristic for that fan and adjusts it according to the speed of the fan at 125 seconds.

System is a property that plots a curve of $P_{tot} = RQ^2$, where R is set such that the curve passes through the fan duty point. It is usual to plot the fan characteristic and system characteristic together using curve definitions like the following:

```
fandata  fanchar      a2w-calc4   CTA-N@125
fandata  system       a2w-calc4   cta-N@125
```

Figure 3.6 is an example of these two curves plotted together.

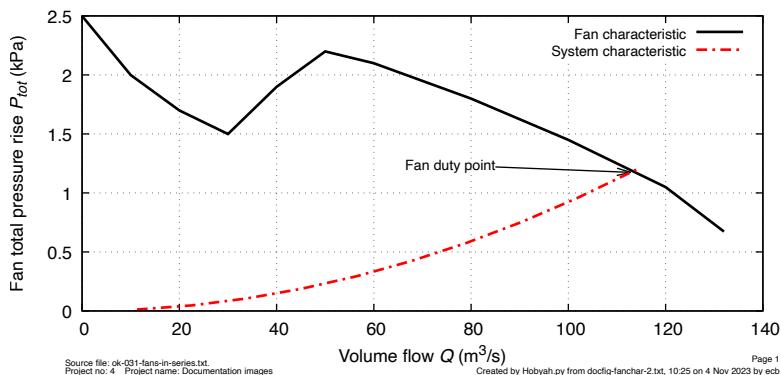


Figure 3.6: Fan and system characteristics of fan CTA-N at 125 seconds

The curves plotted in Figure 3.6 are total pressure characteristics. There is another way to look at fan pressures (fan static pressure). If you have set the fan diameter in the **fanchar** block (see Section 3.17.2) you can plot fan static pressure and system static pressure instead of fan total pressure and system total pressure.

Figure 3.7 shows total pressure characteristics and static pressure characteristics side by side. These graphs are for the same fan, in the same system, at the same instant in time.

The only reason the fan static pressure rise and system static pressure can be plotted is so that I can use Figure 3.7 to explain to my engineers why they should never use fan static pressure.

The keywords used to plot the curves on the right-hand graph are **fanchar-cursed** and **system-cursed**, just to emphasize how bad an idea it is to use fan static pressure rise.

3.25.5 Icons curves

The **icons** curve type does not plot a curve; it generates a set of Gnuplot objects (polygons, arrows, etc.) that are written the **.plt** file and are drawn when the graph is generated. For example:

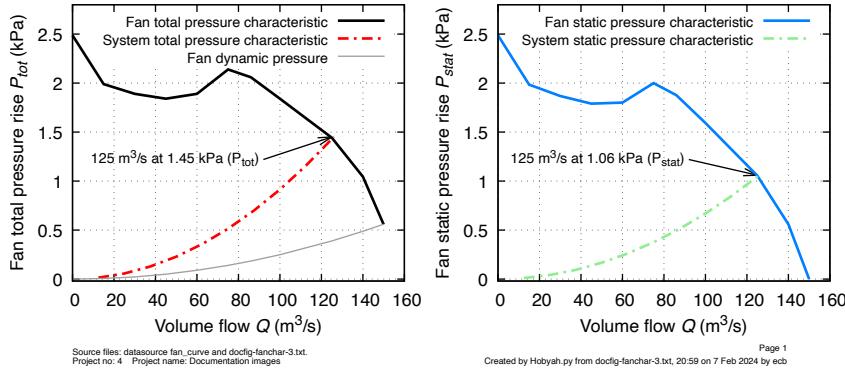


Figure 3.7: Total and static pressure characteristics (same fan, same duty point)

<i>keyword</i>	<i>type of icon</i>	<i>source nickname</i>	<i>route and time</i>
icons	fires	SES_run1	route5@500
icons	trains	SES_run1	route5@500
icons	jetfans	SES_run1	route5@500

There are three types of icon: trains, fires and jet fans. All can be seen in Figure 3.8. The icons are placed at their locations on the vertical profile in a route, so it makes sense to plot vertical profile too.

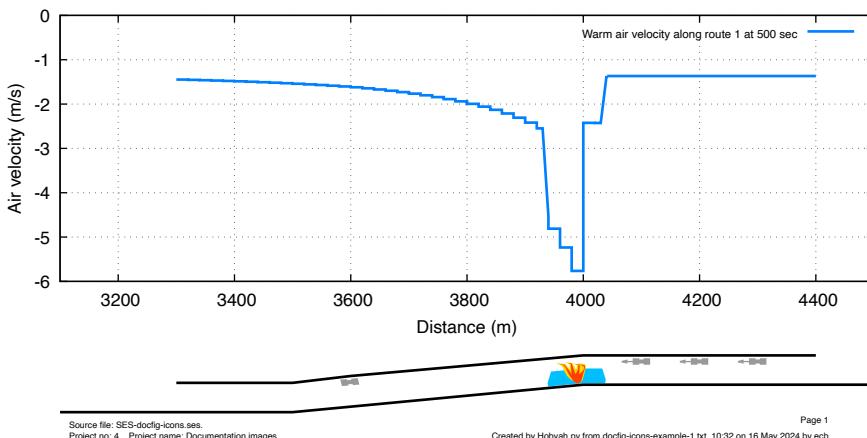


Figure 3.8: Examples of train, fire and jet fan icons

The aim of plotting icons is to let engineers see where things are in a route and what they are doing. For example, the `jetfans` icon has three types: the jet fan carcass on its own, the carcass with an arrow pointing up the route and the carcass with an arrow pointing down the route. These correspond to a jet fan that is off, a jet fan blowing up the route and a jet fan blowing down the route. Visual cues like this are invaluable for quickly figuring out whether you set some of the jet fans blowing the

wrong way in the input file, or turned them on at the wrong time.

Icons can also be plotted relative to the graph frame rather than on a vertical profile using the optional argument. The default is to place them near the bottom of the frame, as in Figure 3.9 (in this example the flow direction has been reversed by activating a different bank of jet fans to blow in the other direction; note the direction of the flames compared to Figure 3.8).

Icons are particularly useful in timeloop blocks (see Section 3.27), where they can be used to show the state of the system at each instant in time and animate the movement of trains.

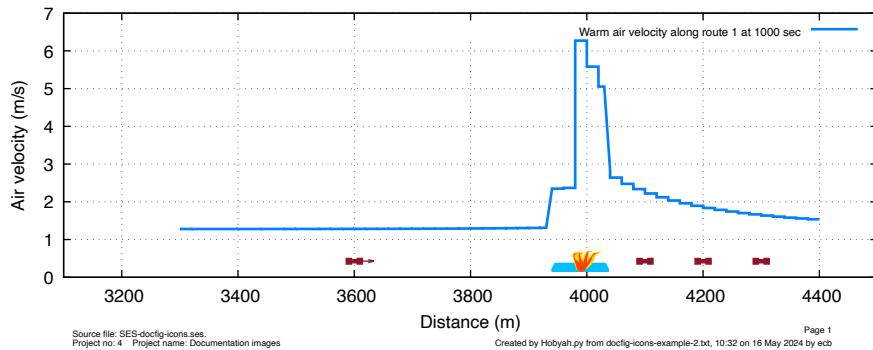


Figure 3.9: Examples of icons on the graph frame

Likewise, the fire icon can show flames going straight up (meaning that the airflow past the fire is 0.5 m/s or less), or bent to one side or the other to indicate the direction of airflow past the fire. The size of the fire icon is large when the fire is burning and small when it is not.

There are four mandatory entries on the `icons` line, which are similar to the entries in `profile` curves:

keyword	type of icon	source nickname	route and time
<code>icons</code>	<code>fires</code>	<code>SES_run1</code>	<code>route5@500</code>
<code>icons</code>	<code>trains</code>	<code>SES_run1</code>	<code>route5@500</code>
<code>icons</code>	<code>jetfans</code>	<code>SES_run1</code>	<code>route5@500</code>

The second entry sets what kind of icon to plot.

The third entry sets the nickname of the source of the data. This could be the nickname of an SES file or Hobyah file (the nicknames are set in the `files` block, see Section 3.9) or the nickname `calc`, which—like all the other plot types—means “use the data in the calculation run by this Hobyah input file”.

The fourth entry sets which route to plot along and what time to plot at. This applies to both Hobyah and SES, although the wording is slightly different as routes in Hobyah have names and routes in SES have numbers.

- Along a Hobyah route at a time e.g., `eastbound@100`
- Along an SES route at a time e.g., `route5@120`

Icons have specific optional entries that control how the icons are displayed:

- `profile`, `float`, `height`, `colour`, `color`, `aspect` and `length`

The `profile` optional entry

The `profile` optional entry takes one of three values: `route`, `stack` and `flat`.

If `profile:=route`, the icons are plotted on the Y axis at the height set by the set by the route vertical profile. This means that if a fire is at 1000 m along a route and the route elevation there is +2.2 m, the base of the fire icon is plotted on the graph axes at X=1000, Y=2.2.

If `profile:=stack`, then the icons are plotted at the height the vertical profile set by the profile developed from the stack heights in SES segments. When plotting this type of curve from Hobyah data the vertical profile is the route profile but only the underground part of the route profile is shown.

If `profile:=flat`, then the icons are plotted along the base of the graph frame, independent of the values on the Y axis.

The default is `profile:=route`.

The `float` optional entry

The `float` optional entry sets a value that moves the icon up from its base position. If the `profile` setting is `route` or `stack` the value is a distance in metres (or feet, if you are plotting in US units). If the `profile` setting is `flat`, the value is a fraction of the height of the graph frame.

For example, `float := 0.5` would move the icon 0.5 m up if the `profile` optional entry is `route` or `stack`. If the `profile` optional entry is `flat`, it would move the icon half-way up the height of the graph frame.

The `colour/color` optional entries

The `colour` and `color` optional entries set the colour of the train icons and the jet fan icons (the fire icon's colours are always red and orange).

The value to give is a six digit hexadecimal RGB value (e.g. `colour:=F926a7`) or the name of a valid gnuplot colour name (e.g. `colour:=red`).

Gnuplot has a number of predefined names of colours (but note that different versions of gnuplot may have different names). To find the available colour names, run gnuplot in an interactive terminal and use the `show colordnames` command. The first few lines of output are given below.

```
gnuplot> show colordnames
There are 111 predefined color names:
white          #ffffff = 255 255 255
```

<code>black</code>	<code>#000000 = 0 0 0</code>
<code>dark-grey</code>	<code>#a0a0a0 = 160 160 160</code>
<code>red</code>	<code>#ff0000 = 255 0 0</code>
<code>web-green</code>	<code>#00c000 = 0 192 0</code>
<code>web-blue</code>	<code>#0080ff = 0 128 255</code>
<code>dark-magenta</code>	<code>#c000ff = 192 0 255</code>
<code>dark-cyan</code>	<code>#00eeee = 0 238 238</code>

If both options are given, the value assigned to `colour` is used.

The length optional entry

The `length` optional entry sets the width of fire icons and jet fan icons. It does not affect train icons (which always have the length of the train they represent).

`Length:=50` would assign a width of 50 m to a fire icon or jet fan icon (or 50 ft when plotting in US units). The defaults are 90 m for fire icons and 40 m for jet fan icons.

3.25.6 Curve optional entries

Optional entries control which gnuplot axes to plot on, the gnuplot linestyle and line thickness and can scale, translate and clip the curve being plotted.

Selecting the axes

Gnuplot has two X axes and two Y axes. You can select which combination to plot a curve on with the `axes` optional entry:

- If there is an optional entry `xaxes:=x1y1`, the curve is plotted on the X-axis at the base of the graph frame and the Y-axis at the left hand side of the graph frame.
- If there is an optional entry `xaxes:=x1y2`, the curve is plotted on the X-axis at the base of the graph frame and the Y-axis at the right hand side of the graph frame.
- If there is an optional entry `xaxes:=x2y1`, the curve is plotted on the X-axis at the top of the graph frame and the Y-axis at the left hand side of the graph frame.
- If there is an optional entry `xaxes:=x2y2`, the curve is plotted on the X-axis at the top of the graph frame and the Y-axis at the right hand side of the graph frame.

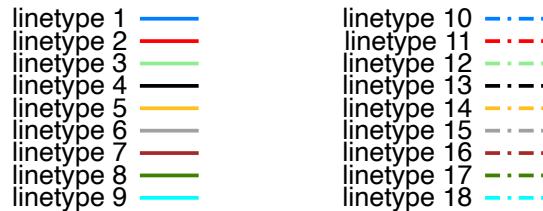
The default (if you don't have an `xaxes` optional entry) is `xaxes:=x1y1`.

Selecting the line type and line width

The gnuplot line type number and line width can be selected by the `lt:<num>` and `lw:<num>` optional argument (“linetype” and “linewidth” in gnuplot parlance

respectively—check gnuplot’s documentation for more details). <Num> represents an integer.

Gnuplot has a large set of predefined linetypes. At the start of each .plt file Hobyah sets the definition of linetypes 1 to 18 to linestyles that I prefer. These are shown in Figure 3.10.



Source file: datasource_line.
Project no: 4 Project name: Documentation images

Created by Hobyah.py from docfig-lineties-example.txt, 18:28 on 18 Jun 2024 by ecb
Page 1

Figure 3.10: Hobyah’s 18 predefined linetypes

Any linetypes over 18 will be gnuplot’s default linetypes (which vary with the terminal type).

You can define your own temporary linetypes (gnuplot’s `set style line` command) and permanent linetypes (gnuplot’s `set linetype` command) inside a `verbatim` block. The temporary linetypes are reset on each new page.

The values on the X axis can be adjusted and clipped by five optional arguments:

- If there is an optional entry `xmult:=<num1>`, the X values in the source are multiplied by <num1>.
- If there is an optional entry `xdiv:=<num2>`, the X values in the source are divided by <num2>.
- If there is an optional entry `xoffset:=<num3>`, the X values have <num3> added to them.
- If there is an optional entry `xstart:=<num4>`, the X values below <num4> are not plotted.
- If there is an optional entry `xstart:=<num5>`, the X values above <num5> are not plotted.

The sequence of operation is as follows:

1. The multiplier and divisor are applied first,
2. The offset is applied second,
3. The clipping rules are applied last.

The values on the Y axis can be adjusted by three optional arguments:

- If there is an optional entry `ymult:=<num1>`, the Y values in the source are multiplied by <num1>.

- If there is an optional entry `ydiv:=<num2>`, the Y values in the source are divided by `<num2>`.
- If there is an optional entry `yoffset:=<num3>`, the Y values have `<num3>` added to them.

The sequence of operation is as follows:

1. The multiplier and divisor are applied first,
2. The offset is applied second.

3.26 The image block

Image blocks can be used to include pictures on the page (.jpg files, .png files or .tiff files). They can be used for things like company logos, photos or a scan of test data from a technical paper.

Images are plotted by sending Gnuplot commands to turn off the annotations (axis numbers, axis labels, titles, key, frame, grid) then generating a standalone Gnuplot graph that plots the image.

Examples of all the capabilities of the block are given in the test file `ok-011-images.txt`.

The block has three keywords: a setting-out point, a width, and the filename of the image to plot. An example:

```
begin image
    leftbase 0.9 0.85
    width 0.05
    filename running-right.png
end image
```

This places the image named “running-right.png” up in the top right-hand corner of the page. The bottom left corner of the image is 90% of the way across the page and 85% up from the base. The width of the image is 5% of the width of the page. The height of the image will be set by the image’s aspect ratio.

There are five ways of defining the location of the image and two ways of setting its size, along with several optional arguments. These are described below.

3.26.1 Location and sizing

Five setting-out points are available: `lefttop`, `leftmid`, `leftbase`, `midbase` and `rightbase`. Their locations are illustrated in Figure 3.11.

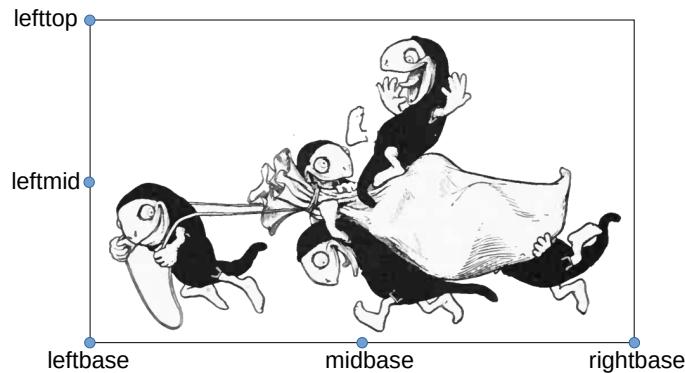


Figure 3.11: Image setting-out location keywords (with Hobyahs)

This means that one of the following five entries must be in the block:

<i>basepoint</i>	<i>width</i>	<i>height</i>
<i>keyword</i>	<i>fraction</i>	<i>fraction</i>
leftbase	0.95	0.05
midbase	0.95	0.05
rightbase	0.95	0.05
leftmid	0.95	0.05
lefttop	0.95	0.05

The names are easy to forget. A useful mnemonic is to note that the first part of each keyword sets a location along the X axis, namely **left**, **mid** and **right** followed by a location along the Y axis (**base**, **mid** and **top**). This matches the arrangement of the two numbers after the keyword (X first, then Y).

As an aside, I did briefly look into whether the four other logical setting-out points could be used (tentatively named **midmid**, **midtop**, **rightmid** and **righttop**) but there doesn't seem to be any way to implement them.

The two numbers after the keyword are the X and Y values that the relevant location on the image border is fixed at. The locations are fractions of the gnuplot page width and height respectively.

Optional arguments

Leftbase, **midbase**, **rightbase**, **leftmid** and **lefttop** all have four optional arguments; **ratio**, **border**, **namecheck** and **rotate**.

The optional argument **ratio** sets the aspect ratio of the image. An aspect ratio of 1 makes the image square, aspect ratios under 1 makes the image wide and short, aspect ratios above 1 makes the image tall and narrow. Aspect ratio **-1** means “use the aspect ratio of the image” (this is the default).

The optional argument **border:=on** causes a black border to be drawn around the image. The default is **border:=off**.

The optional argument **namecheck:=on** causes the name of the image to be added to the list of sources that appear in the page footer. It is useful when an image has technical content, like Figure 1.1. The default is **namecheck:=off**, which is appropriate for images like company logos and Figure 3.11.

The optional argument **rotate:=<some value>** rotates the image around its centroid. The values are in degrees. Positive values rotate the image clockwise, negative values rotate it widdershins. The rotation is done by Gnuplot, which appears to have a minimum rotation angle of 0.01 radians ($\approx 0.0573^\circ$).

3.26.2 The **width** and **height** keywords

Two ways of defining the size of the image are provided: **width** and **height**. Only one may be used, not both.

By default the image will keep its natural aspect ratio. If a width is specified the height will be adjusted to match the aspect ratio: if a height is set the width will be

adjusted instead. The values are fractions of the page width or page height.

Pointing to image locations

If only the name of an image file is given (no path) the image is assumed to be in the same folder as the input file (note that this may not be the current working directory).

Full file paths are also accepted, such as `/Users/JDB/Documents/Hobyahs.png` or `C:\Users\John.Batten\Documents\Hobyahs.png`.

If a partial file path is given (such as `../..\\Hobyahs.png`) is given, the directory that the input file is in is prepended to it before passing it to Gnuplot. The use of `..\\` twice means that the image will be looked for two folders up from the folder with the input file in it, which can be handy.

If the file path starts with a `~` (the symbol used to represent a user's home directory on linux and macOS) Hobyah will figure out the current user's home drive and home directory from environment variables and substitute it for the `~`. This is probably the most useful form for files that have to be passed between different users.

3.27 The timeloop block

The `timeloop` block defines one page and plots it at multiple times. This is useful because a sequence of pages with the same curve plotted at different times can be made into a flipbook or video sequence that is much more informative than a series of graphs. See `ok-032-fans-in-parallel.pdf` or `ok-032-fans-in-parallel.gif` for an example of the output of a `timeloop` block.

All the settings used in the `page` block can be used in the `timeloop` block (`pageunits` and `pageorientation`).

If the `begin timeloop` command has the word `ignore` after it (as in `begin timeloop ignore`) the timeloop will not be processed.

The `timeloop` block has three mandatory settings: a start time, a finish time and a timestep. These must appear before the `first` graph or image defined in the timeloop, for the reasons given in Section 7.15.

```
begin timeloop
  start 120
  stop 360
  step 5
  begin graph
    <lines of graph definition>
  end graph
end timeloop
```

To make it easy to remember the words all begin with `st`: `start`, `stop` and `step`. They can be in any order.

This example produces a `.pdf` file with 49 pages in it. The first page shows the state

of the system at 120 seconds, the second page shows the state at 125 seconds and so on until the last page showing the state at 360 seconds.

If a given time in the range is not in the output file, the page will be plotted at the closest time below the requested time.

Curve definitions inside the **timeloop** block uses a special variable to refer to the time of the current page. This special variable is named ***time** and is used in the curve definitions that require the user to set a time. For example:

```
profile  qwarm  source_file  route5@*time
icons    trains   source_file  route5@*time
```

When processing the first page of our example file, ***time** would be set to 120 seconds. This line of input would be turned into

```
profile  qwarm  source_file  route5@120
icons    trains   source_file  route5@120
```

Which means “plot the warm volume flow at 120 seconds on route 5 and plot the locations of trains on route 5 at 120 seconds.”

When processing the second page, ***time** would be 125. This line of input would be treated as

```
profile  qwarm  source_file  route5@125
icons    trains   source_file  route5@125
```

And so on for the third, fourth and however many pages there are.

Timeloops are mostly used to illustrate how trains move along routes, how fans move up and down their characteristics and how airflow drops as fires (in SES) warm up the walls on the downwind side of fires. I gained a lot of insights from timeloops that static images cannot provide.

3.28 The **filesloop** block

The **filesloop** block defines one page and plots the page many times over multiple input files. This is useful because a sequence of pages with the same graphs plotted for different files can be used to show the performance of multiple smoke control runs in which only the fire location, train location and fan operations change.

All the settings used in the **page** block can be used in the **filesloop** block (**pageunits** and **pageorientation**).

If the **begin filesloop** command has the word **ignore** after it (as in **begin filesloop ignore**) the filesloop will not be processed.

The **filesloop** block has three ways of defining which files to plot over. One of these must be placed between the line with **begin filesloop** on it and the first **begin graph** or **begin image** block:

1. A line with **all files** on it.
2. A **nicknames** block containing a list of nicknames to plot at.
3. An **exclude** block containing a list of nicknames to not plot at.

The nicknames are those set in the **files** block plus the nickname **calc**.

Examples of the usage of each given below.

```
begin filesloop
  all files
  begin graph
    <lines of graph definition>
  end graph
end filesloop
```

This generates a **.pdf** file with one page for each of the files in the **files** block (see Section 3.9)) and then from the current file (if the current file has a **.hbn** file).

```
begin filesloop
  begin nicknames
    myfile1  myfile2  myfile3
    myfile5  myfile6
  end nicknames
  begin graph
    <lines of graph definition>
  end graph
end filesloop
```

This generates a **.pdf** file with one page for each of the files in the **files** block with those five nicknames.

```
begin filesloop
  begin exclude
    myfile2  myfile4
    myfile6 calc
  end exclude
  begin graph
    <lines of graph definition>
  end graph
end filesloop
```

This generates a **.pdf** file with one page for each of the files in the **files** block except the files assigned the nicknames **myfile2**, **myfile4** and **myfile12**. **Calc** is excluded, so a page for the file being processed is not generated.

The **fileloop** block uses a special variable to refer to the file nickname in curve definitions, ***name**. ***Name** works in much the same way as ***time** works in **timeloop** blocks.

For example:

```
profile  qwarm  *name   route5@3600
icons    trains  *name   route5@3600
```

When processing the first page of our example file, `*name` would be set to the nickname of the first file and the page processed. On the second page, it would be set to the nickname of the second file and so on.

If you do not have a `begin files...end files` block in the file, the only nickname available is `calc`.

3.29 Behind the scenes in the plot block

3.29.1 Introduction

This section gives some background on how data is plotted by Hobyah that may be useful for anyone wanting to probe into the curves that Hobyah plots with a view to finding mistakes I may have made.

Whenever Hobyah is run it creates a subfolder named `ancillaries`. It stores data files that users don't usually need to look at in the subfolder. There are two types of file:

- Gnuplot input files (`.plt` files) with input from the `page`, `timeloop`, `filesloop`, `graph` and `image` blocks
- one or more curve data files (`.txt` or `.csv` files)

Having access to these files is handy when investigating oddities and debugging problematic input files.

There are three types of `.plt` file. One of them has the same name as the Hobyah input file and holds all the Gnuplot commands that build the graphs in the `page` blocks (there is only one of these files).

The second type holds the Gnuplot commands that build the items (graphs, images) in one `timeloop` block (there is one of these for each `timeloop` block, numbered from 1).

The third type holds the Gnuplot commands that build the graphs in one `filesloop` block (there is one of these for each `filesloop` block, their numbering is shared with the `timeloop` blocks).

Say that a Hobyah input file named `foo.txt` has a `plots` block with 20 `page` blocks, three `timeloop` blocks and one `filesloop` block. It generates five `.plt` files:

- `foo=plt`, which holds Gnuplot commands to generate the 20 pages in one pdf file,
- `foo-lp1=plt`, which holds Gnuplot commands for the first timeloop block,
- `foo-lp2=plt`, which holds Gnuplot commands for the second timeloop block,
- `foo-lp3=plt`, which holds Gnuplot commands for the third timeloop block and
- `foo-lp4=plt`, which holds Gnuplot commands for the filesloop block.

3.29.2 Looking inside .plt files

Each .plt file contains a set of Gnuplot commands generated from the contents of the plots block. The .plt files are extensively commented, with the start of each page, graph, image and verbatim block identified.

Some lines have comments that let you correlate the contents of the .plt file file to the line in the input file that it was generated by. This useful when Gnuplot complains about something wrong in the .plt file. For example, take the following `verbatim` block:

```
begin verbatim
    set label 1 "/*0.5 Nose passes sensor}" at first 5.2, 1500
    set label 2 "/*0.5 Tail passes sensor}" at first 7.6. -400
    set label 3 "/*0.5 Reflected nose entry wave arrives}" at first 9.4m 120
end verbatim
```

Let's assume that these are on lines 139–143 line of entry in the input file. The line setting label 3 is not valid, because '9.4, 120' has been mis-typed as '9.4m 120'. Hobyah does not catch that error because the error is in a `verbatim` block. Gnuplot will raise an error with the message

```
"foo.plt" line 327: undefined variable: m
```

Now we have to open the .plt file in a text editor and jump to the 327th line. When we get there we can see the text that the program wrote when it processed the verbatim block:

```
set label 3 "/*0.5 Reflected nose entry wave arrives}" at first 9.4m 120
```

After a bit of scrutiny we spot the “m” in place of a comma after the “9.4”. We could fix it here and rerun Gnuplot, but it would be better to fix it in the input file that generated the .plt file. Fortunately the lines surrounding the faulty line have comments to help us figure out where in the input file that is:

```
# Start of a verbatim block, at line 139
    set label 1 "/*0.5 Nose passes sensor}" at first 5.2, 1500
    set label 2 "/*0.5 Tail passes sensor}" at first 7.6, -400
    set label 3 "/*0.5 Reflected nose entry wave arrives}" at first 9.4m 120
# End of a verbatim block, at line 143
```

The first and last lines in that example were written out by Hobyah to help you figure out that the faulty line is at line 142 in the Hobyah file. I only recently started adding these pointers in automatically-generated .plt files and I find them amazingly helpful every time I foul up when writing `verbatim` blocks in my Hobyah input files.

3.30 Curve data files

The curve data files have names based on the .plt file name that uses them, the page number, graph number and curve number. An input file like `foo.txt` would generate

curve data files for the page blocks with names like `foo-p4-g2-c3.txt`. This file would hold data for the 3rd curve plotted on the 2nd graph on page 4.

Each curve data file has the same form: values are separated by commas and comments are preceded by `#`. The `.txt` files can be loaded into spreadsheets by treating them as if they were `.csv` files⁹.

At the top of each curve data file is a block of comment text that holds QA information.

A comprehensive explanation of the QA data held in the files is given in `verification+validation.pdf`.

⁹There is an entry in the `settings` block that allows users to change all the curve data file extensions from `.txt` to `.csv`. I prefer `.txt`, but I know that a lot of other engineers prefer `.csv`. See Section 3.3.13.

Chapter 4

Interpreting Hobyah output

4.1 Printed output from the airflow calculation

Whenever a calculation time matches one of the print times, blocks of data are printed to the `log` file and can be reviewed by users.

The values are printed as columns of data for each gridpoint in each Hobyah segment. For example:

```
"westbound1" (24th segment in the tunnel, seg_ID 52) at 499.0 seconds
grid- distance  velocity  P_static  P_dyn  P_total  celerity  density  volume flow
 point   (m)      (m/s)     (Pa)     (Pa)     (Pa)     (m/s)    (kg/m^3)   (m^3/s)
    0    12760.00   1.68    101325.9   1.69    101327.6  343.8209   1.2000   172.967
    1    12780.00   1.68    101325.7   1.69    101327.4  343.8208   1.2000   172.967
    2    12800.00   1.68    101325.6   1.69    101327.3  343.8207   1.2000   172.967
    3    12820.00   1.68    101325.4   1.69    101327.1  343.8207   1.2000   172.968
    4    12840.00   1.68    101325.3   1.69    101327.0  343.8206   1.2000   172.968
    5    12860.00   1.68    101325.1   1.69    101326.8  343.8205   1.2000   172.968
    6    12880.00   1.68    101325.0   1.69    101326.7  343.8204   1.2000   172.968

"xp1" (1st segment in the tunnel, seg_ID 53) at 499.0 seconds
grid- distance  velocity  P_static  P_dyn  P_total  celerity  density  volume flow
 point   (m)      (m/s)     (Pa)     (Pa)     (Pa)     (m/s)    (kg/m^3)   (m^3/s)
    0      0.00    3.84    101318.1   8.84    101326.9  343.8171   1.1999   1.804
    1     30.00    3.84    101318.1   8.84    101327.0  343.8171   1.1999   1.804
```

Each time a new set of conditions are printed, a header with the time is printed:

```
Print time: 500.0 seconds
"eastbound1" (1st segment in the tunnel, seg_ID 1) at 500.0 seconds
grid- distance  velocity  P_static  P_dyn  P_total  celerity  density  volume flow
 point   (m)      (m/s)     (Pa)     (Pa)     (Pa)     (m/s)    (kg/m^3)   (m^3/s)
    0    10020.00  -1.52    101325.0   1.39    101326.4  343.8204   1.2000  -156.935
    1    10040.00  -1.52    101325.1   1.39    101326.5  343.8205   1.2000  -156.935
```

The method of characteristics calculates the celerity and the velocity. All the other values are calculated from those two, the outside air conditions and the segment area.

4.2 Printed output for traffic

I constantly find errors in my traffic drag calculations; so do most of the engineers I've worked with. In an attempt to help avoid mistakes, I made Hobyah print a considerable amount of data to the screen and to the logfile about how it is calculating traffic.

Printed output for the traffictypes block

When the `traffictypes` block is processed, the entries in it are printed to the log file and to the screen. First item to be printed is a statement of whether a tunnel blockage correction should be applied, followed by the definition of traffic types: vehicle name, cross-sectional area, drag factor, any speed limits caused by vehicle types having to slow on high gradients and PCU values at each speed.

```
Road traffic drag calculation will not include a blockage term.
The vehicle drag factors below are assumed to already account for blockage.
Vehicle   Area   Drag factor   PCU      Mass      Vehicle speed limits on uphill grades
      name     (m^2)       (-)      value    (tonnes)      Limiting gradient      Speed limit
-----
car        2.0      0.4        1.0      Not used      Flat to vertical:      no speed limit
lcv        4.0      0.9        1.0      Not used      Flat to vertical:      no speed limit
hgv        6.0      1.0      Varies      22          Flat to vertical:      no speed limit
Vehicle type "hgv" occupies different amounts of space at different speeds:
<=10.0 km/h,           vehicle type occupies 3.0 PCU
>10.0 km/h,           vehicle type occupies 2.0 PCU
```

Mass is also included (set by an optional entry) but does nothing yet (mass is only useful when doing pollution calculations, which Hobyah does not have yet).

Printed output for the trafficsteady blocks

Next to be printed are blocks generated whenever traffic is assigned to a route in a `trafficsteady` block. There are two types of printout: one for moving traffic and one for traffic at standstill. Moving traffic just gives the vehicle flowrates, the speed and the extents, as in the following example:

```
Route "one" has 90.0 km/h traffic in it as follows:
car: 1800.0 veh/hr
lcv: 220.0 veh/hr
hgv: 300.0 veh/hr (using 2.0 PCU/veh at 90.0 km/h)
Total vehicle flowrate: 2320.0 veh/hr
This is equivalent to: 2620.0 PCU/hr
* The tunnels start at chainage 10000 and stop at chainage 13000.
* The vehicles start at chainage 10000 and stop at chainage 13000.
```

The calculation of vehicle densities at standstill is deceptively easy to get wrong. I constantly foul up vehicle density calculations (probably because I do them so rarely). When processing stationary traffic, the program writes the calculations out in detail in the .log file and to the screen. The level of detail it prints is intended to help users of Hobyah avoid making the same mistakes as me whenever I have to either convert between veh/km and PCU/km or figure out what the mean PCU value for a given mix of traffic mix is. The following is an example of the printout for stationary traffic:

```
Route "two" has stationary traffic at 165 PCU/lane-km
Mean PCU value calculation:
  1*1600 + 1*205 + 3*291
  ----- = 1.278 PCU/veh
  1600 + 205 + 291
So 165 PCU/lane-km is equivalent to 129.141 veh/lane-km.
  car:    98.58 veh/lane-km (98.58 PCU/lane-km)
  lcv:    12.63 veh/lane-km (12.63 PCU/lane-km)
  hgv:    17.93 veh/lane-km (53.79 PCU/lane-km)
* The tunnels start at chainage 10000 and stop at chainage 10200.
* The vehicles start at chainage 10040.01 and stop at chainage 10119.99.
```

In the case of moving traffic, the traffic fills the tunnels in the routes from up end to down end. In stationary traffic the extents of the queue of traffic may start and stop inside the tunnels. The last two lines of each type of printout give the extents.

Printed output for traffic in tunnels

Data for the traffic in tunnels must be printed, because more than one route may put traffic through a tunnel.

When the routes are in the same `trafficsteady` block they share a common set of lanes and the vehicles in each route mix together.

When the routes are in different `trafficsteady` blocks they are assumed to be in separate groups of lanes (think of single-tube, bidirectional road tunnels like Saltash or Mont Blanc: traffic in them would need to be modelled in two `trafficsteady` blocks).

We have to handle the possibility of traffic from different routes combining. So we need to decide which routes share a set of lanes and which don't.

```
Tunnel "filled" has traffic:
  Trafficsteady block "going-east" has stationary traffic
  in tunnel "filled".  Densities are calculated as
  follows (for each block of lanes in the route) using
  a base traffic density of 165 PCU/lane-km:
Lane/traffic block: | 1st
  Start distance: | 0      m
  Stop distance:  | 200    m
"eastbound" traffic: | Is here
  car flow (1 PCU): | 1600   veh/hr
  lcv flow (1 PCU): | 205    veh/hr
  hgv flow (3 PCU): | 291    veh/hr
```

```

Vehicle flow totals: | 2096  veh/hr
Mean PCU value:    | 1.278  PCU/veh
car PCU-flow:      | 1600   PCU/hr
lcv PCU-flow:      | 205    PCU/hr
hgv PCU-flow:      | 873    PCU/hr
PCU flow totals:  | 2678   PCU/hr
Lane counts:       | 2      lanes
car density:       | 197.16  veh/km
lcv density:       | 25.26   veh/km
hgv density:       | 35.86   veh/km

```

The program prints the densities of vehicles in each tunnel:

```

Road traffic in tunnel "filled":
Start | Stop | | Vehicle | Gradient | Vehicle
distance | distance | Speed | density | vehicles | type
(m) | (m) | (km/h) | (veh/km) | are on |
0 | 200 | 0 | 197.162061 | 0 | car
0 | 200 | 0 | 25.261389 | 0 | lcv
0 | 200 | 0 | 35.85885 | 0 | hgv

```

In this file (in which traffic fills the tunnel end to end) these four blocks of information are overkill. But in tunnels with multiple slip roads and different streams of traffic merging and splitting, the printouts are useful for checking the arithmetic of the calculation and ensuring that the correct numbers of vehicles are present.

Once the program has split the tunnels into segments, it prints out segment-based traffic drag data (along with the segment-based jet fan thrust term). In the case of the traffic, these are expressed as pressure loss factors per metre length of the segment with the corresponding speed of the traffic in km/h:

```

Segment-based traffic drag and jet fan thrust terms.
Seg_ID 1 (0 m to 280 m in tunnel "first"):
Traffic: 80.01 m to 159.99m, 0 km/h with drag 0.00450314319482 zeta/m
Seg_ID 2 (11000 m to 11200 m in tunnel "second"):
Traffic: 11000 m to 11200m, 60 km/h with drag 0.000885003376665 zeta/m
Seg_ID 3 (0 m to 200 m in tunnel "third"):
Traffic: 0 m to 200 m, 0 km/h with drag 0.004362795620595 zeta/m

```

These are for the Hobyah validation and verification document, they are less relevant to users (though they may be instructive).

Chapter 5

classSES.py

`classSES.py` is a Python class that can be used to manipulate the contents of binary files produced from the `SESconv.py` program. The contents of the binary file are in SI units.

Its main use is to produce data from SES runs that can be plotted in graphs in Hobyah files, but it can also be used to generate new SES input files from binary files.

5.1 Using it in Python sessions

The best way of understanding `classSES.py` is to import it into a Python session running in Terminal and start trying out its functions. If you have a poor understanding of Python, the points below may not make sense, so go and learn to program in Python before reading any further.

A class instance is created by calling `classSES.SESdata` with three arguments:

1. A path definition,
2. The filename of a `.sbn` file generated by `SESconv.py`, and
3. The handle of a log file that can be written to (not the name of the log file, the handle).

The path definition can be an absolute one (such as `/Users/tester/Documents` or `C:\Users\tester\Documents`) or a relative one (`subfolder`). If it is a relative path it is assumed to start from the current working directory.

The filename must include the extension `.sbn`.

I generally import the class, open a log file handle to a disposable file, then generate the class:

```
>>> import classSES  
>>> log = open("log_discard.txt", "w")  
>>> binfile = classSES.SESdata("test-files", "S-file-valid.sbn", log)
```

If I am developing code in the class, I use `importlib` so that I can reload the module after editing it, then regenerate the class:

```
>>> import importlib
>>> importlib.reload(classSES)
>>> binfile = classSES.SESdata("test-files", "S-file-valid.bin", log)
```

I use these commands so often while developing `classSES.py` that they are stored at the end of `classSES.py` so that it is easy to copy them and paste them into a Terminal session.

5.2 Describe()

`Describe()` is a function inside `classSES.py` that prints a description of a variable stored in the file. An example of its use:

```
>>> import classSES as clS
>>> log = open("log_discard.txt", "w")
>>> s005 = clS.SESdata("test-files", "SES-005-impl-impl-5.sbn", log)
>>> s005.Describe("seg_vels")
A pandas DataFrame giving the air velocities in the open
tunnel in every segment at every timestep. The columns
are integers (the segment numbers). The indices are the
print times given as floats.
The values are in m^3/s.

>>>
```

5.3 PrettyClassPrint()

`PrettyClassPrint()` is a function inside `classSES.py` that prints some or all of the contents of the binary file. I use it place of writing (and trying to keep updated) a document giving the exact definition of each entry in a binary file.

`PrettyClassPrint()` can be used in three ways.

1. If you give it no arguments it will print the user-visible contents of a binary file in a (hopefully) human-readable form:

Command: `>>> binfile.PrettyClassPrint()`

Output: <an endless screed of stuff>

I find this useful when I want to search for a particular phrase that turned up in a stack trace or error message and I have no idea where the phrase came from.

2. If you give it a variable name that does not exist, it will write a list of some of the top-level variable names to help you out.

Command: >>> binfile.PrettyClassPrint("f")

Output:

```
> Tried to print the contents of "f" in
> "S-file-valid.bin" but it doesn't exist.
> Here are the valid names:
> binversion_string, binversion, file_path,
> file_name, ses_name, prn_name, prog_type,
> script_date, script_name, when_who, comments,
> header, footer, settings_dict, form2_dict,
> form3_dict, form4_dict, form5_dict, form6_dict,
> form7_fans, form7_JFs, form8_dict, form9_dict,
> form10_dict, form11_dict, form12_dict, form13_dict,
> print_times, summary_times, sec_seg_dict,
> sec_DPs, seg_flows, seg_vels, subseg_names,
> subseg_humids, subseg_lat, subseg_sens, subseg_temps,
> subpoint_keys, subpoint_areas, subpoint_coldflows,
> subpoint_coldvels, subpoint_warmflows, subpoint_warmvels,
> route_num, train_type, train_locn, train_speed,
> train_accel, train_coeff, train_TE, motoramps,
> lineamps, flywh_spd, ac, decel_temp,
> pwr_all, heat_reject1, train_modev, pwr_aux,
> pwr_prop, pwr_regen, pwr_flywh, pwr_accel,
> pwr_decel, pwr_mech, heat_mech, heat_sens,
> heat_lat, train_eff, heat_reject and train_aerodrag.
```

So putting a random letter in the argument generates a list of variable names that you can use in the command, which is handy.

3. If you give it the name of a top-level variable in the class it will print only the contents of that (in this case, all instances of form 7 jet fans):

Command: >>> binfile.PrettyClassPrint("form7_JFs")

Output:

```
Printing the contents of "form7_JFs" in "S-file-valid.bin".
dict "self.form7_JFs":
    subdict "1":
        "volflow": 29.9999995342848
        "insteff": 0.75
        "jet_speed": -27.999944000000003
        "fan_start": 40.0
        "fan_stop": 900.0
        "static_thrust": 1007.9979824352
Printed the contents of "form7_JFs" only.
```

These are the most useful view of the contents. One dictionary (in the case of an input variable like `form7_JFs`) or one pandas database in the case of an output variable like `sec_DPs`.

If I am developing code that involves SES binary files, I usually fire up a Python interpreter and load a binary file into the class. `PrettyClassPrint()` gives a convenient way of figuring out what name I assigned to a particular value when writing the SES postprocessor—more convenient than searching through `SESconv.py` in a text editor. In the example from form 7C above, I used “`volflow`” for the jet fan volume flow (because I use that word everywhere) but I used “`jet_speed`” with an underscore between the words for jet velocity. I often forget what names I assigned to things, so `PrettyClassPrint()` is a good way to figure it out.

5.4 WriteInputFile()

`WriteInputFile()` is a function used to write out an SES input file based on the contents of the class. It relies on you having set up the contents correctly.

For example: let’s say that you want to add a new type of jet fan (Form 7C). You can set that up by adding a new entry in `form7_JFs`. You will also need to add one to the variable named `jftypes` in `settings_dict`, the program is not smart enough to update it for you.

The new file may be in US units or SI units. At the moment it writes files formatted for SES v4.1, offline-SES, OpenSES v4.3 and Aurecon SES. Support for writing SVS 6.6.2 input files is on the to-do list and has been partially implemented.

Chapter 6

Plot properties

6.1 Plotting

The syntax of plotting is described in Section 3.25. This section just describes the properties, gives example syntax and sample output.

The following are links to the subsections in this document that describe different types of plot:

- Section 6.2—Hobyah transient properties in tunnels
- Section 6.3—fixed properties in routes and tunnels (both Hobyah and SES)
- Section 6.4—Hobyah transient properties at fans
- Section 6.5—Hobyah transient properties at dampers
- Section 6.6—icons for trains, jet fans and fires
- Section 6.8—SES transient properties in tunnels
- Section 6.9—SES transient properties at trains
- Section 6.10—SES transient properties from ECZ estimates

6.2 Plots of Hobyah transient properties in tunnels

This section describes the curves of transient properties in the tunnels from Hobyah. These may be plotted by referencing a chainage along a route, a distance along a tunnel, or a distance from the down end of a (possibly moving) train.

The following lists the keywords of transient properties at gridpoints and train boundaries from the method of characteristics calculation.

- **annulus**: area in the annulus around trains
- **celerity**: speed of sound

- **density**: air density
- **massflow**: mass flow
- **pdiff**: difference in air total pressure between two points
- **pdiffstat**: difference in air static pressure between two points
- **pstat**: gauge air static pressure
- **pstatabs**: absolute air static pressure
- **ptot**: gauge air total pressure
- **ptotabs**: absolute air total pressure
- **velocity**: air velocity
- **volflow**: volume flow

It is useful to negate some properties (multiply them by -1). This can be done by using the optional argument `ymult:=-1`, but it is easier let users just prepend a minus sign to the keyword. This makes no sense for scalars like `density`, but does make sense for vectors like `velocity`. With that in mind, the following keywords are also valid:

- **-massflow**: mass flow multiplied by -1
- **-pdiff**: difference in air total pressure, multiplied by -1
- **-pdiffstat**: difference in air static pressure, multiplied by -1
- **-pstat**: gauge air static pressure multiplied by -1
- **-ptot**: gauge air total pressure multiplied by -1
- **-velocity**: air velocity multiplied by -1
- **-volflow**: volume flow multiplied by -1

All but two of these properties are plotted at one point. The exceptions are `-pdiff` and `-pdiffstat`, which give the difference in total pressure and static pressure between two points.

Sometimes those two points are obvious, like the pressure difference across a junction, area change or damper.

Sometimes the two points are arbitrary, defined by a keyword that takes two locations and assigns a name to them that can be used in curve definitions. This is usually used for pressure difference across cross-passages.

6.2.1 The velocity and -velocity plot types

Velocity is the air velocity at the gridpoints in the tunnels and at the train ends.

It is a vector property, so the keyword **-velocity** is also valid. **-Velocity** returns the values of velocity multiplied by -1 .

Typical curve definitions:

```
keyword      property      nickname      source@distance
transient    velocity      calc          mainline4@10165
transient    velocity      calc          eastbound@11205
keyword      property      nickname      source@time
profile      velocity      calc          mainline4@490
profile      velocity      calc          eastbound@490
```

Typical output:

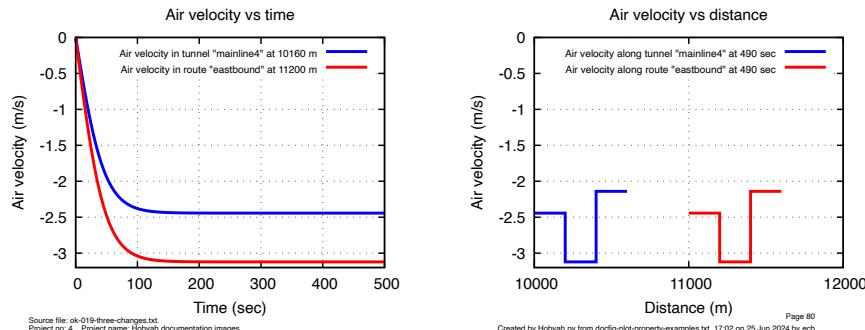


Figure 6.1: Air velocities vs. time and distance

Note that in the transient curves in Figure 6.1, the desired locations (distance 10165 m and chainage 11205 m) are not in the graph key - the program has plotted the values at the nearest gridpoints, which happen to be at distance 10160 m and chainage 11200 m. This is a common occurrence.

- Curve types where it is used: **transient**, **profile**
- Where plotted: at fixed gridpoints, at train end boundaries, along tunnels and along routes.
- Units: m/s, fpm

Data for **velocity** are stored in two lists.

The first list is called **v_array**. Each entry in the list is a Pandas array that holds data for the gridpoints of one segment. The indices of each pandas array are plot time and the columns are gridpoint locations (distances in the segment). The index and column are both real numbers. The first gridpoint is at the back end of the segment and the last gridpoint is at the forward end. The details of which segment belongs to which tunnel are stored in a dictionary named **segs2tuns**.

The second list is called **v_ends** and stores the values at the four gridpoints at the two ends of a train. The list holds one pandas array for each train, with indices of time

and columns that are the four text keys `up_open`, `up_ann`, `down_ann` and `down_open`. These are short for “at the up end of the train in the open tunnel”, “at the up end of the train in the annulus”, “at the down end of the train in the annulus” and “at the up end of the train in the open tunnel”.

6.2.2 The celerity plot type

Celerity is the speed of sound at the gridpoints in the tunnels and at the train ends. It is not useful for engineering work; it is only worth plotting when doing software development or verifying the method of characteristics calculation. It is more useful to plot the two properties that are derived from celerity (pressure and density).

Typical curve definitions:

```
keyword      property      nickname      source@distance
transient    celerity     calc          mainline4@10160
transient    celerity     calc          eastbound@11200
keyword      property      nickname      source@time
profile      celerity     calc          mainline4@490
profile      celerity     calc          eastbound@490
```

Typical output:

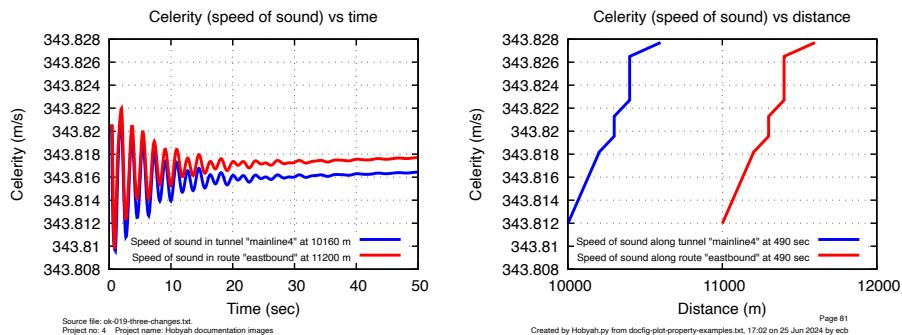


Figure 6.2: Speed of sound vs. time and distance

The speed of sound is ≈ 343.82 m/s at sea level. Hobyah calculates minor variations in it that correlate to changes in pressure and density.

- Curve types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints, at train end boundaries, along tunnels and along routes.
- Units: m/s, fpm

Data for `celerity` are stored in two lists.

The first list is called `c_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers). The first gridpoint is at the

back end of the segment and the last gridpoint is at the forward end. The details of which segment belongs to which tunnel are stored in a dictionary named `segs2tuns`.

The second list is called `c_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.3 The density plot type

Density is the air density at the gridpoints in the tunnels and at the train ends.

Typical curve definitions:

```
keyword      property      nickname      source@distance
transient    density       calc          mainline4@10160
transient    density       calc          eastbound@11200
keyword      property      nickname      source@time
profile      density       calc          mainline4@490
profile      density       calc          eastbound@490
```

Typical output:

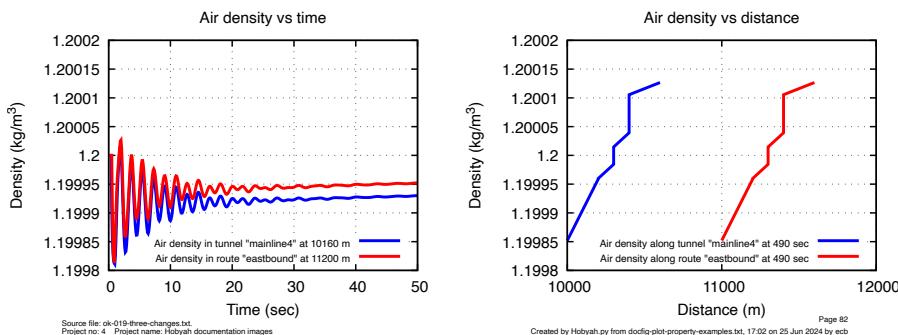


Figure 6.3: Air density vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints, at train end boundaries, along tunnels and along routes.
- Units: kg/m^3 , lb/ft^3

Data for `density` are stored in two lists.

The first list is called `dens_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `dens_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.4 The pdiff plot type

`Pdiff` is the difference in total (stagnation) pressure between two points in the model. The two points are typically on either side of some feature like a damper, pressure loss or fan, but arbitrary locations can be selected and given a name to plot against (this feature is useful for getting the total pressure difference across escape cross-passages).

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@discard</i>
<code>transient</code>	<code>pdiff</code>	<code>file7</code>	<code>f20999</code>
<code>transient</code>	<code>pdiff</code>	<code>file7</code>	<code>d20999</code>

Typical output:

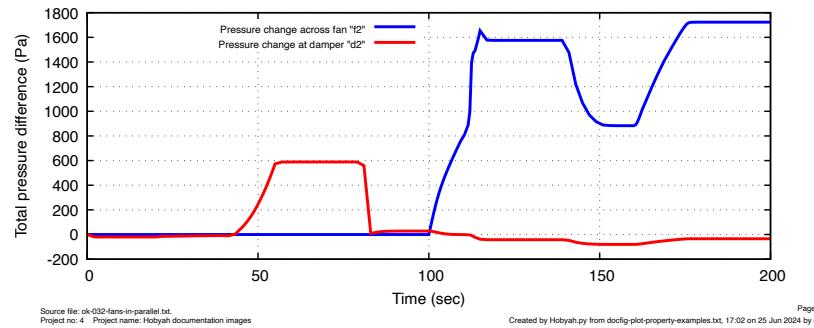


Figure 6.4: Pressure difference vs. time

- Curve types where it is used: `transient`
- Where plotted: at two gridpoints on either side of an entity like a fan or damper.
- Units: Pa, inches of water

It is a vector property, so the keyword `-pdiff` is also valid. `-Pdiff` returns the values of differential total pressure multiplied by -1 .

Data for `pdiff` are calculated on the fly from values in `p_tot_bin`.

6.2.5 The pstat and -pstat plot types

Pstat is the gauge static pressure of air in the tunnels and at the train ends. The datum is the outside air pressure set in the `settings` block.

Typical curve definitions:

```
keyword      property      nickname      source@location
transient    pstat        file7        Fan1_duct@135
transient    pstat        file7        second@10121
keyword      property      nickname      source@time
profile      pstat        file7        Fan1_duct@-1
profile      pstat        file7        second@-1      xoffset:=-9770
```

Typical output:

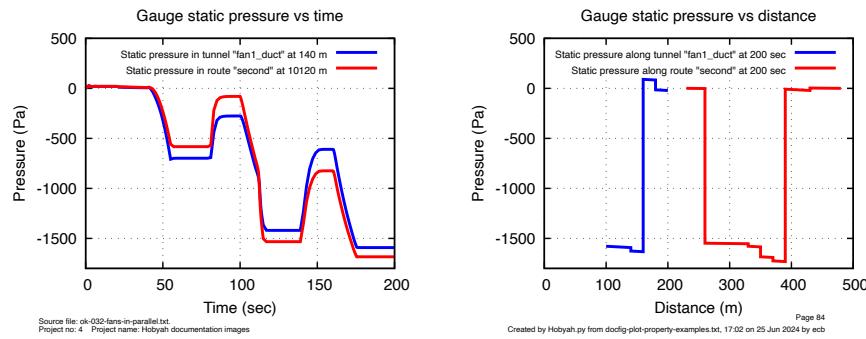


Figure 6.5: Gauge static pressure vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: Pa, inches of water

It is a vector property, so the keyword `-pstat` is also valid. `-Pstat` returns the values of air pressure multiplied by -1 .

Data for `pstat` are stored in two lists.

The first list is called `p_stat_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `p_stat_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.6 The ptot and -ptot plot types

Ptot is the gauge total pressure of air in the tunnels and at the train ends. The datum is the outside air pressure set in the `settings` block.

Typical curve definitions:

```
keyword      property      nickname      source@location
transient    ptot          file7         Fan1.duct@135
transient    ptot          file7         second@10121
keyword      property      nickname      source@time
profile      ptot          file7         Fan1_duct@-1
profile      ptot          file7         second@-1      xoffset:=-9770
```

Typical output:

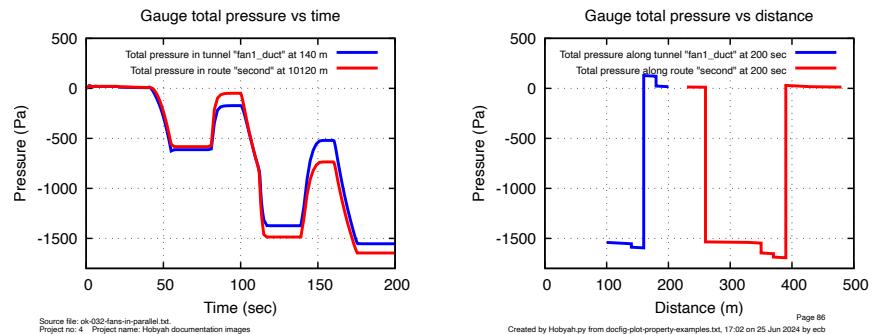


Figure 6.6: Total pressure vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: Pa, inches of water

It is a vector property, so the keyword `-ptot` is also valid. `-Ptot` returns the values of air pressure multiplied by `-1`.

Data for `ptot` are stored in two lists.

The first list is called `p_tot_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `p_tot_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.7 The pstatabs plot type

Pstatabs is the absolute static pressure of air in the tunnels and at the train ends.

Typical curve definitions:

```
keyword      property      nickname      source@location
transient    pstatabs     file7        Fan3_duct@10135
transient    pstatabs     file7        first@10121
keyword      property      nickname      source@time
profile      pstatabs     file7        Fan1_duct@-1
profile      pstatabs     file7        first@-1
```

Typical output:

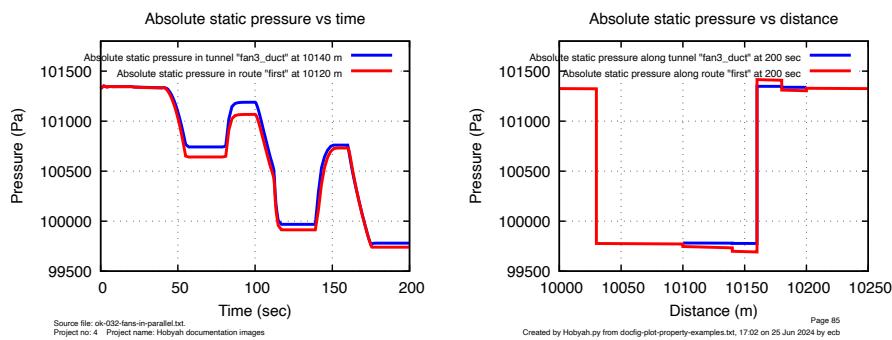


Figure 6.7: Absolute static pressure vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: Pa, inches of water

Data for `pstatabs` are stored in two lists.

The first list is called `p_statabs_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `p_statabs_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.8 The ptotabs plot type

Ptotabs is the absolute total pressure of air in the tunnels and at the train ends.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@location</i>
transient	ptotabs	file7	Fan3_duct@10135
transient	ptotabs	file7	first@10121
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	ptotabs	file7	Fan1_duct@-1
profile	ptotabs	file7	first@-1

Typical output:

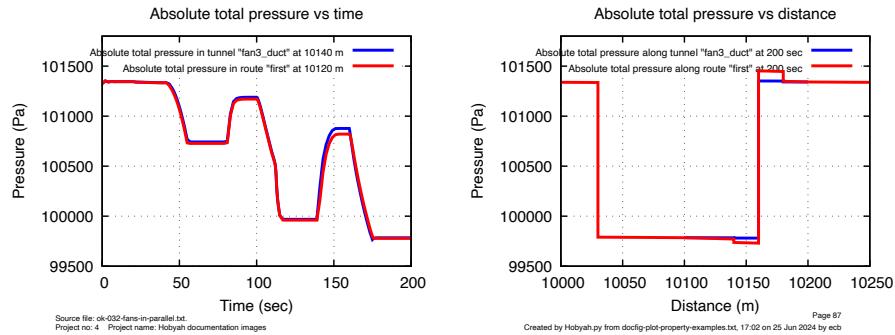


Figure 6.8: Absolute total pressure vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: Pa, inches of water

Data for `ptotabs` are stored in two lists.

The first list is called `p_totabs_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `p_totabs_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.9 The `volflow` and `-volflow` plot types

`volflow` is the volume flow of air in the tunnels and at the train ends. The volume flow in Hobyah always accounts for the volume flow of trains.

Typical curve definitions:

```
keyword      property      nickname      source@location
transient    volflow       file7        Fan3_duct@10135
transient    volflow       file7        first@10121
keyword      property      nickname      source@time
profile      volflow       file7        Fan1_duct@-1
profile      volflow       file7        first@-1
```

Typical output:

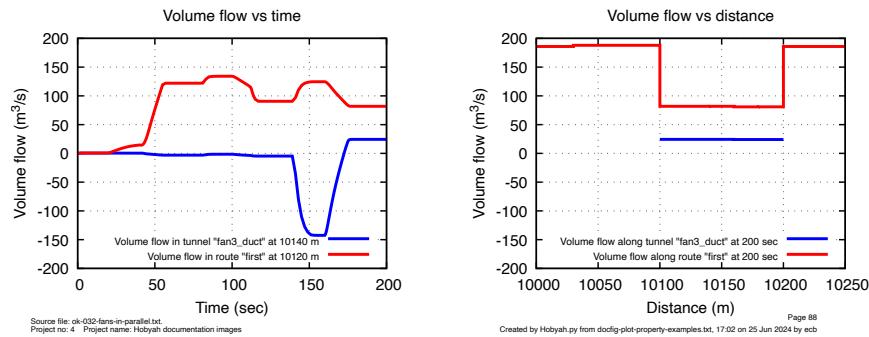


Figure 6.9: Volume flow vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: m^3/s , cfm

It is a vector property, so the keyword `-volflow` is also valid. `-Volflow` returns the values of volume flow multiplied by -1 .

Data for `volflow` are stored in two lists.

The first list is called `q_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `q_ends` and stores the values at the four gridpoints at the two ends of a train.

6.2.10 The `massflow` and `-massflow` plot types

`massflow` is the mass flow in the tunnels and at the train ends. The mass flow at a point always accounts for the product of local air density multiplied by the local air volume flow.

It is usually only used to pick up problems with the method of characteristics calculation.

Typical curve definitions:

```
keyword      property      nickname      source@location
transient    massflow      file7        Fan3_duct@10135
transient    massflow      file7        first@10121
keyword      property      nickname      source@time
profile     massflow      file7        Fan1_duct@-1
profile     massflow      file7        first@-1
```

Typical output:

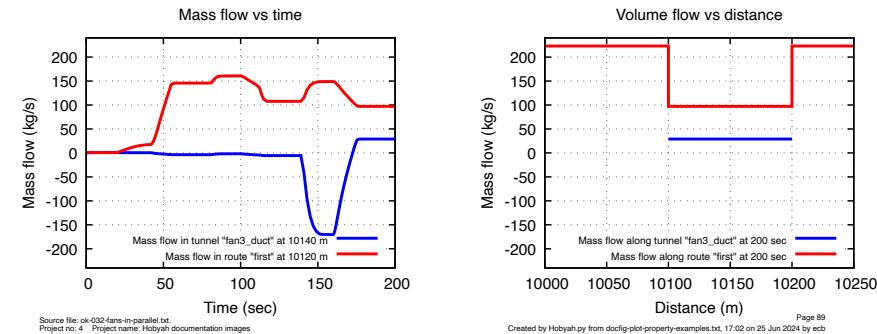


Figure 6.10: Mass flow vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: at fixed gridpoints and train end boundaries
- Units: kg/s, lb/s

Note that the mass flow may not be constant. This is usually due to the change of air density between gridpoints, but density profiles can be useful for catching failures in the calculation (see Section 7.3 for an example).

It is a vector property, so the keyword `-massflow` is also valid. `-Massflow` returns the values of mass flow multiplied by `-1`.

Data for `massflow` are stored in two lists.

The first list is called `m_array`. Each entry in the list is a Pandas array that holds data for one segment. The indices of each pandas array are plot time and the columns are gridpoint locations in the tunnel (both as real numbers).

The second list is called `m_ends` and stores the values at the four gridpoints at the two ends of a train.

6.3 Plots of fixed properties along routes/tunnels

Hobyah has the ability to plot profiles of route data and tunnel data along individual tunnels and along routes. When plotting in tunnels the X values are the distances in the tunnel. When plotting in routes the X values are the chainages in the route.

Hobyah can also plot profiles of SES route data and SES tunnel data along SES routes.

Some of these are useful (such as segment area vs. distance), some are less useful (such as SES section number or SES segment number vs. distance).

These types of plots are particularly useful for checking that the gradients in an SES route (form 8C) match the gradients in segments calculated from the stack heights in form 3A. Figure 6.11 plots two vertical profiles and two gradients along a route. Two curves are the gradients and vertical profile along the route. The other two curves are the gradients and vertical profile in the segments that the route runs through, calculated from the lengths and stack heights of the segments.

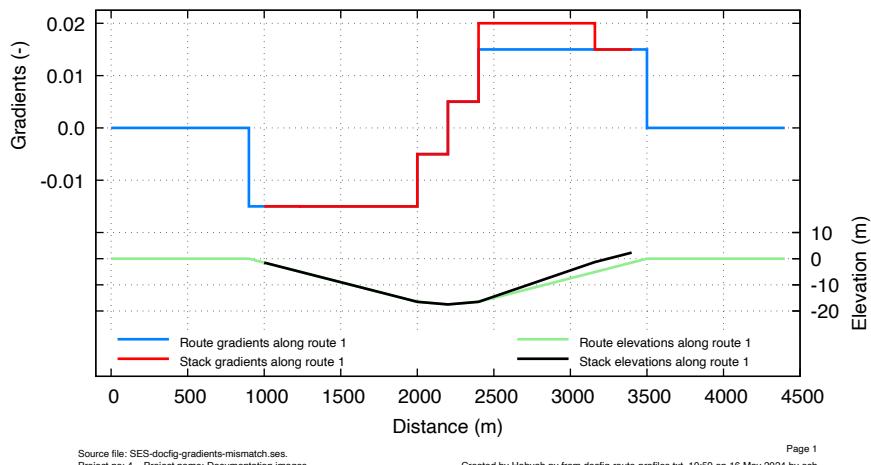


Figure 6.11: Comparison of route and segment profiles

The segment in which the vertical profiles do not match sticks out like a sore thumb. Plots like this one are a useful way to catch such errors on projects.

The following lists the keywords of fixed properties in tunnels and routes in Hobyah runs.

- **elevations**—Profile of elevations along a route
- **gradients**—Profile of gradients along a route
- **speedlimits**—Profile of speed limits along a route
- **lanes**—Profile of the count of traffic lanes along a route
- **area**—Profile of open tunnel area (ignoring the presence of trains) along a route or tunnel
- **perimeter**—Profile of tunnel perimeter (ignoring the presence of trains) along a route or tunnel

- **D_h**—Profile of tunnel hydraulic diameter along a route or tunnel
- **roughness**—Profile of tunnel roughness height along a route or tunnel
- **Darcy**—Profile of fully-turbulent Darcy friction factor λ (four times Fanning friction factor) along a route or tunnel
- **Fanning**—Profile of fully-turbulent Fanning friction factor c_f (one quarter of Darcy friction factor) along a route or tunnel
- **Atkinson**—Profile of fully-turbulent Atkinson friction factor k along a route or tunnel

The following lists the keywords of fixed properties along routes in SES and SVS runs.

- **elevations**—Profile of elevations along a route
- **stacks**—Profile of the stack heights in the tunnels along a route
- **gradients**—Profile of gradients along a route
- **stackgrads**—Profile of the gradients calculated from segment lengths and stack heights along a route
- **speedlimits**—Profile of the speed limits along a route
- **radius**—Profile of the track radii along a route
- **sectors**—Profile of the energy sector numbers along a route
- **coasting**—Profile of the coasting rules along a route
- **SVSregen2**—Profile of the regenerative braking fractions along a route (SVS only). See also the property **SVSregen1** (the regenerative braking fractions used by a train)
- **sections**—Profile of the section numbers along a route
- **segment**—Profile of the segment numbers along a route
- **subsegs**—Profile of the count of subsegments in the segments along a route
- **sublength**—Profile of the lengths of subsegments along a route
- **fireseg**—Profile of the fire segment on/off switch along a route
- **area**—Profile of open tunnel area (ignoring the presence of trains) along a route
- **perimeter**—Profile of open tunnel perimeter (ignoring the presence of trains) along a route
- **D_h**—Profile of tunnel hydraulic diameter along a route or tunnel
- **roughness**—Profile of tunnel roughness height along a route
- **Darcy**—Profile of fully-turbulent Darcy friction factor λ (four times Fanning friction factor) along a route
- **Fanning**—Profile of fully-turbulent Fanning friction factor c_f (one quarter of Darcy friction factor) along a route
- **Atkinson**—Profile of fully-turbulent Atkinson friction factor k along a route
- **wetted**—Profile of the wetted perimeter along a route

6.3.1 The elevations plot type

Elevations is a profile plot of the track elevation against distance on a route.

It makes no sense to try to plot elevation against time, so the time value in the curve definition is ignored.

The profile of **elevations** starts at the route origin and stops at the end of the route.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2.

When plotting from Hobyah runs, the definition is the name of a route. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword      property      nickname      source@discard
profile     elevations    file8        eastbound@0  # Hobyah route name
profile     elevations    file1        route1@0  # SES route number
```

Typical output:

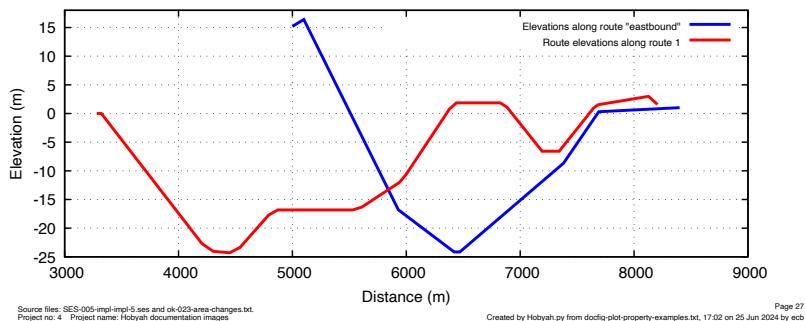


Figure 6.12: Elevation of the tracks along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: m, ft

In Hobyah, data for **elevations** are stored in a pair of lists in each route dictionary. The lists are named **elevgrad_chs** (X-values) and **elevations** (Y-values). In SES, data for **elevations** are stored in a pair of lists in each form 8 dictionary. The lists are named **single_chs** (X-values) and **elevations** (Y-values).

6.3.2 The stacks plot type

Stacks is a profile plot of the elevation of the segments against distance on an SES route. It is calculated from the segment lengths, segment stack heights and the orientation of the segments in the route.

It makes no sense to try to plot elevation against time, so the time value in the curve definition is ignored.

The profile of **stacks** starts at the up end of the first segment in the route and ends at the down end of the last segment in the route. It is useful to plot the route elevations and the stack elevations (**elevations** and **stacks**) on the same graph: this is an easy way to pick up stack heights in form 3B that don't match the track gradients.

Stacks can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    stacks      file1       route100
```

Typical output:

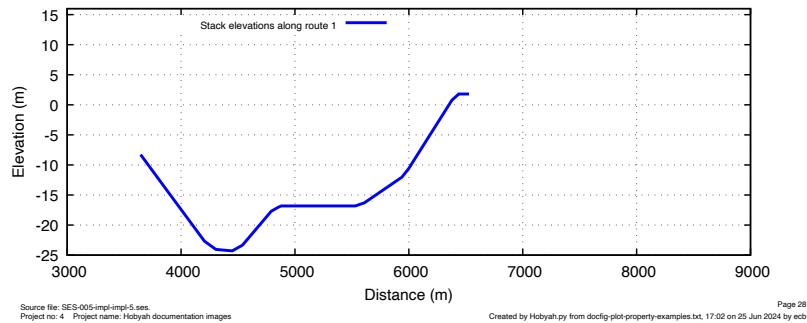


Figure 6.13: SES segment elevations along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: m, ft

Data for **stacks** are stored in a pair of lists in each form 8 dictionary. The lists are named **seg_chs** (X-values) and **seg_elevs** (Y-values).

6.3.3 The gradients plot type

Gradients is a profile plot of the track gradients on a route against distance. The gradients are shown as fractions (-1 to +1), not percentages, in order to be consistent with how Hobyah stores gradients. If you want to plot them as percentages, use a `ymult:=100` or `y2mult:=100` optional argument in the curve definition.

The **gradients** are shown with step changes at each change of value. The curve can be plotted along SES routes and from Hobyah routes.

It makes no sense to try to plot gradients against time, so the time value in the curve definition is ignored.

The profile starts at the route origin and stops at the end of the route.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2.

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    gradients   file8      eastbound@0  # Hobyah route name
profile    gradients   file1      route100    # SES route number
```

Typical output:

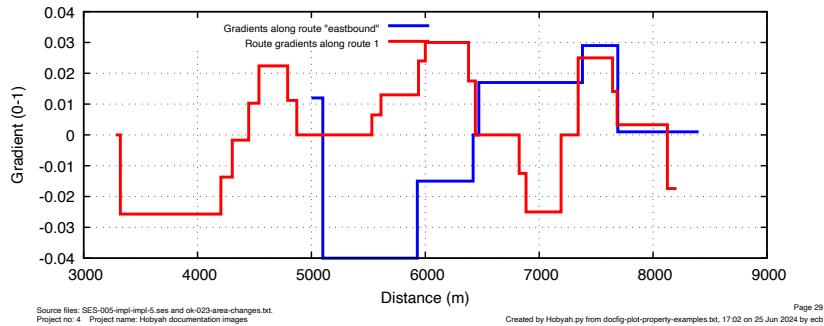


Figure 6.14: Gradients of the tracks along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: fraction (-1 to +1)

In Hobyah, data for **gradients** are stored in a pair of lists in each route dictionary. The lists are named `elevgrad_chs` (X-values) and `gradients2` (Y-values). In SES, data for **gradients** are stored in a pair of lists in each form 8 dictionary. The lists are named `paired_chs` (X-values) and `gradient2` (Y-values).

6.3.4 The `stackgrads` plot type

`Stackgrads` is a profile plot of the segment stack gradients on a route against distance. The stack gradients are shown as fractions (-1 to $+1$), not percentages, in order to be consistent with how Hobyah handles gradients. If you want to plot them as percentages, use a `ymult:=100` or `y2mult:=100` optional argument in the curve definition.

The `stackgrads` are shown with step changes at each change of gradient.

It makes no sense to try to plot stack gradients against time, so the time value in the curve definition is ignored.

The profile starts at the up end of the first section in the route and ends at the down end of the last section in the route. It is usual to plot `stackgrads` and `gradients` on the same curve so that mismatches can be picked up and corrected.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

`Stackgrads` can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword      property      nickname      source@discard
profile     stackgrads    file1        route1@00      ymult:= 100
```

Typical output (showing gradients in percent due to the `ymult` optional entry):

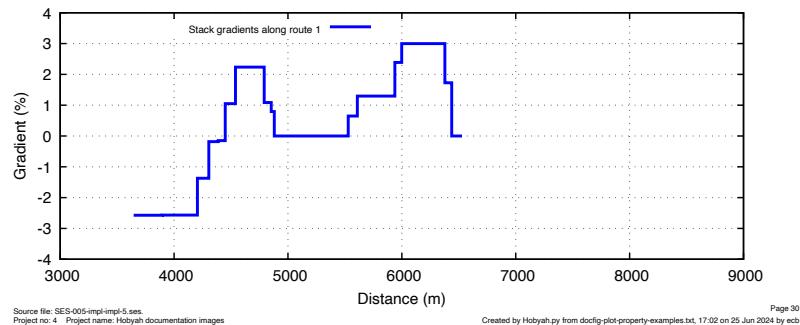


Figure 6.15: Gradients of the segments along a route

- Curve types where it is used: `profile`
- Where plotted: along a route.
- Units: fraction (-1 to $+1$)

Data for `stackgrads` are stored in a pair of lists in each form 8 dictionary. The lists are named `stack_chs` (X-values) and `stack_grads` (Y-values).

6.3.5 The speedlimits plot type

Speedlimits is a profile plot of the speed limits on a route against distance.

The speed limits are shown with step changes at each change of speed limit.

It makes no sense to try to plot speed limits against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

It is usually plotted alongside train speed, as in the example below.

Typical curve definitions:

```
keyword      property      nickname      source@discard
profile     speedlimits    calc          offslip@00    # Hobyah route name
profile     speedlimits    file1         route1@00    # SES route number
transient   speed         file1         train1@00   xaxis:= chainage
```

Typical output:

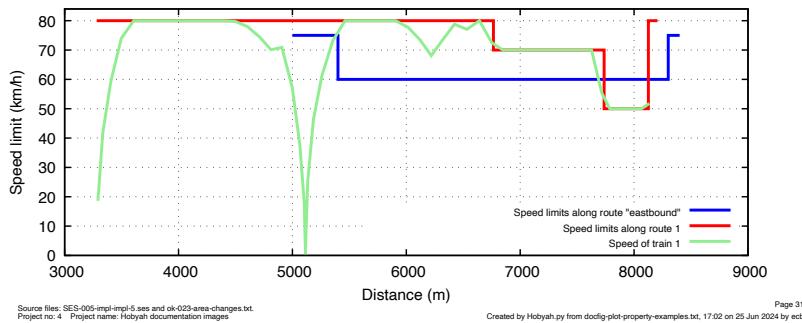


Figure 6.16: Speed limits along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: km/h, mph

In Hobyah, data for **speedlimits** are stored in a pair of lists in each route dictionary. The lists are named **speed_chs** (X-values) and **speed_plots** (Y-values). In SES, data for **speedlimits** are stored in a pair of lists in each form 8 dictionary. The lists are named **paired_chs** (X-values) and **speedlimit2** (Y-values).

6.3.6 The lanes plot type

`Lanes` is a profile plot of the count of lanes in a (road tunnel) route against distance.

The profile shows a step change at each change of count of lanes.

It makes no sense to try to plot the count of lanes against time, so the time value in the curve definition is ignored.

When plotting from Hobyah runs, the definition is the name of a route. It cannot be plotted from SES.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    lanes       calc        offslip@0      # Hobyah route name
```

Typical output:

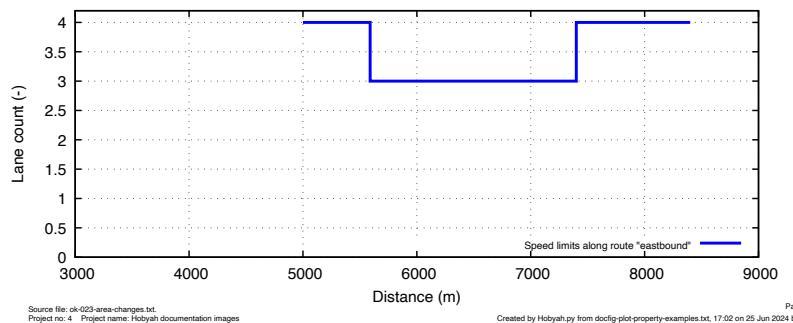


Figure 6.17: Count of lanes along a route

- Curve types where it is used: `profile`
- Where plotted: along a route.
- Units: none

Data for `lanes` are stored in a pair of lists in each route dictionary. The lists are named `lane_chs` (X-values) and `lane_plots` (Y-values).

6.3.7 The radius plot type

Radius is a profile plot of the track radius along a route against distance. If a section of track is straight, it will have zero radius.

The track radii are shown with step changes at each change of radius.

It makes no sense to try to plot track radius against time, so the time value in the curve definition is ignored.

This plot type is almost completely useless. The energy dissipated as heat as the wheels squeal as they go around curves is a tiny fraction of all the other heat sources on the train.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Radius can be set in Hobyah and plotted from its output, but only because Hobyah passes the radius values in SES form 8C when it writes a new SES input file. **Radius** does not affect the Hobyah calculation.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    radius      calc        EBmain@00
profile    radius      file1      route1@00
```

Typical output:

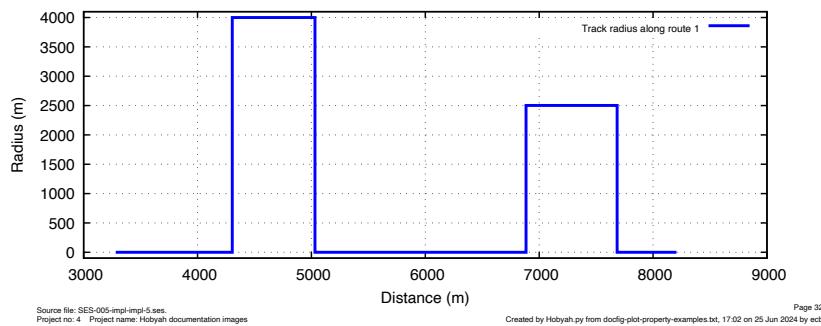


Figure 6.18: Radii of the tracks along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: m, ft

Data for **radius** are stored in a pair of lists in each form 8 dictionary. The lists are named **paired_chs** (X-values) and **radius2** (Y-values).

6.3.8 The sectors plot type

Sectors is a profile plot of the energy sectors along an SES route against distance.

The track sectors are shown with step changes at each change of sector.

It makes no sense to try to plot track sectors against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Sectors can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    sectors     file1       route100
```

Typical output:

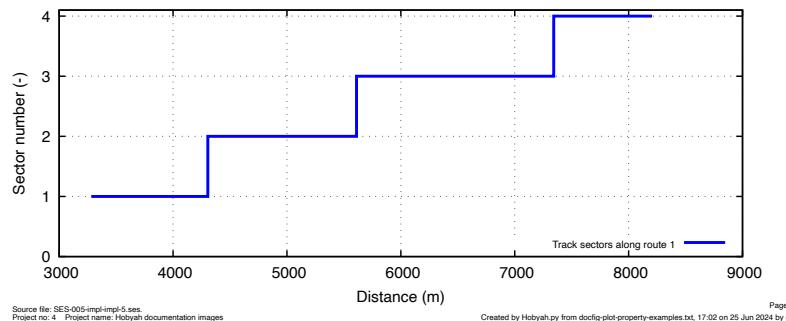


Figure 6.19: Energy sectors along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range 0–50

Data for **sectors** are stored in a pair of lists in each form 8 dictionary. The lists are named **paired_chs** (X-values) and **sector2** (Y-values).

6.3.9 The coasting plot type

Coasting is a profile plot of the coasting rules along an SES route against distance. If coasting is allowed in a section of track it has a value of one. If coasting is not allowed it has a value of zero.

It makes no sense to try to plot the coasting rules against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Coasting can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    coasting    file1        route100
```

Typical output:

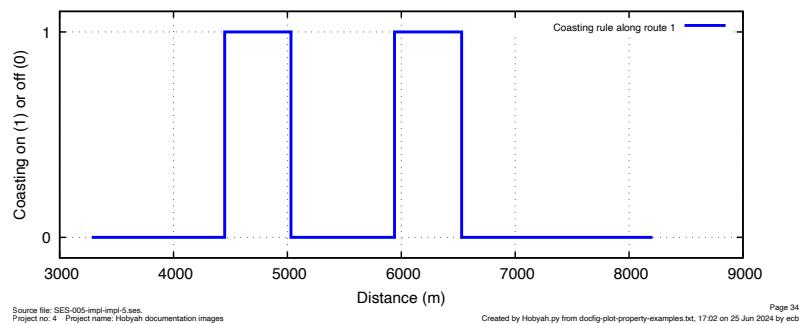


Figure 6.20: Coasting rules of the tracks along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range 0 or 1

Data for **coasting** are stored in a pair of lists in each form 8 dictionary. The lists are named **paired_chs** (X-values) and **coasting2** (Y-values).

6.3.10 The `svsregen2` plot type

`SVSregen2` is a profile plot of the regenerative braking fraction along an SVS route against distance. The values are fractions between 0 and 1. The feature was added in SVS to allow train regen braking to vary with a train's location along a route.

It makes no sense to try to plot the regenerative braking fraction against time, so the time value in the curve definition is ignored.

It is only available in simulations in which the train performance option in form 1C is one or two.

`SVSregen2` can only be plotted from SVS runs, there is no equivalent in SES runs or Hobyah runs.

A related property is `SVSregen1`, which is plotted on a train against time or chainage.

Typical curve definitions:

```
keyword      property      nickname      source@discard
profile     SVSregen2    file1        route1@0
```

Typical output:

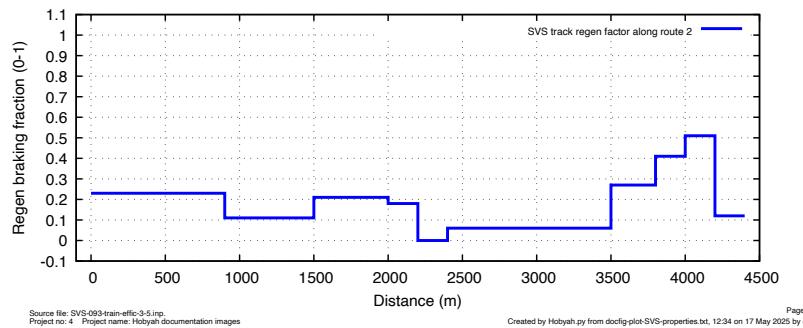


Figure 6.21: Regen braking fractions along a route

- Curve types where it is used: `profile`
- Where plotted: along a route.
- Units: dimensionless, range 0 or 1

Data for `SVSregen2` are stored in a pair of lists in each form 8 dictionary. The lists are named `paired_chs` (X-values) and `regen2` (Y-values).

6.3.11 The sections plot type

Sections is a profile plot of the section numbers that an SES route runs through against distance. The first section in the route starts at the location of the first node on the route (set in form 8F) and the plot stops at the location of the last node. Negative section numbers mean that the section is reversed in the route.

It makes no sense to try to plot the sections against time, so the time value in the curve definition is ignored. The plot is completely useless apart from verification work.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Sections can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    sections    file1        route100
```

Typical output:

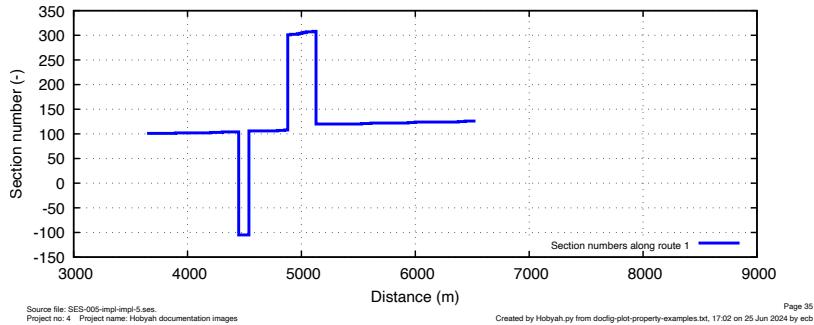


Figure 6.22: Section numbers along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range –999 to 999

Data for **sections** are stored in a pair of lists in each form 8 dictionary. The lists are named **sec_chs** (X-values) and **sec_list** (Y-values).

6.3.12 The segments plot type

Segments is a profile plot of the segment numbers that an SES route runs through against distance. The first segment in the route starts at the location of the first node on the route (set in form 8F) and the plot stops at the location of the last note. Negative segment numbers mean that the segment is reversed in the route.

It makes no sense to try to plot the segments against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Segments can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    segments    file1        route1@00
```

Typical output:

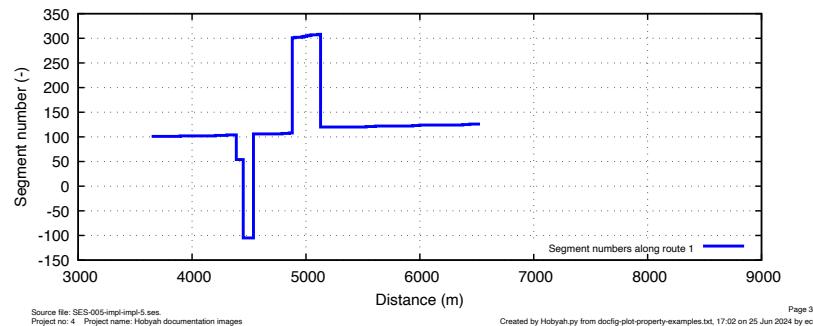


Figure 6.23: Section numbers along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range –999 to 999

Data for **segments** are stored in a pair of lists in each form 8 dictionary. The lists are named **seg_chs** (X-values) and **seg_list** (Y-values).

6.3.13 The subsegs plot type

subsegs is a profile plot of the count of subsegments in the segments in an SES route. It is not as useful as a closely-related curve type, **sublength**.

It makes no sense to try to plot the count of subsegments against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Subsegs can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    subsegs     file1        route1@00
```

Typical output:

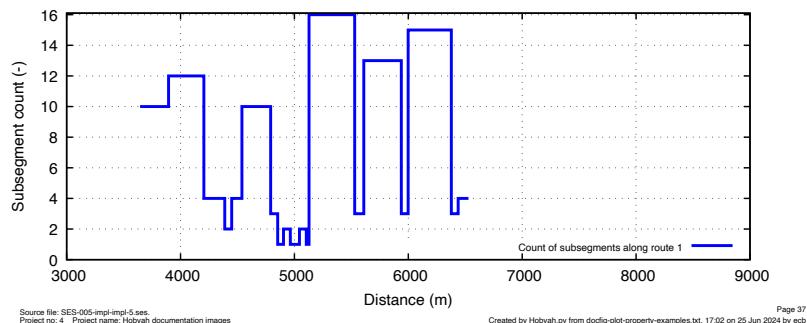


Figure 6.24: Subsegment counts along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range 1 to infinity

Data for **subsegs** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the count of subsegments in each segment that the route runs through.

6.3.14 The sublength plot type

Sublength is a profile plot of the length of subsegments in segments against distance along an SES route. It is particularly useful for ensuring that the subsegments are all roughly the same length in runs involving temperature calculations.

It makes no sense to try to plot the subsegment length against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Sublength can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword      property      nickname      source@discard
profile     sublength     file1         route1@0
```

Typical output:

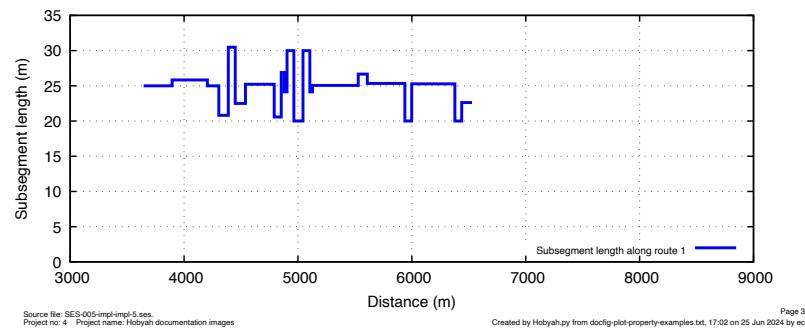


Figure 6.25: Lengths of subsegments along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, range 1 to infinity

Data for **sublength** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the length of subsegments in each segment that the route runs through.

6.3.15 The fireseg plot type

Fireseg is a profile plot of the fire segment/non-fire segment setting against distance along an SES route.

It makes no sense to try to plot the **fireseg** switch against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Fireseg can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    fireseg      file3        route1@00
```

Typical output:

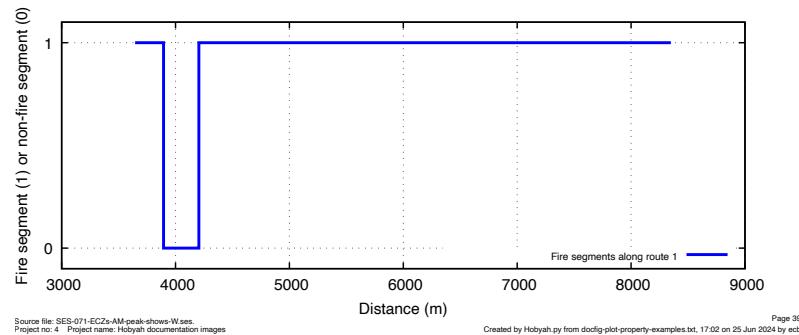


Figure 6.26: Fire segment indicators along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless, could be 0 or 1

Data for **fireseg** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the fire segment/non-fire segment setting in each segment that the route runs through.

6.3.16 The area plot type

Area is a profile plot of the open tunnel area in segments against distance along a route or tunnel. It does not account for the presence of trains: to see the annulus area around trains use the **annulus** plot type instead.

It makes no sense to try to plot the tunnel area against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    area        file8       eastbound@00  # Hobyah route name
profile    area        file8       mainline1@00 # Hobyah tunnel name
profile    area        file1       route1@00   # SES route number
```

Typical output:

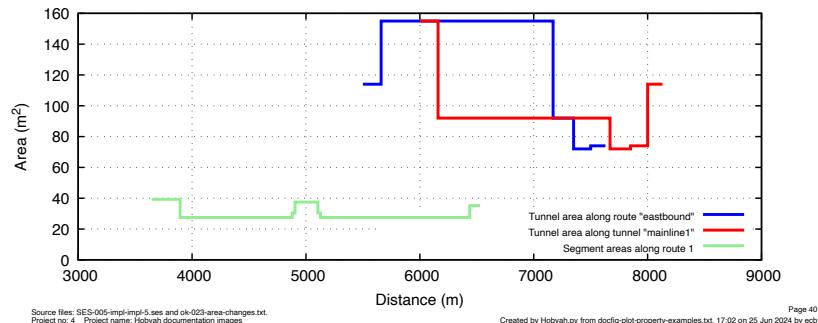


Figure 6.27: Tunnel areas along a route

- Curve types where it is used: **profile**
- Where plotted: along a route (in SES), along tunnels or routes (in Hobyah).
- Units: m², ft²

Data for **area** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the area of each segment that the route runs through.

6.3.17 The perimeter plot type

Perimeter is a profile plot of the tunnel perimeter distance along a route or tunnel.

It makes no sense to try to plot the tunnel perimeter against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    perimeter   file8      eastbound00  # Hobyah route name
profile    perimeter   file8      mainline100 # Hobyah tunnel name
profile    perimeter   file1      route100   # SES route number
```

Typical output:

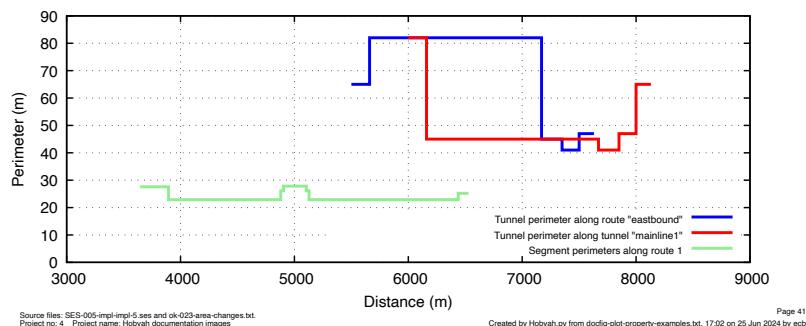


Figure 6.28: Tunnel perimeters along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: m, ft

Data for **perimeter** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the perimeter of each segment that the route runs through.

6.3.18 The roughness plot type

Roughness is a profile plot of the mean tunnel roughness height in segments against distance along a route. In SES it is the “weighted average roughness length” in the .PRN files.

Hobyah allows users to input roughness heights or fixed friction factor in sectype definitions. If a fixed friction factor has been set in a sectype, the roughness height is back-calculated from the friction factor using Colebrook’s expression for the fully-turbulent friction factor.

It makes no sense to try to plot roughness against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    roughness   file8       eastbound00  # Hobyah route name
profile    roughness   file8       mainline100 # Hobyah tunnel name
profile    roughness   file1       route100   # SES route number
```

Typical output:

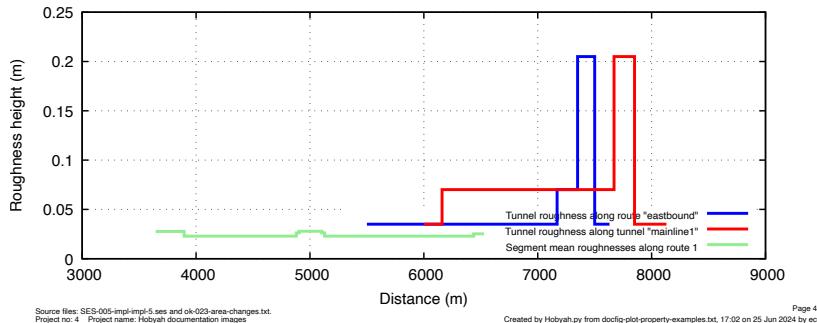


Figure 6.29: Roughness along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: m, ft

Note that the roughness heights are in metres, not millimetres. The reason for using metres is so that roughness heights and fixed friction factors (in Hobyah) are in the

same range as each other. If you want to see mm, use a `ymult:=1000` or `y2mult:=1000` optional argument in the curve definition.

Data for `roughness` are calculated on the fly from a list of segment chainages in the form 8 dictionary, `seg_chs` (the X-values) and by interrogating the form 3 dictionary for the mean roughness of each segment that the route runs through.

6.3.19 The Darcy plot type

Darcy is a profile plot of the fully turbulent Darcy friction factor in segments against distance along a route. In SES is the “Fully turbulent friction factor (from average roughness)” in the .PRN files. In Hobyah it is either the specified friction factor in a **sectype** definition or the fully turbulent friction factor calculated from the roughness height, area and perimeter in a **sectype** definition.

It makes no sense to try to plot fully turbulent Darcy friction factor against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    Darcy      file8       eastbound@0  # Hobyah route name
profile    Darcy      file8       mainline1@0 # Hobyah tunnel name
profile    Darcy      file1       route1@0   # SES route number
```

Typical output:

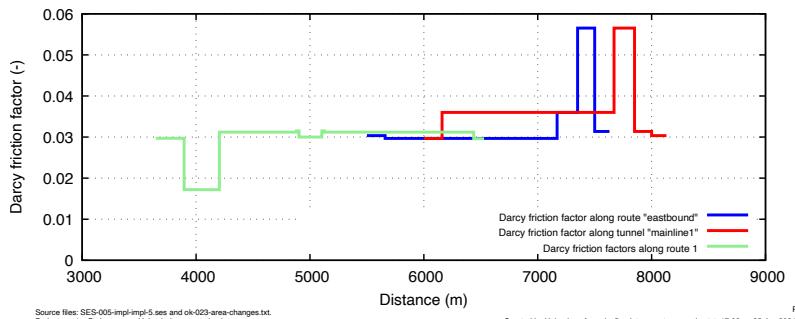


Figure 6.30: Darcy friction factor λ along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless. 0 to a maximum of ≈ 0.5 .

Data for **Darcy** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the fully turbulent Darcy friction factors of each segment that the route runs through.

6.3.20 The Fanning plot type

Fanning is a profile plot of the fully turbulent Fanning friction factor in segments against distance along a route. In SES is the “Fully turbulent friction factor (from average roughness)” in the .PRN files divided by four. In Hobyah it is either the specified friction factor in a **sectype** definition or the fully turbulent friction factor calculated from the roughness height, area and perimeter in a **sectype** definition.

It makes no sense to try to plot the fully turbulent Fanning friction factor against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

When plotting from Hobyah runs, the definition is the name of a route or the name of a tunnel. When plotting from SES runs, the definition is the word **route** followed by the SES route number.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile   Fanning    file8      eastbound@00  # Hobyah route name
profile   Fanning    file8      mainline1@00 # Hobyah tunnel name
profile   Fanning    file1      route1@00   # SES route number
```

Typical output:

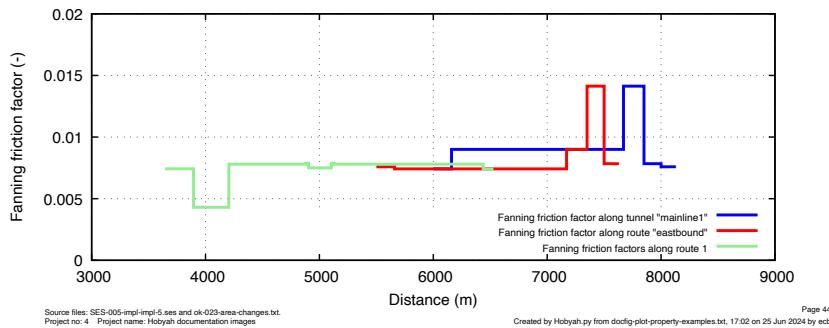


Figure 6.31: Fanning friction factor c_f along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: dimensionless. 0 to a maximum of ≈ 0.1 .

Data for **Fanning** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg_chs** (the X-values) and by interrogating the form 3 dictionary for the fully turbulent Darcy friction factors of each segment that the route runs through and dividing them by four.

6.3.21 The wetted plot type

Wetted is a profile plot of the percentage of the tunnel perimeter that is kept wet by groundwater inflow in segments. It is plotted as percentage against distance along a route. It is also the best example of a “mostly harmless” parameter in SES. Nobody cares a dingo’s kidney about what percentage of the tunnel walls are permanently wetted by groundwater, except under extremely rare circumstances.

It makes no sense to try to plot the wetted perimeter percentage against time, so the time value in the curve definition is ignored.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero (this is only because there are no route definitions in other simulations).

Wetted can only be plotted from SES runs, there is no equivalent in Hobyah runs.

Typical curve definitions:

```
keyword    property    nickname    source@discard
profile    wetted      file1       route100
```

Typical output:

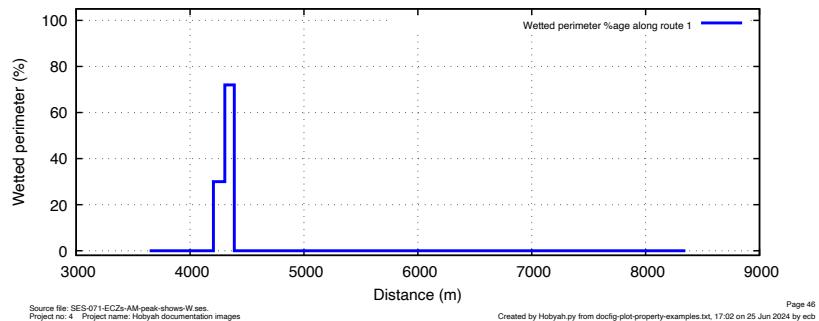


Figure 6.32: Wetted perimeter percentage along a route

- Curve types where it is used: **profile**
- Where plotted: along a route.
- Units: percentage, 0 to 100.

Data for **wetted** are calculated on the fly from a list of segment chainages in the form 8 dictionary, **seg.chs** (the X-values) and by interrogating the form 3 dictionary for the wetted perimeters.

6.4 Plots of Hobyah fan properties

This section describes the behaviour of fans at an instant in time. These may be plotted by referencing the name of the fan and a time. These properties are all plotted on the flow-pressure plane.

See also the `pdiff` plot type, which plots the difference in total pressure and static pressure across objects like fans, dampers and junctions.

Two types of fan characteristic can be plotted: fan total pressure and fan static pressure. The fan characteristic and system characteristic can be plotted, giving four curves in all. Figure 6.33 shows the four together.

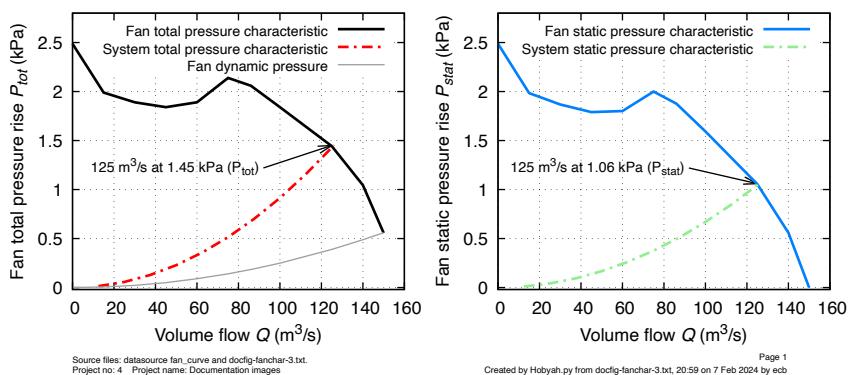


Figure 6.33: Total and static pressure characteristics side by side

See also Section 3.25.4 for more explanation of the difference between fan total pressure and fan static pressure.

6.4.1 The fanchar plot type

Fanchar is the fan total pressure characteristic of a fan, plotted on the flow–total pressure plane. An example can be seen in Figure 3.6. It is a user-defined curve that is adjusted by the fan laws to match the air density at the fan and the fan speed at each instant in time.

Typical curve definition:

```
keyword    property    nickname    fan name@time
fandata    fanchar     file1        20155
```

Typical output:

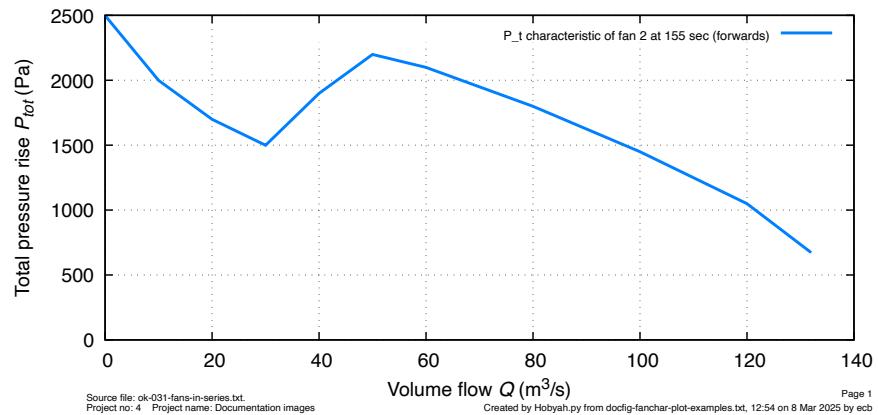


Figure 6.34: Fan total pressure characteristic at an instant in time

- Plot types where it is used: **fanchar**
- Where plotted: at fans
- Units: Volume flows in m^3/s & fan total pressures in Pa. Or volume flows in cfm and fan total pressures in inches of water gauge.

Data for **fanchar** are stored in dictionay named **tunnelfans_dict**. The keys are the name of the fan. Note that this array holds pressure differences for many named items such as fans, dampers and jet fans.

6.4.2 The system plot type

System is the system characteristic of a fan plotted as a curved line on the flow–total pressure plane. An example can be seen in Figure 3.6. It is a curve of the form $\Delta P = RQ^2$ where ΔP is the fan total pressure rise, Q is the volume flow and R is the Atkinson resistance of the system.

Typical curve definition:

```
keyword    property    nickname    fan name@time
fandata    system      file1        2@155
```

Typical output:

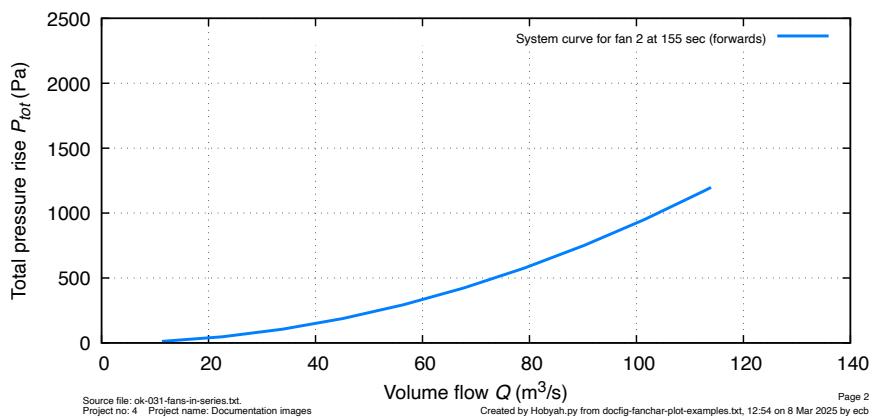


Figure 6.35: System total pressure characteristic at an instant in time

- Plot types where it is used: **fandata**
- Where plotted: at fans
- Units: Pa, inches of water gauge

Data for **system** are stored in Pandas array named **p_diff_dict**. The indices are the plot times and the columns are the name of the fan. Note that this array holds pressure differences for many named items such as fans, dampers and jet fans.

6.4.3 The fanchar-cursed plot type

Fanchar-cursed is the fan characteristic of a fan, plotted on the flow–static pressure plane. An example can be seen in Figure 3.7. It is a user-defined curve that is adjusted by the fan laws to match the air density at the fan and the fan speed at each instant in time.

It can only be plotted from Hobyah data, and only if the user sets a fan diameter in the **fanchar** block.

Typical curve definition:

```
keyword      property      nickname   fan name@time
fadata     fanchar-cursed    file1        2@155
```

Typical output:

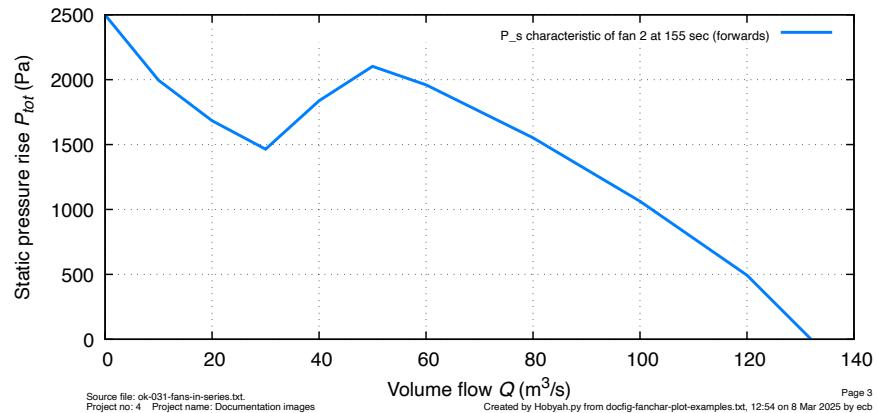


Figure 6.36: Fan static pressure characteristic at an instant in time

- Plot types where it is used: **fadata**
- Where plotted: at fans
- Units: Volume flows in m^3/s & fan static pressures in Pa. Or volume flows in cfm and fan static pressures in inches of water gauge.

The name includes the term **-cursed** because I do not like the concept of fan static pressure. Use fan total pressure instead, folks!

Data for **fanchar-cursed** are calculated on the fly from the fan total pressure characteristic, the air density at the fan outlet, the volume flows in the fan characteristic, the fan diameter and the fan speed at each instant in time.

6.4.4 The system-cursed plot type

System-cursed is the system characteristic of a fan plotted as a curved line on the flow–static pressure plane. An example can be seen in Figure 3.7. It is a curve of the form $\Delta P = RQ^2$ where ΔP is the fan static pressure rise, Q is a volume flow and R is the Atkinson resistance of the system.

It can only be plotted from Hobyah data, and only if the user sets a fan diameter in the **fandata** block.

Typical curve definition:

```
keyword      property      nickname    fan name@time
fandata    system-cursed   file1        20155
```

Typical output:

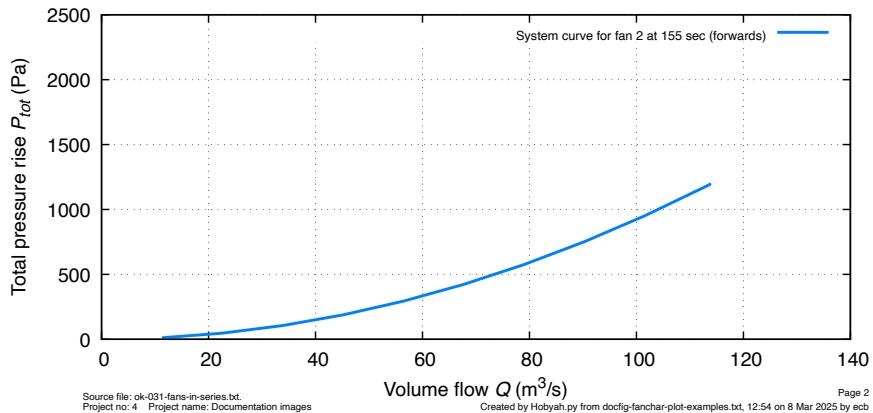


Figure 6.37: System static pressure characteristic at an instant in time

- Plot types where it is used: **fandata**
- Where plotted: at fans
- Units: Pa, inches of water gauge

Data for **system-cursed** are calculated on the fly from the system total pressure loss, the volume flow at the fan outlet, the fan area and the air density at the fan outlet.

6.5 Plots of Hobyah damper properties

This section describes the plotting of the transient behaviour of dampers. These are plotted by referencing the name of the damper.

See also the `pdiff` plot type, which plots the difference in total pressure and static pressure across objects like fans, dampers and junctions.

6.5.1 The damper `area_d` plot type

`Area_d` is a plot of the area of a damper against time. It can only be plotted at dampers of type `damper1`; dampers of type `damper2` do not use area to set the flow resistance. I would have liked to use just `area`, but distinguishing between the area at dampers and the area along tunnels and routes was too difficult.

Typical curve definition:

```
keyword      property    nickname   damper name@discard
transient    area_d     file1       D3@155
```

Typical output:

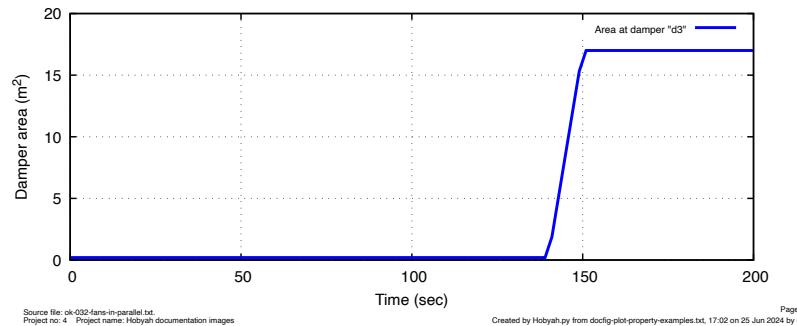


Figure 6.38: Damper area variation with time

- Plot types where it is used: `transient`
- Where plotted: at dampers
- Units: Area in m² or ft².

Data for `area_d` at dampers are stored in a Pandas dataframe called `areas_bin` with indices of plot time (as real numbers) and columns of damper name (as strings).

6.5.2 The damper `zeta_bf` k-factor plot type

`Zeta_bf` is a plot of the k-factor (pressure loss factor) against time for flow through a damper in the positive direction (from the back end of the tunnel to the forward end). It can only be plotted at dampers of type `damper1`; dampers of type `damper2` do not use `zeta_bf` to set the flow resistance.

Typical curve definition:

```
keyword      property      nickname      damper name@discard
transient    zeta_bf      file1          D3@155
```

Typical output:

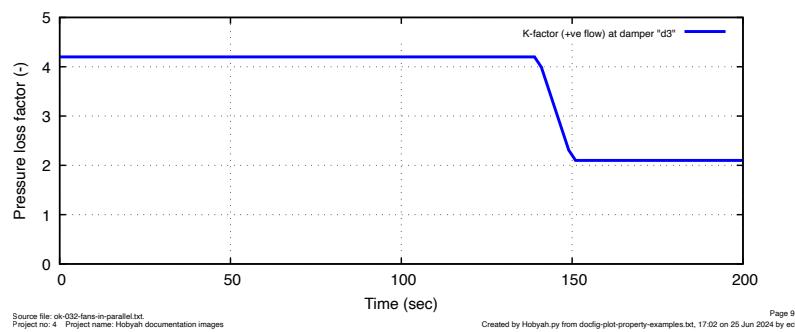


Figure 6.39: Damper k-factor (+ve flow direction, back to forward) variation

- Plot types where it is used: `transient`
- Where plotted: at dampers
- Units: dimensionless.

Data for `zeta_bf` at dampers are stored in a Pandas dataframe called `zetabs_bf_bin` with indices of plot time (as real numbers) and columns of damper name (as strings).

6.5.3 The damper zeta_fb k-factor plot type

Zeta_fb is a plot of the k-factor (pressure loss factor) against time for flow through a damper from the back end of the tunnel to the forward end. It can only be plotted at dampers of type **damper1**; dampers of type **damper2** do not use **zeta_fb** to set the flow resistance.

Typical curve definition:

```
keyword      property      nickname      damper name@discard
transient    zeta_fb      file1          D3@155
```

Typical output:

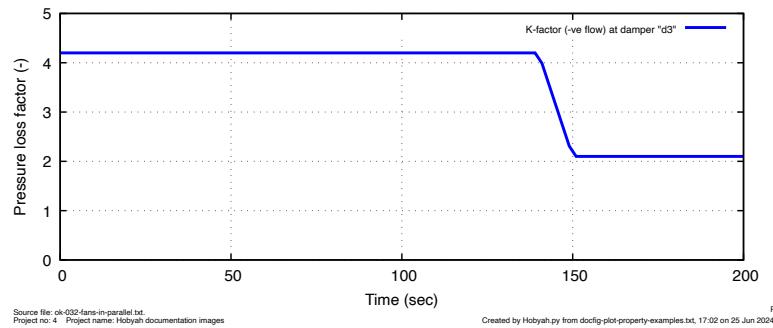


Figure 6.40: Damper k-factor (–ve flow direction, forward to back) variation

- Plot types where it is used: **transient**
- Where plotted: at dampers
- Units: dimensionless.

Data for **zeta_fb** at dampers are stored in a Pandas dataframe called **zetas_fb_bin** with indices of plot time (as real numbers) and columns of damper name (as strings).

6.5.4 The damper R_bf resistance plot type

R_bf is a plot of the flow resistance (Atkinson resistance $\equiv \Delta P/Q^2$) against time for flow through a damper in the positive direction (from the back end of the tunnel to the forward end). It can be plotted at dampers of type **damper1** and **damper2**.

Typical curve definition:

```
keyword      property      nickname      damper name@discard
transient    R_bf        file1          D2@155
```

Typical output:

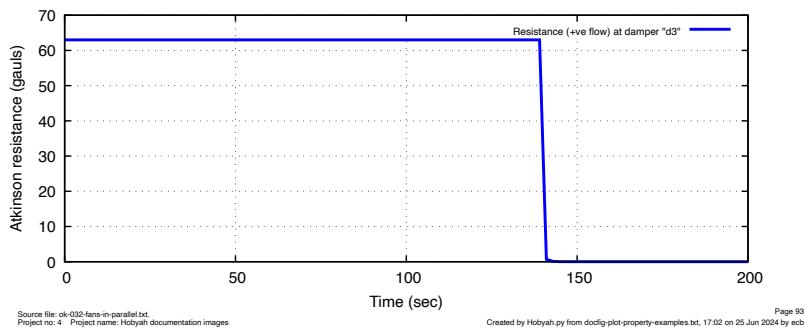


Figure 6.41: Damper resistance (+ve flow direction, back to forward) variation

- Plot types where it is used: **transient**
- Where plotted: at dampers
- Units: $N \cdot s^2/m^8$, also known as “gauls” and $Pa \cdot s^2/m^6$

Data for R_bf at dampers are stored in a Pandas dataframe called **R.bf_bin** with indices of plot time (as real numbers) and columns of damper name (as strings).

6.5.5 The damper R_fb resistance plot type

R_fb is a plot of the flow resistance (Atkinson resistance $\equiv \Delta P/Q^2$) against time for flow through a damper in the negative direction (from the forward end of the tunnel to the back end). It can be plotted at dampers of type `damper1` and `damper2`.

Typical curve definition:

```
keyword      property      nickname      damper name@discard
transient    R_fb        file1          D30155
```

Typical output:

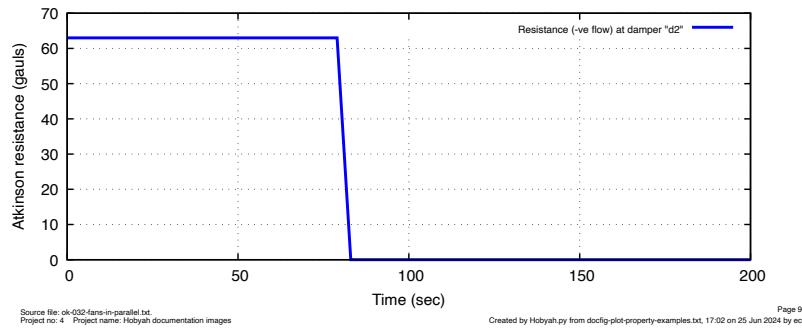


Figure 6.42: Damper resistance (–ve flow direction, forward to back) variation

- Plot types where it is used: `transient`
- Where plotted: at dampers
- Units: $N \cdot s^2/m^8$, also known as “gauls” and $Pa \cdot s^2/m^6$

Data for R_fb at dampers are stored in a Pandas dataframe called `R_fb.bin` with indices of plot time (as real numbers) and columns of damper name (as strings).

6.6 Plots of icons

Icons are a useful way of getting information across to nonspecialists. They let you plot the locations of trains, fires and jet fans alongside profiles. See Figures 3.8 and 3.9 in Section 3.25.5. That section also explains in details the many optional arguments specific to icons that can be used to adjust their size, shape and location.

6.6.1 Trains, fires and jetfans icons

`Trains` is a means of showing the locations of trains along a route. It does not plot a graph, instead it adds gnuplot polygons (icons) to show the locations of each train in a route.

Likewise, `fires` is a means of showing the locations of fires along an SES route and `jetfans` is a means of showing the locations of jetfans along a route.

Train icons have specific optional entries that control how the icons are displayed:

- `profile`, `float`, `height`, `colour`, `color`, `aspect` and `length`

See Section 3.25.5 for a fuller explanation of the `icons` curve type and its optional entries.

Typical curve definitions:

<i>keyword</i>	<i>type of icon</i>	<i>source nickname</i>	<i>route and time</i>	<i>optional argument</i>
icons	trains	SES_run1	route5@500	<code>profile:=route</code>
icons	trains	SES_run1	route5@500	<code>profile:=stacks axes:=x1y2</code>
icons	trains	SES_run1	route5@500	<code>profile:=flat</code>
icons	fires	SES_run1	route5@500	
icons	jetfans	SES_run1	route5@500	

Typical output:

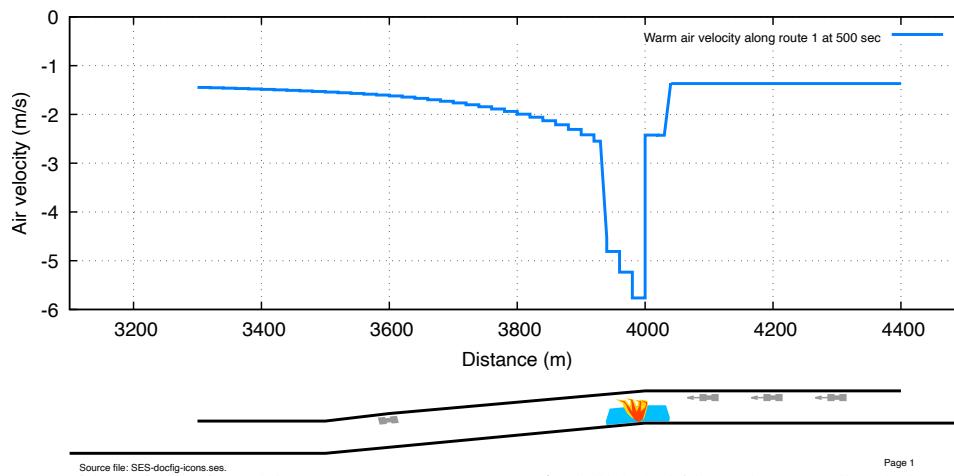


Figure 6.43: Icons curve types don't generate curves, just icons

6.7 Plots of Hobyah traffic properties

This section describes the plotting of the behaviour of road traffic (vehicle densities and vehicle flowrates).

Vehicle density and vehicle flowrate can be plotted at a point against time, or along a route at an instant in time.

The plot keywords match the names of the traffic types given in the `traffictypes` block (see Section 3.19). These are typically things like `car_p` (petrol cars), `car_d` (diesel cars), `LCV_p` (petrol light commercial vehicles), `LCV_d` (diesel light commercial vehicles) and `HGV`, depending on how far you want to break down the traffic mix.

6.7.1 The <trafficname>.flow plot type

<trafficname>.flow is the rate of flow of traffic given the name <trafficname> in the **traffictypes** block (see Section 3.19). The traffic names are typically **car**, **LCV** and **HGV**, which would map to plot properties named **car_flow**, **LCV_flow** and **HGV_flow** instead of the <trafficname>.flow property in the title of this section.

What can I say? If you let the names of plot properties depend on the names defined in the **traffictypes** block, you're going to have to let things get meta.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@location</i>
transient	<trafficname>.flow	file7	eastbound@15200
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	<trafficname>.flow	file7	westbound@230
profile	car_flow	file7	westbound@230
profile	LCV_flow	file7	westbound@230
profile	HGV_flow	file7	westbound@230

Typical output:

Figure 6.44: Traffic flow vs. time and distance

- Plot types where it is used: **transient**, **profile**
- Where plotted: at fixed gridpoints and along routes
- Units: vehicles per hour

6.7.2 The <trafficname>_dens plot type

<trafficname>_dens is the density of traffic given the name <trafficname> in the **traffictypes** block (see Section 3.19). The traffic names are typically car, LCV and HGV, which would map to plot properties named car.dens, LCV.dens and HGV.dens instead of the <trafficname>_dens property in the title of this section.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@location</i>
transient	<trafficname>_dens	file7	eastbound@15200
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	<trafficname>_dens	file7	westbound@230
profile	car.dens	file7	westbound@230
profile	LCV.dens	file7	westbound@230
profile	HGV.dens	file7	westbound@230

Typical output:

Figure 6.45: Traffic density vs. time and distance

- Plot types where it is used: **transient**, **profile**
- Where plotted: at fixed gridpoints and along routes
- Units: vehicles per km

6.8 Plots of SES transient properties in tunnels

This section describes the curves of transient properties in the tunnels from SES.

Some properties vary in each subsegment, such as temperature; some vary in each segment, such as air velocity; and one (pressure) varies in each section.

`SESconv.py` generates some derived properties that account for the presence of trains and density changes. These properties (annulus area, volume flow in the annulus and air velocity in the annulus) are calculated at each subpoint in the subsegments.

Engineers who have used SES will be familiar with the concept of sections, segments and subsegments (they are explained in sections 3 and 5 of the SES User's Manual). But you will be unfamiliar with subpoints.

Subpoints are a finer division of subsegments. Subpoints are used in `SESconv.py` to track the effect of having trains passing through subsegments. SES calculates annulus areas and air velocities/volume flows in the annulus around a train for its pressure calculation but does not print them. `SESconv.py` calculates them and stores their values at the back end, midpoint and forward end of each subsegment.

These three points (back end, midpoint and forward end) in each subsegment have been termed "subpoints" during the development of Hobyah and are illustrated in Figure 6.46. It shows two segments (107 and 301) that each have two subsegments.

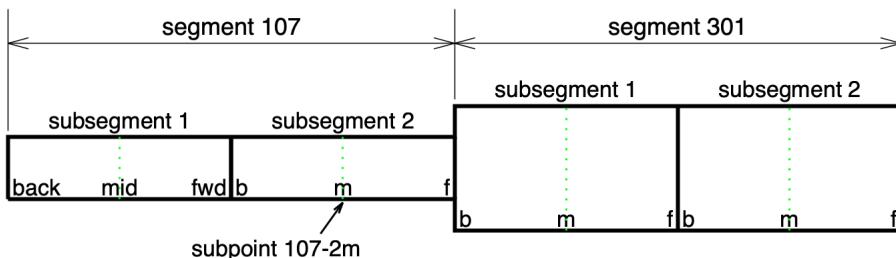


Figure 6.46: Subpoints within subsegments

The subsegment identifiers are 107-1, 107-2, 301-1 and 301-2 (same as in the SES output file but without spaces).

Subpoints identify the segment number, subsegment number and "b", "m" or "f" to signify the back end, midpoint or forward end of the subsegment, like 107-2m in Figure 6.46. I generally use subsegments that are 20–30 m apart so this puts the subpoints 10–15 m apart. I have reviewed engineering submissions from engineers who chose to use the longest subsegments possible in SES (one subsegment per segment) and while those runs worked for ECZ calculations, they don't work for fire runs.

Properties in tunnels can be plotted in three ways:

- as transient plots at a fixed points in a tunnel,
- as profiles along a route at a fixed time and
- as transient plots on a moving train (plotted against time or train location).

The following lists the keywords in segments, a short description of the property and which locator it uses (section, segment, subsegment or subpoint).

- **qSES**: cold volume flow in the open tunnel (segments)
- **qcold**: cold volume flow in the open tunnel and in the annulus around trains (subpoints)
- **qwarm**: warm volume flow in the open tunnel and in the annulus around trains (subpoints)
- **vSES**: cold air velocity in the open tunnel (segments)
- **vcold**: cold air velocity in the open tunnel and in the annulus around trains (subpoints)
- **vwarm**: warm air velocity in the open tunnel and in the annulus around trains (subpoints)
- **dp**: total pressure drop (sections)
- **DB**: dry-bulb air temperature (subsegments)
- **wall_temp**: tunnel wall surface temperature (subsegments)
- **W**: humidity of the air in kg/kg (subsegments)
- **sens**: sensible heat gain to the air (subsegments)
- **sens_pul**: sensible heat gain per unit length to the air (subsegments)
- **lat**: latent heat gain to the air (subsegments)
- **lat_pul**: latent heat gain per unit length to the air (subsegments)
- **annulus**: annulus area around trains (subpoints)

The following is a sample of curve definitions that can be used to plot transient curves at a stationary point or moving point:

<i>keyword</i>	<i>property</i>	<i>nickname of source</i>	<i>locator</i>
transient	dp	slugflow1	sec508 # SES section
transient	qses	slugflow1	403 # SES segment
transient	db	slugflow1	612-5 # SES subsegment
transient	vcold	slugflow1	123-2m # SES subpoint
transient	vcold	slugflow1	route5@1304.5
transient	vcold	slugflow1	train12@noseopen

The first four reference a specific SES section, segment, subsegment and subpoint.

The fifth is a distance along a route. In this case Hobyah will figure out which plot location is closest to chainage 1304.5 on route 5 and use that (the curve data file will contain lines of QA data that let you know which subpoint was selected so that you can check whether it is the correct one or not). There are a few rules that the software follows in cases of ambiguity:

- If the point is in the open air, a default value is used (such as zero velocity or the outside air temperature).
- If the chainage in a Hobyah route plot is exactly on a boundary (boundaries have two or more gridpoints at the same chainage), error 7121 is raised, telling you to move the plot location off the boundary so it can figure out which gridpoint to use.
- If the chainage in an SES route plot is exactly on the boundary between two subsegments, the section, segment, subsegment or subpoint on the up side of the route is used.
- If the distance along a Hobyah tunnel is exactly halfway along a cell, the properties at the gridpoint closest to the back end of the cell are used.
- If the chainage in an SES route plot is exactly half way between two subpoints, the subpoint on the up side of the route is used.
- If the distance along an SES section or an SES segment is exactly on the boundary between two subsegments, the section, segment, subsegment or subpoint closest to the back end of the segment is used.

The sixth is a moving plot point. The phrase `train12@noseopen` means “whichever plot location in the open tunnel in front of the nose of train 12 is the closest” as the train moves through the tunnel. Plots relative to trains are useful when calculating volume flows of air relative to the intakes of diesel locomotives and air temperatures near the intakes of air conditioning units on trains travelling through the tunnels. They are not implemented yet (as they are tricky to verify that you’ve programmed every case correctly).

Sections may have multiple segments, segments may have multiple subsegments and subsegments each have three subpoints. Hobyah can figure out the correct key to use if you use a key that is more detailed than is needed.

For example, let’s say that you want to plot the pressure drop in section 502. The correct key to use in the curve definition is `sec502`. Let’s say that section 502 contains three segments numbered 505, 506 and 507, each with two subsegments.

The least ambiguous definition is

```
transient      dp <source file> sec502
```

But Hobyah knows that section 502 contains segments 505, 506 and 507. So the following segment-based definitions are allowed:

```
transient      dp <source file> 505
transient      dp <source file> 506
transient      dp <source file> 507
```

Hobyah will spot that “505”, “506” and “507” are all integers and interpret them as segment numbers, figure out that they all belong to section 502 and substitute “sec502” for them. The same goes for subsegment definitions and subpoint definitions:

```
transient      dp <source file> 505-1      # subsegment definitions
transient      dp <source file> 505-2
transient      dp <source file> 507-1
transient      dp <source file> 507-2
transient      dp <source file> 506-1b      # subpoint definitions
transient      dp <source file> 506-1m
transient      dp <source file> 506-1f
```

These all get narrowed from their subsegment/subpoint definitions to the section number, **sec502**.

The same approach works for segment-based properties: if you specify a subsegment or subpoint location it will figure out which segment to use.

6.8.1 The qSES and -qSES plot types

Qses is the volume flow of air in the open tunnel, ignoring the presence of trains. It is a cold volume flow (i.e. no correction for subsegment air density). It has the same value on both sides of a fire.

Qses is rarely used. **Qcold** and **qwarm** (see below) are more useful. Its only real use is to provide traceability: values of **qses** can be directly compared to the volume flow printed for each segment in the SES output file.

In some circumstances it is useful to be able to multiply the values being plotted by -1 . The easiest way to do this is to use the **-qses** keyword instead of **qses**, as **classSES.py** will do the multiplying for you.

Typical curve definitions:

keyword	property	nickname	source
transient	qSES	file3	103-1
keyword	property	nickname	source@time
profile	qses	file3	route1@900

Typical output:

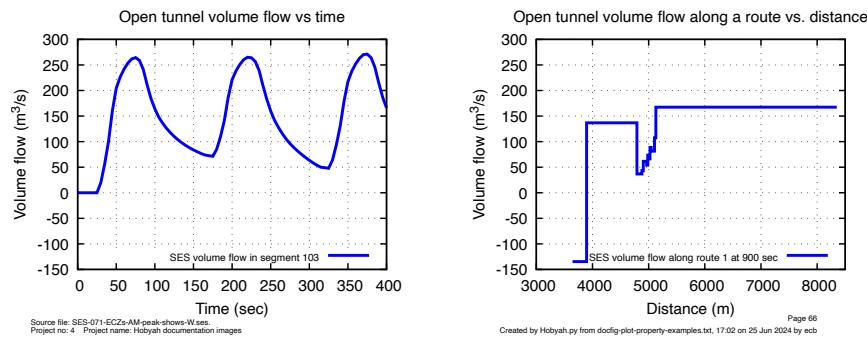


Figure 6.47: SES volume flow vs. time and distance

- Plot types where it is used: **transient**, **profile**
- Where plotted: one value for each segment.
- Units: m^3/s , cfm

Data for **qses** are stored in a Pandas array called **seg_volfows** with indices of plot time (as real numbers) and columns of segment number (as integers).

6.8.2 The `qcold` and `-qcold` plot types

`qcold` is the volume flow of air in the tunnel or the annulus around trains. It is a cold volume flow, meaning that it has the same value on both sides of a fire (i.e. no correction for subsegment air density).

Values of `qcold` at a point are calculated by taking the volume flow of air `qSES` at that point and adding/subtracting the volume flow of moving trains at that point. The “volume flow of moving trains” is a bit counterintuitive the first time you hear of it, so the following example may help.

Say we have a train moving through a tunnel at an instant in time. The train induces an airflow of $100 \text{ m}^3/\text{s}$ in front of it. The train has area 9 m^2 and a speed of 60 km/h (16.667 m/s). The train is displacing $9 \times 16.667 = 150 \text{ m}^3/\text{s}$ of air volume flow. The volume flow of air in the annulus (relative to the tunnel wall) is $100 - 150 = -50 \text{ m}^3/\text{s}$. So -50 is what `qcold` returns.¹

Typical curve definitions:

```
keyword      property      nickname      source
transient    qcold        file3         103-1
keyword      property      nickname      source@time
profile      qcold        file3         route1@900
```

Typical output:

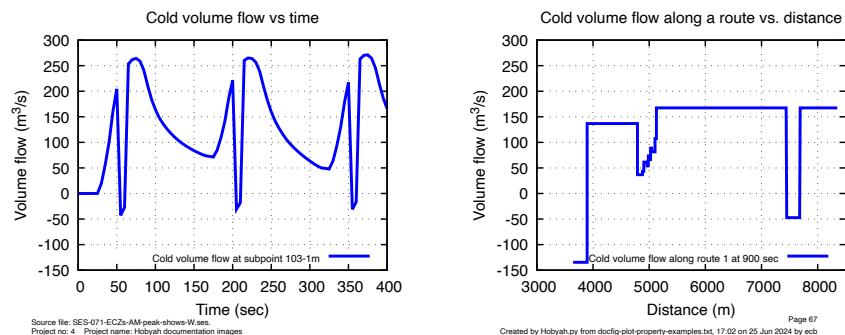


Figure 6.48: Cold volume flow vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: m^3/s , cfm

Figure 6.48 can be compared to Figure 6.47—the only difference is the effect of moving trains.

If you use `-qcold` instead of `qcold`, `classSES.py` will multiply the values by -1 .

¹I could put in a more detailed explanation with diagrams. But I've talked through this with dozens of graduate mechanical engineers over the years. I eventually figured out that the best way to give them a gut-level understanding was to order them to figure it out on paper three or four times. Go and do the same: it's not a pleasant experience, but it works.

Data for `qcold` are stored in a Pandas array called `subpoint_coldflows` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m")

6.8.3 The `qtrains` plot type

`Qtrains` is the volume flow of trains.

Its only purpose is to verify how the calculation of volume flows in the annulus around moving trains is done. See Section 5 of the verification and validation document, where it is used.

- Curve types where it is used: `transient`
- Where plotted: one value for each train. It can be plotted against time or against train nose location.
- Units: m^3/s , cfm

Data for `qtrains` are calculated on the fly whenever it is plotted (by multiplying the train speed in m/s by the area of the train type in m^2).

6.8.4 The `qwarm` and `-qwarm` plot types

`Qwarm` is the volume flow of air in the tunnel or the annulus around trains. It is a warm volume flow, meaning that it has different value on both sides of a fire (i.e. correcting for subsegment air density).

Values of `qwarm` at a point are calculated by taking the volume flow of air `qcold` at that point and dividing by the density factor of air in the subsegment. This replicates a calculation used in SES to adjust for density in the friction term of each subsegment.

Take an example. Say that outside the tunnel the air temperature is 20 °C and that in the segment that has the fire the air temperature is 300 °C. SES calculates a density correction for the fire subsegment:

$$\text{correction} = \frac{300 + 273.15}{20 + 273.15} = 1.955. \quad (6.1)$$

`SESconv.py` replicates that calculation and uses the density correction to turn cold volume flow upwind of the fire to warm volume flow downwind of the fire. The calculation is verified in the document `verification+validation.pdf`.

Typical curve definitions:

keyword	property	nickname	source
transient	<code>qwarm</code>	<code>file2</code>	<code>120-3</code>
keyword	property	nickname	<code>source@time</code>
profile	<code>qwarm</code>	<code>file2</code>	<code>route1@900</code>

Typical output:

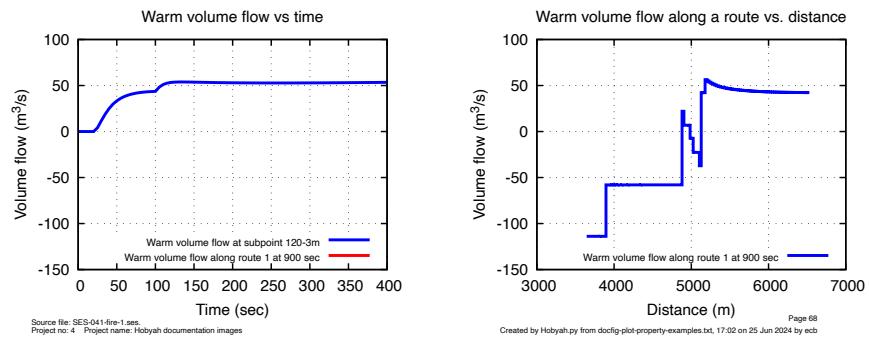


Figure 6.49: Warm volume flow vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: m^3/s , cfm

If you use `-qwarm` instead of `qwarm`, `classSES.py` will multiply the values by -1 .

Data for `qwarm` are stored in a Pandas array called `subpoint_warmflows` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m")

6.8.5 The vSES and -vSES plot types

VSES is the air velocity in the open tunnel, ignoring the presence of trains. It is a cold air velocity, it has the same value on both sides of a fire (no correction for subsegment density).

Vses is not used often; **vcold** and **vwarm** are more useful. Its only real use is traceability: values of **vses** can be directly compared to the air velocities printed in the SES output file.

Typical curve definitions:

```
keyword      property      nickname      source
transient    vSES          file2         203-1
keyword      property      nickname      source@time
profile      vses          file2         route2@600
```

Typical output:

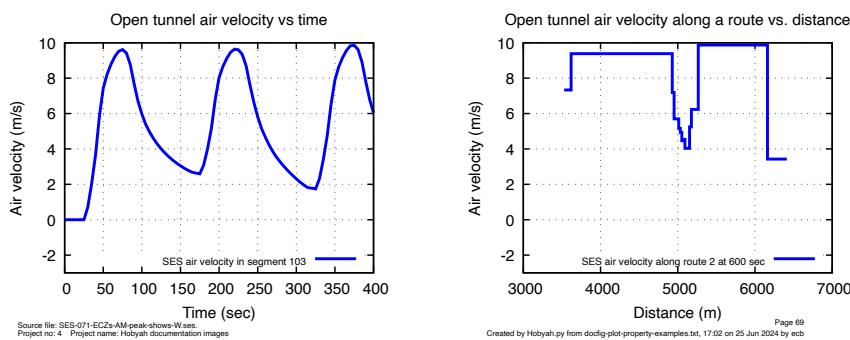


Figure 6.50: Open tunnel air velocity vs. time and distance

- Plot types where it is used: **transient**, **profile**
- Where plotted: one value for each segment.
- Units: m/s, fpm

If you use **-vses** instead of **vses**, **classSES.py** will multiply the values by **-1**.

Data for **vses** are stored in a Pandas array called **seg_vels** with indices of plot time (as real numbers) and columns of segment number (as integers).

6.8.6 The `vcold` and `-vcold` plot types

`vcold` is the air velocity in the tunnel or the annulus around trains. It is a cold air velocity, it has the same value on both sides of a fire (no correction for air density in subsegments).

Values of `vcold` at a point are calculated by taking the volume flow from `qcold` at that point and dividing by the annulus area there.

Typical curve definitions:

```
keyword      property      nickname      source
transient    vcold        file2         203-1
keyword      property      nickname      source@time
profile      vcold        file2         route2@600
```

Typical output:

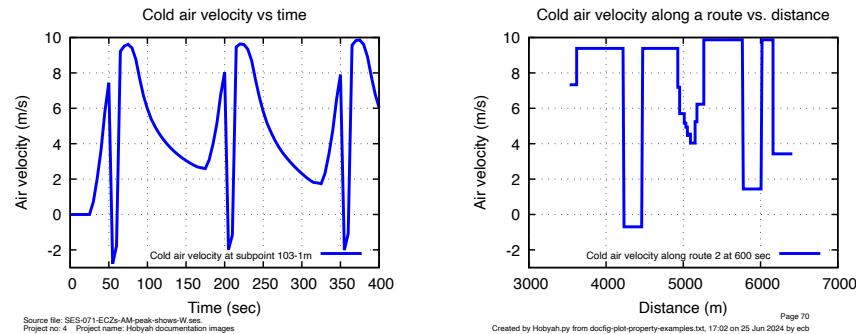


Figure 6.51: Cold air velocity vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: one value for each segment.
- Units: m/s, fpm

Figure 6.51 can be compared to Figure 6.50—the only difference is the effect of moving trains.

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: m/s, fpm

If you use `-vcold` instead of `vcold`, `classSES.py` will multiply the values by -1 .

Data for `vcold` are stored in a Pandas array called `subpoint_coldvels` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m")

6.8.7 The `vwarm` and `-vwarm` plot types

`Vwarm` is the air velocity in the tunnel or the annulus around trains. It is a warm air velocity, by which it is meant that there is a correction for local subsegment air density.

Values of `vwarm` at a point are calculated by taking the volume flow from `qwarm` at each subpoint and dividing by the annulus area there.

Typical curve definitions:

```
keyword      property      nickname      source
transient    vcold        file2         203-1
keyword      property      nickname      source@time
profile      vcold        file2         route2@600
```

Typical output:

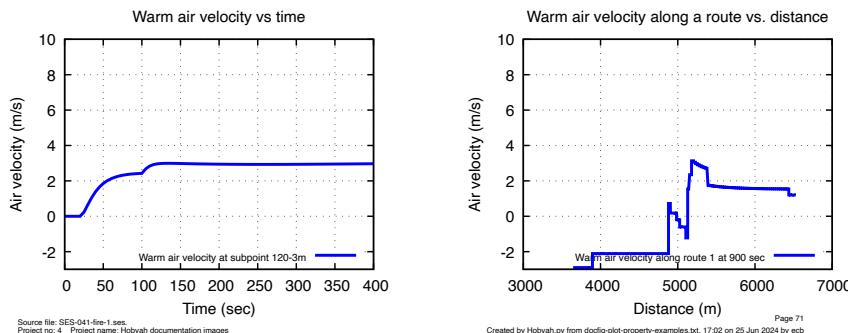


Figure 6.52: Warm air velocity vs. time and distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: one value for each segment.
- Units: m/s, fpm

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: m/s, fpm

If you use `-vwarm` instead of `vwarm`, `classSES.py` will multiply the values by -1 .

Data for `vwarm` are stored in a Pandas array called `subpoint_warmvels` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m")

6.8.8 An SES volume flow/air velocity recap

This section draws together the points made in subsections 6.8.1 to 6.8.7 describing the curve types `qSES`, `qcold`, `qwarm`, `vSES`, `vcold` and `vwarm`.

`QSES` and `vSES` are the volume flow and air velocity printed in the SES output file. Both ignore the effects of passing trains reducing the tunnel area and density variations due to fires. They are not useful for engineering calculations, but they are useful for checking that what is in the `.PRN`, `.TMP` or `.OUT` file has been interpreted correctly by `SESconv.py`.

Users of SES may appreciate a refresher about what `qSES`, `qcold`, `qwarm`, `vSES`, `vcold` and `vwarm` are and how they differ from one another.

- `qSES` is the volume flow printed in SES the output file. It ignores the volume flow of moving trains and is not adjusted for subsegment air density.
- `qcold` is the volume flow at a subpoint, accounting for the volume flow of passing trains. It is not adjusted for subsegment air density.
- `qwarm` is the volume flow at a subpoint, accounting for the volume flow of passing trains and the subsegment air density.
- `vSES` is the air velocity printed in the SES output file. It ignores the presence of trains and is not adjusted for subsegment air density.
- `vcold` is the air velocity at a subpoint, accounting for the volume flow of passing trains and the annulus area. It is not adjusted for subsegment air density.
- `vwarm` is the volume flow at a subpoint, accounting for the volume flow of passing trains, the annulus area and the subsegment air density.

Figure 6.53 shows two graphs plotted from SES that illustrate the difference between `qcold` and `qwarm` and the difference between `vcold` and `vwarm`.

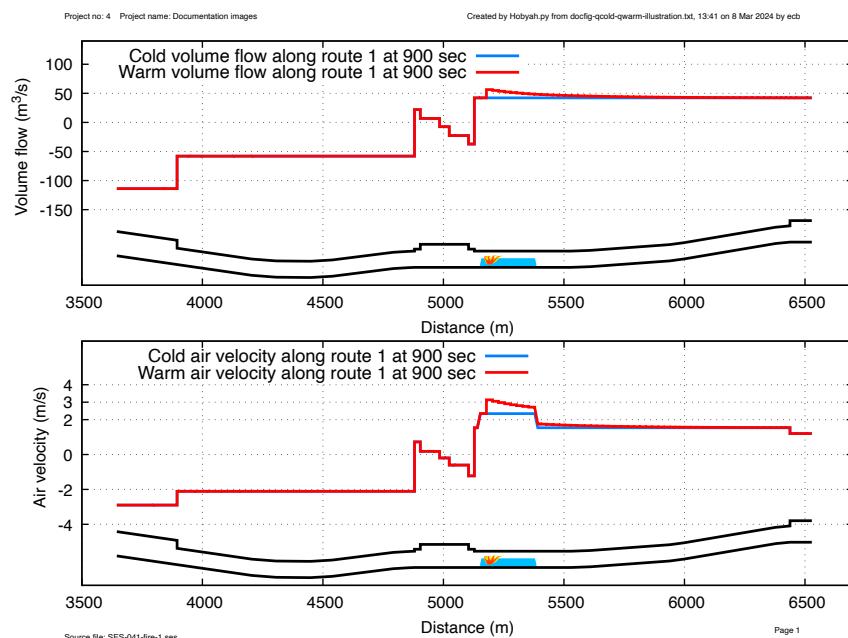


Figure 6.53: Comparison of warm and cold volume flows & velocities

The system being modelled is a rail tunnel around 3 km long, with a station in the middle. There are vent shafts at the ends of the station. The schematic at the base of the two graphs shows the layout: a vertical profile showing the tunnel's gradients. There is a train stopped in the tunnel, on fire. The shafts at each end of the station are supplying air to the tunnels with the aim of blowing the smoke away from the station.

The upper graph shows the cold and warm volume flows. They are identical until the line for the warm volume flow reaches the fire, at which point the converter factors in the change of air density as the air flows over the fire.

The lower graph shows the cold and warm air velocities. Both air velocities are low in the open tunnel and increase when the curves reach the annulus area around the train. The warm air velocity increases once the curve reaches the fire location, then drops off gradually as the heat in the air transfers to the tunnel walls downwind of the fire.

6.8.9 The `dp` and `-dp` plot types

The `dp` curve keyword plots the total pressure change across sections.

When plotted against time, it is the pressure change in a particular section. When plotted against distance along a route, it is the cumulative change in pressure, assuming zero pressure at the route's entry portal.

If you want to plot a profile of the individual total pressure changes along a route instead of the cumulative pressure profile, use the `dp_indiv` keyword instead of `dp`.

Typical curve definitions:

```
keyword      property      nickname      source
transient    dp           file2         203-1
keyword      property      nickname      source@time
profile      dp           file2         route20600
```

Typical output:

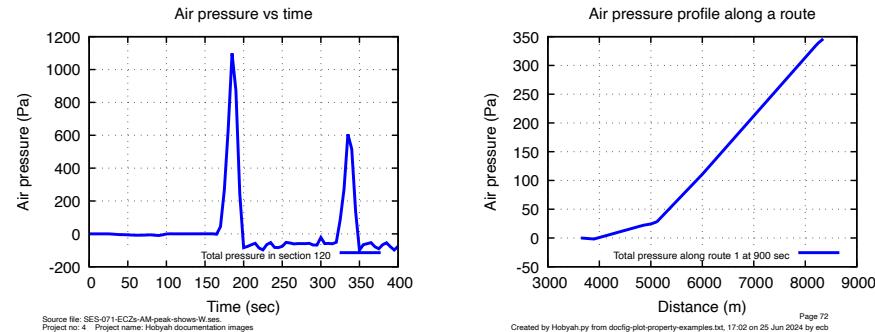


Figure 6.54: Air pressure vs. time and pressure profile vs. distance

- Plot types where it is used: `transient`, `profile`
- Where plotted: one value for each segment.
- Units: Pa, inches of mercury

- Curve types where it is used: `profile`
- Where plotted: one value in each section.
- Units: Pa, inches of mercury

Data for `dp` are stored in a Pandas array called `sec_DPs` with indices of plot time (as real numbers) and columns of section number (as strings, like "`sec235`").

6.8.10 The `dp_indiv` plot type

The `dp_indiv` curve keyword plots the total pressure change across individual sections.

If you want to plot a profile of cumulative total pressure change along a route, use the `dp` keyword instead of `dp_indiv`.

Typical curve definitions:

```
keyword    property    nickname    source@time
profile    dp_indiv    file2        route20600
```

Typical output:

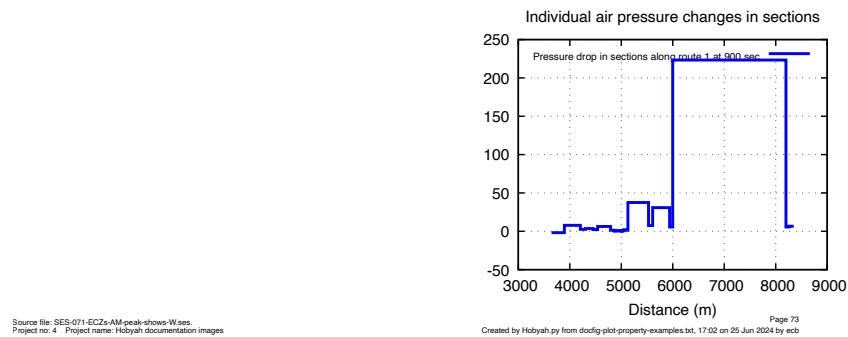


Figure 6.55: Air pressure changes across segments vs. distance

- Plot types where it is used: `transient, profile`
- Where plotted: one value for each segment.
- Units: m/s, fpm
- Curve types where it is used: `transient, profile`
- Where plotted: one value in each section.
- Units: Pa, inches of mercury

Data for `dp_indiv` are stored in a Pandas array called `sec_DPs` with indices of plot time (as real numbers) and columns of section number (as strings, like "`sec235`").

Pressure can only be plotted from SES runs in which the supplementary print option in form 1C is 3 or 5.

6.8.11 The DB plot type

DB is the dry-bulb temperature of air in the tunnel.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source</i>
transient	DB	file2	203-1
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	db	file2	route20600

Typical output:

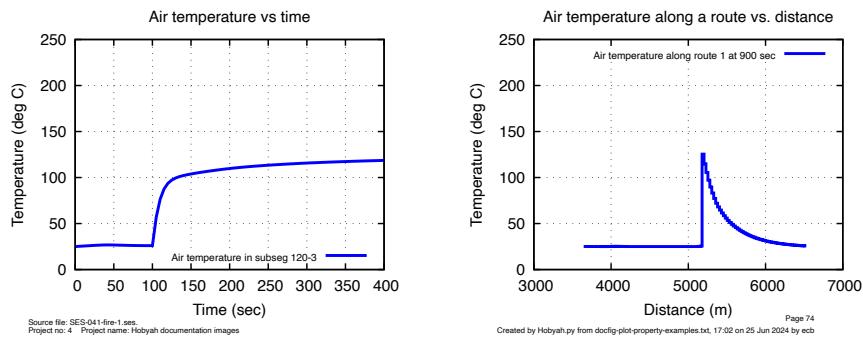


Figure 6.56: Dry bulb temperature vs. time and distance

Dry bulb temperature can be plotted along cooling pipes in SVS runs. This is so that the air temperature and pipe coolant temperature along a pipe can be plotted next to one another.

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source</i>
transient	DB	file2	203-1
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	db	file2	route20600

Typical output:

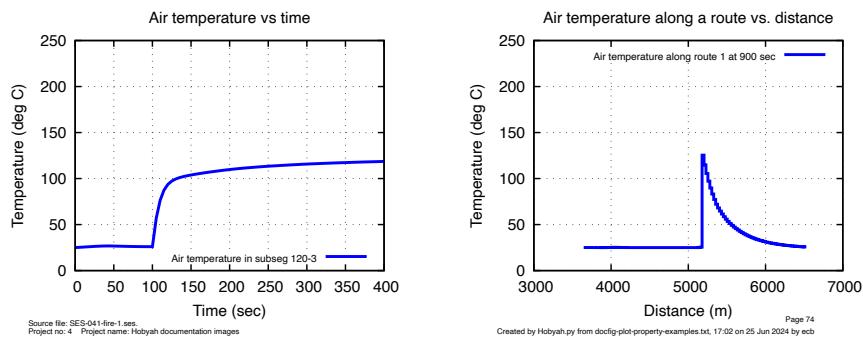


Figure 6.57: Dry bulb temperature vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value in each subsegment.
- Units: °C, °F

Data for DB are stored in a Pandas array called `subseg.temps` with indices of plot time (as real numbers) and columns of subsegment number (as strings, like "102-7")

6.8.12 The wall_temp plot type

`Wall_temp` is the temperature of the inside surface (intrados) of the tunnel walls.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source</i>
<code>transient</code>	<code>wall_temp</code>	<code>file2</code>	<code>203-1</code>
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
<code>profile</code>	<code>wall_temp</code>	<code>file2</code>	<code>route2@600</code>

Typical output:

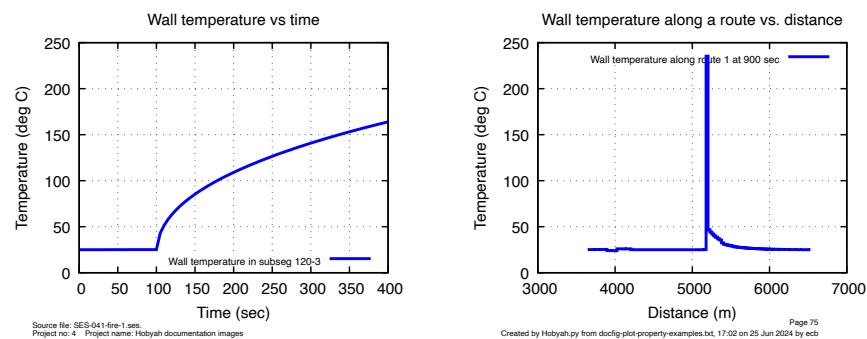


Figure 6.58: Wall temperature vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value in each subsegment.
- Units: °C, °F

Data for `wall_temp` are stored in a Pandas array called `subseg_walltemps` with indices of plot time (as real numbers) and columns of subsegment number (as strings, like "102-7").

Values of `wall_temp` may be updated under the following circumstances:

- after a PM peak hour environmental control zone (ECZ) estimate
- after an AM peak hour ECZ estimate
- at every wall temperature timestep in a fire run

6.8.13 The W plot type

W is the water content of air (kg of water per kg of dry air), also known as humidity ratio.

Typical curve definitions:

```
keyword      property      nickname      source
transient    W            file2         203-1
keyword      property      nickname      source@time
profile     W            file2         route2@600
```

Typical output:

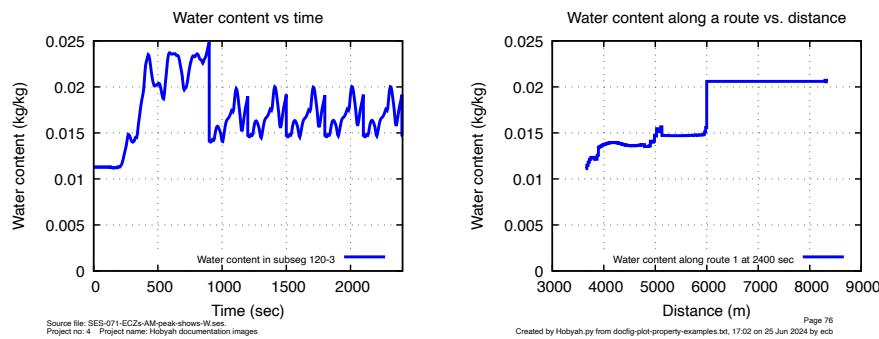


Figure 6.59: Humidity vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value in each subsegment.
- Units: kg/kg, lb/lb

Hobyah does not plot humidities as wet bulb (WB) temperatures or relative humidities (RH), for the following reasons:

- When using a mixture of abbreviated prints and detailed prints, the abbreviated prints are always given as water content even if you want to see WB temperature or RH.
- The routines used in SES to calculate wet bulb temperature and relative humidity were written in the 1970s. Allegedly more accurate routines have been developed in the decades since by the likes of ASHRAE and CIBSE: I don't want to have to figure out what is the best one to use. Far easier just to stick to water content in kg/kg (\equiv lb/lb, which is handy).

Data for W are stored in a Pandas array called `subseg_humids` with indices of plot time (as real numbers) and columns of subsegment number (as strings, like "102-7")

6.8.14 The sens and sens_pul plot types

`Sens` is the sensible heat gain in subsegments (watts per subsegment). `Sens_pul` is the sensible heat gain expressed per unit length: watts per metre in SI units (BTU/hr-ft in US units), because watts per subsegment is mostly useless for plotting profiles. On many occasions I've written incredibly complex spreadsheets to generate heat gains in segments for SES form 3D. I always worry about these spreadsheets because it is so easy to get the arithmetic wrong. Being able to plot a profile in W/m directly from the output makes checking the input of an engineering calculation much easier.

There is one issue to be aware of when checking that the values in the output files against the values in the input file. In SES v4.1 the values of heat gains in the runtime printout are rounded to the nearest 0.1 BTU/sec (equivalent to the nearest 105.4 watts). Those printouts have poor accuracy.

If you find a mismatch between the values you put into the input file and the values that the output states, it may be due to this rounding. I've often had heat gains that happen to get rounded badly in the printouts. For example, consider having 14 W/m for lighting/ third rail heat gains in a 26 m long subsegment. This is $14 \times 26 = 364$ W (0.3453 BTU/sec) in the subsegment. When printed in the SES output file, it is rounded to 0.3 BTU/sec. The converter turns this into 316 W, about 13% lower than the 364 W in the input file².

Typical curve definitions:

keyword	property	nickname	source
transient	sens	file3	120-3
transient	sens_pul	file3	120-3
keyword	property	nickname	source@time
profile	sens	file3	route1@2400
profile	sens_pul	file3	route1@2400

Typical output:

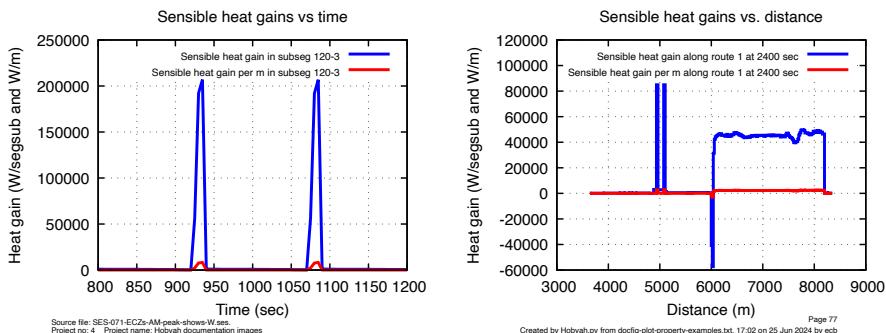


Figure 6.60: Sensible heat gains vs. time and distance

²Rounding in output files can be a serious problem when generating new input files from output files. It often makes sense to change the Fortran `FORMAT` statements in modified versions of SES so that the output is printed to more decimal places. In the case of the heat gains, I changed the `FORMAT` statement from “`F14.1`” to “`1X, F13.4`”, which means that my modified version of SES `offline-SES` prints heat gains to the nearest 0.0001 BTU/sec (0.1 W).

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value in each subsegment.
- Units: `sens` is in watts per segment and BTU/hr per segment. `Sens_pul` are in watts/m and BTU/hr-ft.

Data for `sens` and `sens_pul` are stored in a Pandas array called `subseg_sens` with indices of plot time (as real numbers) and columns of subsegment number (as strings, like "102-7"). The values are in W and when `sens_pul` is used the contents are converted to W/m or BTU/hr-ft on the fly.

6.8.15 The lat and lat_pul plot types

`Lat` is the latent heat gain in subsegments (watts per subsegment). `Lat_pul` is the latent heat gain expressed per unit length: watts per metre in SI units (BTU/hr-ft in US units), because watts per subsegment is mostly useless for plotting profiles.

Typical curve definitions:

```

keyword      property      nickname      source
transient    lat          file2         353-2
transient    lat_pul     file2         353-2
keyword      property      nickname      source@time
profile      lat          file2         route2@600
profile      lat_pul     file2         route2@600

```

Typical output:

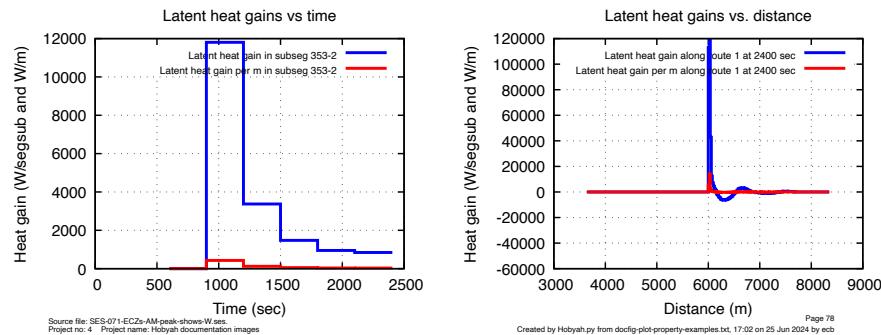


Figure 6.61: Latent heat gains vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value in each subsegment.
- Units: `lat` is in watts per segment and BTU/hr per segment. `Lat_pul` are in watts/m and BTU/hr-ft.

Data for `lat` and `lat_pul` are stored in a Pandas array called `subseg_lat` with indices of plot time (as real numbers) and columns of subsegment number (as strings, like "102-7"). The values are in W and when `lat_pul` is used the contents are converted to W/m or BTU/hr-ft on the fly.

6.8.16 The annulus plot type

Annulus is the annulus area around any trains passing across subpoints in segments. If there are no trains crossing the subpoint it will be the area of the open tunnel. There are values for every print timestep, accounting for the presence of trains at each time.

Typical curve definitions:

```
keyword      property      nickname      source
transient    annulus       file3         203-1
keyword      property      nickname      source@time
profile      annulus       file3         route2@600
```

Typical output:

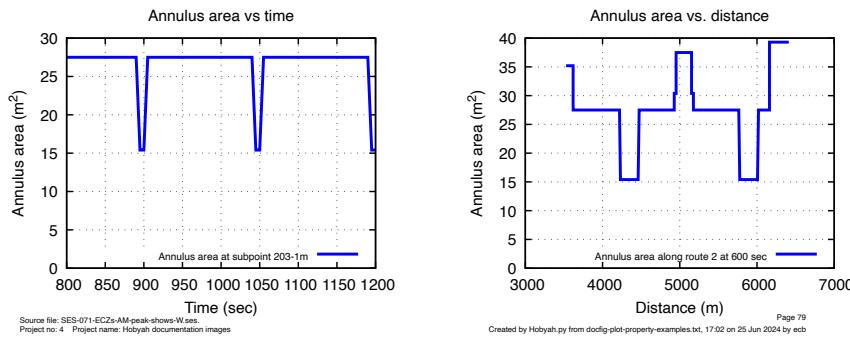


Figure 6.62: Annulus area vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: m², ft²

Data for **annulus** are stored in a Pandas array called `subpoint_areas` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m")

6.9 Plots of SES train properties

If your SES run has trains in it, you can plot the properties of trains as they move through the system. These are transient properties such as train speed, acceleration, tractive effort and heat flows. Because trains may be moving, you can plot the properties against train down end chainage at each time instead of just against time.

It is often useful to plot something against time **and** against train down end chainage, as the two types of graph give different insights into the same data.

For example, Figure 6.63 shows two views of the speed of the same train. The graph on the left is train speed versus time, the graph on the right is the same train's speed versus distance.

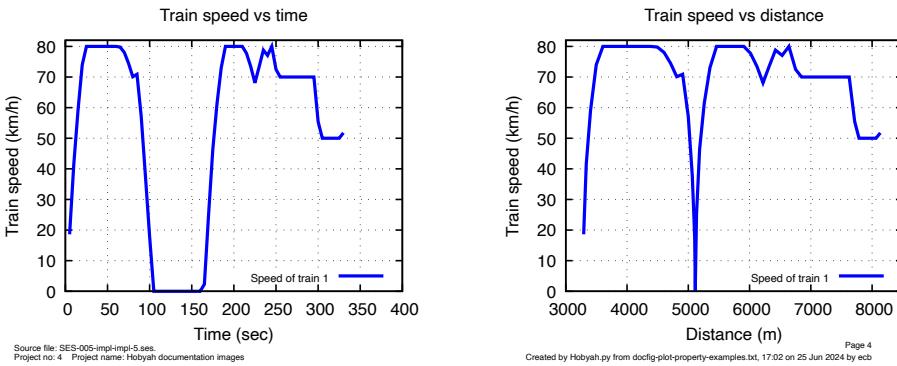


Figure 6.63: Plots of a train property (speed) vs. time and vs. distance

The graph on the left tells you that the train is stationary for around 60 seconds. If you are running a simulation in which you wanted the trains to dwell for 75 seconds, the graph on the left helps you spot your mistake. The graph on the right tells you that the train stops with its nose at approximate chainage 5100. If you know that the headwall of your station is at 6000 m instead of 5100 m, the graph on the right lets you know you made a mistake in setting the dwell points. Plotted output like these graphs help you catch your mistakes early in the design process.

When plotting a train's intrinsic properties you use the `transient` keyword (see Section 3.25 with the name of the property, the nickname of the source file and a phrase giving the train number). For consistency with other transient plot types you must include an @ symbol and a number after the @, but the number is ignored. The graph on the left in Figure 6.63 was plotted with the following curve definition:

```
keyword    property    nickname    source@discard
transient   speed      slugflow1   train1@0
```

The graph on the right was plotted with the same curve definition but with an optional argument telling the program to set the X-axis values to train nose chainage instead of time:

```
keyword    property    nickname    source@discard
transient   speed      slugflow1   train1@0      xaxis:=distance
```

6.9.1 The accel plot type

Accel is the acceleration rate of a train (+ve values) and the deceleration rate (-ve values).

In SES, it is only available in simulations in which the train performance option in form 1C is not zero.

Typical curve definitions:

```
transient    accel    file1    train100
transient    accel    file1    train1@987    xaxis:=distance
```

Typical output:

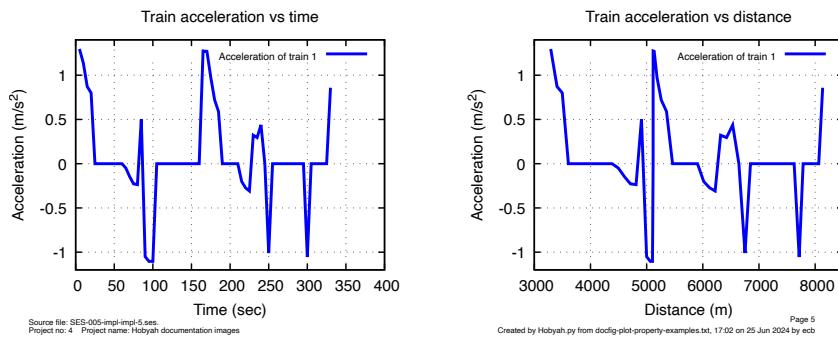


Figure 6.64: Train acceleration curves

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: m/s^2 , mph/sec

Data for **accel** are stored in a Pandas array called **train_accel** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.2 The speed plot type

Speed is the speed of a given train number (in km/h, not m/s).

In SES, it is only available in simulations in which the train performance option in form 1C is not zero.

Typical curve definitions:

```
keyword      property    nickname   source@discard
transient    speed       file1      train1@0
transient    speed       file1      train1@0373  xaxis:=distance
```

Typical output:

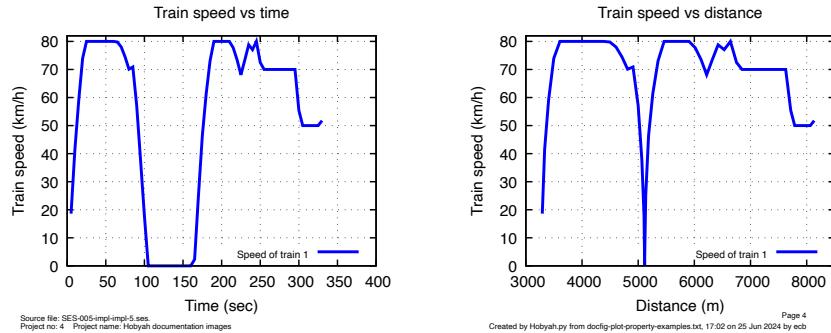


Figure 6.65: Train speed curves

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: km/h, mph

Data for `speed` are stored in a Pandas array called `train_speed` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.3 The route plot type

Route is the number of the route (form 8) that a given train number follows (1 or above).

In SES, it is only available in simulations in which the train performance option in form 1C is not zero.

In the (unlikely) event that SES launches more than 100 trains in a given simulation, a train number may be re-used on a different route. In this case the plotter will show the route of the first train to use the train number from when it enters until it leaves, then show the route of the second train to use the same train number. This can be confusing.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    route        file1         train100
transient    route        file1         train100      xaxis:=distance
```

Typical output:

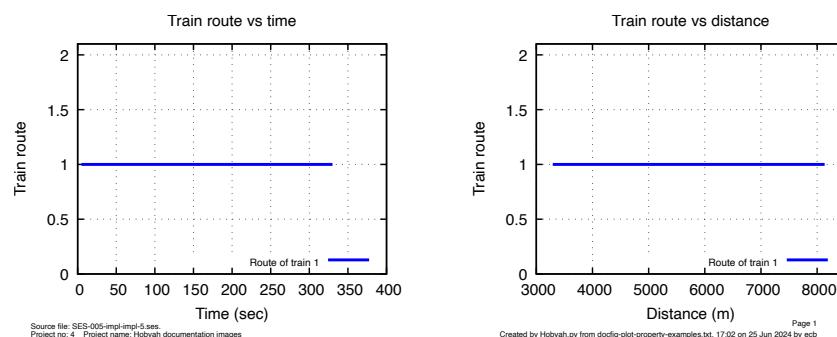


Figure 6.66: Train route vs. time and distance

Plotting this curve is usually pointless. But it's a plottable property that is intrinsic to trains, so it needs to have its section in the manual.

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: dimensionless

Data for **route** are stored in a Pandas array called **route_num** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.4 The type plot type

Type is the number of the train type (form 9) that a given train number follows (1 or above).

In SES, it is only available in simulations in which the train performance option in form 1C is not zero.

Typical curve definitions:

```
keyword      property    nickname   source@discard
transient    route       file1      train2@0
transient    route       file1      train2@0      xaxis:=distance
```

Typical output:

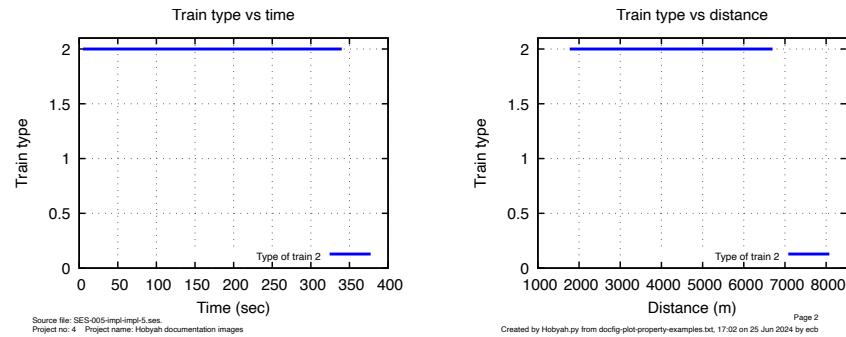


Figure 6.67: Train type vs. time and distance

Like the `route` plot type, plotting this curve is usually pointless.

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: dimensionless

Data for type are stored in a Pandas array called `train_type` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.5 The chainage and time plot types

Chainage and **time** both plot the location in a route of the down end of a given train (given as train number, 1 to 99 and 0). See Section 7.7 for an explanation of the odd numbering of trains in SES.

Chainage plots the location on the Y-axis and has time on the X-axis.

Time plots the location on the X-axis and has time on the Y-axis.

Some rail engineers like to plot train locations the first way, other rail engineers like to plot train locations the second way. Having both available makes it easier to work with both types of rail engineer.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    chainage     file1        train200
transient    time         file1        train200
```

Typical output:

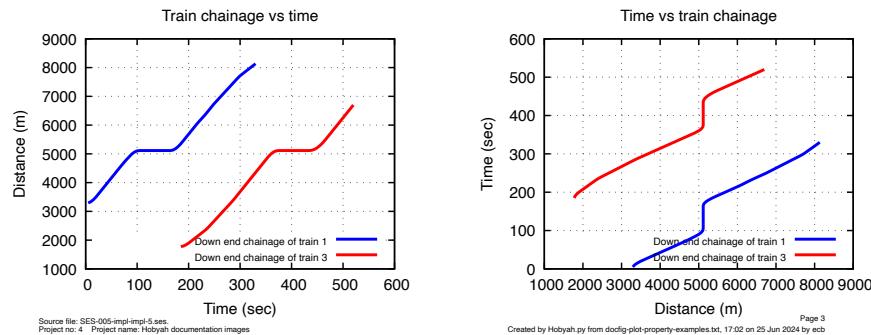


Figure 6.68: Train waterfall diagrams

Results from both keywords are only available in simulations in which the train performance option in form 1C is not zero.

- Curve types where they are used: **transient**
- Where plotted: on a train.
- Units: m, ft

Data for **chainage** and **time** are stored in a Pandas array called **train_locn** with indices of plot time (as real numbers) and columns of train number (as integers). If the train has not entered the system at a particular time (or has left) the value for that train at that time will be NaN (Not a Number in the IEEE-754/IEC-60559 floating point standard). Gnuplot ignores NaN values.

6.9.6 The aerodrag plot type

Aerodrag is the aerodynamic drag of a given train, made up of skin friction, pressure change at the nose and pressure change at the tail.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    aerodrag      file1        train1@0
transient    aerodrag      file1        train1@0      xaxis:=distance
```

Typical output:

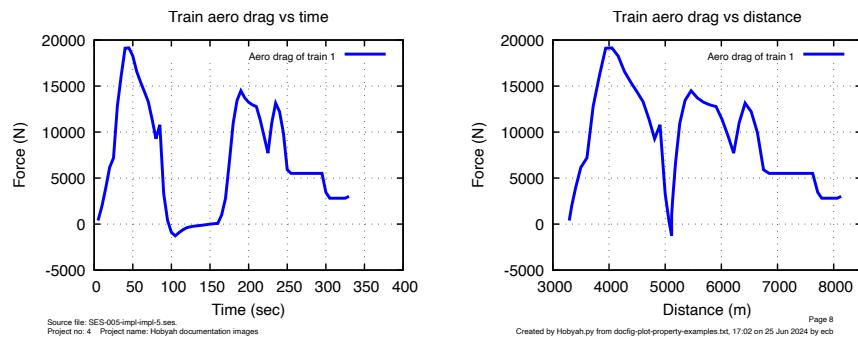


Figure 6.69: Train drag vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: N, lb

Data for `aerodrag` are stored in a Pandas array called `train_aerodrag` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.7 The cd plot type

`Cd` is the aerodynamic drag coefficient of a given train. It is the aerodynamic drag (in newtons) divided by the train cross-sectional area and the dynamic pressure calculated from train speed.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero. When trains are moving very slowly the printing of the drag coefficient often overflows the print field size (Fortran prints the overflowed value as `*****`). When the printing overflows, the value of `cd` is set to zero. When trains are stationary the value is also set to zero.

Typical curve definitions:

```
keyword      property    nickname   source@discard
transient    Cd          file1      train100
transient    cd          file1      train100      xaxis:=distance
```

Typical output:

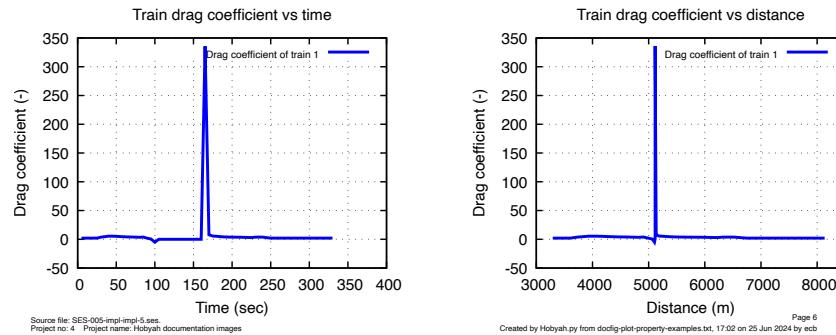


Figure 6.70: Train drag coefficient vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: dimensionless

Data for `cd` are stored in a Pandas array called `train_coeff` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.8 The TE plot type

TE is an acronym for tractive effort, the force required to push (or retard) a given train. When a train is accelerating, climbing a hill, or maintaining speed the tractive effort is probably positive. When a train is decelerating the tractive effort is negative.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero. When the train performance option is 3 (explicit heat calculation) the tractive effort is always zero (with that train performance option, there is no way to calculate the tractive effort).

When printed in the output file the tractive effort is given as N/motor. When stored in the .sbn file it is in N/train (i.e. the converter multiplies by the count of motors on the train).

Typical curve definitions:

```
keyword      property    nickname   source@discard
transient    TE          file1      train1@00
transient    TE          file1      train1@00      xaxis:=distance
```

Typical output:

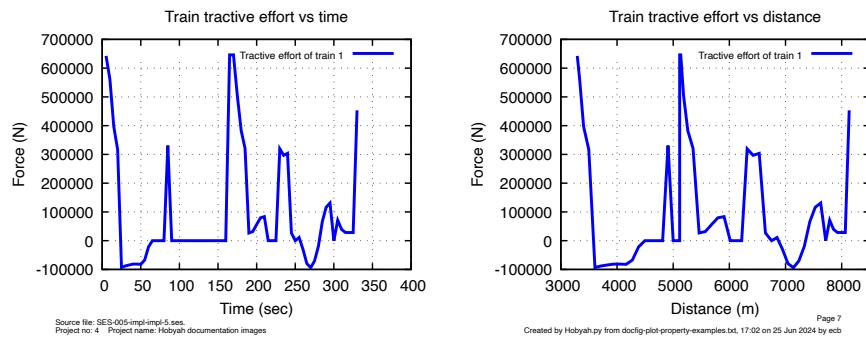


Figure 6.71: Train tractive effort vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: N/train, lb/train

Data for TE are stored in a Pandas array called `train_TE`³ with indices of plot time (as real numbers) and columns of train number (as integers).

³Note that the names of Pandas arrays are case-sensitive (because Python is case-sensitive). When looking for the tractive effort, a call to `train_te` would not work in `classSES.py`.

6.9.9 The motoramps plot type

Motoramps is the current running through each motor in a train. It is only useful to those trying to learn about the traction power calculation in SES.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the motor amps of trains are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    motoramps    file1        train1@0
transient    motoramps    file1        train1@0      xaxis:=distance
```

Typical output:

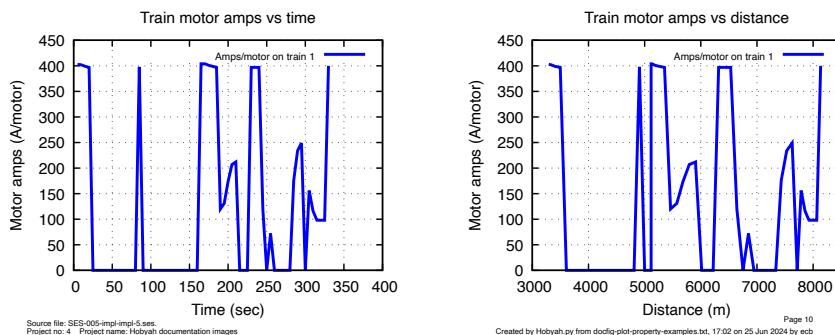


Figure 6.72: Train motor current vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: amperes

Data for `motoramps` are stored in a Pandas array called `motor_amps` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.10 The lineamps plot type

Lineamps is the total current entering the pantographs or current-collecting shoes of a train. It is only useful to those trying to learn about the traction power calculation in SES. For reasons unclear, SES limits the line current (amps per powered car) to twice the motor current (amps per motor).

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the line amps of trains are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    lineamps      file1        train1@0
transient    lineamps      file1        train1@0      xaxis:=distance
```

Typical output:

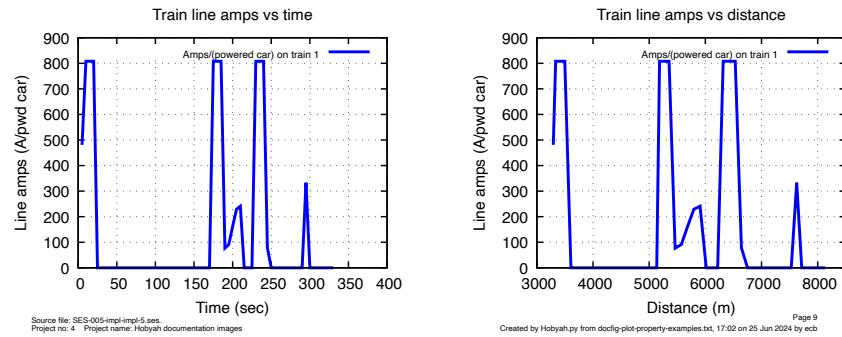


Figure 6.73: Train tractive effort vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: amperes

Data for `lineamps` are stored in a Pandas array called `line_amps` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.11 The `flywh_rpm` plot type

`Flywh_rpm` is the rotational speed of the flywheels storing energy on the train.

In SES, it is only available in simulations in which the train performance option in form 1C is not zero and the onboard flywheel simulation option in form 9H is 2.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the rpm of flywheels on trains are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    flywh_rpm    file1        train1@0
transient    flywh_rpm    file1        train1@0      xaxis:=distance
```

Typical output:

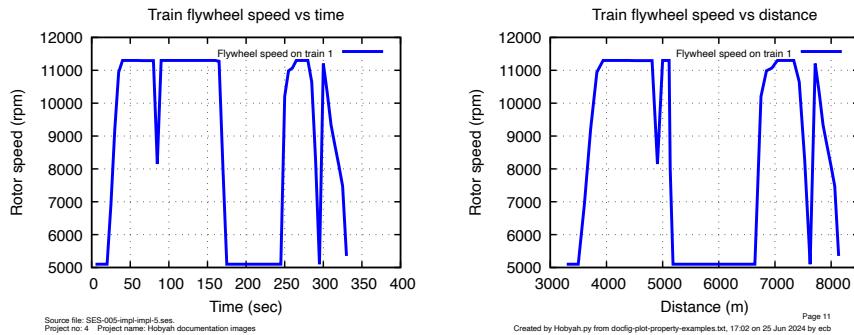


Figure 6.74: Train onboard flywheel rpm vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: rpm

Data for `flywh_rpm` are stored in a Pandas array called `flywh_rpm` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.12 The `acceltemp` plot type

`Acceltemp` is the temperature of the acceleration grid. The acceleration grid is the name used for a lumped parameter (a thermal mass) that represents all the elements on the train (motor carcass, bearings, wheels, inverter heat sinks, cables etc.) that are warmed by the electrical or frictional losses in the traction power system.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the acceleration grid temperatures are always zero.

The acceleration grid temperature is a useful property to plot, as it can pick up initial grid temperatures that are too high or too low (see page 8–36 and 8–37 of the SES v4.1 User Manual).

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    acceltemp    file1        train1@0
transient    acceltemp    file1        train1@0      xaxis:=distance
```

Typical output:

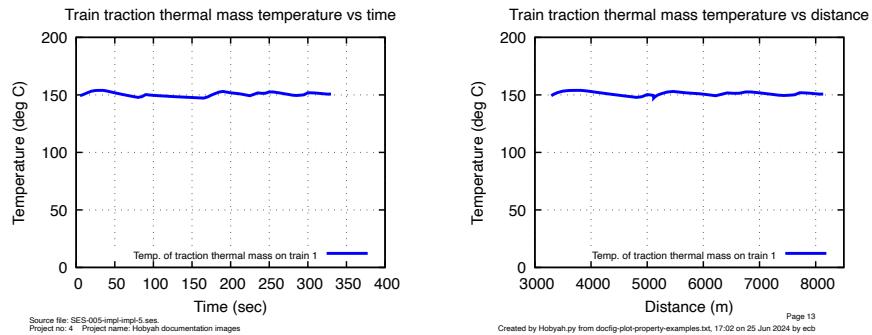


Figure 6.75: Train acceleration grid temperature vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: °C, °F

Data for `acceltemp` are stored in a Pandas array called `accel_temp` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.13 The deceltemp plot type

Deceltemp is the temperature of the deceleration grid. The deceleration grid is the name used for a lumped parameter (a thermal mass) that represents all the elements on the train (resistors, friction brakes, wheels etc.) that warm up when the train brakes.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the deceleration grid temperatures are always zero.

The deceleration grid temperature is a useful property to plot, as it can pick up initial grid temperatures that are too high or too low (see page 8–36 and 8–37 of the SES v4.1 User Manual).

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    deceltemp     file1        train1@0
transient    deceltemp     file1        train1@0      xaxis:=distance
```

Typical output:

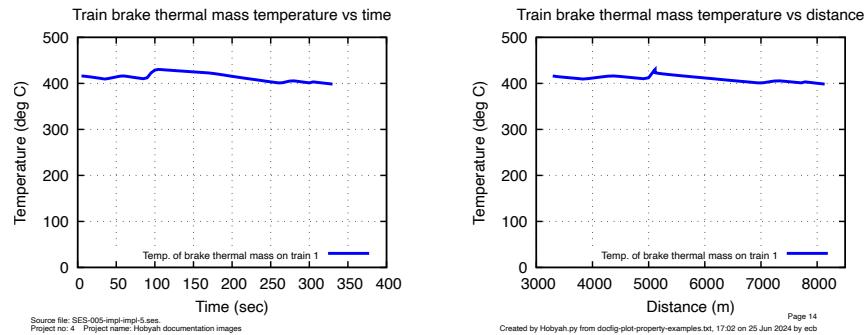


Figure 6.76: Train deceleration grid temperature vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: °C, °F

Data for **deceltemp** are stored in a Pandas array called **decel_temp** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.14 The pwr_all plot type

`Pwr_all` is the heat from passengers plus the power used to overcome all the losses on the train: losses in the traction system, unregenerated energy in the braking system and mechanical resistances⁴. Within SES it is the variable `HETGEN` (in routine `PRINT.FOR`), which is the sum of the following train performance arrays: `QAXSV` (sensible heat from people on the train), `QACCV` (power wasted in the traction power system), `QDECV` (power not regenerated in the braking system) and `RMHTV` (power to overcome rolling resistances and curve resistance).

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the powers are always zero.

When printed in SES output it is given as watts per metre of train length. This is inconvenient, so in `.sbn` files it is stored as watts per train.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_all      file1        train1@0
transient    pwr_all      file1        train1@0      xaxis:=distance
```

Typical output:

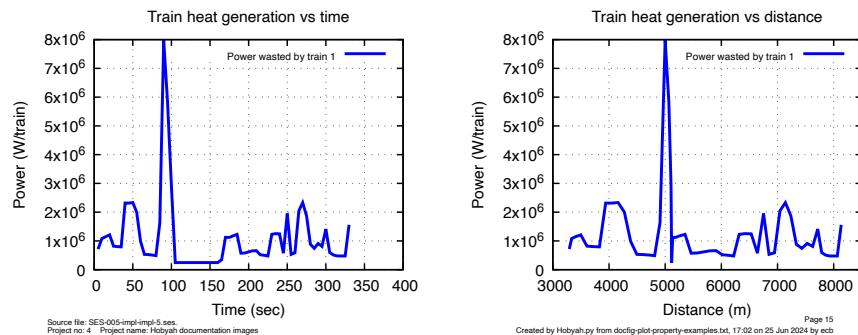


Figure 6.77: Train heat generated vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W, BTU/hr

Data for `pwr_all` are stored in a Pandas array called `pwr_all` with indices of plot time (as real numbers) and columns of train number (as integers).

⁴The power to overcome aerodynamic resistances is not included here.

6.9.15 The heat2air plot type

Heat2air is the heat emitted to the air from the acceleration grid, from the deceleration grid, from mechanical resistances and from people on the train.

Within SES it is the variable **QTRPF** (in routine **PRINT.FOR**), which is the sum of the following: **QAXSV** (sensible heat from passengers), **RMHTV** (heat generated by rolling resistances and curve resistance, heat emitted by the acceleration grid and the deceleration grid).

It is important to grasp the difference between **pwr.all** and **heat2air**. The former is the sum of the heat and power wasted as the train moves; some of that power causes the acceleration or deceleration grids to warm up. The latter is the heat sent to the air by passengers, mechanical resistances and the cooling of the acceleration and deceleration grids.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2. When the train performance option is 3, the heat flows are always zero.

When printed in SES output it is given as watts per metre of train length, but in **.sbn** files it is stored as watts per train.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    heat2air     file1        train100
transient    heat2air     file1        train100      xaxis:=distance
```

Typical output:

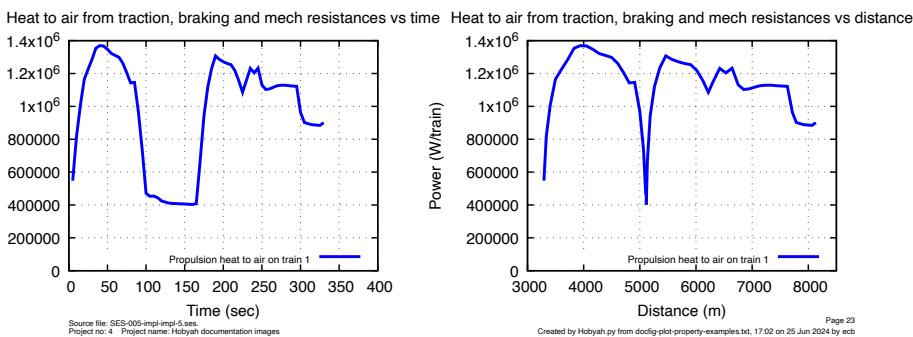


Figure 6.78: Train heat rejected to air vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: W, BTU/hr

Data for **heat2air** are stored in a Pandas array called **heat.reject** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.16 The mode plot type

`Mode` is an integer that indicates what a particular train is doing in terms of train performance. It has eight values (zero to seven), as follows:

0. Stopped at a station
1. Maintaining constant speed
2. Accelerating at full available power
3. Braking with a variable deceleration rate
4. Braking with a constant deceleration rate
5. Coasting
6. Maintaining minimum speed in a track section where coasting is permitted
7. Attempting to maintain constant minimum speed in a track section where coasting is permitted

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the train modes are always zero.

The curve plotted by `mode` is plotted with step-changes in value. This may be misleading if you plot it at long time intervals: for example, the `mode` plot may show a value of 0 long after you know a train has clearly started to move. Best to plot it at short time intervals if you don't want to confuse yourself.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    mode          file1         train1@00
transient    mode          file1         train1@00      xaxis:=distance
```

Typical output:

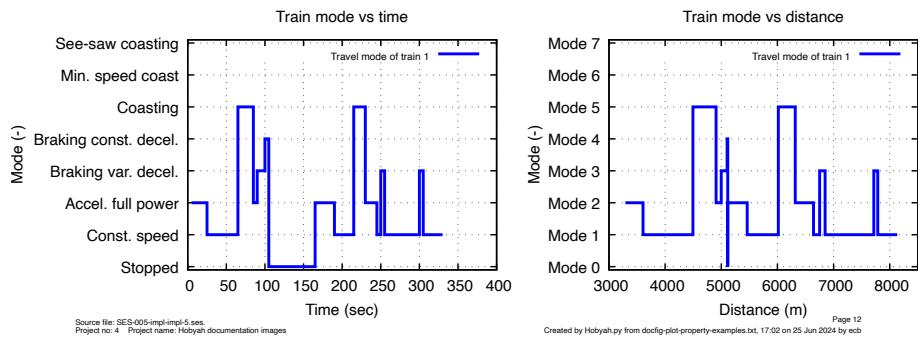


Figure 6.79: Train traction mode vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: dimensionless

Whenever I need to plot `MODE`, I usually use the following `verbatim` block to set custom Y axis labels:

```
begin verbatim
    set ytics ("Stopped" 0, "Const. speed" 1, "Accel. full power" 2, \
               "Braking var. decel." 3, "Braking const. decel." 4, "Coasting" 5, \
               "Min. speed coast" 6, "See-saw coasting" 7)
end verbatim
```

Data for `mode` are stored in a Pandas array called `train_modev` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.17 The `pwr_aux` plot type

`Pwr_aux` is the power drawn by the train auxiliary power systems from the pantographs, for things like lighting, airconditioning and compressors.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the auxiliary power loads are always zero.

When passengers get on a train the auxiliary load rises (more load on the aircon), and when passengers get off the train it falls.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_aux      file1         train1@0
transient    pwr_aux      file1         train1@0      xaxis:=distance
```

Typical output:

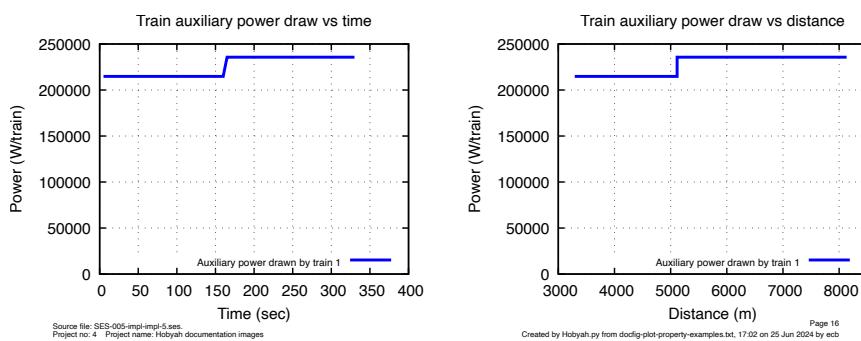


Figure 6.80: Train auxiliary power load vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W (in both SI and US files)

Data for `pwr_aux` are stored in a Pandas array called `pwr_aux` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.18 The `pwr_prop` plot type

`Pwr_prop` is the power drawn by the traction power system from the pantographs.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the traction power loads are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_prop     file1        train1@00
transient    pwr_prop     file1        train1@00      xaxis:=distance
```

Typical output:

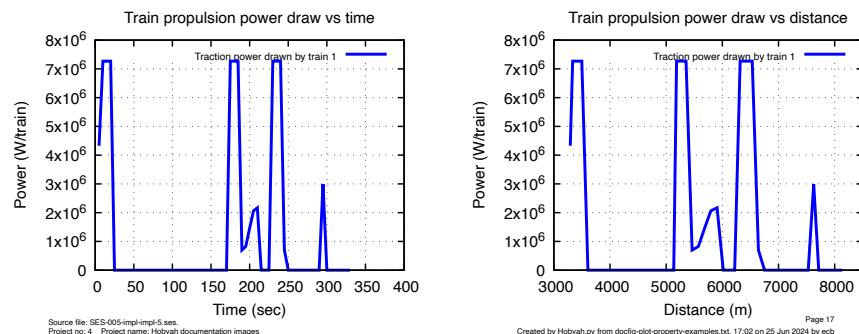


Figure 6.81: Train propulsion power vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W (in both SI and US files)

Data for `pwr_prop` are stored in a Pandas array called `pwr_prop` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.19 The pwr_regen plot type

Pwr_regen is the power regenerated to the pantograph by the traction power system acting as a generator.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the regenerated powers are always zero.

If a train type has flywheels, it appears that **pwr_regen** is zero even if the train type has been told to regenerate power. See Figure 6.82.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_regen    file1        train1@0
transient    pwr_regen    file1        train2@0
transient    pwr_regen    file1        train1@0      xaxis:=distance
transient    pwr_regen    file1        train2@0      xaxis:=distance
```

Typical output:

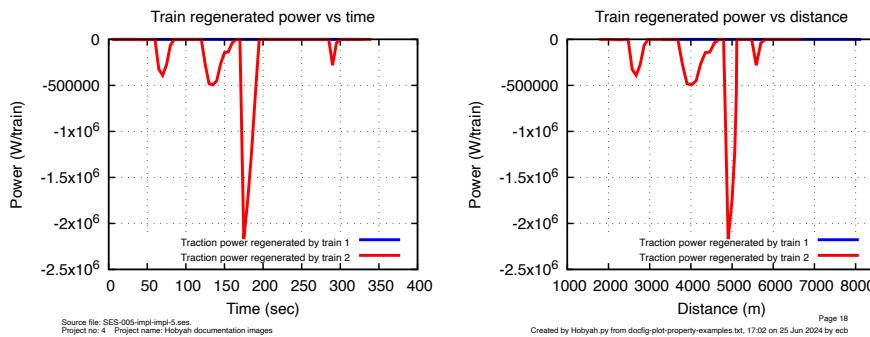


Figure 6.82: Train regenerated power vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: W (in both SI and US files)

Data for **pwr_regen** are stored in a Pandas array called **pwr_regen** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.20 The pwr_flywh plot type

`Pwr_flywh` is the power drawn by the traction power system from the onboard flywheel. When power is regenerated to the flywheel it is not negative (as you might expect), it is zero.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the power flows to/from flywheels are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_flywh    file1        train1@0
transient    pwr_flywh    file1        train1@0      xaxis:=distance
```

Typical output:

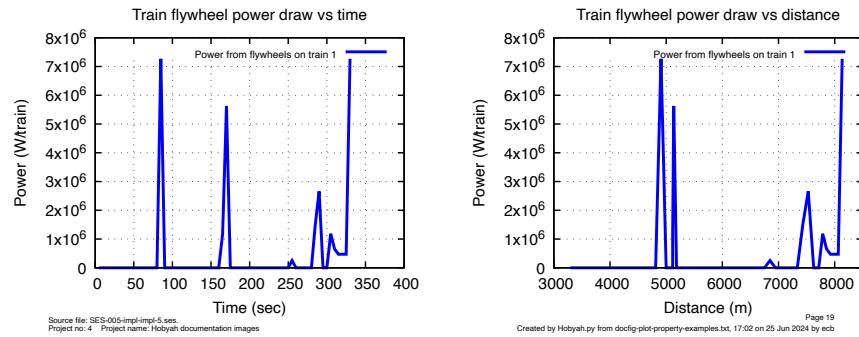


Figure 6.83: Train flywheel power draw vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W (in both SI and US files)

Data for `pwr_flywh` are stored in a Pandas array called `pwr_flywh` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.21 The pwr_accel plot type

Pwr_accel is the waste power in the traction system. It is sent to the acceleration grid and warms the grid up.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the power flows to/from grids are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_accel    file1        train1@0
transient    pwr_accel    file1        train1@0      xaxis:=distance
```

Typical output:

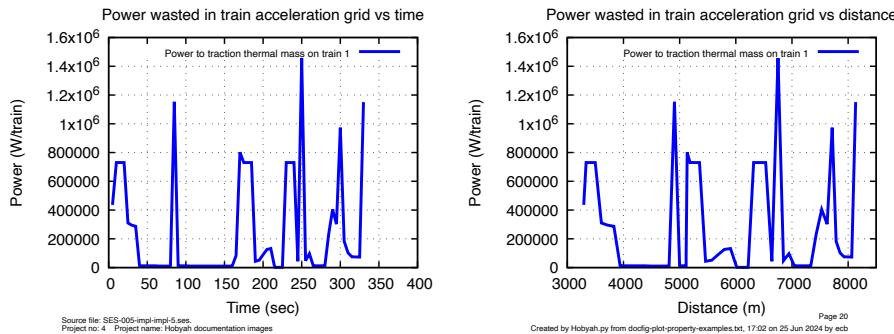


Figure 6.84: Train power to acceleration grid vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: W, BTU/sec

Data for **pwr_accel** are stored in a Pandas array called **pwr_accel** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.22 The pwr_decel plot type

`Pwr_decel` is the power in the braking system that is sent to the deceleration grid and warms the grid up.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the power flows to/from grids are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_decel    file1        train1@0
transient    pwr_decel    file1        train1@0      xaxis:=distance
```

Typical output:

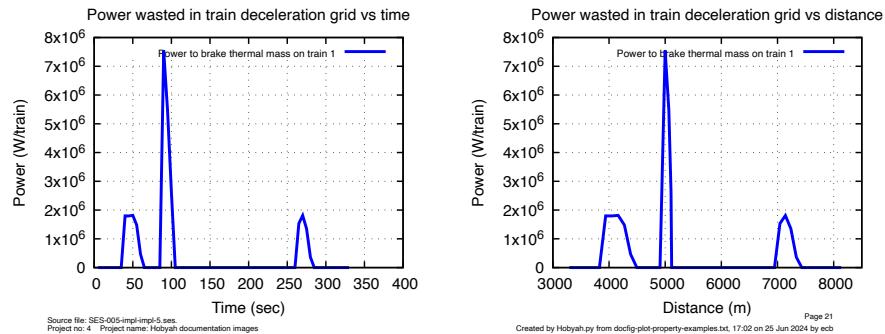


Figure 6.85: Train power to deceleration grid vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W, BTU/sec

Data for `pwr_decel` are stored in a Pandas array called `pwr_decel` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.23 The pwr_mech plot type

Pwr_mech is the power to overcome mechanical resistances (static resistance, rolling resistance and curve resistance). It is converted into heat to the tunnel air.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, the power to overcome mechanical resistances are always zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    pwr_mech     file1        train1@0
transient    pwr_mech     file1        train1@0      xaxis:=distance
```

Typical output:

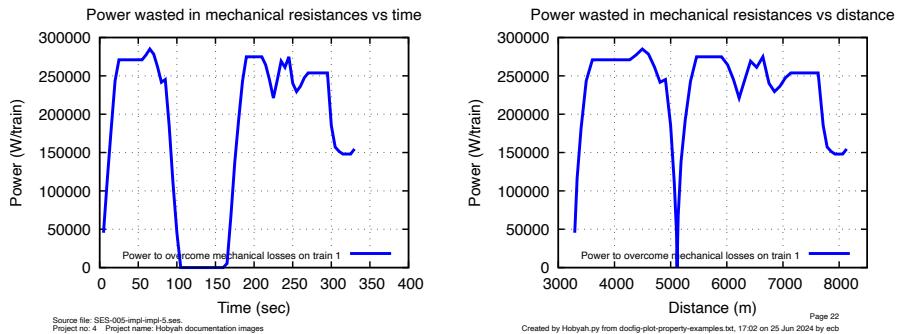


Figure 6.86: Train tractive effort vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: W, BTU/sec

Data for **pwr_mech** are stored in a Pandas array called **pwr_mech** with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.24 The heat_ADM plot type

`Heat_ADM` is the heat transferred to the tunnel air from the acceleration grid, the deceleration grid and the mechanical resistances (static resistance, rolling resistance and curve resistance). The “ADM” is an acronym of Acceleration, Deceleration & Mechanical.

Within SES it is the variable `QTRPV` (calculated in routine `GRID.FOR`).

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, these heat gains to the air are always printed as zero.

When printed in SES output it is given as watts per metre of train length, but in `.sbn` files it is stored as watts per train.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    heat_adm     file1        train1@00
transient    heat_ADM    file1        train1@00      xaxis:=distance
```

Typical output:

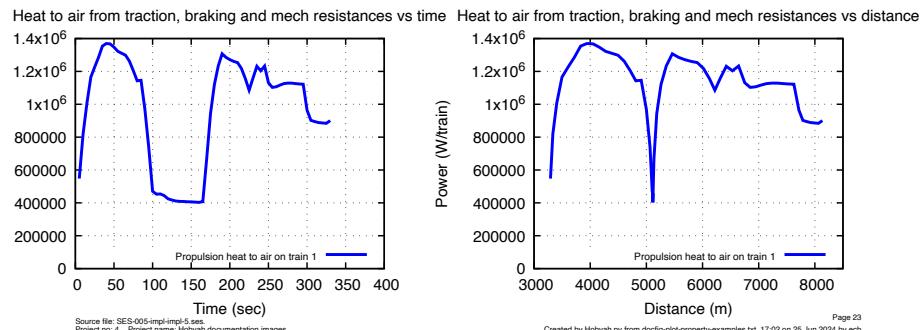


Figure 6.87: Train ADM heat to air vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W, BTU/sec

Data for `heat_adm` are stored in a Pandas array called `heat_adm` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.25 The `heat_sens` plot type

`heat_sens` is the sensible heat transferred to the tunnel air from the auxiliary loads and the passengers.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, these heat gains to the air are always printed as zero.

When printed in SES output it is given as watts per metre of train length, but in `.sbn` files it is stored as watts per train.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    heat_sens    file1        train1@0
transient    heat_sens    file1        train1@0      xaxis:=distance
```

Typical output:

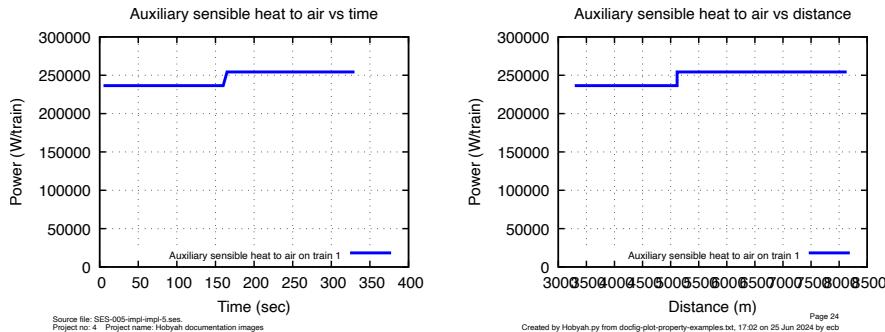


Figure 6.88: Train sensible heat from auxiliary loads and passengers vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W, BTU/sec

Data for `heat_sens` are stored in a Pandas array called `heat_sens` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.26 The heat_lat plot type

`Heat_lat` is the latent heat transferred to the tunnel air from trains' air conditioning systems.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more. When the train performance option is 3, these heat gains to the air are always printed as zero.

When printed in SES output it is given as watts per metre of train length, but in `.sbn` files it is stored as watts per train.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    heat_lat     file1        train1@0
transient    heat_lat     file1        train1@0      xaxis:=distance
```

Typical output:

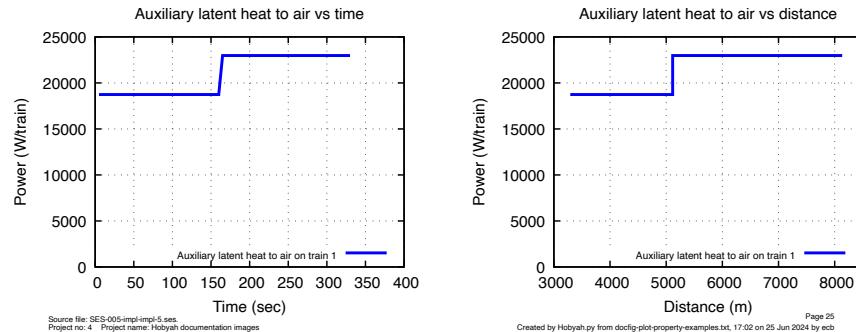


Figure 6.89: Train latent heat from passengers vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: W, BTU/sec

Data for `heat_lat` are stored in a Pandas array called `heat_lat` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.27 The effic1 plot type

Effic1 is the efficiency of the traction power system, expressed as a fraction 0–1.

In SES, it is only available in simulations in which the train performance option in form 1C is 1 or 2 and in which the supplementary print option (also form 1C) is 2 or more.

It is calculated by `SESconv.py` from the train tractive effort “TE” (N/train), the speed of the train “speed” (km/h) and the power sent to the acceleration grid by the traction power system “pwr_accel” (W/train). It is a two-step calculation. First the power delivered at the wheel–rail interface is calculated:

$$\text{Wheel_power} = \text{TE} \times \frac{\text{speed}}{3.6}. \quad (6.2)$$

then the efficiency is calculated as the power delivered divided by the sum of the power delivered and the power sent to the acceleration grid:

$$\text{effic1} = \frac{\text{Wheel_power}}{\text{Wheel_power} + \text{pwr_accel}}. \quad (6.3)$$

The factor of 3.6 in equation (6.2) converts train speeds in km/h to m/s.

Effic1 is a genuinely useful thing to plot, as it tells you the true traction efficiency of the trains you are using. A fuller discussion of **effic1** can be found in the notes on SES (`SES-notes.pdf` in the “Documentation” folder on github).

If the traction system is disengaged (i.e. the train is stopped, braking or coasting), then **effic1** is set to zero.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    effic1       file1        train100
transient    effic1       file1        train100      xaxis:=distance
```

Typical output:

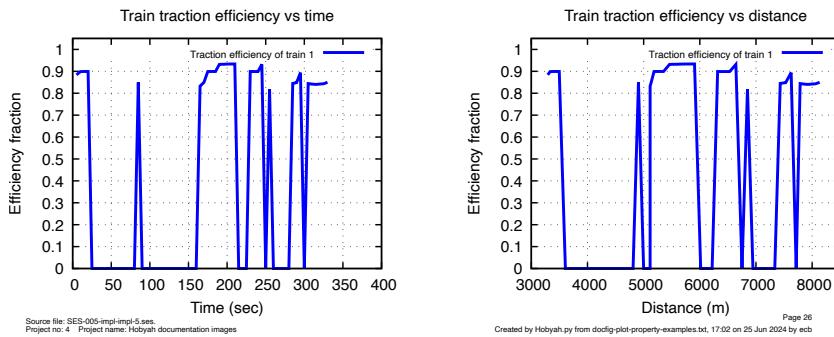


Figure 6.90: Train traction efficiency vs. time and distance

- Curve types where it is used: `transient`

- Where plotted: on a train.
- Units: dimensionless, range 0 to 1

Data for `effic1` are stored in a Pandas array called `train_effic` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.28 The SVSeffic plot type

SVSeffic is the efficiency of the traction power system, expressed as a fraction 0–1. Its values are only valid when using SVS train control option 3; when you use train control options 1 or 2 are used in SVS, the values printed in the .OUT file are zero.

The values are interpolated from the figures in SVS form 9H-A.

It is only available in simulations in which the train performance option in form 1C is 1 or 2.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    svseffic     file1        train100
transient    SVSeffic     file1        train100      xaxis:=distance
```

Typical output:

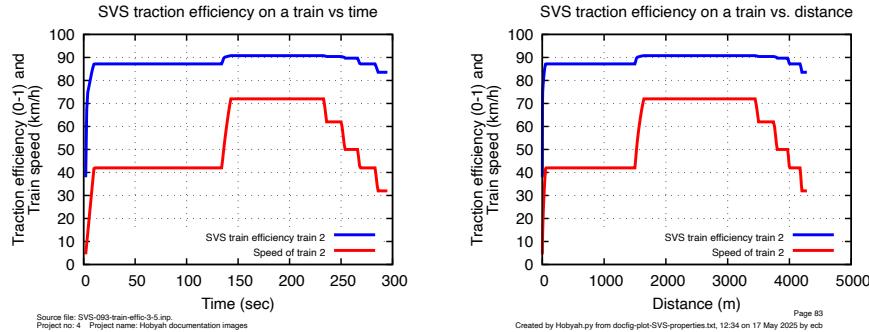


Figure 6.91: Train traction efficiency vs. time and distance

- Curve types where it is used: `transient`
- Where plotted: on a train.
- Units: dimensionless, range 0 to 1

Data for **SVSeffic** are stored in a Pandas array called `train_SVSeff` with indices of plot time (as real numbers) and columns of train number (as integers).

6.9.29 The SVSregen1 plot type

SVSregen1 is the regenerative braking efficiency used by a train as it travels along its route, expressed as a fraction 0–1.

Its values are only valid when using SVS train control option 3; when train control options 1 or 2 are used its values are always zero.

Its values are taken either from SVS form 9H-A (the default value) or form 8C (values in different track sections). If the value in the track section that the train nose is above zero, that value is used. If the track section value is zero, the default value is used.

It is only available in simulations in which the train performance option in form 1C is 1 or 2.

A related property is **SVSregen2**, which is plotted along a route against chainage.

In SVS output files the values are printed as percentages (0–100). For consistency with plot types **SVSregen2** (form 8C) and how the default regen braking efficiency is set in form 9H-A, the percentage values recorded over the course of the run are divided by 100 so the plots are given as fractions 0–1 instead of as percentages.

Typical curve definitions:

```
keyword      property      nickname      source@discard
transient    svsregen1    file1        train1@0
transient    SVSregen1   file1        train1@0      xaxis:=distance
```

Typical output:

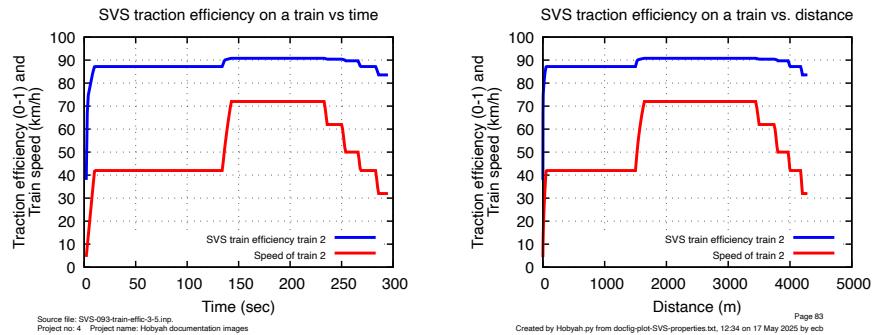


Figure 6.92: Train traction efficiency vs. time and distance

- Curve types where it is used: **transient**
- Where plotted: on a train.
- Units: dimensionless, range 0 to 1

Data for **SVSregen1** are stored in a Pandas array called **train_SVSregen** with indices of plot time (as real numbers) and columns of train number (as integers).

6.10 Plots of SES Environmental Control Zone data

6.10.1 The meanDB_AM plot type

MeanDB_AM is a dry-bulb air temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the air temperature during the morning peak hour.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

It should be ignored in runs that calculate off-hour ECZ estimates and in peak hour ECZ estimates in which the design hour in form 1B is 12 or above.

You can plot the property at a fixed location (a subsegment) against time using a **transient** curve. This lets you see the convergence of the temperature over each ECZ, as in the graph on the left-hand side of Figure 6.93. Each step-change in temperature occurs at an ECZ estimate and the graph shows an example of good convergence.

You can also plot the temperature along a route at a fixed point in time, as in the graph on the right-hand side of Figure 6.93. This graph shows a route running through an uncontrolled zone (before 3700 m to 6000 m) and a controlled zone with a design air temperature of 25 °C (6000 m to 8200 m).

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanDB_AM    file3         54-2          # Transient plot in subsegment 2 of segment 54
keyword      property      nickname      source@time
profile      meanDB_AM    file3         route1@-1     # Profile plot along route 1 at last timestep
```

Typical output:

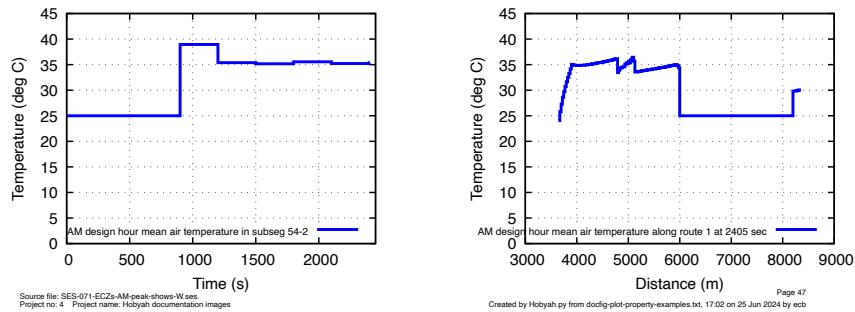


Figure 6.93: Morning peak dry-bulb temperatures vs. time and distance

- Curve types where it is used: **transient**, **profile**
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `meanDB_AM` are stored in a Pandas array called `airT_AM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.2 The `meanDB_PM` plot type

`MeanDB_PM` is a dry-bulb air temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the temperature during the evening peak hour.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

It should be ignored in runs that calculate off-hour ECZ estimates and in peak hour ECZ estimates in which the design hour in form 1B is less than 12.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanDB_PM    file4        54-2
keyword      property      nickname      source@time
profile      meanDB_PM    file4        route1@-1
```

Typical output:

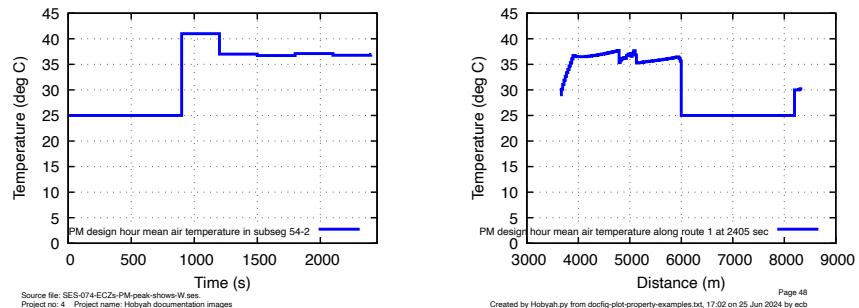


Figure 6.94: Evening peak dry-bulb temperatures vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `meanDB_PM` are stored in a Pandas array called `airT_PM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.3 The meanDB_off plot type

`MeanDB_off` is a dry-bulb air temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the temperature during an off-hour.

In SES, it is only available in simulations in which environmental control zone estimates are carried out for off hour simulations (not peak hour simulations).

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanDB_off   file5        54-2
keyword      property      nickname      source@time
profile     meanDB_off   file5        route1@-1
```

Typical output:

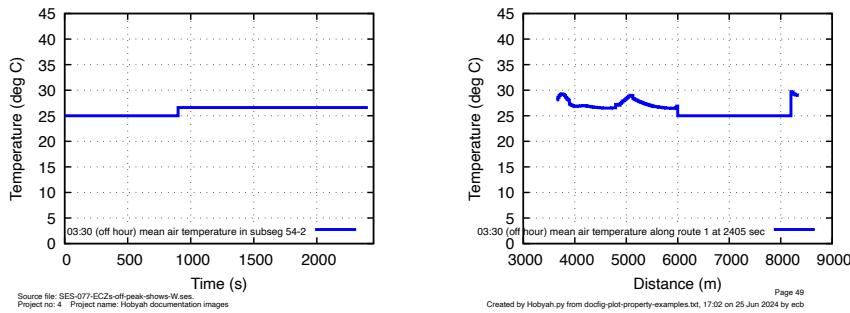


Figure 6.95: Off-hour dry-bulb temperatures vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `meanDB_off` are stored in a Pandas array called `airT_off` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.4 The meanW_AM plot type

`meanW_AM` is a humidity ratio in a subsegment. It is the mean humidity over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the humidity during the morning peak hour.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

It should be ignored in runs that calculate off-hour ECZ estimates and in peak hour ECZ estimates in which the design hour in form 1B is 12 or above.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanW_AM     file3          54-2
keyword      property      nickname      source@time
profile      meanW_AM     file3          route1@-1
```

Typical output:

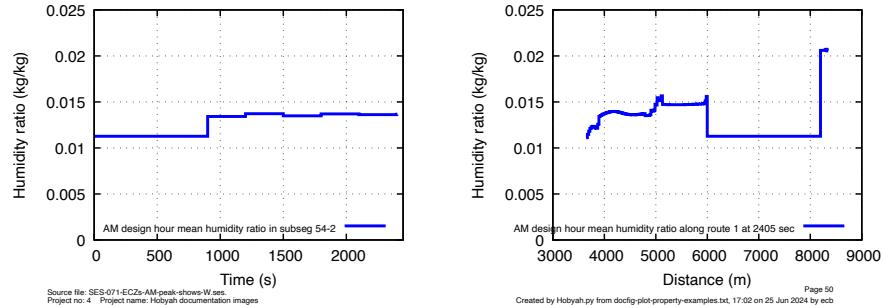


Figure 6.96: Morning peak humidity ratios vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: kg/kg, lb/lb

Data for `meanW_AM` are stored in a Pandas array called `airW_AM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.5 The meanW_PM plot type

`meanW_PM` is a humidity ratio in a subsegment. It is the mean humidity over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the humidity during the evening peak hour.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

It should be ignored in runs that calculate off-hour ECZ estimates and in peak hour ECZ estimates in which the design hour in form 1B is less than 12.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanW_PM     file4        54-2
keyword      property      nickname      source@time
profile      meanW_PM     file4        route1@-1
```

Typical output:

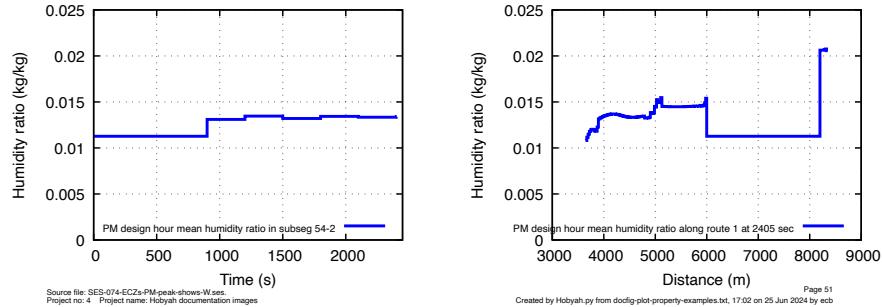


Figure 6.97: Evening peak humidity ratios vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: kg/kg, lb/lb

Data for `meanW_PM` are stored in a Pandas array called `airW_PM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.6 The meanW_off plot type

`meanW_off` is a humidity ratio in a subsegment. It is the mean humidity over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the temperature during an off-hour time (not morning peak or evening peak).

In SES, it is only available in simulations in which environmental control zone estimates are carried out for off hour simulations (not peak hour simulations).

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    meanW_PM     file5        54-2
keyword      property      nickname      source@time
profile     meanW_PM     file5        route1@-1
```

Typical output:

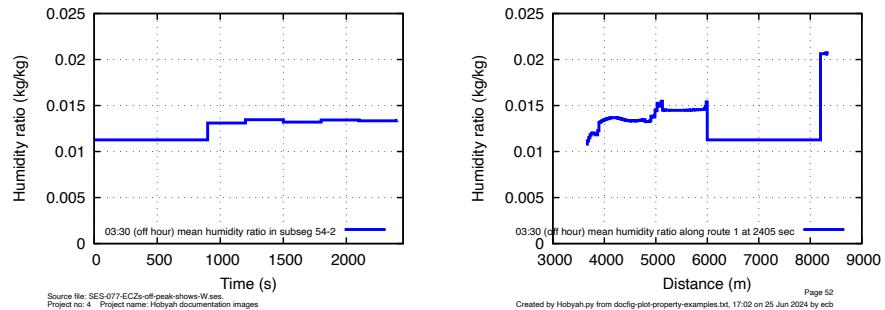


Figure 6.98: Off-hour humidity ratios vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: kg/kg, lb/lb

Data for `meanW_off` are stored in a Pandas array called `airW_off` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.7 The wallT_used plot type

`WallT_used` is a wall temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the temperature during whatever type of ECZ estimate is being calculated. Its values depend on what kind of ECZ estimate is being carried out and what the design hour is. The rules are as follows:

- If a peak hour ECZ estimate is being carried out and the design hour in form 1B is below 12 (before midday), it is equal to `wallT_AM`.
- If a peak hour ECZ estimate is being carried out and the design hour in form 1B is 12 or higher, it is equal to `wallT_PM`.
- If an off-hour ECZ estimate is being carried out, its values are the same as the initial wall temperatures and do not change throughout the run.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    wallT_used   file3         54-2          # AM peak run
transient    wallT_used   file4         54-2          # PM peak run
transient    wallT_used   file5         54-2          # off hour run
keyword      property      nickname      source@time
profile     wallT_used   file3         route10-1
profile     wallT_used   file4         route10-1
profile     wallT_used   file5         route10-1
```

Typical output:

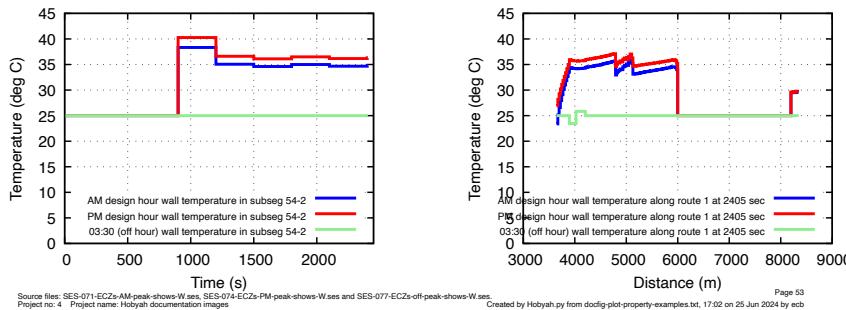


Figure 6.99: Wall temperatures vs. time and distance

The autokey used in the plots is adjusted to reflect the type of ECZ estimate being conducted and show the design hour.

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `wallT_used` are stored in a Pandas array called `wallT_used` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the

full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.8 The wallT_AM plot type

`wallT_AM` is a wall temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the wall temperature during the morning peak hour.

It should only be used for debugging and development; much better to use the plot type `wallT_used` in design work. To emphasize this, when trying to plot `wallT_AM` from a PM peak hour or off-hour ECZ run, the autokey is changed to include the word "Non-valid". If you plot something and the autokey includes "non-valid", start looking into your run.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    wallT_AM     file3          54-2
keyword      property      nickname      source@time
profile      wallT_AM     file3          route1@-1
```

Typical output:

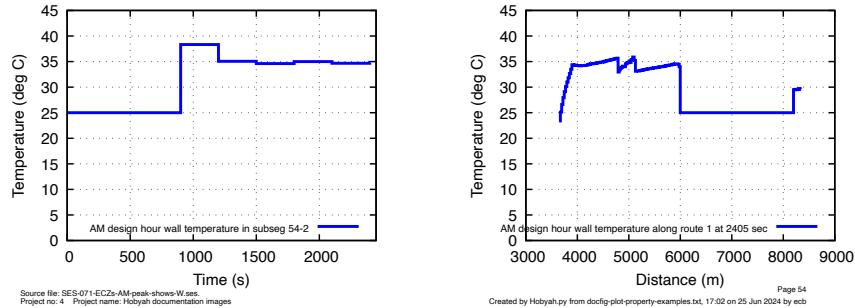


Figure 6.100: AM peak-hour wall temperatures vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `wallT_AM` are stored in a Pandas array called `wallT_AM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.9 The wallT_PM plot type

`WallT_PM` is a wall temperature in a subsegment. It is the mean temperature over the duration of an environmental control zone estimate (ECZ estimate), adjusted to represent the wall temperature during the evening peak hour.

It should only be used for debugging and development; much better to use the plot type `wallT_used` in design work. To emphasize this, when trying to plot `wallT_PM` from an AM peak hour or off-hour ECZ run, the autokey is changed to include the word "Non-valid". If you plot something and the autokey includes "non-valid", start looking into your run.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    wallT_PM     file4          54-2
keyword      property      nickname      source@time
profile      wallT_PM     file4          route1@-1
```

Typical output:

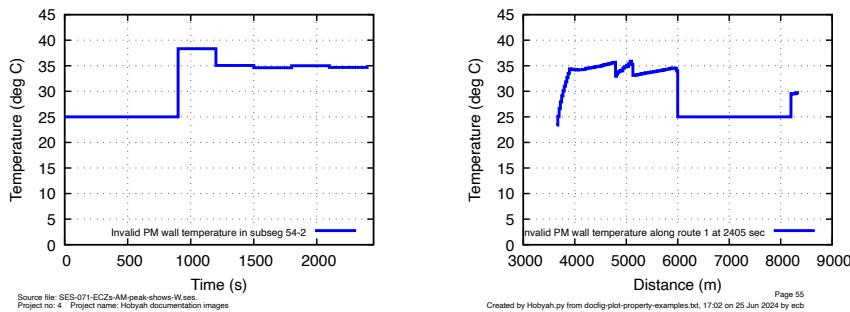


Figure 6.101: PM peak-hour all temperatures vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: °C, °F.

Data for `wallT_PM` are stored in a Pandas array called `wallT_PM` with indices of time (as real numbers) and columns of subsegment identifiers. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.10 The `misc_sens` plot type

`misc_sens` is the sensible heat load from trains, viscous heating and miscellaneous heat loads over the course of an environmental control zone estimate. It is calculated and printed in `ACEST2.FOR`.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    misc_sens    file4        352-2
keyword      property      nickname      zone
transient    misc_sens    file4        zone1
keyword      property      nickname      source@time
profile      misc_sens    file4        route1@-1
```

Typical output:

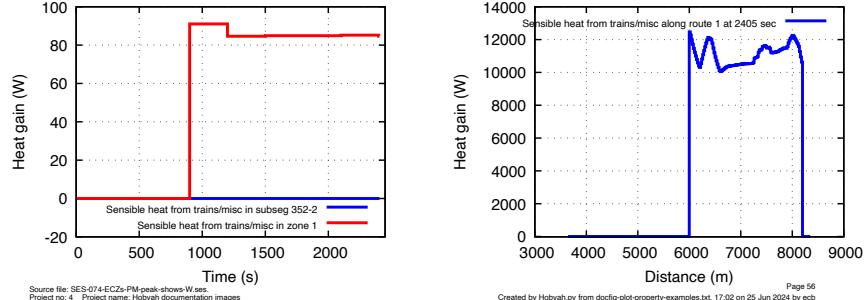


Figure 6.102: Sensible heat loads from trains/misc. vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for `misc_sens` are stored in a Pandas array called `misc_sens` with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.11 The `misc_lat` plot type

`misc_lat` is the latent heat load from trains, viscous heating and miscellaneous heat loads over the course of an environmental control zone estimate. It is calculated and printed in `ACEST2.FOR`.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    misc_lat     file4        352-2
keyword      property      nickname      zone
transient    misc_lat     file4        zone1
keyword      property      nickname      source@time
profile      misc_lat     file4        route1@-1
```

Typical output:

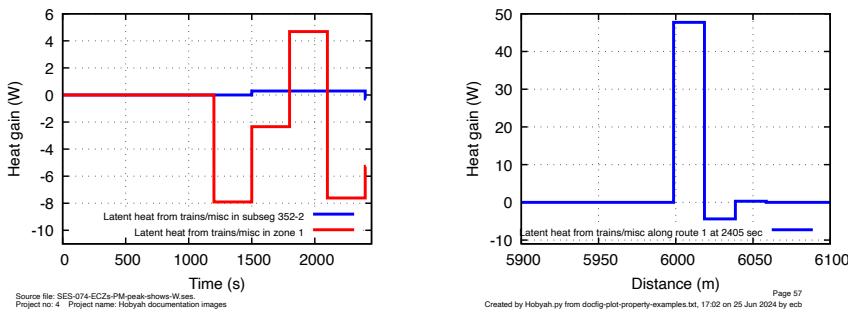


Figure 6.103: Latent heat loads from trains/mic. vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for `misc_lat` are stored in a Pandas array called `misc_lat` with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.12 The steady_sens plot type

Steady_sens is the sensible heat set by steady heat loads in subsegments (form 3D) over the course of an environmental control zone estimate. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    steady_sens   file4        352-2
keyword      property      nickname      zone
transient    steady_sens   file4        zone1
keyword      property      nickname      source@time
profile     steady_sens   file4        route1@-1
```

Typical output:

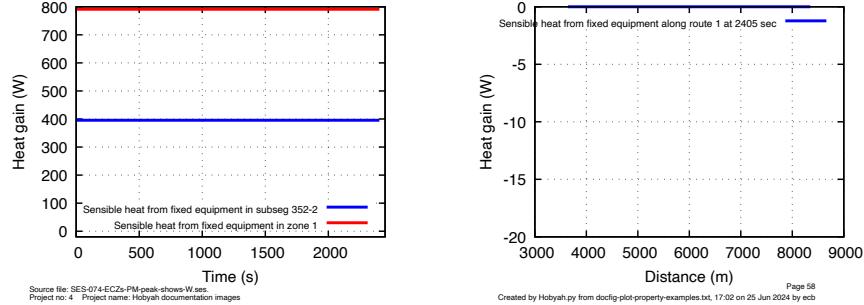


Figure 6.104: Sensible steady heat loads. vs. time and distance

- Curve types where it is used: **transient, profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **steady_sens** are stored in a Pandas array called **steady_sens** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.13 The steady_lat plot type

Steady_lat is the latent heat set by steady heat loads in subsegments (form 3D) over the course of an environmental control zone estimate. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    steady_lat    file4        352-2
keyword      property      nickname      zone
transient    steady_lat    file4        zone1
keyword      property      nickname      source@time
profile      steady_lat    file4        route1@-1
```

Typical output:

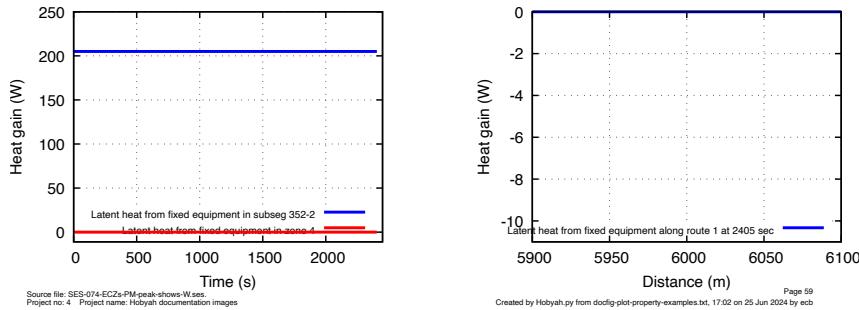


Figure 6.105: Latent heat loads from steady heat loads vs. time and distance

- Curve types where it is used: **transient, profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **steady_lat** are stored in a Pandas array called **steady_lat** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.14 The ground_sens plot type

Ground_sens is the sensible heat load from the ground over the course of an environmental control zone estimate. It is usually negative but may be positive in ground near hot springs like the Gellért spa in Budapest. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    ground_sens   file4        352-2
            keyword      property      nickname      zone
transient    ground_sens   file4        zone1
            keyword      property      nickname      source@time
profile      ground_sens   file4        route1@-1
```

Typical output:

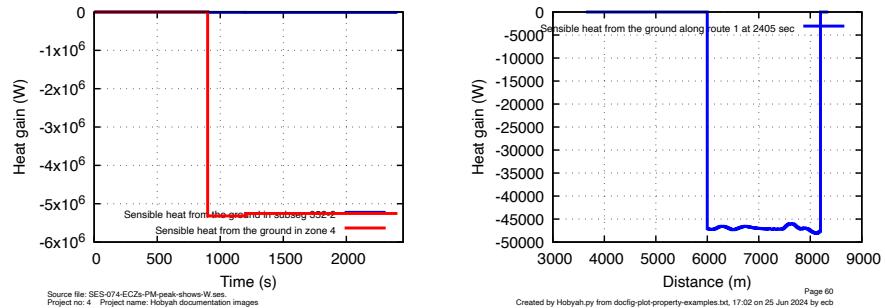


Figure 6.106: Sensible heat loads from the surrounding ground. vs. time and distance

- Curve types where it is used: **transient**, **profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **ground_sens** are stored in a Pandas array called **ground_sens** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.15 The airex_sens plot type

`airex_sens` is the sensible heat load from air exchange over the course of an environmental control zone estimate. It is calculated and printed in `ACEST2.FOR`.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    airex_sens   file4        352-2
keyword      property      nickname      zone
transient    airex_sens   file4        zone1
keyword      property      nickname      source@time
profile      airex_sens   file4        route1@-1
```

Typical output:

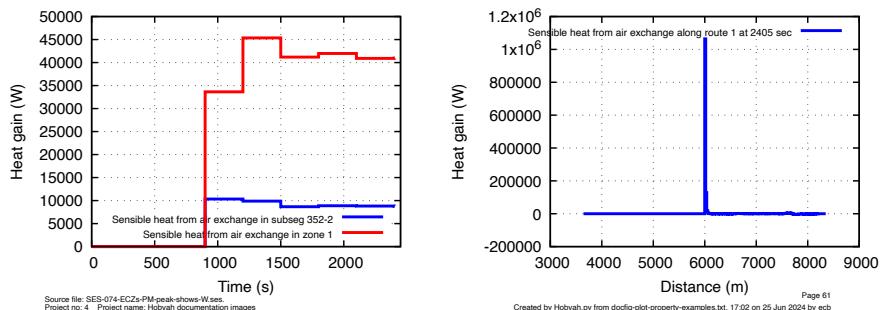


Figure 6.107: Sensible heat loads from trains and air exchange

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for `airex_sens` are stored in a Pandas array called `airex_sens` with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.16 The airex_lat plot type

`airex_lat` is the latent heat load from air exchange over the course of an environmental control zone estimate. It is calculated and printed in `ACEST2.FOR`.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```

keyword      property      nickname      segment-subseg
transient    airex_lat    file4        352-2
keyword      property      nickname      zone
transient    airex_lat    file4        zone1
keyword      property      nickname      source@time
profile      airex_lat    file4        route1@-1
  
```

Typical output:

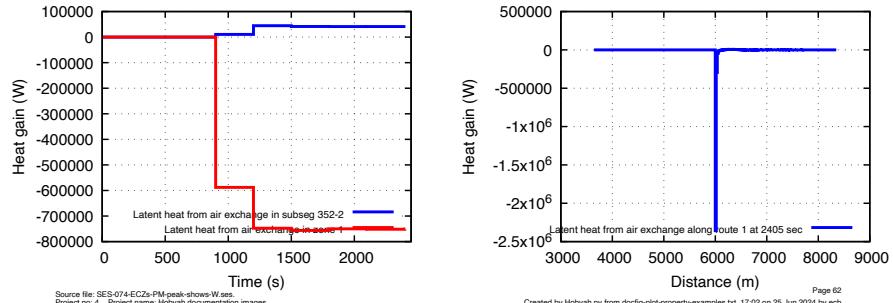


Figure 6.108: Latent heat loads from trains and air exchange

- Curve types where it is used: `transient`, `profile`
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for `airex_lat` are stored in a Pandas array called `airex_lat` with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.17 The HVAC_sens plot type

HVAC_sens is the sensible heat load from the air conditioning in a controlled zone over the course of an environmental control zone estimate. Positive values mean heating, negative values mean cooling. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    HVAC_sens    file4        352-2
keyword      property      nickname      zone
transient    HVAC_sens    file4        zone1
keyword      property      nickname      source@time
profile      HVAC_sens    file4        route1@-1
```

Typical output:

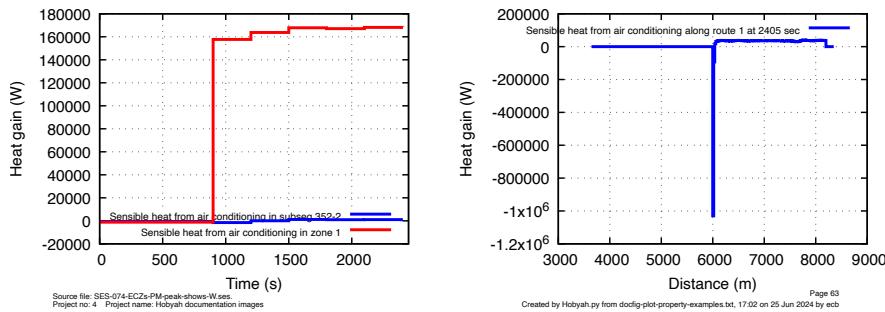


Figure 6.109: Sensible heat loads from HVAC vs. time and distance

- Curve types where it is used: **transient**, **profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **HVAC_sens** are stored in a Pandas array called **HVAC_sens** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.18 The HVAC_lat plot type

HVAC_lat is the latent heat load from the air conditioning in a controlled zone over the course of an environmental control zone estimate. Positive values mean heating, negative values mean cooling. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    HVAC_lat     file4        352-2
            keyword      nickname      zone
            transient   HVAC_lat     file4        zone1
            keyword      property      nickname      source@time
            profile     HVAC_lat     file4        route10-1
```

Typical output:

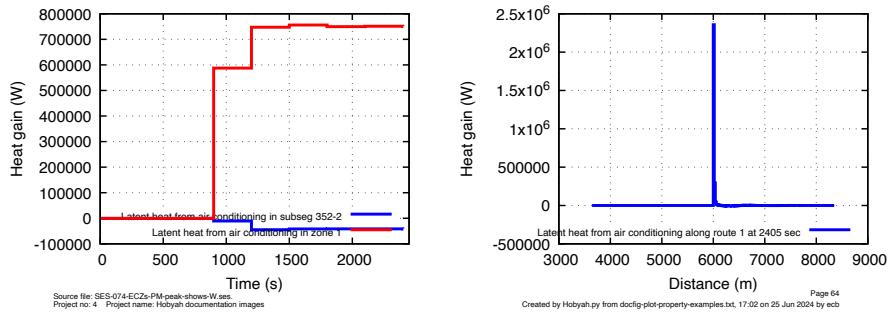


Figure 6.110: Latent heat loads from HVAC vs. time and distance

- Curve types where it is used: **transient, profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **HVAC_lat** are stored in a Pandas array called **HVAC_lat** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.10.19 The HVAC_total plot type

HVAC_sens is the total heat load from the air conditioning in a controlled zone over the course of an environmental control zone estimate. It is the sum of the sensible and latent heat loads. It is calculated and printed in **ACEST2.FOR**.

In SES, it is only available in simulations in which environmental control zone estimates are carried out.

Transient curves can be plotted in individual subsegments and the sum of all the subsegments in a controlled zone. Transient curves can be plotted in individual sub-segments and the sum of all the subsegments in a controlled zone.

If a segment is in an uncontrolled zone, its value is zero. If a zone is not a controlled zone an error will be raised.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>segment-subseg</i>
transient	HVAC_total	file4	352-2
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>zone</i>
transient	HVAC_total	file4	zone1
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	HVAC_total	file4	route1@-1

Typical output:

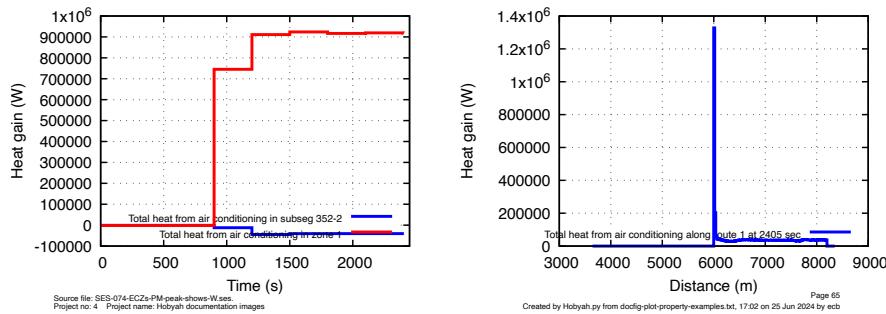


Figure 6.111: Total heat loads from HVAC vs. time and distance

- Curve types where it is used: **transient, profile**
- Where plotted: in a subsegment and along a route.
- Units: watts, BTU/h

Data for **HVAC_total** are stored in a Pandas array called **HVAC_total** with indices of time (as real numbers) and columns of subsegment identifiers and the word “zone” followed by a zone number, e.g. “zone3”. The times are not the full set of plot times, but the start time, the times each ECZ estimate is carried out and the finish time.

6.11 Plots of SVS properties

6.11.1 The pipetemp plot type

Pipetemp is the temperature of the working fluid in SVS cooling pipes. The pipe coolant temperatures are given at the pipe inlet and at the down end of each subsegment: these are converted into temperature values at subpoints.

Cooling pipes can pass through both line segments and vent segments, while SES routes can only pass through line segments.

As a result of this there are two ways of plotting temperature change along the pipes at an instant in time: plotting along the length of the pipe (as if the pipe was a route) and plotting along routes that parts of the pipe passes through.

For convenience, the dry bulb temperature (DB) can be plotted along pipes, see Section 6.8.11.

Typical curve definitions:

<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>subpoint</i>
transient	pipetemp	file3	4-3m
<i>keyword</i>	<i>property</i>	<i>nickname</i>	<i>source@time</i>
profile	pipetemp	file3	pipe2@600
profile	pipetemp	file3	route1@600

Typical output:

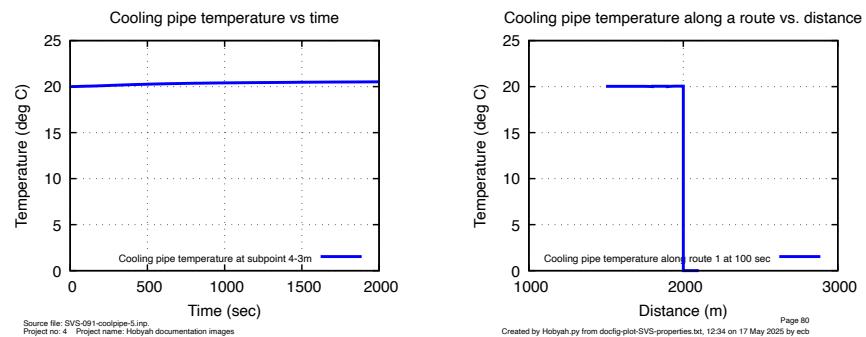


Figure 6.112: Cooling pipe temperature vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subpoint in the subsegments.
- Units: °C, °F

Data for `pipetemp` are stored in a Pandas array called `subpoint_SVSpipe_temps` with indices of plot time (as real numbers) and columns of subpoint identifiers (as strings, like "102-7m").

6.11.2 The pipe_ht plot type

`Pipe_ht` is the heat transfer to cooling pipes. The heat transfers are given in each subsegment as watts per subsegment.

Cooling pipes can pass through both line segments and vent segments, while SES routes can only pass through line segments.

As a result of this there are two ways of plotting temperature change along the pipes at an instant in time: plotting along the length of the pipe (as if the pipe was a route) and plotting along routes that parts of the pipe passes through.

Typical curve definitions:

```
keyword      property      nickname      segment-subseg
transient    pipetemp     file3          4-3
keyword      property      nickname      source@time
profile      pipetemp     file3          pipe2@600
profile      pipetemp     file3          route1@600
```

Typical output:

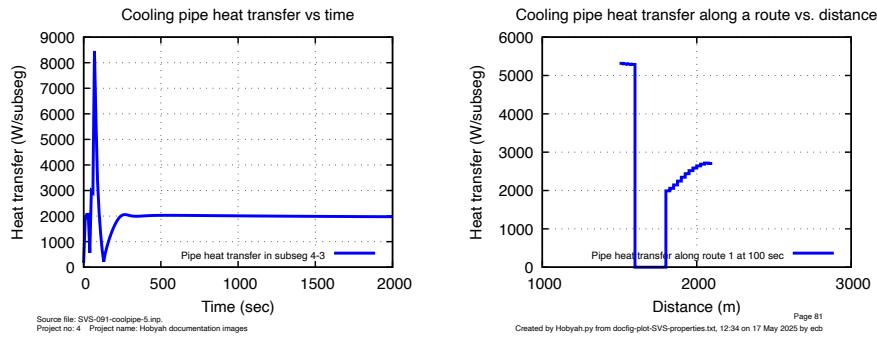


Figure 6.113: Cooling pipe temperature vs. time and distance

- Curve types where it is used: `transient`, `profile`
- Where plotted: one value at each subsegment.
- Units: m^2 , ft^2

Data for `pipe_ht` are stored in a Pandas array called `subseg_SVSpipe_HT` with indices of plot time (as real numbers) and columns of subsegment identifiers (as strings, like "102-7").

Chapter 7

Miscellany

This chapter holds sections that expand on points that may occasionally be of interest but which would break the flow of earlier sections if they were included there.

7.1 What's a Hobyah?

Hobyah! Hobyah! Hobyah! Tear down the house, eat up the old man and woman, and carry off the little girl!

That quote is a war-cry in a folk tale—*The Hobyahs*—that was a favourite of mine when I was about four years old.

Read it here: <https://archive.org/details/moreenglishfairy00jacocuala/page/118>

Trigger warnings: **animal cruelty, murder, kidnapping, genocide.**

Hobyahs are your run-of-the-mill, pointlessly-cruel, monsters-who-raid. The image below left shows some Hobyahs heading off at night to raid the standard “house at the edge of the forest” that is a staple of such tales.



The hero of *The Hobyahs* is Little Dog Turpie (LDT). LDT saves his owners each night by barking loud enough to scare off the approaching Hobyahs (middle image). His master (the old man in the quote) misunderstands what's going on—each morning he cuts off one of the dog's limbs as punishment for keeping him awake (children's tales before the late 20th century were 100% nightmare fuel for no apparent reason). LDT's master eventually runs out of patience and makes LDT take a dirt nap.

The next night there is no LDT to scare off the Hobyahs. They duly tear down the house, eat up the old man and woman, and carry off the little girl in a sack (above right). Toddler me loved this story. I cherish the memory of the cheerfully gruesome way that my father used to read it aloud.

Nowadays, parents are a bit more honest about the intent of bedtime stories: they read *Go the Fuck to Sleep* to their kids instead. Progress!

7.2 Minimum cell size in characteristic calculations

This section describes how the program sets its cell size. There are five values in the input file's `settings` block, as follows.

- `p_atm` (default 101325 Pa if you don't set a value)
- `rho_atm` (default 1.2 kg/m³ if you don't set a value)
- `gamma` (default 1.4 if you don't set a value)
- `max_vel` (default 20 m/s if you don't set a value)
- `aero_step` (which has no default, so we'll use $\Delta t = 0.05$ seconds)

These are turned into a minimum distance between gridpoints as follows.

First the pressure, density and ratio of specific heats (`gamma`) are turned into a celerity,

$$c = \sqrt{\frac{\gamma P_{atm}}{\rho_{atm}}} \quad (7.1)$$

$$\Rightarrow c = \sqrt{\frac{1.4 \times 101325}{1.2}} \quad (7.2)$$

$$\Rightarrow c = 343.8 \text{ m/s.} \quad (7.3)$$

Next the maximum slope of the forwards characteristic ($u + c$ in the $x-t$ plane) is calculated. In this case u is the value in `max_vel`;

$$\text{slope} = u + c = 20 + 343.8 = 363.8 \text{ m/s.} \quad (7.4)$$

The minimum value of cell length is the slope multiplied by the timestep,

$$\text{slope} * \Delta t = 363.8 \times 0.05 = 18.19 \text{ m.} \quad (7.5)$$

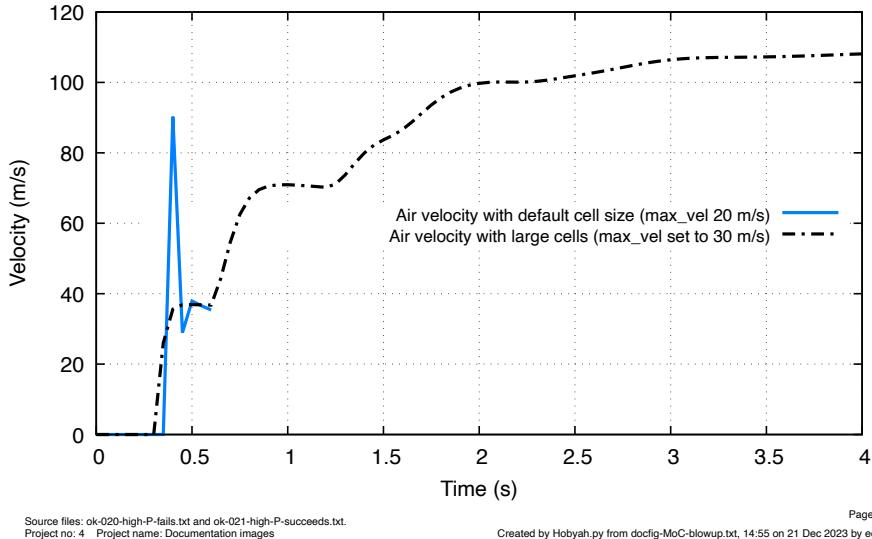
Hobyah creates gridpoints that at least this far apart. If you had a tunnel that was 36.3 m long it would have one segment (because $18.19 \times 2 = 36.38$ m). If you had a tunnel 36.4 m long it would have two segments (when divvying up a computational domain you have to draw lines somewhere).

In the event of the value of $u + c$ exceeding 363.8 m/s, the base of the characteristic lies outside the cell and the calculation may become unstable and blow up.

If that happens, the easiest solution is to set a larger value for `max_vel` in the `settings` block. If a value of 30 m/s was set, the cell length would be $0.05 \times (30 + 343.8) = 18.69$ m.

A rule of thumb that I use all the time is to use a timestep of 0.05 seconds and place entities at least 20 m apart.

Figure 7.1 shows the results of two calculations. The only difference between the two files is that the one in which the calculation does not blow up has `max_vel` set to 30 m/s, and the one that does blow up has the default (20 m/s). This calculation is a pretty extreme test for the method of characteristics: a fixed pressure of 18,000 Pa is applied to one end of a relatively short tunnel full of stationary air. The successful calculation generates airspeeds of over 100 m/s in the tunnel in the first four seconds. The other calculation fails.

Figure 7.1: Effect of increasing the `max_vel` setting

If the calculation in Figure 7.1 that failed was to run for one timestep more the program would fail with the following message:

```
> Fouled up when calculating with two character-
> istics, ended up with a gridpoint that had the
> following values:
>   celerity = NaN
>   velocity = NaN
> If either of these are "NaN" or there are runtime
> warnings printed to the screen, try increasing
> max_vel in the "settings" block to force larger
> cell sizes. See also the Miscellany chapter in
> the User Manual for more details.
> Try setting the aero_time to .60000 seconds and
> plotting celerity and velocity values. This
> will show the state of the system at the last
> stable timestep.
```

A lot of useful information can be gathered about such failures by setting the `aero_time` to the value suggested in the error message, which is the timestep before the calculation blew up.

7.3 Convergence problems

Although the method of characteristics is very stable, it does occasionally encounter stability problems that don't stop the calculation. `Hobyah.py` will issue warnings when this happens.

Typically, these warning messages are copied directly from the main junction solver code (`SciPy.optimize.fsolve`) but they can also be caused by large losses at portals open to atmosphere.

Sometimes the stability problems in the calculation are not a problem in the real world. When the calculation of pressure rise across a fan doesn't converge when the duty point is crossing the stall hump while the fan is changing speed (see the test file named `ok-031-fan-test.txt`, which raises three "didn't converge" error messages when the duty point of fan 2 crosses its stall hump). This is a transient feature (you can just see a wiggle in the air velocity caused by it between 115 seconds and 116 seconds in `ok-031-fans-in-series-1p1.pdf`). The calculations at that time are incorrect, but once the duty point moves away from the stall hump the calculation gets back to normal.

But sometimes the instabilities are caused by something that must be changed because the results are inaccurate. For example, when there are high pressure losses at portals open to atmosphere, sometimes the MoC calculation hunts as it tries to get to a stable solution. In the calculation `kb-001-portal-zetas.txt` the solver hops back and forwards between two possible solutions at every time step as soon as the air velocity gets high enough. It writes over 900 red flags to the terminal and to the log file. The velocity, mass flow and pressure profiles in the file "`kb-001-portal-zetas.pdf`" are deeply suspect (especially the mass flow profile).

So what should you take from this? Four things:

1. If Hobyah writes a message to the screen saying "The calculation suffered from stability problems", do not ignore it.
2. You should plot output that you can use to gauge whether this is a genuine problem.
3. If it turns out that a particular entity in a tunnel wrote a screed of messages at multiple time steps, then you have a problem. The calculation is not going to converge.
4. A good test is to plot mass flows along the segment in which the calculation did not converge. If the calculation can't conserve mass, then it's probably wrong. Don't go overboard with this though: pressure waves can cause fluctuations in mass flow as the cells in high pressure regions have higher density and store more mass than the same size cells in low pressure regions. And there can be minor mismatches in mass flow (on the order of grams/second) in correct calculations.

7.4 How to submit bug reports

All software has bugs and users will find them, no matter how much time a programmer spends trying to catch them first. So if you are using Hobyah and you find a bug, what should you do? Submit a good bug report on github!

A good bug report has the following characteristics:

- The bug report doesn't include anything telling me how to fix the bug. Including code showing how to fix the bug can cause copyright issues, which is something I want to avoid. Most users are likely to have signed employment contracts that assign the copyright of all the code they develop to their employer. I want to keep Hobyah free of legal entanglements, so don't send code suggestions!
- The bug report has as few input files as possible (ideally two). One input file allows the problem to occur, the other does not.
- The files have as few differences as possible (ideally no more than one difference).
- The input files are accompanied by as few external files (`.csv` files, image files, binary files) as possible.

An example of two files with minimal differences are `ok-020-high-P-fails.txt` and `ok-021-high-P-succeeds.txt`. There are two differences between these input files (which were used to illustrate a known stability problem in Figure 7.1).

7.5 Stability of calculations

The software has the following known issues:

- When a fan characteristic has a stall hump and the system resistance puts the duty point near the stall point, the solver hunts and cannot converge on a stable answer (just like fans with a stall hump in real life: they hunt). In many cases it gives up and delivers unrealistic results like the wobble you can see in the pressure profile in the test file `ok-032-fans-in-parallel-lp1.pdf` between 114.8 seconds and 116 seconds.
- When large pressure loss factors are applied at portals with a fixed pressure, the solver sometimes fail to calculate. This is a bug in the routines `OpenEnd` and `PortalCelerity`, which solves for c_N and u_N at the portal gridpoint using one characteristic and the pressure change between the atmospheric reservoir and the portal gridpoint. A workaround is to not place large pressure loss values at portals; instead, to place them at `loss1` entities a few tens of metres inside the portals. See the test file `kb-001-portal-zetas.txt`.

7.6 Limits on use cases

7.6.1 Variable properties

Some method of characteristics calculations are extremely sophisticated and allow parameters to vary with distance, pressure and time.

For example, there are solvers that allow area to vary with distance (useful for modelling angled tunnel walls) and others that allow area to vary with pressure (useful for modelling blood vessels). Hobyah is not sophisticated. It has the following limitations:

- It does not calculate temperature.

- Tunnels do not have distributed addition of mass flow along the length.
- Angled tunnel walls cannot be modelled.
- It cannot calculate micro-pressure waves (see Section 7.6.3 below for more details).
- Area cannot vary with pressure (Hobbyah is emphatically not intended for modelling blood vessels).

7.6.2 Explicit terms in jet fan thrust and traffic drag

The thrust of jet fans and the drag of traffic are calculated from the following expressions

$$E_{\text{traffic}} = -\frac{1}{2A_t} \sum_{i=1}^N [D_{v_i} c_{d_i} A_{v_i} (v_{v_i} - v_t)] |v_{v_i} - v_t| \quad (7.6)$$

$$E_{\text{jet fans}} = -\frac{T_f}{\rho_0} \frac{N_f \eta_f}{A_t} D_{pt} \frac{(v_f - v_t)}{|v_f|} \quad (7.7)$$

in which the tunnel air velocity v_t is not the the air velocity at the current timestep, but the air velocity at the previous timestep.

This kind of calculation (called an explicit calculation by anyone familiar with writing finite element software) is always wrong. But—and this is crucial—this kind of calculation is rarely wrong enough to worry about.

A fully implicit calculation would use v_t at the current timestep to calculate the traffic drag and jet fan thrust. Implicit calculations are difficult to program because:

- In implicit calculations the value of v_t at the current timestep affects the traffic drag & fan thrust.
- In implicit calculations the traffic drag and fan thrust both affect v_t .
- Traffic/jet fans may be next to junctions. Modifying the current value of v_t in a cell next to a junction affects the entire flow split at the junction.

The key question is: how wrong is the explicit calculation? To be more specific: how different is the jet fan derating term calculated from air velocity at the current timestep compared to the derating term calculated from air velocity at the previous timestep?

Let's put some numbers in here. I generally never use timesteps longer than 0.1 seconds, so that sets the timestep. The file `docfig-explicit-airspeeds.txt` has a short (100 m) tunnel with two 2 kN jet fans in it. These jet fans have a relatively low outlet velocity (just 27 m/s). Figure 7.2 shows the results of starting those two fans in a tunnel with air at rest. Within 20 seconds of starting up, the jet fans generate an air velocity of just under 10 m/s. This is a more rapid rate of rise than would occur in the real world (first, we wouldn't design jet fans to generate 10 m/s in the tunnel and second, we'd probably stagger their start times for electrical reasons).

The lower graph in Figure 7.2 shows a detail of the highest rate of rise of air velocity, which is about 1.2 m/s^2 (between 4.5 seconds and 5.5 seconds). This equates to a change in speed of 0.11 m/s over one calculation timestep.

This allows us to calculate the jet fan derating term at time 4.6 seconds using the air velocity at time 4.6 s (an implicit calculation) and at time 4.5 s (an explicit calculation)

and calculate what percentage of error we have in the explicit calculation. The velocity derating factor for jet fans is given by

$$\text{derating} = \left(1 - \frac{v_t}{U_j}\right) \quad (7.8)$$

where

v_t = velocity derating factor

v_t = tunnel air velocity

U_j = jet velocity.

In the implicit calculation, $v_t = 1$ m/s (air velocity at 4.5 seconds). In the explicit calculation, $v_t = 1.12$ m/s (air velocity at 4.6 seconds). The derating factors for these values of v_t are:

$$\text{implicit calculation derating} = \left(1 - \frac{1.12}{27}\right) = 0.959 \quad (7.9)$$

$$\text{explicit calculation derating} = \left(1 - \frac{1}{27}\right) = 0.963 \quad (7.10)$$

$0.963/0.959 = 1.005$, so the explicit calculation miscalculates the derating factor by 0.5% at the maximum rate of change of air velocity in this simulation. This is the maximum inexactness: at later timesteps, when the rate of change of air velocity approaches steady-state, the error reduces to zero.

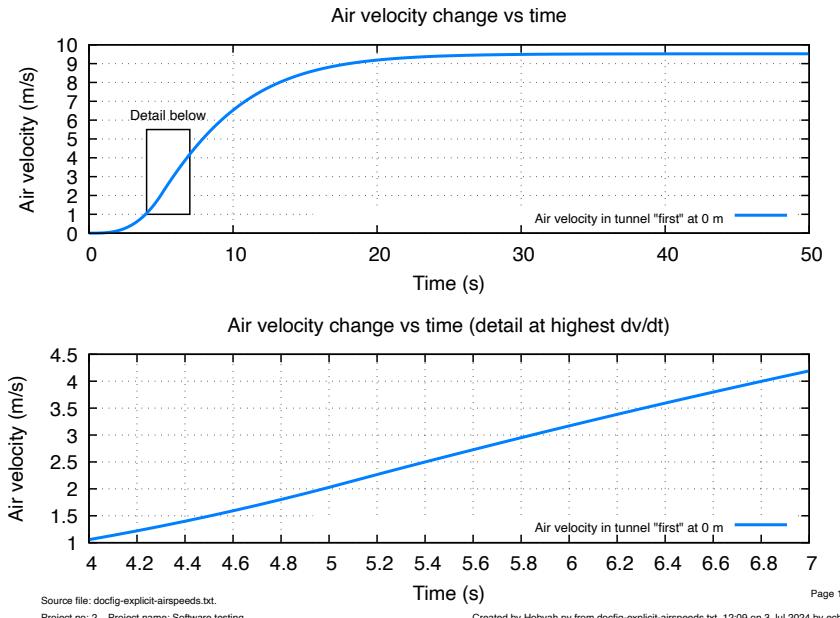


Figure 7.2: Air velocity changes example

An inexactness of 0.5% in the derating factor due to using an explicit scheme in this “worse than it would be in a real-world design” is considered insignificant, so the explicit scheme can be used.

7.6.3 Micro-pressure waves

In long rail tunnels that carry high speed trains, a loud noise like a gunshot may be emitted from the exit portal shortly after a train enters. This is caused by a micro-pressure wave (MPW), also known as a tunnel boom. If not assessed correctly and mitigated properly, MPWs may cause unacceptable disturbances at buildings near the exit portal (by rattling doors/cracking windows every time a train goes through the tunnel).

Improper assessment may cause a railway operator to have to introduce a permanent speed restriction (PSR) in a tunnel in order to avoid adversely affecting nearby buildings.

Having to impose PSRs after running the first few test trains through the tunnels on a new railway is not what any railway operator wants to do. The operator may well conclude that the tunnels have been designed & built by consultants & contractors who may be incompetent.

So assessment of MPWs is important for figuring out what mitigation measures to use (such as a larger tunnel diameter, porous materials like ballast along the tunnel, extremely long train noses, portal hoods, air chambers—the list goes on and on).

High speed rail tunnel projects that do not assess MPWs usually find out that MPWs are a problem only when they start running test trains, which is far too late to for the project to do anything but wring its collective hands, weep and wail piteously, start frantically building and adjusting Helmholtz resonators at the daylight portals and (of course) lawyer up.

Hobyah is not suitable for assessing MPWs. If you are designing a high speed rail tunnel, do not try to use Hobyah to assess MPWs (the calculation methods used in Hobyah cannot assess them).

Instead, find an expert in MPWs and ask them for advice. I am not an expert in MPWs but I may be able to put you in touch with people who are.

7.7 SES train numbering

Train numbers in SES are tricky. Although any number of trains can pass through the system, the train numbers printed in the output are restricted to two digits.

The first train to enter an SES run is given train number 1, the second is given 2 and so on.

As the count of trains goes up it breaches the two-digit limit at train 100. The “1” in train 100 gets lopped off in the Fortran printing scheme and train 100 is printed as train 00. The next train to enter will be train 101 (printed as train 01) if the original train 1 has left the system. If the original train 1 is still in the system, the next train will be given the number of the first train to have left the system.

`Hobyah.py` and `SESconv.py` work within SES’s 100-train framework by allocating train arrays that can hold data for up to 100 trains at each timestep. Before a train enters, its entries in the Pandas dataframe are set to NaN. Once it enters the system the entries are changed to numbers. Once it leaves, its entries go back to being NaNs.

If another train enters with the same number, the entries go back to being numbers for that new train. If a train never enters, its entries in the dataframes are removed before saving to the `.sbn` file (no point storing columns entirely made up of NaNs for train numbers that aren't used).

I did think about getting `SESconv.py` to keep a count of which trains had left and which had not and try to predict which train number would be available next, but ultimately decided it was a waste of time at the moment—I've only ever used more than 100 trains in test files, never in design files.

I see that the `Development` branch of OpenSES has a new array for tracking train numbers (part of their work on developing JSON output from SES), so perhaps a better way of handling train numbers will come out of that work.

7.8 Limits

This section describes the limits of the software. Some of it is what the software can and cannot do, some of it is numerical limitations.

7.8.1 Maximum number of air paths at junctions

Hobyah is set up so that it can handle up to six air paths at nodes and joins. This means that six tunnel ends can be connected to one `node` and four air paths can be socketed into the side of a tunnel at a `join`.

Six is an arbitrary limit. I chose it because six air paths covers most of the cases I've had to deal with in my professional life and if I need more, it is relatively easy to put in short, low-loss tunnel sections that can relax the restriction without causing calculation issues.

I had to write 128 test files for the junctions with six air paths in order to cover every combination of endedness (six back ends joining together, six forward end joining together, 5 back plus 1 forward etc.) with each permutation of flow direction. It was a slog, even with a text editor that can search and replace across hundreds of files (a quick shoutout to Sublime Text: you're worth every cent I paid for your licence).

If you need more than six tunnel ends at a junction because you have a vent adit and six or more duty fans in parallel, put in a fake junction that lets up to five fans connect to one short, low-loss tunnel and another (up to) five fans connect to another short, low-loss tunnel. An example of this can be found in test file `ok-033-nodes-in-parallel.txt`, which is based on a tunnel vent shaft I worked on that has six extract fans and a connection to a road tunnel for pollution extract—seven air paths in all.

7.8.2 Limits on the size of numbers

General limits

As a general rule, floating point numbers and integers should not exceed the limits of signed 64 bit numbers, because that is the size limit used in the Fortran code. So no integers larger than $\pm 9,223,372,036,854,775,807$ and no floating point numbers larger than $\pm 1.7 \times 10^{308}$. If you're getting anywhere close to those, something has almost certainly gone wrong with your calculation.

How to avoid Gnuplot numbering clashes

Gnuplot can be told to generate labels, polygons and arrows. You can either let gnuplot number them for you or specify a number for a label, polygon or arrow. Experienced users of gnuplot usually set the numbers themselves (so that they can turn the labels, polygons and arrows on and off in the .plt files that they write). Users of Hobyah can pass commands to gnuplot to generate labels, polygons and arrows and commands to turn them on and off in `verbatim` blocks or the one-line `verbatim` command.

`Hobyah.py`'s `icons` curve type (see Section 3.25.5) generate a collection of numbered labels, polygons and arrows, so that Hobyah can activate and de-activate its icons that represent jet fans, trains, fires and whatnot.

Advanced users of gnuplot can generate numbered labels, polygons and arrows in `verbatim` blocks. If we're not careful, Hobyah's `icons` may set numbers that overwrite the numbers that experienced gnuplot users use in their `verbatim` blocks.

So there needs to be a way of ensuring that the numbering system used by Hobyah's routines don't clash with the numbering system used by advanced users in their `verbatim` blocks.

I reckon that the best solution is this: we agree to stick to different ranges of numbers.

Hobyah assumes that it has exclusive use of all the numbers above two billion for its icons, labels and arrows. This leaves the range 1–1,999,999,999 for user-defined Gnuplot features in `verbatim` blocks.

So if you are an experienced user of Gnuplot and want to put labels, arrows and objects in `verbatim` blocks, best to keep your label numbers below 2,000,000,000. If you start using numbers above that, figuring out the resulting clashes will be time-consuming.

If you need to use numbers above 2,000,000,000 in your `verbatim` blocks, you can edit the `Hobyah.py` script, search for the text “`poly_num =`” (without quotes) and increase the number assigned to that variable.

But be aware that when Gnuplot runs on 32-bit machines, the largest valid Gnuplot label number is $2^{31} - 1$ (2,147,483,647). Old versions of Gnuplot running on 64-bit machines may have this limitation too. See Section 7.10 for more details.

7.8.3 Limits on time accuracy

Times are rounded to eight decimal places, to make it easier to find and remove duplicate times when merging lists of floating point numbers that may have been generated in different ways.

This ought not be a problem as timesteps in tunnel ventilation calculations are generally in the range 0.001 to 0.1 seconds. If you want to use timesteps shorter than 10 nanoseconds, you will need to spend an enormous of time understanding the source before you use it (look for the variable named `time_accuracy` in the code). You may also need to modify all instances of the `math.isclose` function, because `math.isclose()` defaults to treating two numbers as equal if the relative difference between them is under 10^{-9} .

This is probably something that will only need to be considered by engineers forking Hobyah to use in a field other than tunnel ventilation: other fields may use extremely short timesteps in the method of characteristics.

7.8.4 Deciding when to merge boundaries

Train ends are moving boundaries, and sometimes they will be so close to fixed boundaries that it makes sense to treat the two as coinciding. The reasoning behind the decision on whether to treat them separately or as coinciding is described in `MoC.pdf`.

It is implemented in the code by a variable named `close_dist`, which is a distance in metres. If a train end is closer than this distance to another boundary (such as a node, join, sectype change, fan, loss coefficient or portal door) the boundaries are treated as if they coincide.

This seems to be a big deal in the tunnel ventilation field, and something that a lot of experts in the field who have written method of characteristics programs skimmed over without detailed explanations in their papers.

I expect to get called out for the choices I made when treating this aspect of the method of characteristics. I've tentatively pegged it as the leading contender for me raising the input file version number (see Section 3.3.1) when someone with more experience than me demonstrates that the code I wrote was bad.

7.8.5 Why are route profiles/gradients extended?

Those of you looking at route plot data (such as elevation or gradients) may notice that route gradients continue beyond the final datapoint specified in your input file.

This is a side-effect of something included in the function `ProcessRoute` to ensure that trains never run out of track gradients. In runs with trains, the track elevations are extrapolated to just after the longest train length past the last point in the vertical profile. In runs without trains, the “maximum train length” variable (called `max_trlen` in the code) is set to 1000 m, so the vertical profile and gradients are extended by 1000 m.

7.9 A Gnuplot problem with image files

I have had one problem with Gnuplot that may be worth describing in case someone else has the same issue. On one machine I had a version of Gnuplot that claimed it was able to handle image files. When that version tried to read an image it discovered that it could not. Its error message was not too helpful (“**Gnuplot cannot read png/gif/jpeg images**”). A web search found someone who had encountered this before and found the cause: Gnuplot had been compiled without support for the graphics library **libgd**.

See <https://sourceforge.net/p/gnuplot/bugs/2255/> for more details. For what it is worth, I compared the compilation settings of the version of Gnuplot that failed to the compilation settings of a version of Gnuplot that worked. This first transcript is from the machine that couldn’t show images:

```
Gnuplot> show version long
G N U P L O T
Version 5.0 patchlevel 3    last modified 2016-02-21
Copyright (C) 1986-1993, 1998, 2004, 2007-2016
Thomas Williams, Colin Kelley and many others
Compile options:
-READLINE +LIBREADLINE +HISTORY
-BACKWARDS_COMPATIBILITY +BINARY_DATA
+LIBCERF -LIBGD
-USE_CWDRC +X11 +X11_POLYGON +MULTIBYTE +X11_EXTERNAL +USE_MOUSE
+HIDDEN3D_QUADTREE +DATASTRINGS +HISTOGRAMS +OBJECTS +STRINGVARS
+MACROS +THIN_SPLINES +IMAGE +USER_LINETYPES +STATS +EXTERNAL_FUNCTIONS
MAX_PARALLEL_AXES=7
```

I guess that the compile option **-LIBGD** above means that there is no support for the **libgd** library. This second transcript is from the machine that could show images:

```
Gnuplot> show version long
G N U P L O T
Version 5.4 patchlevel 3    last modified 2021-12-24
Copyright (C) 1986-1993, 1998, 2004, 2007-2021
Thomas Williams, Colin Kelley and many others
Compile options:
-READLINE +LIBREADLINE +HISTORY
+OBJECTS +STATS +EXTERNAL_FUNCTIONS
+LIBCERF +GD_PNG +GD_JPEG +GD_TTF +GD_GIF +ANIMATION
-USE_CWDRC +USE_MOUSE +HIDDEN3D_QUADTREE
```

In the above transcript I guess that the compile options **+GD_PNG**, **+GD_JPEG**, **+GD_TTF** and **+GD_GIF** mean that the **libgd** library can process .png, .jpg, .tif and .gif filetypes.

The first transcript was from a .tar file downloaded from the Gnuplot website in 2016. The second transcript was from an installation downloaded by the **Homebrew** package manager in 2022.

If you find you cannot get your Gnuplot to show images, put the **show version long** command into Gnuplot and look for “**GD**” in the compile options section of the text it returns. Good luck with getting a version that includes **libgd** support.

7.10 Gnuplot number limits

Before version 5.4, Gnuplot used 32-bit integers to store its label numbers (version 5.4 switches to 64-bit integers where available). So with older versions of Gnuplot (or on computers with 32-bit processors like older Raspberry Pi models), label numbers cannot be higher than $2^{31} - 1$ (2,147,483,647).

If you create a .plt file with the following in it

```
set label 2147483647 "this label number works in Gnuplot" at 0, 0
set label 2147483648 "this label number fails in Gnuplot" at 0, 1
plot sin(x) with lines
```

and run an old version of Gnuplot, it will generate the following error message:

```
set label 2147483648 "this label number fails in Gnuplot" at 0, 1
^
"label-number-too-large.plt" line 9: tag must be > zero
```

Gnuplot looked at the 32-bit signed integer 2,147,483,648 and treated it as $-2,147,483,647$. Hence the complaint the tag number must be over zero. I owe a tip of the hat to Stack Overflow for keeping the answer to this puzzle available.

7.11 Customising the Gnuplot terminal

This section is intended for knowledgeable users of Gnuplot. It describes how Hobyah sets the default terminal and default output file, how you can prevent it setting them and what you can use to set a custom terminal and output file.

Hobyah writes text at the top of its Gnuplot .plt files to set the terminal, name the output file and set the `multiplot` flag. The default is the following Gnuplot commands (assuming the Hobyah input file you are running is named `foo.txt`):

```
set terminal pdfcairo size 29.7cm, 21cm linewidth 1 font "Helvetica, 18" enhanced
set output "/Users/tester/Documents/sandboxes/docs/foo.pdf"
set multiplot
```

If you set `pagesize none` at the top of the `plots` block, Hobyah will still write these lines to the .plt file but will comment them out. It will also define three Gnuplot variables with the path, name and namestem of the file that it would have written to:

```
# set terminal pdfcairo size 29.7cm, 21cm linewidth 1 font "Helvetica, 18" enhanced
# set output "/Users/tester/Documents/sandboxes/docs/foo.pdf"
# set multiplot
outpath = "/Users/tester/Documents/sandboxes/docs/"
outname = "foo.pdf"
outstem = "foo"
```

You will have no output file defined and will use Gnuplot's default terminal. This may sound crazy but it's there for a good reason: it lets knowledgeable users of Gnuplot set a custom terminal type without having to edit the .plt files that Hobyah generates.

Instead, knowledgeable users of Gnuplot can set the file name and terminal type in a `verbatim` block (see Section 3.24.5) at the start of the first block in the first page (a `graph` block or `image` block, whichever comes first).

Similar `verbatim` blocks will be needed at the start of the first block in every `timeloop` and `filesloop` definition, as these generate different Gnuplot .plt files.

The following snippet gives the entries that I usually use:

```
begin plots
  pagesize none
  begin page
    begin graph
      begin verbatim
        # Set a custom terminal, page size, linewidth and text size
        unset multiplot
        set terminal png size 1024, 768 linewidth 1 \
                      font "default, 9.5" enhanced
        set output outpath.outstem.".png"
        set multiplot
      end verbatim
    <the rest of the graph definition goes here>
```

Note that the entry `set output outpath.outstem.".png"` in the `verbatim` block accesses the variables `outpath` and `outstem` that Hobyah already wrote to the file. `Outpath` and `outstem` make this feature portable.

Hobyah does not check the contents of `verbatim` blocks for erroneous gnuplot commands; such faults are way above Hobyah's pay grade. Hobyah will not warn you if you use `pagesize none` but forget to put commands in a `verbatim` block to give the necessary settings. In that case Gnuplot will use its default terminal and output, which will vary across different systems. When I foul up, I usually get a long screed of .pdf commands printed in my Terminal window.

7.12 Roughness heights

Hobyah uses roughness heights in metres. Prospective users may make the point that SVS v6 uses millimetres, so why not be consistent with SVS? It's mostly down to personal preference: I've used roughness heights in metres for all of my professional life. Also:

- Most of the Moody charts I've used in my career express relative roughness (ε/D_h) as metres per metre, not millimetres per metre.
- The first tunnel ventilation program I used was the Mott MacDonald Aero program, which uses metres for roughness heights. I'm pretty sure Thermotun does too.
- Before writing a program to convert SI input files to US units for Aurecon SES in 2016, I did a poll of my colleagues to see if they had any strong opinions about

what multipliers to put on units. Most of them strongly preferred consistent units across the input file (all distances in metres, all powers in watts). The only debate was whether to put fire sizes in form 4 in watts or megawatts.¹

7.13 Why show three types of friction factor?

Hobyah lets you use one of three types of friction factor in your input file: Fanning friction factor c_f , Darcy friction factor λ and Atkinson friction factor k . Whichever one you choose to set, it prints all three to the `log` file so you can compare them.

The reason Hobyah shows both Fanning friction factor (c_f) and Darcy friction factor (λ) is because Fanning and Darcy friction factor have both been around since the mid-19th century and are often given the same symbol (f) in modern engineering textbooks and papers, despite Darcy friction factor being four times larger than Fanning friction factor.

This factor of four problem has led to a lot of confusion over the years, particularly in the field of high-speed rail tunnels. See the file “`friction-rant.pdf`” in the Hobyah documentation folder for more details of the relationship between the two.

As far as Atkinson friction factor is concerned: I wish I could put my hand on my heart and claim that I included Atkinson friction factor k to boost co-operation between the mine ventilation and tunnel ventilation professions. That would be a truly worthy reason to include it.

But the ugly truth is this: I included Atkinson friction factor k to wind up some of my ex-colleagues. Specifically, those ex-colleagues who used to throw their toys out of the pram whenever I asked them to check a calculation that referred to full-scale test data from mine ventilation papers.

7.14 A damper2 problem

<Placeholder>

Need to write a file to illustrate this issue (linear interpolation of Atkinson resistance is less representative of a damper closing than linear interpolation of area and zeta in `damper1` entries).

7.15 Ignored input in the plots block

Some input in the `plots` block is ignored.

The `plots` block has lines of input that define plot settings as well as blocks that define pages and loops.

¹We ended up choosing MW for fire sizes in form 4 because MW means fewer zeros. Be honest: can you tell whether 100000000 W is 10 MW or 100 MW at a glance? Of course not, you need to spend a little time counting the zeros. Using 100 MW or 10 MW in form 4 input made it much easier to spot an extra or absent zero in the fire size.

The lines of input that define the plot settings begin with one of the following keywords: `pagesize`, `orientation`, `plotunits`, `font`, `fontsize`, `linewidth` and `basemargins`.

Only the plot settings before the first `begin page`, `begin timeloop` or `begin filesloop` command are processed.

The following example has two `pagesize` commands in it. The first is processed, the second is ignored.

```
begin plots
    pagesize A3 # This line is processed because it appears before the first sub-block.

    begin page # This is the start of the first sub-block
        begin graph
            <definitions of graphical results>
        end graph
    end page

    pagesize A4 # This line is ignored because it appears after the first sub-block.
    begin page # This page will be A3, not A4
        begin graph
            <definitions of graphical results>
        end graph
    end page
end plots
```

The reason is that the `ProcessPlots` routine processes the lines of input at the top of the block and stops when it finds the first line starting with `begin`. Then it processes all the `page`, `timeloop` and `filesloop` blocks without checking for lines of input outside those blocks.

If you think that's not how the program should behave, you are welcome to rewrite it to handle `pagesize`, `orientation`, `plotunits`, `font`, `fontsize`, `linewidth` and `basemargins` commands between page definitions. Good luck!

7.16 Command line options

I run Hobyah from the command line 99% of the time, and hardly ever drag and drop input files from a folder onto an icon that is linked to a Python script. I use the command line a lot because I can run a subset of filenames by passing regular expressions to my program on macOS's command line interface.

Hobyah has six command line options, which are described below.

7.16.1 Command line option `-debug1`

Command line option `-debug1` is for programmers. It causes the program to print an enormous amount of information to the screen, to help with debugging problems.

7.16.2 Command line option `-nocalc`

Command line option `-nocalc` prevents the program from carrying out a calculation.

I use this all the time when I've carried out a long calculation and only want to rerun the file to adjust the plots.

I typically run a calculation, look at the output, then realize that I think that the calculation is correct but I want to change graph extents, or I didn't plot the correct curve on one graph, or I want additional plots based on what I can see from my plots. Rerunning the file with the `-nocalc` option means that I can regenerate the plots without having to take the time to recalculate.

On a related note, you should be aware that when a calculation is run, the existing `.hbn` file is only deleted AFTER the calculation is finished. If you forget to include the `-nocalc` option and accidentally start recalculating, you can stop the run by pressing `<ctrl> C`. If you manage to press `<ctrl> C` before the calculation ends, the existing `.hbn` file will be unchanged. You can usually rerun with the `-nocalc` option and access a valid `.hbn` file.

If you usually drag and drop files and don't want to rerun the calculation every time, you can also prevent the program calculating by setting `runtype plot` instead of `runtype calc` in the `settings` block (see Section 3.3.2) for more details.

7.16.3 Command line option `-nofortran`

This command line option tells the program not to load the Fortran compressible flow routines but to use the Python ones instead.

It is mostly used during code development when ensuring that the Python and Fortran routines calculate the same results.

New capabilities are developed in Python then ported to Fortran, so `-nofortran` is used to test whether the Python and Fortran routines give the same answer.

You do not need to use this option if you do not have the Fortran routines available on your system. If the Fortran routines can not be imported, the program will fall back on the equivalent Python routines automatically.

7.16.4 Command line option `-serial`

Hobyah can be told to process multiple input files. By default it tries to process these in parallel, assigning one core of your computer's CPU to each file.

This is good (because it processes a long list of input files in a short time) but also bad (because if one input file fails or complains about something, the runtime transcript in the PowerShell/Terminal window is a mishmash of all the lines written by each file as it is processed).

This `-serial` command line option tells the program to process each file in sequence, so that when an error occurs it is easier to trace which input file caused it. `-serial` is also used when running the fault test files that generate the transcript of error

messages in Appendix C of this document.

7.16.5 Command line option `-showerrors`

This command line option tells the program to not print certain messages to the screen. It is used when running the fault test files that generate the transcript of error messages in Appendix C.

7.16.6 Command line option `-dudbins`

Command line option `-dudbins` is for programmers. It tells the program to generate three `.hbn` files with faults in them. The faulty files are used to test error messages in `classHobyah.py`.

Appendix A

Using the Fortran routines

A.1 Compiling the Fortran

The Fortran routines that Hobyah can use to calculate are held in a file named `compressible.f95`. They are compiled into a form that Python can access them by the SciPy module's `f2py` function.

You may not need to compile the Fortran at all; the executables uploaded to GitHub (for Windows and Apple silicon) may be suitable for your machine. If you think that might be the case, skip to Section A.2.

If you are on some other system, it is not essential to compile the Fortran routines: there are equivalent Python routines that the program can use. The only advantage of the Fortran routines is that they are much faster than the Python routines.

If the executables uploaded to GitHub are not suitable for your machine, you are not familiar with Fortran and your employer has an IT department, then please point your company's IT department at this section of the document and get them to do the work for you. Someone, somewhere in your IT department will either know how to handle SciPy and Fortran or be smart enough to figure it out from this section and from SciPy's online help.

For me, SciPy's `f2py` method worked first time on macOS and Linux. On Windows I tried and failed to get `f2py` and `gfortran` to talk to each other for a long time. The `f2py` documentation has a page specifically discussing using `f2py` on Windows (<https://numpy.org/doc/stable/f2py/windows/index.html>). I ended up installing the Microsoft Build Tools for Visual Studio as per that page's recommendation. Once the Build Tools were installed on my Windows machine, `f2py` was able to put the routines in a form that `gfortran` accepted and I was able to compile the Fortran routines.

One oddity that I did encounter was that on some machines, `f2py` builds the Fortran executable and generates a Windows dynamic linked library (a `.dll` file) that it places in a folder two levels down from the Fortran executable, presumably for a good reason. Unfortunately having the `.dll` file in a folder that Python doesn't search in means that none of the compressible flow routines worked. I found it necessary to copy the

.dll file to the same folder as the .dyd file to get the compressible flow routines to work on Windows.

If you don't have access to an IT department you might well be out of luck: I can't promise to solve your problems. The best I can offer is the following (assuming you are familiar with command-line interfaces and the Python interpreter):

First, ensure you have Python, Python's `numpy`, `Pandas` and `SciPy` modules installed¹. Next, ensure you have a suitable Fortran compiler installed. I use the open-source `gfortran` compiler on macOS, Linux and Windows.

Assuming you have a suitable set of programs installed, open `cmd.exe` (Windows) or Terminal (macOS and Linux). Navigate to the folder that contains `compressible.f95`.

Issue the command

```
python3 -m numpy.f2py -c compressible.f95 -m compressible
```

On some systems the first word may need to be `python` instead of `python3`:

```
python -m numpy.f2py -c compressible.f95 -m compressible
```

If you get a message saying that it can't find `numpy.f2py`, your last chance is to use

```
f2py -c compressible.f95 -m compressible
```

If that doesn't work, I'm afraid you're stuck with using Hobyah's Python routines unless you can get in touch with someone with a lot of experience of `f2py`.

If one of the above works, then the program will issue a long screed of messages (`f2py`'s runtime transcript).

If you are lucky, the words `Removing build directory` will be on the last line and a new binary file with a name like `compressible.cPython-39-darwin.so` or `compressible.p310-win_amd64.pyd` will have been created in the working folder. There may be warnings (when I compile I get six warnings, but none of them seem to be serious enough to stop the compilation). Now check if a new subfolder named `compressible` has been created in the same folder as the `.pyd` file. If it has, drill down into its subfolders until you encounter a `.dll` file. Copy the `.dll` file to the same folder as the `.pyd` file (this is an ugly hack that I hope to get a workaround for some day).

If you are unlucky, `Removing build directory` will not be the last line of the transcript. Instead there will be a different message describing a problem with the compilation, some mismatch between the compiler and `f2py`. `F2py`'s documentation (<https://numpy.org/doc/stable/f2py/index.html> has extensive notes on troubleshooting).

There is a particularly helpful section on using it on Windows (<https://numpy.org/doc/stable/f2py/windows/index.html>).

¹You can check which Python modules are installed by typing `help("modules")` in a Python interpreter.

If that doesn't help, your next port of call should be the website Stack Overflow.

Stack Overflow is full of questions posed by new Ph.D candidates (mostly studying physics and engineering) who are trying and failing to compile godawful Fortran code that their Ph.D supervisor's Ph.D supervisor wrote in the 1960s for Concorde, Polaris, stellar evolution or suchlike.

If you find them of little help, your best bet is to start searching for people who have encountered similar problems online (once again, Stack Overflow is probably your best bet). I can't give you much more assistance than the words above.

A.2 Placing the executable

Assuming that you have an executable that you think is suitable (a `.so` file on macOS or `.dyd` and `.d11` files on Windows) you will need to put it/them in a place where Python can find it.

The best thing to do is create a folder to hold all the `.py` files in the suite and put the `.so` file (macOS) or the `.dyd` and `.d11` files (Windows) in them.

In my "test the program" account, the whole Python installation is owned by my `tester` account, so all those folders are accessible to the user. But I reckon that the most logical one to put the `.so` file into is the `site-packages` folder.

Assuming that you manage to build a `.so` or `.pyd` file and put it in a suitable folder, it is worth checking that the compressible module can be accessed by Python. Start a Python session from the command line and give it the following command:

```
>>> import compressible
```

If you don't get an error message, it was imported successfully.

Ask for help on the module:

```
>>> help(compressible)
```

It should spit out a screed of information about the Fortran functions in the file and the variables passed to those functions.

```
>>> help(compressible)
Help on module compressible:

NAME
    compressible

DESCRIPTION
    This module 'compressible' is auto-generated with f2py (version:1.24.3).
    Functions:
        calccelestiy1 = calccelestiy1(p,rho,gamma)
        calccelestiy2 = calccelestiy2(p,p_atm,c_atm,gammapsi)
        colebrook = colebrook(rr37,re)
```

```

fricfac = fricfac(d_h,roughness,rr,rr37,veloc,fric_app_num)
extrapolate = extrapolate(x_mid,x_vals,y_vals,count=shape(x_vals, 0))
c_b,u_b = baseback1(dt_dx,c_m,u_m,c_r,u_r,c_n,u_n)
c_f,u_f = basefwd1(dt_dx,c_m,u_m,c_l,u_l,c_n,u_n)
c_n,converged = fixedvel(u_n,back_end,dt_dx,d_h,roughness,rr,rr37,fric_const,max_iter,fix)
c_n,u_n,converged = openend(c_outside,zeta_in,zeta_out,back_end,dt_dx,d_h,roughness,rr,rr37)
c_inside = portalcelerity(c_outside,u_inside,zeta_in,zeta_out,psi)
c_new,u_new,message = celvel1(c_array,u_array,d_h,roughness,rr,rr37,zeta_back_bf,zeta_back_af)
char1,char2,continuity,p_stag = twoaymoc2a(varies,fixed)
char1,char2,char3,continuity,p_stag1,p_stag2 = threeaymoc2a(varies,fixed)
char1,char2,char3,char4,continuity,p_stag1,p_stag2,p_stag3 = fouraymoc2a(varies,fixed)
char1,char2,char3,char4,char5,continuity,p_stag1,p_stag2,p_stag3,p_stag4 = fiveaymoc2a(varies,fixed)
char1,char2,char3,char4,char5,char6,continuity,p_stag1,p_stag2,p_stag3,p_stag4,p_stag5 = sixaymoc2a(varies,fixed)
char1,char2,continuity,p_stag = twoaymoc2fan(varies,fixed)
local_drags = trafficadjust(dists,vels,trcalc_data,gpoints=shape(dists, 0),blocks=shape(dists, 0))

.
.

DATA
__f2py_numpy_version__ = '1.24.3'
baseback1 = <fortran function baseback1>
basefwd1 = <fortran function basefwd1>
calccelerity1 = <fortran calccelerity1>
calccelerity2 = <fortran calccelerity2>
celvel1 = <fortran function celvel1>
colebrook = <fortran colebrook>
extrapolate = <fortran extrapolate>
fiveaymoc2a = <fortran function fiveaymoc2a>
fixedvel = <fortran function fixedvel>
fouraymoc2a = <fortran function fouraymoc2a>
fricfac = <fortran fricfac>
openend = <fortran function openend>
portalcelerity = <fortran function portalcelerity>
sixaymoc2a = <fortran function sixaymoc2a>
threeaymoc2a = <fortran function threeaymoc2a>
trafficadjust = <fortran function trafficadjust>
twoaymoc2a = <fortran function twoaymoc2a>
twoaymoc2fan = <fortran function twoaymoc2fan>

VERSION
1.24.3

FILE
/Users/tester/Documents/sandboxes/Hobyah/compressible.cpython-39-darwin.so

(END)

```

These are low-level calculations that take celerity (variable names beginning with `c_`), velocity (variable names beginning with `v_`) and values like hydraulic diameter and roughness. They carry out the method of characteristics calculations much faster than the Python routines can. All these routines are math-intensive ones with simple inputs (integers, floats and arrays; no lists, tuples or dictionaries). Calculations like that are Fortran's meat and drink; it can calculate them far faster than Python can.

For example, the Fortran routine `celvel1` takes an array of celerities and velocities

in one segment and calculates values of celerity and velocity at the next timestep in all internal gridpoints. It runs in about 5% of the time it takes the Python routine `celvel1` to run.

Of course, the Fortran routines do not speed up the entire program twentyfold. Python does all the work of interpreting the input file, generating the plots, deciding how to handle the boundaries between tunnels and calculating the cell-based traffic drag terms and jet fan thrust terms at every timestep. The time taken to do those tasks is the same regardless of whether you use the Fortran calculation routines or the Python calculation routines. When running a set of 50 test files (a mixture of doing calculations and generating plots) the Python-only version processed them in 291 seconds and the Python+Fortran version processed them in 66 seconds. Only a fourfold reduction.

Appendix B

References

This list of references is in alphabetical order (based on the surnames in author order).

1. Edgell, G, “*Pressure transients in tunnels, extension of the theory to irreversible, non-adiabatic flow*”, University of Leeds Department of Civil Engineering report, 1974
2. Fox, J A, “*The use of the digital computer in the solution of waterhammer problems*”, Paper 7020, J. Instn. Civ. Engrs, 1968
3. Fox, J A, and Henson, D A, “*The prediction of the magnitudes of pressure transients generated by a train entering a single tunnel*”, Paper 7635, J. Instn. Civ. Engrs, 1971
4. Fox, J A, “*Hydraulic analysis of unsteady flow in pipe networks*”, Macmillan Press, 1977
5. Fox, J A and Higton, N N, “*Pressure transient predictions in railway tunnel complexes*”, Proceedings of the 3rd International Symposium on the Aerodynamics and Ventilation of Vehicle Tunnels (ISAVVT), BHRA, 1979
6. Gawthorpe, R G and Pope, C W, “*The measurement and interpretation of transient pressures generated by trains in tunnels*”, Proceedings of the 2nd International Symposium on the Aerodynamics and Ventilation of Vehicle Tunnels (ISAVVT), BHRA, 1976
7. Gawthorpe, R G, “*Aerodynamics of trains in tunnels*”, Railway Engineer International, 1978
8. Henson, D A and Fox, J A, “*An investigation of the transient flows in tunnel complexes of the type proposed for the Channel Tunnel*”, Proc. Instn. Mech. Engrs vol. 188, 1974 (in two parts)
9. Kestin, J and Glass, J S, “*Application of the Method of Characteristics to the Transient Flow of Gases*”, Proc. Instn. Mech. Engrs vol. 161, 1949
10. Lister, M, “*The numerical solution of hyperbolic partial differential equations by the method of characteristics*”, in Ralston and Wilf, “*Mathematical Methods for Digital Computers*”, Wiley, 1960
11. Mizuno, A, Azuma, T, Ichikawa, A and Kanoh, T, “*Appropriate JF installation interval obtained by booster capacity measurements*”, Proceedings of the 10th International Symposium on the Aerodynamics and Ventilation of Vehicle Tunnels (ISAVVT), BHR Group, 2000

12. PIARC report 05.02.B, “*Vehicle Emissions, Air Demand, Environment, Longitudinal Ventilation*”, 1996
13. Pope, C W, personal communication on contraction coefficients, January 2010
14. Vardy, A E, “*The use of airshafts for the alleviation of pressure transients in railway tunnels*”, Proceedings of the 2nd International Symposium the Aerodynamics and Ventilation of Vehicle Tunnels (ISAVVT)”, BHRA, 1976
15. Vardy, A E, “*On the use of the method of characteristics for the solution of unsteady flows in networks*”, Proceedings of the 2nd International Symposium on Pressure Surges, BHRA 1976
16. Vardy, A E, “*Aerodynamic drag on trains in tunnels*”, Proc. Instn. Mech. Engrs vol. 182, 1999 (in two parts)
17. Woods, W A and Gawthorpe, R G, “*The Train and Tunnel—A large scale unsteady flow machine*”, 2nd International JSME Symposium, 1972
18. Woods, W A and Pope, C W, “*On the range of validities of simplified one dimensional theories for calculating unsteady flow in railway tunnels*”, Proceedings of the 3rd International Symposium the Aerodynamics and Ventilation of Vehicle Tunnels (ISAVVT)”, BHRA, 1979
19. Woods, W A and Pope, C W, “*A generalised flow prediction method for the unsteady flow generated by a train in a single-track tunnel*”, Journal of Wind Engineering and Industrial Aerodynamics, vol. 7, pp. 331-360, 1981¹

¹An earlier version of this paper was presented at a joint ASME/CSME conference in 1979.

Appendix C

Error message transcripts

The following is a list of error messages from various programs in the suite. Some are aimed at users, some are aimed at programmers.

This is not the full list of error messages in the program, as a few cannot be triggered by input files. Some can only be raised while new features are being added to the source code (most of those errors are in blocks that would otherwise have a dangling `else` clause).

Error numbers are mostly allocated in blocks of 20, with one block per routine (there are a few exceptions, though).

A few errors in the transcript have the same number. This is usually because there may be more than one logical test that triggers an error, but it may be formatting of the message (error 2086 has several formats, all of which have input files that trigger them).

The names of the test files that trigger errors have the following characteristics:

1. The filenames all begin with `fault-`.
2. The filenames all contain the number of the fault they raise, e.g. `fault-2068`. If there are several files that raise the same error but through different logic paths, each file has a letter after the number, e.g. `fault-2086c`.
3. The filenames end with the name of the function definition in which the error is raised, e.g. `fault-2086c-ProcessCalc.txt`.

Quality assurance of the program's error messages was made considerably easier by lashing together the script `_error-statements.py`. Scripts like `_error-statements.py` keep programmers honest: that script helps me ensure that every new error I write is sorted into one category (needs a file or files to test it) or another category (has a record of why no test file is needed for this error). `_error-statements.py` is discussed in more detail in the verification & validation document.

```

> *Error* type 2001 ****
> Skipping "fault-2001-ProcessFile.txs", because it
> doesn't end with the extension ".txt".

> *Error* type 2002 ****
> Skipping "fault-2002-ProcessFile.txt", because you
> do not have permission to read it.

> *Error* type 2003 ****
> Skipping "fault-2003-nonexistent.txt" in folder
> "/Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/",
> because it doesn't exist.

> *Error* type 2004 ****
> Skipping "fault-2004-processFile.txt", because it
> is in a folder where you do not have permission
> to create the required "ancillaries" subfolder.

> *Error* type 2005 ****
> Skipping "fault-2005-ProcessFile.txt", because you
> do not have permission to write to its logfile.

> *Error* type 2101 ****
> Came across an unrecognised keyword in "fault-2101a-ProcessBlock.txt".
> The keyword is "rfictiontype". Valid keywords
> are as follows:
>   version, runtype, frictiontype, frictionapprox,
>   units, qa1, qa2, qa3, p_atm, rho_atm, aero_step,
>   aero_time, footer, header, plotnames, images,
>   keytextscales, max_vel, gamma, rise_time, time_accuracy,
>   g, autokeys, solver, min_area and jetfancounts.
> Faulty line of input (9th) is
>   rfictiontype Darcy

> *Error* type 2101 ****
> Came across an unrecognised keyword in "fault-2101b-ProcessBlock.txt".
> The keyword is "dud". Valid keywords
> are as follows:
>   transient, profile, waterfall, property, userdata,
>   icons, fandata, calc, begin, end, title, xlabel,
>   ylabel, x2label, y2label, xrange, yrange,
>   x2range, y2range, margins, verbatim, data
>   and allroutes.
> Faulty line of input (40th) is
>   Dud line of input

> *Error* type 2102 ****
> Came across a faulty setting in "fault-2102-ProcessBlock.txt".
> The setting has too few entries in it
> (expected 2 words).
> Faulty line of input (11th) is
>   frictiontype

> *Error* type 2103 ****

```

```

> Came across a faulty setting in "fault-2103-ProcessBlock.txt".
> The setting has too few entries in it
> (expected 4 words).
> Note that this error can be triggered when
> when there are instances of := (marking the
> optional entries) but not enough keys. For
> what it's worth, here are the words taken
> from the line as optional entries:
>     tbm := 0.3
>     zeta_fb := 0.42
> Faulty line of input (32nd) is
>     change 10060 TBM           := 0.3    zeta_fb := 0.42

> *Error* type 2104 ****
> Came across an unrecognised entry in "fault-2104-ProcessBlock.txt".
> The 1st entry for keyword "runtype" was "fail".
> Valid entries for this are:
>     calc and plot.
> Faulty line of input (9th) is
>     runtype fail

> *Error* type 2105 ****
> Came across a faulty setting in "fault-2105-ProcessBlock.txt".
> The setting has too many entries in it
> (expected 2 words).
> Faulty line of input (10th) is
>     frictiontype Darcy Weisbach

> *Error* type 2106 ****
> Came across a duplicate keyword in "fault-2106-ProcessBlock.txt".
> The keyword "version" has been used already in
> this "settings" block and you must have
> only one of them per block.
> Duplicate lines of input (7th and 8th) are
>     version 1
>     version 1 # Sets the same entry a second time

> *Error* type 2107 ****
> Came across an invalid optional entry in
> "fault-2107a-ProcessBlock.txt".
> The keyword "float" has no valid
> optional entries, but you've set one.
> Please remove it.
> Faulty line of input (17th) is
>     float 42 option_1 := 42

> *Error* type 2107 ****
> Came across an invalid optional entry in
> "fault-2107b-ProcessBlock.txt".
> The keyword "float" has no valid
> optional entries, but you've set 3.
> Please remove them.
> Faulty line of input (17th) is
>     float 42 option_1 := 42, option_2 := 42 option_3 := 42

```

```
> *Error* type 2108 ****
> Came across an invalid optional entry in
> "fault-2108a-ProcessBlock.txt".
> The keyword "name" cannot use the
> optional entry "dud_option", there is one
> valid optional entry for this keyword,
> "option_1".
> Faulty line of input (18th) is
>     name testing dud_option := 42

> *Error* type 2108 ****
> Came across an invalid optional entry in
> "fault-2108b-ProcessBlock.txt".
> The keyword "any" cannot use the
> optional entry "dud_option", the only valid
> optional entries for this keyword are:
> option_1, option_2, tuple1 and tuplemany.
> Faulty line of input (19th) is
>     any 0.0 dud_option := 42

> *Error* type 2109 ****
> Came across an invalid optional entry in
> "fault-2109a-ProcessBlock.txt".
> The optional entry "tuple1" was assigned
> the value "invert". There is one
> valid optional entry for this keyword,
> "reverse".
> Faulty line of input (18th) is
>     any 0.0 tuple1 := invert

> *Error* type 2109 ****
> Came across an invalid optional entry in
> "fault-2109b-ProcessBlock.txt".
> The optional entry "tuplemany" was assigned
> the value "ten". The only valid
> optional entries for this keyword are:
> one, two, three, four, five, six, seven, eight
> and nine.
> Faulty line of input (18th) is
>     any 0.0 tuplemany := ten

> *Error* type 2110 ****
> Came across a tunnel definition
> in "fault-2110-ProcessBlock.txt"
> that lacked a required entry, "fwd".
> Please add a line to define it.
> Faulty line of input (24th) is
>     begin tunnel Dopey

> *Error* type 2111 ****
> Came across a testblock definition
> in "fault-2111-ProcessBlock.txt"
> that lacked one of two required (but
```

```
> mutually exclusive) entries:  
> req1a or req1b.  
> Please add a line to define one of them.  
> The faulty block of input started at the 17th line:  
> begin testblock  
  
> *Error* type 2112 ****  
> Came across a testblock definition  
> in "fault-2112-ProcessBlock.txt" that  
> had too many mutually exclusive entries.  
> The following entries are mutually exclusive:  
> req1a and req1b.  
> Please remove all but one of them.  
> Faulty lines of input (18th and 19th) are  
>     req1a word  
>     req1b word  
  
> *Error* type 2113 ****  
> In the file named "fault-2113a-ProcessBlock.txt"  
> the name of tunnel "o'leary" is  
> not valid because it contains a single  
> quote ('), which is a reserved symbol.  
> Please edit the file to remove it.  
> Faulty line of input (22nd) is  
> begin tunnel O'Leary  
  
> *Error* type 2113 ****  
> In the file named "fault-2113b-ProcessBlock.txt"  
> the name of tunnel ""bob"" is  
> not valid because it contains a double  
> quote ("), which is a reserved symbol.  
> Please edit the file to remove it.  
> Faulty line of input (22nd) is  
> begin tunnel "Bob"  
  
> *Error* type 2113 ****  
> In the file named "fault-2113c-ProcessBlock.txt"  
> the name of route "@kinson" is  
> not valid because it contains an @,  
> which is a reserved symbol.  
> Please edit the file to remove it.  
> Faulty line of input (27th) is  
> begin route @kinson  
  
> *Error* type 2121 ****  
> There is no setting for input file version number in  
> the file "fault-2121-ProcessSettings.txt".  
> Please add "version 1" (without double quotes) to  
> the settings block (it starts on the 6th line).  
  
> *Error* type 2122 ****  
> There is no setting for the run type in the file  
> "fault-2122-ProcessSettings.txt".  
> Please add "runtype plot" or "runtype calc" (without
```

```
> double quotes) to the settings block (it starts on
> the 7th line).

> *Error* type 2123 *****
> There is no setting for friction type in the file
> "fault-2123-ProcessSettings.txt".
> Please add "frictiontype Darcy", "frictiontype Fanning"
> or "frictiontype Atkinson" (without double quotes) to
> the settings block (it starts on the 7th line).
> You must make a choice.
> If you are unsure which to use (or were unaware that
> there is more than one type of friction factor) then
> please read "friction-rant.pdf" in Hobyah's
> documentation folder.

> *Error* type 2124 *****
> There is no setting for the aero timestep in the
> file "fault-2124-ProcessSettings.txt".
> In files that run a calculation an aero timestep
> must be set. Something like "aero_step 0.05" to
> set a timestep in seconds. Please add one to the
> "settings" block (it starts on the 6th line).
> Alternatively, change the runtype to "plot" in
> the "settings" block so that the calculation is
> skipped.

> *Error* type 2125 *****
> The aero timestep in the file "fault-2125-ProcessSettings.txt"
> is too high (over 0.25 seconds).
> Please set a value less than that (a typical
> range is 0.005 to 0.1 seconds).
> Faulty line of input (13th) is
>     aero_step 0.27

> *Error* type 2126 *****
> There is no setting for the aero runtime in the
> file "fault-2126-ProcessSettings.txt".
> In files that run a calculation an aero runtime
> must be set. Something like "aero_time 700" to
> set the run duration in seconds. Please add one
> to the "settings" block (it starts on the 6th line).
> Alternatively, change the runtype to "plot" in
> the "settings" block so that the calculation is
> skipped.

> *Error* type 2141 *****
> The "begin files" block in "fault-2141-ProcessPlotFiles.txt"
> has been given the type "misspelled".
> Only "numbered", "nicknames" or "2syllables" are
> permitted here.
> Faulty line of input (16th) is
>     begin files misspelled

> *Error* type 2142 *****
```

```

> There is an invalid entry in a "begin files"
> block in "fault-2142-ProcessPlotFiles.txt".
> The file "Incorrect-extension.bin" in the
> "files" block has an incorrect extension or
> no extension. Filenames must end in ".hbn"
> or ".sbn" (signifying a Hobyah binary file or
> an SESconv binary file respectively).
> Faulty line of input (17th) is
>     Incorrect-extension.bin

> *Error* type 2143 *****
> The "files" block in "fault-2143-ProcessPlotFiles.txt"
> has been told to generate nicknames from
> two syllables in the file name but there
> are not enough syllables available in the
> file named "too-few.hbn".
> Please either give the file name three
> or more syllables separated by dashes (a
> name like "WGT-012-fire-12080ab.txt" has
> four syllables ("WGT", "012", "fire" and
> "12080ab").
> Faulty line of input (18th) is
>     too-few.hbn

> *Error* type 2144 *****
> The "files" block in "fault-2144-ProcessPlotFiles.txt"
> has been told to generate one-word nicknames
> from two syllables in the file name. The
> syllables in the nickname "2 nd-5th"
> (generated from "1st-2 nd-3rd-4th-5th.hbn"
> has spaces in it, which is not allowed.
> Please rename the file or use a different
> naming scheme.
> Faulty line of input (17th) is
>     1st-2 nd-3rd-4th-5th.hbn

> *Error* type 2145 *****
> The "files" block in "fault-2145-ProcessPlotFiles.txt"
> has been told to read a nickname and a
> file name, but one of the entries had
> nothing but the nickname, "Fake-output-file2.hbn".
> Please add a file name or choose another
> way of generating nicknames.
> Faulty line of input (20th) is
>     Fake-output-file2.hbn

> *Error* type 2146 *****
> The "files" block in "fault-2146-ProcessPlotFiles.txt"
> has been told to read a nickname and a
> file name. One of the nicknames was
> "calc", which is not allowed because
> it is a word reserved for other uses.
> Please choose a different word.
> Faulty line of input (16th) is

```

```
>      calc      SES-041-fire-1.sbn

> *Error* type 2147 ****
> The "files" block in "fault-2147-ProcessPlotFiles.txt"
> has duplicate nicknames: "compressible"
> appears twice (note that nicknames are not
> case sensitive). Please rename one of them.
> Faulty line of input (17th) is
>      compressible SES-041-fire-1.sbn

> *Error* type 2148 ****
> Came across a file name that consisted of
> nothing but the extension ".hbn" in the
> "files" block of "fault-2148-ProcessPlotFiles.txt".
> Please give it a name.
> Faulty line of input (16th) is
>      .hbn

> *Error* type 2149 ****
> One of the binary output files in the "files"
> block of "fault-2149-ProcessPlotFiles.txt" doesn't exist
> in the same folder as the input file.
> The missing file is "Fake-output-file3.hbn".
> Please either run "Fake-output-file3.txt"
> and copy the binary to the correct folder.
> Alternatively, remove it from the list of files.

> Faulty line of input (16th) is
>      Fake-output-file3.hbn

> *Error* type 2150 ****
> One of the binary output files in the "files"
> block of "fault-2150-ProcessPlotFiles.txt" exists
> but you don't have permission to read it.
> The locked file is "H-2150-output-file-locked.hbn".
> Please either change its permissions or remove
> it from the list of files and remove any
> references to it in your plots.
> Faulty line of input (16th) is
>      H-2150-output-file-locked.hbn

> *Error* type 2181 ****
> Came across a faulty line of input in
> "fault-2181-ProcessConstants.txt".
> The line set a constant ("*dud_name").
> This contains a "*", which is not allowed
> because it fouls up the syntax for defining
> lists and the syntax of the "plots" block.
> Please edit the file to rename the constant.
> Faulty line of input (18th) is
>      *dud_name 12.5

> *Error* type 2182 ****
> Came across a faulty line of input in
```

```
> "fault-2182-ProcessConstants.txt".
> The line set a constant ("dud,name").
> This contains a comma, which is not allowed
> (it fouls up the syntax of the "data" blocks,
> which use commas to separate the entries).
> Please edit the file to rename the constant.
> Faulty line of input (19th) is
>     dud,name 12.5

> *Error* type 2183 ****
> Came across a faulty line of input in
> "fault-2183a-ProcessConstants.txt".
> The line set a constant with the name "duration".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
> Faulty line of input (18th) is
>     duration 4

> *Error* type 2183 ****
> Came across a faulty line of input in
> "fault-2183b-ProcessConstants.txt".
> The line set a constant with the name "range".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
> Faulty line of input (18th) is
>     range 4

> *Error* type 2183 ****
> Came across a faulty line of input in
> "fault-2183c-ProcessConstants.txt".
> The line set a constant with the name "startstepcount".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
> Faulty line of input (18th) is
>     startstepcount 4

> *Error* type 2183 ****
> Came across a faulty line of input in
> "fault-2183d-ProcessConstants.txt".
> The line set a constant with the name "startstopcount".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
```

```
> Faulty line of input (18th) is
>      startstopcount      4

> *Error* type 2183 *****
> Came across a faulty line of input in
> "fault-2183e-ProcessConstants.txt".
> The line set a constant with the name "up_ptl".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
> Faulty line of input (18th) is
>      up_ptl      4

> *Error* type 2183 *****
> Came across a faulty line of input in
> "fault-2183f-ProcessConstants.txt".
> The line set a constant with the name "down_ptl".
> This is a reserved word that cannot be used
> as the name of one of your constants because
> it is the name of an internal constant.
> Please edit the file to rename the constant
> to something else.
> Faulty line of input (18th) is
>      down_ptl      4

> *Error* type 2184 *****
> Came across a faulty line of input in
> "fault-2184-ProcessConstants.txt".
> The line set a constant named "2".
> The name of the constant was a number, which
> is not allowed. It is way too confusing for
> users if you are allowed to define a constant
> named "2" and give it the value 4 (which is
> what you just tried to do).
>
> If you wrote a file specifically to see what
> would happen if you did this, then I doff my hat
> to you. Full marks for sneakiness!
> Now, please edit the file to rename your constant
> so that its name can't be construed as a number.
> Faulty line of input (19th) is
>      2      4

> *Error* type 2201 *****
> In the file named "fault-2201-ProcessTunnel.txt"
> the name of the sectype at the back end
> of the tunnel named "dopey" does not
> exist (the name given was "BLT").
> Please edit the file to correct it. For
> what it's worth, here are the names of
> the sectype(s) you've set:
>      tbm.
```

```

> Faulty line of input (22nd) is
>     back 10000 portal 20  BLT      # incorrect section name

> *Error* type 2202 ****
> In the file named "fault-2202-ProcessTunnel.txt"
> the name of the sectype in a change of
> sectype in tunnel "dopey" does not
> exist (the name given was "blt").
> Please edit the file to correct it. For
> what it's worth, here are the names of
> the sectype(s) you've set:
>     tbm.

> Faulty line of input (23rd) is
>     change 14060    BLT      # wrong name of new section

> *Error* type 2203 ****
> In the file named "fault-2203-ProcessTunnel.txt"
> the name of tunnel "-dopey" is not
> valid because it starts with a "-", which
> is a reserved character for routes.
> Please edit the file to correct it.

> Faulty line of input (21st) is
>     begin tunnel -Dopey

> *Error* type 2204 ****
> In the file named "fault-2204-ProcessTunnel.txt"
> the tunnel named "dopey" has its
> back end located above its forward end,
> which is not allowed.
> Please edit the file to correct it.

> Faulty lines of input (22nd and 23rd) are
>     back 15000 portal 20  TBM
>     fwd  12000 portal 0

> *Error* type 2205 ****
> In the file named "fault-2205-ProcessTunnel.txt",
> the tunnel named "dopey" had an
> entity ("change") at 14060.0 that is
> not inside tunnel (which goes from
> 10000 to 12130).
> Please edit the file to correct it.

> Faulty line of input (23rd) is
>     change 14060    TBM

> *Error* type 2206 ****
> In the file named "fault-2206-ProcessTunnel.txt"
> tunnel "medium-hadron-collider" started and ended at
> the same node (named "samenode"). This is not
> allowed. Please edit the file to correct this,
> either by correcting the node names or splitting
> the tunnel in two.

> Faulty lines of input (22nd and 23rd) are
>     back  0 node samenode MHC
>     fwd  9000 node samenode      # It's one-third of the diameter of the LHC

```

```
> *Error* type 2207 ****
> In the file named "fault-2207-ProcessTunnel.txt"
> there were two joins with the same name in
> tunnel "westbound1", both called "xp1wb".
> Please edit the file to distinguish between
> them.
> Faulty lines of input (39th and 40th) are
>     join 10100    XP1wb
>     join 10120    XP1wb

> *Error* type 2208 ****
> In the file named "fault-2208-ProcessTunnel.txt"
> there were two joins with the same name in
> tunnel "westbound1", both called "xp5wb".
> Please edit the file to distinguish between
> them. Note that one or both may have been
> generated by a "joins" command, so it may not
> be immediately obvious how the conflict occurred.
> Faulty lines of input (43rd and 44th) are
>     joins "[10240] + startstepcount(10360, 120, 2)"  XP*wb  start:=3
>     joins startstepcount(10580,120,3)  XP*wb  start:=5

> *Error* type 2209 ****
> Came across two entities that are at the
> same location in "fault-2209a-ProcessTunnel.txt".
> Entities can't be closer than one metre.
> Faulty lines of input (27th and 28th) are
>     change 11380 TBM
>     change 11380 East-C+C

> *Error* type 2209 ****
> Came across two entities that are at the
> same location in "fault-2209b-ProcessTunnel.txt".
> Entities can't be closer than 3.28 feet.
> Faulty lines of input (28th and 29th) are
>     change 11380 TBM
>     change 11380 East-C+C

> *Error* type 2210 ****
> Came across two entities that are too close
> in "fault-2210a-ProcessTunnel.txt".
> Entities can't be closer than one metre.
> Faulty lines of input (26th and 28th) are
>     back 10000 portal 0 West-C+C
>     change 10001 TBM

> *Error* type 2210 ****
> Came across two entities that are too close
> in "fault-2210b-ProcessTunnel.txt".
> Entities can't be closer than 3.28 feet.
> Faulty lines of input (28th and 30th) are
>     back 10000 portal 0 West-C+C
>     change 10002 TBM
```

```

> *Error* type 2221 ****
> Came across location identifier that was not
> a valid constant in "fault-2221-CheckRangeAndSI.txt".
> The constant named "*time" is a special one that
> can only be used inside timeloops. You probably
> copied and pasted the graph from a loop definition
> into a page definition. Please change all the
> instances of "*time" to a number or to the name
> of a constant in the "constants" block.
> Faulty line of input (48th) is
>         profile vSES slugflow route1@*time

> *Error* type 2222 ****
> Came across a version number that was not
> an integer in "fault-2222a-CheckRangeAndSI.txt".
> The 1st entry for keyword "version" was "1.2".
> The only valid entries there are integers.
> Faulty line of input (7th) is
>         version 1.2 # Should be an integer

> *Error* type 2222 ****
> Came across any number (testblock) that was not
> an integer in "fault-2222b-CheckRangeAndSI.txt".
> The 1st entry for keyword "int" was "real_neg",
> referring to a constant of that name which
> was not an integer. The only valid entries
> there are integers or integer constants.
> Faulty lines of input (20th and 26th) are
>         real_neg -1.2
>         int real_neg # Should be an integer

> *Error* type 2223 ****
> Came across outside air density that was not
> a real number in "fault-2223a-CheckRangeAndSI.txt".
> The 1st entry for keyword "rho_atm" was "1.2.3"
> The only valid entries there are real numbers
> and constants, and there is not a constant named
> "1.2.3".
> Faulty line of input (13th) is
>         rho_atm 1.2.3 # Should be a number

> *Error* type 2223 ****
> Came across a multiplier on Y that was not
> a real number in "fault-2223b-CheckRangeAndSI.txt".
> The optional entry "ymult" was assigned
> the value "test_constant",
> referring to a constant of that name which
> was not a number. The only valid entries
> there are real numbers or constants (there
> are no constants defined).
> The following constants are defined:
>     test_constant.
> Faulty lines of input (17th and 29th) are

```

```

>      test_constant [1, 2, 3]
>          profile vcold calc route1@time1 ymult:=test_constant

> *Error* type 2223 ****
> Came across a multiplier on Y that was not
> a real number in "fault-2223c-CheckRangeAndSI.txt".
> The optional entry "ymult" was assigned
> the value "[1,"
> The only valid entries there are real numbers
> and constants, and there is not a constant named
> "[1,".
> Faulty line of input (25th) is
>      profile vcold calc route1@time1 ymult := [1, 2, 3]

> *Error* type 2224 ****
> Came across a number (testblock) that should
> have been negative but was not, in
> "fault-2224a-CheckRangeAndSI.txt"
> The 1st entry for keyword "--" was "0".
> The only valid entries there are negative numbers.
> Faulty line of input (16th) is
>      - 0  # Should be a negative number

> *Error* type 2224 ****
> Came across a number (testblock) that should
> have been negative but was not, in
> "fault-2224b-CheckRangeAndSI.txt"
> The 1st entry for keyword "--" was "zero",
> referring to a constant of that name which
> was not negative.
> Faulty lines of input (26th and 19th) are
>      - zero
>      zero 0

> *Error* type 2224 ****
> Came across a number (testblock) that should
> have been negative but was not, in
> "fault-2224c-CheckRangeAndSI.txt"
> The 1st entry for keyword "--" was "real_pos",
> referring to a constant of that name which
> was not negative.
> Faulty lines of input (26th and 21st) are
>      - real_pos  # Should be a negative number
>      real_pos 1.

> *Error* type 2225 ****
> Came across a number (testblock) that should
> have been negative or zero but was not, in
> "fault-2225a-CheckRangeAndSI.txt".
> The 1st entry for keyword "-0" was "2".
> The only valid entries there are negative numbers
> or zero.
> Faulty line of input (18th) is
>      -0  2  # Should be a negative number or zero

```

```

> *Error* type 2225 ****
> Came across a number (testblock) that should
> have been negative or zero but was not, in
> "fault-2225b-CheckRangeAndSI.txt".
> The 1st entry for keyword "-0" was "real_pos",
> referring to a constant of that name which
> was positive.
> Faulty lines of input (21st and 27th) are
>     real_pos 1.
>     -0    real_pos

> *Error* type 2226 ****
> Came across a number (testblock) that should
> have been positive or zero but was not, in
> "fault-2226a-CheckRangeAndSI.txt".
> The 1st entry for keyword "0+" was "-1".
> The only valid entries there are positive numbers
> or zero.
> Faulty line of input (18th) is
>     0+    -1    # Should be a positive number or zero

> *Error* type 2226 ****
> Came across a number (testblock) that should
> have been positive or zero but was not, in
> "fault-2226b-CheckRangeAndSI.txt".
> The 1st entry for keyword "0+" was "real_neg",
> referring to a constant of that name which
> was negative.
> Faulty lines of input (20th and 27th) are
>     real_neg -1.
>     0+    real_neg

> *Error* type 2227 ****
> Came across a number (testblock) that should
> have been positive but was not, in
> "fault-2227a-CheckRangeAndSI.txt".
> The 1st entry for keyword "+" was "0".
> The only valid entries there are positive numbers
> (not negative numbers or zero).
> Faulty line of input (16th) is
>     +    0    # Should be a positive number

> *Error* type 2227 ****
> Came across a number (testblock) that should
> have been positive but was not, in
> "fault-2227b-CheckRangeAndSI.txt".
> The 1st entry for keyword "+" was "-2".
> The only valid entries there are positive numbers
> (not negative numbers or zero).
> Faulty line of input (16th) is
>     +    -2    # Should be a positive number

> *Error* type 2227 ****

```

```
> Came across a number (testblock) that should
> have been positive but was not, in
> "fault-2227c-CheckRangeAndSI.txt".
> The 1st entry for keyword "+" was "real_neg",
> referring to a constant of that name which
> was not positive.
> Faulty lines of input (20th and 26th) are
>     real_neg -1.
>     + real_neg

> *Error* type 2241 ****
> The "invalid" datasource in "fault-2241-ProcessUserData.txt"
> started with a mixture of column names, blank
> entries, numbers and the names of constants.
> Please edit the file to either remove the
> column names or remove the numbers, blanks
> and names of constants. For what it's worth
> the following named constants were used:
>     my_const.
> Faulty line of input (30th) is
>     my_const, 1.0, , col4_name, col5_name

> *Error* type 2242 ****
> The "invalid" datasource in "fault-2242-ProcessUserData.txt"
> started with a mixture of column names, numbers
> and the names of constants. Please edit the
> file to either remove the column names or
> remove the numbers and names of constants. For
> what it's worth the following named constants
> were used:
>     my_const.
> Faulty line of input (30th) is
>     My_Const, 1.0, col3_name, col4_name, col5_name

> *Error* type 2243 ****
> The "invalid" datasource in "fault-2243-ProcessUserData.txt"
> started with a mixture of column names, numbers
> and blank entries. Please edit the file to
> either remove the column names or remove the
> numbers and blank entries.

> Faulty line of input (29th) is
>     43.5, , col3_name, col4_name, col5_name

> *Error* type 2244 ****
> The "invalid" datasource in "fault-2244-ProcessUserData.txt"
> started with a mixture of column names, blanks
> and the names of constants. Please edit the
> file to either remove the column names or
> remove the blanks and names of constants. For
> what it's worth the following named constants
> were used:
>     my_const.
> Faulty line of input (30th) is
```

```
>      my_const,      ,    col3_name,  col4_name,  col5_name

> *Error* type 2245 ****
> The "invalid" datasource in "fault-2245-ProcessUserData.txt"
> started with a mixture of column names and the
> names of constants. Please edit the file to
> either remove the column names or remove the
> names of constants. For what it's worth the
> following named constants were used:
>   my_const1, my_const2 and my_const3.
> Faulty line of input (32nd) is
>   my_const1,  my_const2,  my_const3,  col4_name

> *Error* type 2246 ****
> The "invalid" datasource in "fault-2246-ProcessUserData.txt"
> started with a mixture of column names and
> numbers. Please edit the file to either
> remove the column names or remove the numbers.
> Faulty line of input (25th) is
>   col1_name,  3.5,    col3_name,  col4_name

> *Error* type 2247 ****
> The "invalid" datasource in "fault-2247-ProcessUserData.txt"
> started with a mixture of column names and
> blank entries. Please edit the file to fill
> in the blank entries, because if one column
> is given a name they all must be given a name.
> Faulty line of input (25th) is
>   col1_name,      ,    col3_name,  col4_name

> *Error* type 2248 ****
> The "invalid" datasource in "fault-2248a-ProcessUserData.txt"
> had a column name that contained a space.
> Please edit the name "col2 name" to remove
> it, as column names cannot contain spaces.
> Remember that column names must be separated
> by commas (just like the numbers are).
> Faulty line of input (29th) is
>   col1_name,  col2 name,    col3_name,  col4_name

> *Error* type 2248 ****
> The "invalid" datasource in "fault-2248b-ProcessUserData.txt"
> had a column name that contained spaces.
> Please edit the name "col2 name" to remove
> them, as column names cannot contain spaces.
> Remember that column names must be separated
> by commas (just like the numbers are).
> Faulty line of input (29th) is
>   col1_name,  col2 name,    col3_name,  col4_name

> *Error* type 2249 ****
> The "invalid" datasource in "fault-2249-ProcessUserData.txt"
> had a column name that was a duplicate ("test_DP"),
> it appears at the top of the 2nd and 3rd columns.
```

```
> Faulty line of input (24th) is
>     time,      test_DP,    test_DP

> *Error* type 2250 ****
> The "invalid" datasource in "fault-2250-ProcessUserData.txt"
> had a line of entry that contained more entries
> (4 entries) than the first line (3 entries).
> The first line of a data block sets its maximum
> width. Please edit the file to either shorten
> this line or lengthen the header (the 16th line).
> Faulty line of input (27th) is
>     0.2,      340.2,    361.2

> *Error* type 2251 ****
> The "invalid" datasource in "fault-2251-ProcessUserData.txt"
> had an entry that was not a blank entry, a
> number or the name of a constant. It was
> "dud_entry" instead. Please edit the file
> to replace it with something more appropriate.
> Keep in mind that numbers must be separated
> by commas.
> Faulty line of input (31st) is
>     0.2,      a_constant,  dud_entry

> *Error* type 2261 ****
> The "csv" block in "fault-2261-ProcessUserData.txt"
> has been told to read a nickname and a
> file name, but one of the entries had
> nothing but the nickname "oneword.csv".
> Please add a file name after the nickname.
> If the nickname is actually a file name
> then please add a nickname before it.
> Faulty line of input (19th) is
>     oneword.csv

> *Error* type 2262 ****
> There is an invalid entry in a "begin csv"
> block in "fault-2262-ProcessUserData.txt".
> The file "1976-G-Fig-3a.doc" has an
> incorrect extension or no extension. The
> names of CSV files must end in ".csv" or ".txt".
> Faulty line of input (18th) is
>     invalid  1976-G-Fig-3a.doc

> *Error* type 2263 ****
> The "csv" block in "fault-2263-ProcessUserData.txt"
> has been told to read a nickname and a
> file name. One of the nicknames was
> "xrange", which is not allowed because
> it is a word reserved for other uses.
> Please choose a different word.
> Faulty line of input (16th) is
>     xrange  1976-G-Fig-3a-nose-ptot.csv
```

```

> *Error* type 2264 ****
> The "csv" block in "fault-2264-ProcessUserData.txt"
> has duplicate nicknames: "file1"
> appears twice (note that nicknames are not
> case sensitive). Please rename one of them.
> Faulty lines of input (16th and 17th) are
>   file1 1976-G-Fig-3a-nose-ptot.csv
>   file1 1976-G-Fig-3b-tail-pstat.csv

> *Error* type 2265 ****
> Came across a nickname for a .csv file that
> has already been used as the nickname for a
> "begin data...end data" block. The duplicate
> nickname is "untitled_DATA" in the file
> "fault-2265-ProcessUserData.txt".
> Please change one nickname or the other.
> Faulty line of input (26th) is
>   untitled_DATA 1976-G-Fig-3a-nose-ptot.csv

> *Error* type 2266 ****
> Came across a file name that consisted of
> nothing but the extension ".csv" in the
> "csv" block of "fault-2266-ProcessUserData.txt".
> Please give it a name.
> Faulty line of input (16th) is
>   file1 .CSV

> *Error* type 2267 ****
> One of the .csv files in the "csv" block
> of "fault-2267-ProcessUserData.txt" doesn't exist.
> The missing file is "1976-G-Fig-3b-tail-pstat.csv".
> Please edit the file name to correct it or
> remove it from the list of files.
> Faulty line of input (16th) is
>   file1 1976-G-Fig-3b-tail-pstat.csv

> *Error* type 2268 ****
> One of the .csv files in the "csv" block
> of "fault-2268-ProcessUserData.txt" exists but.
> but you don't have permission to read it.
> The locked file is "locked-file.csv".
> Please remove it from the list of files.
> Faulty line of input (16th) is
>   file1 locked-file.csv

> *Error* type 2281 ****
> The animation on loop 1 of "fault-2281-ProcessTimeLoop.txt"
> has a start time that is later than its stop
> time. Please alter the settings so that the
> start time is equal to or before the stop time.
> Faulty lines of input (22nd and 23rd) are
>   start 20
>   stop 10

```

```
> *Error* type 2282 ****
> The animation on loop 1 of "fault-2282-ProcessTimeLoop.txt"
> has a step time that is so close to zero that it is
> considered to be zero. Please alter the step time to
> make it above 1 nanosecond.
> Faulty line of input (24th) is
>     step    0.00000001

> *Error* type 2301 ****
> In the file named "fault-2301-ProcessRoute.txt"
> there was a route named "mainline2", which
> has the same name as a tunnel. Please edit
> the file to change the name of the route.
> Alternatively, change the name of the tunnel.
> Faulty line of input (36th) is
>     begin route Mainline2

> *Error* type 2302 ****
> In the file named "fault-2302-ProcessRoute.txt"
> the route named "through" refers to a
> tunnel that does not exist (the name
> given was "mainline").
> Please edit the file to correct it. For
> what it's worth, here are the names of
> the tunnel(s) you've created:
>     mainline1, mainline2 and offslip.
> Faulty line of input (41st) is
>     Mainline

> *Error* type 2303 ****
> In the file named "fault-2303-ProcessRoute.txt"
> tunnel "mainline1" appears more than
> once in the route named "through".
> Please edit the file to remove the duplicate.
> Faulty line of input (40th) is
>     Mainline1    Mainline2    Mainline1

> *Error* type 2304 ****
> In the file named "fault-2304-ProcessRoute.txt"
> the route named "through" refers to a
> tunnel named "origin". That name is not
> allowed because it leads to a confusing
> state of affairs where "origin" could be
> treated as both a route keyword and as a
> tunnel name.
> Please edit the file to move that line before
> "begin tunnels" or after "end tunnels" and
> rename the tunnel you created called "origin"
> Faulty line of input (41st) is
>     origin    13204

> *Error* type 2305 ****
> In the file named "fault-2305a-ProcessRoute.txt"
> the route named "through" had a set
```

```
> of elevations and a set of gradients. This is
> not allowed, you must define the profile by
> using one or the other, not both.
> Please edit the file to delete the gradients
> block or the elevations block.
> Faulty lines of input (44th and 52nd) are
>   begin gradients fractions
>   begin elevations

> *Error* type 2305 *****
> In the file named "fault-2305b-ProcessRoute.txt"
> the route named "through" had a set
> of elevations and a set of gradients. This is
> not allowed, you must define the profile by
> using one or the other, not both.
> Please edit the file to delete the gradients
> block or the elevations block.
> Faulty lines of input (44th and 52nd) are
>   begin elevations
>   begin gradients fractions

> *Error* type 2306 *****
> Came across faulty input in "fault-2306a-ProcessRoute.txt".
> In route "through" there was an impossible
> gradient (-1.01). Gradients are limited to
> +1 (vertical up) to -1 (vertical down).
> Please edit the file and get the absolute
> value of the gradient under 1. Note that
> gradients in this block are expressed as
> fractions (-1 to +1) not as percentages
> (-100 to +100) because you started the
> block with "begin gradients fractions"
> instead of "begin gradients percentages".
> Faulty line of input (50th) is
>           -1.01    11050

> *Error* type 2306 *****
> Came across faulty input in "fault-2306b-ProcessRoute.txt".
> In route "through" there was an impossible
> gradient (1.02). Gradients are limited to
> +1 (vertical up) to -1 (vertical down).
> Please edit the file and get the absolute
> value of the gradient under 1. Note that
> gradients in this block are expressed as
> fractions (-1 to +1) not as percentages
> (-100 to +100) because you started the
> block with "begin gradients fractions"
> instead of "begin gradients percentages".
> Faulty line of input (50th) is
>           1.02    11050

> *Error* type 2307 *****
> Came across faulty input in "fault-2307a-ProcessRoute.txt".
> In route "through" there was a
```

```
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation -629.7 at chainage 11800.0 and
>   * elevation -639.711 at chainage 11810.0
> These combine to give an impossible value of
> gradient (-1.0011).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty line of input (55th) is
>           11800 -629.7  11810 -639.711

> *Error* type 2307 ****
> Came across faulty input in "fault-2307b-ProcessRoute.txt".
> In route "through" there was a
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation -629.7 at chainage 11800.0 and
>   * elevation -639.711 at chainage 11810.0
> These combine to give an impossible value of
> gradient (-1.0011).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty lines of input (55th and 56th) are
>           11800 -629.7  11810
>           -639.711

> *Error* type 2307 ****
> Came across faulty input in "fault-2307c-ProcessRoute.txt".
> In route "through" there was a
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation -629.7 at chainage 11800.0 and
>   * elevation -639.711 at chainage 11810.0
> These combine to give an impossible value of
> gradient (-1.0011).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty lines of input (55th, 56th and 57th) are
>           11800
>           -629.7
>           11810 -639.711

> *Error* type 2307 ****
> Came across faulty input in "fault-2307d-ProcessRoute.txt".
> In route "through" there was a
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation -629.7 at chainage 11800.0 and
>   * elevation -639.711 at chainage 11810.0
> These combine to give an impossible value of
```

```

> gradient (-1.0011).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty lines of input (55th, 56th and 57th) are
>      11800  -629.7
>      11810
>      -639.711

> *Error* type 2307 ****
> Came across faulty input in "fault-2307e-ProcessRoute.txt".
> In route "through" there was a
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation -629.7 at chainage 11800.0 and
>   * elevation -639.711 at chainage 11810.0
> These combine to give an impossible value of
> gradient (-1.0011).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty lines of input (55th, 56th, 57th and 58th) are
>      11800
>      -629.7
>      11810
>      -639.711

> *Error* type 2307 ****
> Came across faulty input in "fault-2307f-ProcessRoute.txt".
> In route "through" there was a
> "begin elevations" block. The difference
> in height between two points exceeded their
> distance. Details are:
>   * elevation 21.3 at chainage 9000.0 and
>   * elevation 31.5 at chainage 9010.0
> These combine to give an impossible value of
> gradient (1.02).
> Please edit the file to get the absolute
> value of the gradient to be below 1.
> Faulty lines of input (44th, 51st and 52nd) are
>      origin 9000 elevation:= 21.3
>      9010
>      31.5

> *Error* type 2308 ****
> Came across faulty input in "fault-2308a-ProcessRoute.txt".
> In route "atkinson" there was a line starting
> with an unrecognised keyword, "tunnel". Note that
> trying a "tunnel" keyword was a good guess, but
> you need to use a "begin tunnels...end tunnels"
> sub-block in the route definition.> Please edit the file to correct or remove the
> entry.
> Faulty line of input (32nd) is
>      tunnel first

```

```
> *Error* type 2308 ****
> Came across faulty input in "fault-2308b-ProcessRoute.txt".
> In route "atkinson" there was a line starting
> with an unrecognised keyword, "keyword".
> Please edit the file to correct or remove the
> entry.
> Faulty line of input (29th) is
>     keyword

> *Error* type 2309 ****
> Came across missing input in "fault-2309-ProcessRoute.txt".
> In route "atkinson" there was no
> "begin tunnels...end tunnels" block, which
> much makes the route pretty much useless.
> Please edit the file to remove the route, or
> add a "begin tunnels...end tunnels" block
> to the route definition.
> Relevant line of input (27th) is
>     begin route Atkinson

> *Error* type 2310 ****
> Came across faulty input in "fault-2310a-ProcessRoute.txt".
> In route "trainpassage", train schedule "toofarup"
> starts trains at -501 m, which is before the
> route origin. Please change it so that the trains
> start at or between -500 m and 2140 m.
> Faulty line of input (41st) is
>     downstart -501.

> *Error* type 2310 ****
> Came across faulty input in "fault-2310b-ProcessRoute.txt".
> In route "trainpassage", train schedule "toofarup"
> starts trains at -501 ft, which is before the
> route origin. Please change it so that the trains
> start at or between -500 ft and 4420.84 ft.
> Faulty line of input (42nd) is
>     downstart -501.

> *Error* type 2311 ****
> Came across faulty input in "fault-2311a-ProcessRoute.txt".
> In route "trainpassage", train schedule "toofarup"
> starts trains at 11000 m, which is so far
> down the route that the alignment is undefined.
> Please change it so that the trains start at
> or between -500 m and 2140 m.
> Faulty line of input (41st) is
>     downstart 11000.

> *Error* type 2311 ****
> Came across faulty input in "fault-2311b-ProcessRoute.txt".
> In route "trainpassage", train schedule "toofarup"
> starts trains at 11000 ft, which is so far
> down the route that the alignment is undefined.
> Please change it so that the trains start at
```

```

> or between -500 ft and 4420.84 ft.
> Faulty line of input (42nd) is
>      downstart 11000.

> *Error* type 2312 ****
> Came across faulty input in "fault-2312a-ProcessRoute.txt".
> In route "testroute", the list of tunnels
> included two tunnels that do not have a common
> node (tunnels "first" and "second").
> Please either edit the list of tunnels to give
> tunnels with a common node or edit the definitions
> of the tunnel ends to give them a common node.
> Relevant lines of input (39th, 28th and 32nd) are
>      begin tunnels
>      fwd    100   node  my_node
>      back    0     NODE  other_node  raitunnel

> *Error* type 2312 ****
> Came across faulty input in "fault-2312b-ProcessRoute.txt".
> In route "testroute", the list of tunnels
> included two tunnels that do not have a common
> node (tunnels "second" and "third").
> Please either edit the list of tunnels to give
> tunnels with a common node or edit the definitions
> of the tunnel ends to give them a common node.
> Relevant lines of input (44th, 32nd, 33rd and 37th) are
>      begin tunnels
>      back    0     NODE  my_node  raitunnel
>      fwd    100   node  other_node
>      back    0     node  different_node  raitunnel

> *Error* type 2312 ****
> Came across faulty input in "fault-2312c-ProcessRoute.txt".
> In route "testroute", the list of tunnels
> included two tunnels that do not have a common
> node (tunnels "second" and "third").
> Please either edit the list of tunnels to give
> tunnels with a common node or edit the definitions
> of the tunnel ends to give them a common node.
> Relevant lines of input (49th, 32nd, 33rd, 37th and 38th) are
>      begin tunnels
>      back    0     NODE  My_node  raitunnel
>      fwd    100   node  other_node
>      back    0     node  last_node  raitunnel
>      fwd    100   node  different_node

> *Error* type 2313 ****
> Came across faulty input in "fault-2313-ProcessRoute.txt".
> In route "testroute", the list of tunnels
> included two tunnels that have two pairs of common
> nodes (tunnels "second" and "third").
> This is not allowed because it is too difficult
> to figure out which way round they are in the
> route. Please either edit the list of tunnels

```

```

> to give tunnels with one common node, or edit the
> definitions of the tunnel ends to give them one
> common node.

> Relevant lines of input (41st, 29th, 30th, 34th and 35th) are
>     begin tunnels
>     back      0   NODE  My_node   railtunnel
>     fwd       100  node  other_node
>     back      0   node  other_node   railtunnel
>     fwd       100  node  my_node

> *Error* type 2314 *****
> Came across faulty input in "fault-2314-ProcessRoute.txt".
> In route "wb" there was a
> block of speed limits, one of which was
> zero or negative. Please edit the file
> to set all speed limits above zero.
> Faulty line of input (42nd) is
>     80      11900      0  14000

> *Error* type 2315 *****
> In the file named "fault-2315-ProcessRoute.txt"
> the route named "wb" had a
> "gradients" sub-block, but the line that
> started the block did not specify whether
> the gradients were given as fractions or
> percentages, it was just "begin gradients"
> If the numbers that specify your gradients
> are percentages (-100 to +100), then change
> the line to "begin gradients percentages".
> If the numbers that specify your gradients
> are fractions (-1 to +1), then change the
> line to "begin gradients fractions".
> If you don't know whether your gradients are
> percentages or fractions, you should probably
> stop using the program until you figure it
> out, as you have bigger problems.
> Faulty line of input (34th) is
>     begin gradients

> *Error* type 2316 *****
> In the file named "fault-2316-ProcessRoute.txt"
> the route named "wb" had a
> "gradients" sub-block, but the line that
> started the block did not specify whether
> the gradients were given as fractions or
> percentages, it was "begin gradients undefined".
> If the numbers that specify your gradients
> are percentages (-100 to +100), then change
> the line to "begin gradients percentages".
> If the numbers that specify your gradients
> are fractions (-1 to +1), then change the
> line to "begin gradients fractions".
> If you don't know whether your gradients are

```

```

> percentages or fractions, you should probably
> stop using the program until you figure that
> out.
> Faulty line of input (33rd) is
>   begin gradients undefined switcheroo:=true # comment

> *Error* type 2341 ****
> Came across faulty input in "fault-2341a-ProcessNumberList.txt".
> A list of data pairs was given in which one
> of the chainages was not a number or the
> name of a constant, it was "10000**".
> Please edit the file to correct the mistake.
> Faulty line of input (26th) is
>   9500    31.3    10000**    6.3    10900   -29.7

> *Error* type 2341 ****
> Came across faulty input in "fault-2341b-ProcessNumberList.txt".
> A list of data pairs was given in which one
> of the elevations was not a number or the
> name of a constant, it was "6.3**".
> Please edit the file to correct the mistake.
> Faulty line of input (26th) is
>   9500    31.3    10000    6.3**    10900   -29.7

> *Error* type 2342 ****
> Came across faulty input in "fault-2342a-ProcessNumberList.txt".
> A list of data pairs was given in which the
> first entry in each pair must be higher than
> the first entry in the previous pair.
> That didn't happen here: there are two entries,
> "11200.0" followed by "10800.0".
> Please edit the file to correct the mistake.
> Faulty line of input (46th) is
>   11050   -31.2    11200   -29.7    10800   -629.7

> *Error* type 2342 ****
> Came across faulty input in "fault-2342b-ProcessNumberList.txt".
> A list of data pairs was given in which the
> first entry in each pair must be higher than
> the first entry in the previous pair.
> That didn't happen here: there are two entries,
> "11200.0" followed by "10800.0".
> Please edit the file to correct the mistake.
> Faulty lines of input (49th and 50th) are
>   11200   -29.7
>   10800   -629.7

> *Error* type 2345 ****
> Came across faulty input in "fault-2345a-ProcessNumberList.txt".
> A list of data pairs was given in which the
> first entry in each pair must be lower than
> or equal to the first entry in the previous pair.
> That didn't happen here: there are two entries,
> "2.5" followed by "3.0".

```

```

> Please edit the file to correct the mistake.
> Faulty line of input (45th) is
>     vehicle    HGV      6.0   1.2    3   10   2.5  20  3 mass := 22.0

> *Error* type 2346 ****
> Came across faulty input in "fault-2346a-ProcessNumberList.txt".
> A list of data pairs was given in which the
> second entry in each pair must be higher than
> the second entry in the previous pair.
> That didn't happen here: there are two entries,
> "0.04" followed by "0.03".
> Please edit the file to correct the mistake.
> Faulty line of input (46th) is
>     speedgradpairs HGV    120  0.04   40  0.03

> *Error* type 2350 ****
> Came across faulty input in "fault-2350-ProcessNumberList.txt".
> A list of data pairs was given in which there
> were no numbers at all, just the "begin gradients"
> and the "end gradients" on adjacent lines.
> Please edit the file to add at least two entries.
> Faulty lines of input (32nd and 35th) are
>     begin gradients fractions switcheroo:=true  # comment
>     end gradients

> *Error* type 2351 ****
> Came across faulty input in "fault-2351-ProcessNumberList.txt".
> A list of data pairs was given in a "gradients" block
> in which there was just one number.
> Please edit the file to make the list of data pairs
> have at least two entries.

> Faulty line of input (35th) is
>     9500

> *Error* type 2361 ****
> Came across faulty input in "fault-2361-CheckGreater.txt".
> The portal chainage must be higher than the
> route origin chainage, and in route "through"
> it was not.
> Please edit the file to correct the mistake.
> Faulty lines of input (39th and 38th) are
>     portal 10000
>     origin 19000

> *Error* type 2363 ****
> Came across a faulty line of input in
> "fault-2363a-CheckForConstant2.txt".
> The line used a constant named "valid_list"
> that was assigned to a list, not a number.
> In the context of this line of input, the
> constant must be assigned to a number but
> this returned "[1.0, 2.0, 3.0]" instead.
> Please edit the file to remove the name

```

```
> "valid_list" from the line or change the
> definition of "valid_list" to be a number.
> Faulty lines of input (20th and 21st) are
>   valid_list [1,2, 3]
>   dud_list [ 4 , 7 , valid_list ]

> *Error* type 2363 *****
> Came across a faulty line of input in
> "fault-2363b-CheckForConstant2.txt".
> The line used a constant named "valid_list"
> that was assigned to a list, not a number.
> In the context of this line of input, the
> constant must be assigned to a number but
> this returned "[1,2, 3]" instead.
> Please edit the file to remove the name
> "valid_list" from the line or change the
> definition of "valid_list" to be a number.
> Faulty lines of input (22nd and 21st) are
>   valid_list [1,2, 3]
>   dud_list [ 4 , 7 , valid_list ]

> *Error* type 2364 *****
> Came across a faulty line of input in
> "fault-2364-CheckForConstant2.txt".
> The line used a value that should have been
> a real number but which was "slartibartfast".
> Please edit the file to correct the entry
> to a number or the name of a constant.
> Faulty line of input (18th) is
>   dud_list [4, OK_name, slartibartfast]

> *Error* type 2365 *****
> Came across a faulty line of input in
> "fault-2365-CheckForConstant2.txt".
> The line used a constant named "float"
> that was assigned to a real number, not
> an integer. In the context of this line
> of input, the constant must return an
> integer, not "3.0".
> Please edit the file to remove the name
> "float" from the line or change the
> the definition of the constant "float"
> to be an integer.
> Faulty lines of input (18th and 19th) are
>   float 3.0
>   dud_list [1, 3, 5] * float

> *Error* type 2366 *****
> Came across a faulty line of input in
> "fault-2366-CheckForConstant2.txt".
> The line used a list multiplier that should
> have been an integer but which was a real
> number (3.0). Please edit the file to
> change the entry to an integer.
```

```
> Faulty line of input (16th) is
>     dud_list [1, 3, 5] * 3.0

> *Error* type 2381 *****
> Came across a faulty line of input in
> "fault-2381-CheckListAndRange.txt".
> The line set an entry that had one or more
> lists in it separated by "+". One of the
> entries was a number (1.1) instead of a list.
> Please edit the file to make the entry all
> lists, or one number, or one constant.
> Faulty line of input (17th) is
>     dud_list [1,2,1] + [0] * 12 + 1.1

> *Error* type 2382 *****
> Came across a faulty line of input in
> "fault-2382-CheckListAndRange.txt".
> The line set an entry that returns a list
> with a multiplier (*) in it. The multiplier
> was not followed by a number or the name of
> a constant. The faulty term was "[0] * ".
> Please edit the file to add the multiplier.
> Faulty line of input (18th) is
>     dud_list [1,] * mult + [0] *   + [-1, -1]

> *Error* type 2383 *****
> Came across a faulty line of input in
> "fault-2383-CheckListAndRange.txt".
> The line set an entry that returns a list
> A list was started by a "[" but did not end
> in a "]". It was "[-1, 0, -1".
> Please edit the file to add the "]" or
> delete the list.

> Faulty line of input (17th) is
>     dud_list [1,2,1] + [0] * mult + [-1, 0, -1

> *Error* type 2384 *****
> Came across a faulty line of input in
> "fault-2384-CheckListAndRange.txt".
> The line had an entry that included an empty
> list. Please edit the file to put at least
> one entry in it, or delete the empty list.

> Faulty line of input (16th) is
>     dud_list [1,2,1] + [] * 12 + [-1, 0, -1

> *Error* type 2385 *****
> Came across a faulty line of input in
> "fault-2385-CheckListAndRange.txt".
> The line had a list as part of its entry.
> Entries in lists must be separated by
> commas, but the 2nd value was not
> separated from the 3rd value by a comma.
```

```
> Please edit the file to correct the list.

> Faulty line of input (16th) is
>     dud_list [1, 2 1]

> *Error* type 2386 ****
> Came across a faulty line of input in
> "fault-2386-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (range). The keyword
> was not followed by a "(" to mark the start
> of three numbers that define the range.
> Please modify the line to match the following
> syntax:
>     "range (start location, stop location, step interval)"
> Faulty line of input (24th) is
>     dud_list range 1000, 2000, 120

> *Error* type 2387 ****
> Came across a faulty line of input in
> "fault-2387-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstopcount). It started
> with a "(" but did not end in a ")",
> instead it was "]".
> Please edit the file to add a ")" or
> delete the range definition.

> Faulty line of input (22nd) is
>     dud_list startstopcount ( 1000, 2000,9]

> *Error* type 2388 ****
> Came across a faulty line of input in
> "fault-2388-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstepcount). Entries
> in range definitions must be separated by
> commas, but the 2nd value was not
> separated from the 3rd value by a comma.
> Please edit the file to correct this.

> Faulty line of input (21st) is
>     dud_list startstepcount(1000, 120  9)

> *Error* type 2389 ****
> Came across a faulty line of input in
> "fault-2389a-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstepcount). These require
> three numbers to define the range to use,
> but there were too few entries (2). Please
> edit the file to put three entries in the
> range function.
> Faulty line of input (21st) is
```

```
>      dud_list startstepcount(1000, 1209)

> *Error* type 2389 ****
> Came across a faulty line of input in
> "fault-2389b-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstepcount). These require
> three numbers to define the range to use,
> but there were too many entries (4). Please
> edit the file to put three entries in the
> range function.
> Faulty line of input (21st) is
>      dud_list startstepcount(1000, 120, 9, 143)

> *Error* type 2390 ****
> Came across a faulty line of input in
> "fault-2390-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstepcount). These require
> a count of entries that is at least 1, but
> in this case it is -2.
> Please set the counter to 1 or above.
> Faulty line of input (21st) is
>      dud_list startstepcount(1000, 120, -2)

> *Error* type 2391 ****
> Came across a faulty line of input in
> "fault-2391-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (startstepcount). These require
> a step distance to set how far apart the
> entities are. In this case the step distance
> is zero. Please set it to a non-zero value
> (note that the distances can be +ve or -ve).
> Faulty line of input (24th) is
>      dud_list startstepcount(1000, 0.0, 9)

> *Error* type 2392 ****
> Came across a faulty line of input in
> "fault-2392-CheckListAndRange.txt".
> The line set an entry that returns a range
> definition (range) that starts at
> 1000.0 and ends at 2000.0 but wants to get
> there in steps of -120.0.
> Please change the step value so that the
> range function can increment from 1000.0
> to 2000.0.
> Faulty line of input (21st) is
>      dud_list range(1000, 2000, -120)

> *Error* type 2393 ****
> Came across a faulty line of input in
> "fault-2393-CheckListAndRange.txt".
> The line set an entry that returns a range
```

```

> definition (startstopcount) that starts at
> 1000.0 and ends at 1000.0. These are too
> close together to be valid.
> Please change the start or stop value so
> that the range function can work properly.
> Faulty line of input (21st) is
>     dud_list startstopcount(1000, 1000, 1)

> *Error* type 2394 *****
> Came across a faulty line of input in
> "fault-2394a-ProcessConstants.txt".
> The line had an entry that was not a list,
> the name of a constant, a valid number or
> a range definition.
> Instead it was "1.2 , 3.2 ]".
> Please edit the file to correct it.

> Faulty line of input (18th) is
>     dud_entry1 1.2 , 3.2 ] * 12

> *Error* type 2394 *****
> Came across a faulty line of input in
> "fault-2394b-ProcessConstants.txt".
> The line had an entry that was not a list,
> the name of a constant, a valid number or
> a range definition.
> Instead it was "xxx".
> Please edit the file to correct it.

> Faulty line of input (18th) is
>     dud_entry  xXx

> *Error* type 2395 *****
> Came across a faulty line of input in
> "fault-2395-CheckListAndRange.txt".
> The line had a range definition:
>     "range(1000, 2000, 120)"
> and a multiplier (2). Multipliers can
> only be used in conjunction with lists
> or constants that return lists.
> Please edit the file to remove the
> multiplier or change the range to a list.
> Faulty line of input (21st) is
>     dud_list range(1000, 2000, 120) * 2

> *Error* type 2396 *****
> Came across a faulty line of input in
> "fault-2396-CheckListAndRange.txt".
> The line had a constant (mult) that
> returned a number (3) and a multiplier
> (12). Multipliers can only be used in
> conjunction with lists or constants that
> return lists.
> Please edit the file to correct the line or

```

```

> the definition of the constant.
> Faulty lines of input (16th and 18th) are
>   mult 3
>   dud_entry2 [1] + mult * 12

> *Error* type 2397 ****
> Came across a faulty line of input in
> "fault-2397-CheckListAndRange.txt".
> The line had a number (1.2) and a multiplier
> (12). Multipliers can only be used in
> conjunction with lists or constants that
> return lists.
> Please edit the file to remove the
> constant or change the number to a list.
> Faulty line of input (18th) is
>   dud_entry2 [1] + 1.2 * 12

> *Error* type 2401 ****
> The file named "fault-2401-ProcessPlotControl.txt" has
> incorrect entries in its plotcontrol block.
> The numbers in the block must be multiples
> of the timestep (0.1). At least one of the
> entries is not, 0.22.
> Please edit the file to correct this.
> Faulty line of input (28th) is
>   aero [0, 0.1, 0.22] + range(0.3, 4, 0.1)

> *Error* type 2421 ****
> Came across a faulty line of input in
> "fault-2421-ProcessSectypes.txt".
> The line defined a sectype named "Same".
> This is not allowed because that word is
> reserved for an internally-named sectype.
> Please edit the file to rename the sectype.
> Faulty line of input (20th) is
>   Same      40.5 35 -0.036

> *Error* type 2441 ****
> In the file named "fault-2441a-ProcessSectypes.txt"
> two optional arguments in tunnel "dopey"
> set the same custom k-factor. One was "zeta_bf",
> the other was "zeta_in". Please remove one of
> them.
> Faulty line of input (22nd) is
>   back 10000 portal 20 TBM zeta_in := 0.5 zeta_bf := 0.4

> *Error* type 2441 ****
> In the file named "fault-2441b-ProcessSectypes.txt"
> two optional arguments in tunnel "dopey"
> set the same custom k-factor. One was "zeta_fb",
> the other was "zeta_out". Please remove one of
> them.
> Faulty line of input (22nd) is
>   back 10000 portal 20 TBM zeta_out := 0.9 zeta_fb := 1.2

```

```

> *Error* type 2441 ****
> In the file named "fault-2441c-ProcessSectypes.txt"
> two optional arguments in tunnel "dopey"
> set the same custom k-factor. One was "zeta_bf",
> the other was "zeta_out". Please remove one of
> them.
> Faulty line of input (24th) is
>     fwd 12130 portal 0 zeta_out := 0.9 zeta_bf := 1.2

> *Error* type 2441 ****
> In the file named "fault-2441d-ProcessSectypes.txt"
> two optional arguments in tunnel "dopey"
> set the same custom k-factor. One was "zeta_bf",
> the other was "zeta_out". Please remove one of
> them.
> Faulty line of input (24th) is
>     fwd 12130 portal 0 zeta_out := 0.9 zeta_bf := 1.2

> *Error* type 2461 ****
> The file named "fault-2461-ProcessFanCurve.txt" does
> not have an entry that defines characteristics
> to use in fan curve definition "testfan".
> You need to have a "datasource" entry that tells
> the program the nickname of a source of data.
> Alternatively (and this is only recommended if
> you are using SES) you can have an "SES_7B" entry
> with four pairs of pressures and flows that can
> be processed into a characteristic by splines.
> Please add one or two "datasource" or "SES_7B"
> entries. The following nickname(s) of data
> sources are available:
>     fan_curve1, fan_curve2, fan_curve3, fan_curve4
>     and csv_fancurve5.

> *Error* type 2462 ****
> The file named "fault-2462a-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has 3 "datasource"
> entries.
> You can't define more than two characteristics
> (one for forwards mode, one for reverse mode).
> Please pick the two definitions you want the fan
> to use and remove the others.
>
> Faulty lines of input (99th, 100th and 101st) are
>     datasource fan_curve3 q p_Tot direction:= reverse
>     datasource fan_curve4 1 2 direction:= forwards
>     datasource fan_curve5 1 2 direction:= reverse

> *Error* type 2462 ****
> The file named "fault-2462b-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has 3 "SES_7B" entries.
> You can't define more than two characteristics
> (one for forwards mode, one for reverse mode).

```

```

> Please pick the two definitions you want the fan
> to use and remove the others.
>
> Faulty lines of input (99th, 100th and 101st) are
>     SES_7B  3000  0    2300  100    800  200    0  225  direction:= forwards
>     ses_7b  3000  0    2300  100    800  200    0  225  direction:= reverse
>     SES_7B  3000  0    2300  100    800  200    0  225

> *Error* type 2462 ****
> The file named "fault-2462c-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has 2 "datasource"
> entries and one "SES_7B" entry.
> You can't define more than two characteristics
> (one for forwards mode, one for reverse mode).
> Please pick the two definitions you want the fan
> to use and remove the others.
>
> Faulty lines of input (100th, 101st and 99th) are
>     datasource fan_curve3 q p_Tot  direction:= reverse
>     datasource fan_curve4 1 2  direction:= reverse
>     SES_7B  3000  0    2300  100    800  200    0  225  direction:= forwards

> *Error* type 2462 ****
> The file named "fault-2462d-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has one "datasource"
> entry and 2 "SES_7B" entries.
> You can't define more than two characteristics
> (one for forwards mode, one for reverse mode).
> Please pick the two definitions you want the fan
> to use and remove the others.
>
> Faulty lines of input (101st, 99th and 100th) are
>     datasource fan_curve4 1 2
>     SES_7B  3000  0    2300  100    800  200    0  225  direction:= forwards
>     SES_7B  3000  0    2300  100    800  200    0  225  direction:= reverse

> *Error* type 2463 ****
> The file named "fault-2463-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has two
> characteristics.
> When you define two characteristics, you must
> include optional arguments to specify which
> one to use when running in forwards mode and
> which one to use when running in reverse mode.
> Please add the optional argument
>         "direction:=forwards"
> to the one intended for forwards mode or add
>         "direction:=reverse"
> to the other (you can add both if you want).
> Faulty lines of input (100th and 99th) are
>     datasource fan_curve3 q p_Tot
>     SES_7B  3000  0    2300  100    800  200    0  225

> *Error* type 2464 ****

```

```

> The file named "fault-2464a-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has two
> characteristics.
> When you define two characteristics, you must
> include an optional argument to specify which
> curve to use when running in forwards mode and
> which one to use when running in reverse mode.
> Unfortunately, you specified in both entries
> that they should be running in forwards mode.
> Please edit the file to change one so that it
> runs in the opposite direction or remove one
> optional entry (this has the same effect).
> Faulty lines of input (100th and 99th) are
>     datasource fan_curve3 q p_Tot    direction:= forwards
>     SES_7B   3000  0     2300  100    800  200    0  225  direction:= forwards

> *Error* type 2464 ****
> The file named "fault-2464b-ProcessFanCurve.txt" has a fan
> curve definition ("testfan") that has two
> characteristics.
> When you define two characteristics, you must
> include an optional argument to specify which
> curve to use when running in forwards mode and
> which one to use when running in reverse mode.
> Unfortunately, you specified in both entries
> that they should be running in reverse mode.
> Please edit the file to change one so that it
> runs in the opposite direction or remove one
> optional entry (this has the same effect).
> Faulty lines of input (100th and 99th) are
>     datasource fan_curve3 q p_Tot    direction:= reverse
>     SES_7B   3000  0     2300  100    800  200    0  225  direction:= reverse

> *Error* type 2481 ****
> The file named "fault-2481a-GetTabulated.txt"
> tried to read data from a data source (a .csv
> file or "begin data...end data" block in the
> file) that does not exist. The nickname of
> the desired data source was "fan_curve3".
> There were no data sources in the file (no
> "begin data...end data" blocks and no data
> taken from .csv files).
> Please edit the file to add a suitable data
> source or remove the line of entry.
> Faulty line of input (27th) is
>     datasource fan_curve3 q p_Tot    direction:= forwards

> *Error* type 2481 ****
> The file named "fault-2481b-GetTabulated.txt"
> tried to read data from a data source (a .csv
> file or "begin data...end data" block in the
> file) that does not exist. The nickname of
> the desired data source was "non_existent".
> Please edit the file to add a suitable data

```

```
> source or remove the line of entry. For what
> it's worth the following nickname(s) of data
> sources are available:
>   fan_curve1, fan_curve2, fan_curve3, fan_curve4
>   and csv_fancurve5.

> Faulty line of input (101st) is
>   datasource non_existent q p_Tot direction:= forwards

> *Error* type 2482 ****
> The file named "fault-2482-GetTabulated.txt"
> tried to read data from a data source called
> "fan_curve4". The block exists, but
> you want to read the column number -2, which
> is impossible - the lowest column number is 1.
> The block of data has 2 columns of data, so a
> number between 1 and 2 is needed.
> Please edit the file to correct it.

> Faulty line of input (99th) is
>   datasource fan_curve4 1 -2 direction:= forwards

> *Error* type 2483 ****
> The file named "fault-2483-GetTabulated.txt"
> tried to read data from a data source called
> "fan_curve4". The block exists, but
> you want to read the column number 3, which
> is higher than the count of columns available.
> The block of data has 2 columns of data, so a
> number between 1 and 2 is needed.
> Please edit the file to correct it.

> Faulty line of input (100th) is
>   datasource fan_curve4 1 3 direction:= forwards

> *Error* type 2484 ****
> The file named "fault-2484-GetTabulated.txt"
> tried to read data from a data source called
> "fan_curve3". The block exists, but
> you want to read the column named p_stat,
> which does not exist. For what it is worth
> the data block contained the following
> column names:
>   q and p_tot.
> Faulty line of input (99th) is
>   datasource fan_curve3 Q P_stat direction:= forwards

> *Error* type 2485 ****
> The file named "fault-2485a-GetTabulated.txt"
> successfully read data from "fan_curve1",
> but there was a problem with the data. The
> data is intended for a purpose that requires
> no missing entries (such as a fan characteristic
> or the behaviour of a portal door). One of
```

```
> the columns of data has at least one missing
> entry, so the data can't be used. Here is some
> information to help you trace the problem:
>
>   The line calling up the block of data (the 101st
> line) is:
>     datasource fan_curve1  Q  P_tot    direction:= forwards
>
>   The data block was named "fan_curve1" and
> it began on the 26th line of the file:
>     begin data  fan_curve1
>
>   The column with a missing entry was the 2nd column
> (named "p_tot").
>     20.0, 1700.0
>     30.0, 1500.0
>     40.0, ,
>     50.0, 2200.0
>     60.0, 2100.0

> *Error* type 2485 *****
> The file named "fault-2485b-GetTabulated.txt"
> successfully read data from "fan_curve1",
> but there was a problem with the data. The
> data is intended for a purpose that requires
> no missing entries (such as a fan characteristic
> or the behaviour of a portal door). One of
> the columns of data has at least one missing
> entry, so the data can't be used. Here is some
> information to help you trace the problem:
>
>   The line calling up the block of data (the 102nd
> line) is:
>     datasource fan_curve1  1  2    direction:= forwards
>
>   The data block was named "fan_curve1" and
> it began on the 27th line of the file:
>     begin data  fan_curve1
>
>   The column with a missing entry was the 2nd column
> (named "p_tot").
>     20.0, 1700.0
>     30.0, 1500.0
>     40.0, ,
>     50.0, 2200.0
>     60.0, 2100.0

> *Error* type 2485 *****
> The file named "fault-2485c-GetTabulated.txt"
> successfully read data from "fan_curve2",
> but there was a problem with the data. The
> data is intended for a purpose that requires
> no missing entries (such as a fan characteristic
> or the behaviour of a portal door). One of
```

```
> the columns of data has at least one missing
> entry, so the data can't be used. Here is some
> information to help you trace the problem:
>
> The line calling up the block of data (the 102nd
> line) is:
>     datasource fan_curve2    1  2   direction:= forwards
>
> The data block was named "fan_curve2" and
> it began on the 45th line of the file:
>     begin data  fan_curve2
>
> The column with a missing entry was the 2nd column.
>     40.0, 1900.0
>     50.0, 2200.0
>     60.0, , ,
>     80.0, 1800.0
>     100.0, 1450.0

> *Error* type 2485 *****
> The file named "fault-2485d-GetTabulated.txt"
> successfully read data from "fancurve5",
> but there was a problem with the data. The
> data is intended for a purpose that requires
> no missing entries (such as a fan characteristic
> or the behaviour of a portal door). One of
> the columns of data has at least one missing
> entry, so the data can't be used. Here is some
> information to help you trace the problem:
>
> The line calling up the block of data (the 100th
> line) is:
>     datasource fancurve5    1  2   direction:= forwards
>
> The data came from a .csv file named "fan_curve5.csv".
>
> The column with a missing entry was the 2nd column
> in the .csv file:
>     40.0, 1900.0
>     50.0, 2200.0
>     60.0, , ,
>     80.0, 1800.0
>     100.0, 1450.0

> *Error* type 2501 *****
> The file named "fault-2501-BuildFanChar.txt"
> had an SES fan characteristic definition in
> the fan named "testfan". The four pressure
> values in such definitions must all decrease
> as you go from left to right in the line that
> defines the characteristic.
> The 3rd and 4th pressures (800.0 and 1000.0)
> don't follow that rule. Please edit the file
> to correct this.
```

```

> Faulty line of input (25th) is
>     Ses_7b    3000   0     2300  100     800  200  1000  225

> *Error* type 2502 ****
> The file named "fault-2502-BuildFanChar.txt"
> had an SES fan characteristic definition in
> the fan named "testfan". The four flow values
> in such definitions must all increase as you
> go from left to right in the line defining
> the characteristic.
> The 2nd and 3rd flowrates (210.0 and 200.0)
> don't follow that rule. Please edit the file
> to correct this.
> Faulty line of input (26th) is
>     Ses_7b    3000   0     2300  210     800  200   0  225

> *Error* type 2503 ****
> The file named "fault-2503a-BuildFanChar.txt"
> had a fan characteristic definition in the
> fan named "testfan". The flow values must
> increase as you go down the list of entries.
> The 6th and 7th flowrates (60.0 and 50.0)
> don't follow that rule. Please edit the file
> to correct this.
> The line calling up the block of data (the 41st
> line) is:
>     Datasource fan_curve  Q  P_tot
> The data came from the 1st column of the
> datasource "fan_curve"..
> *Error* type 2503 ****
> The file named "fault-2503b-BuildFanChar.txt"
> had a fan characteristic definition in the
> fan named "testfan". The flow values must
> increase as you go down the list of entries.
> The 7th and 8th flowrates (80.0 and 60.0)
> don't follow that rule. Please edit the file
> to correct this.
> The line calling up the block of data (the 29th
> line) is:
>     Datasource fancurve5  6  3      direction:=forwards
> The data came from the 6th column of the
> .csv file "fan_curve5.csv".

> *Error* type 2521 ****
> In the file named "fault-2521-GetCharData.txt"
> you tried to use a fan characteristic in
> tunnel "first", but the fan characteristic
> (named "dudfan") does not exist.
> Please either add a "fanchar" block with this
> name or change its name to one of the fanchars
> defined in the file, which are:
> testfan.
> Faulty line of input (33rd) is

```

```

>      fan1 10220  fan1  dudfan  1.0  start:=60  stop:=200

> *Error* type 2541 ****
> In the file named "fault-2541-CheckTimeIncrease.txt",
> the datasource nicknamed "toofewrows" was used
> to define a damper operating sequence.
> Unfortunately the datasource had fewer than two
> entries in it, which doth not a time-series make.
> Please edit the file to use another column for
> time, use a different datasource, or edit the
> datasource to add more entries.
> Faulty lines of input (30th and 23rd) are
>      damper1  10100  Grant-St-FID2  toofewrows  1  2  3  4
>      begin data toofewrows

> *Error* type 2542 ****
> In the file named "fault-2542-CheckTimeIncrease.txt",
> the datasource nicknamed "toofewrows" was used
> to define a damper operating sequence.
> Unfortunately the column assigned as time (the
> 1st column) did not start at zero time.
> Please edit the file to use another column for
> time, use a different datasource, or edit the
> datasource to start the column at time zero.
> Faulty lines of input (32nd and 24th) are
>      damper1  10100  Grant-St-FID2  toofewrows  1  2  3  4
>      begin data toofewrows

> *Error* type 2543 ****
> In the file named "fault-2543-CheckTimeIncrease.txt",
> the datasource nicknamed "toofewrows" was used
> to define a damper operating sequence.
> Unfortunately in the column assigned as time (the
> column named "time") the times were out of sequence.
> See the 2nd and 3rd times (102.0 and 54.0).
> Please edit the file to use another column for
> time, use a different datasource, or edit the
> datasource to make the time values increase.
> Relevant lines of input (32nd and 23rd) are
>      damper1  10100  Grant-St-FID2  toofewrows  time  area  3  4
>      begin data toofewrows

> *Error* type 2561 ****
> In the file named "fault-2561-CheckEntityName.txt"
> there were two entities with the same name (fan1).
> The first is a fan in tunnel "first", the
> second is a fan in tunnel "second". Please
> edit the file and rename one of them.
> Faulty lines of input (35th and 41st) are
>      fan1 10220  fan1  testfan  1.0  start:=60  stop:=200
>      fan1 10450  FAN1  testfan  -0.8

> *Error* type 2581 ****
> In the file named "fault-2581a-AllPositive.txt",

```

```
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to area (the column named
> "area") had a negative value in it, which is
> not appropriate for areas.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the value.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time   area   3   3
>     begin data damperseq

> *Error* type 2581 ****
> In the file named "fault-2581b-AllPositive.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to area (the column named
> "area") had a zero value in it, which is
> not appropriate for areas.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the value.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time   area   3   3
>     begin data damperseq

> *Error* type 2581 ****
> In the file named "fault-2581c-AllPositive.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to area (the column named
> "area") had 3 negative values in it, which is
> not appropriate for areas.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the values.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time   area   3   3
>     begin data damperseq

> *Error* type 2581 ****
> In the file named "fault-2581d-AllPositive.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to area (the column named
> "area") had 2 zero values in it, which is
> not appropriate for areas.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the values.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time   area   3   3
>     begin data damperseq
```

```
> *Error* type 2581 ****
> In the file named "fault-2581e-AllPositive.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to area (the 2nd column)
> had 3 -ve/zero values in it, which is
> not appropriate for areas.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the values.
> Relevant lines of input (41st and 23rd) are
>     damper1 10100 FID_1    damperseq    time    2    zeta    zeta
>     begin data damperseq

> *Error* type 2582 ****
> In the file named "fault-2582a-NotNegative.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to k-factor (the 3rd column)
> had a negative value in it, which is not
> appropriate for k-factors.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the value.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time    area    3    3
>     begin data damperseq

> *Error* type 2582 ****
> In the file named "fault-2582b-NotNegative.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to k-factor (the 3rd column)
> had 3 negative values in it, which is not
> appropriate for k-factors.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the values.
> Relevant lines of input (40th and 22nd) are
>     damper1 10100 FID_1    damperseq    time    area    3    3
>     begin data damperseq

> *Error* type 2582 ****
> In the file named "fault-2582c-NotNegative.txt",
> the datasource nicknamed "damperseq" was
> used to define a damper operating sequence.
> The column assigned to k-factor (the column named "zeta")
> had 3 negative values in it, which is not
> appropriate for k-factors.
> Please edit the file to use another column, use
> a different datasource, or edit the datasource
> to correct the values.
> Relevant lines of input (41st and 23rd) are
>     damper1 10100 FID_1    damperseq    time    area    zeta    3
```

```

> begin data damperseq

> *Error* type 2621 ****
> Came across faulty input in "fault-2621-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> was defined twice. Please rename one of them or
> delete one of them.
> Faulty lines of input (46th and 36th) are
>   begin schedule Once
>   begin schedule      once

> *Error* type 2622 ****
> Came across faulty input in "fault-2622-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> referred to train type "g+p-fig-5", but no such
> train type exists. Please define it using a
> "begin traintype g+p-fig-5" block in the file.
> Faulty line of input (40th) is
>   traintype G+P-Fig-5

> *Error* type 2623 ****
> Came across faulty input in "fault-2623-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> referred to train type "g+p-fig-6", but no such
> train type exists. Please define it or change
> the train type to one of the following:
>   g+p-fig-5.
> Faulty line of input (45th) is
>   traintype G+P-Fig-6

> *Error* type 2624 ****
> Came across faulty input in "fault-2624a-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> has trains spawning at the same time, which is not
> allowed. The following is a list of the duplicated
> times:
>   100.0 and 200.0.
> Please edit the entries in the "times" keyword to
> to remove the duplicate times.
> Faulty line of input (46th) is
>   times [0, 100, 100, 200, 200.0]

> *Error* type 2624 ****
> Came across faulty input in "fault-2624b-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> has trains spawning at the same time, which is not
> allowed. The following is a list of the duplicated
> times:
>   3600.0.
> Please edit the entries in the "times" keyword to
> to remove the duplicate times.
> You used the range, startstepcount or startstopcount
> function to build the list of times so it may not be
> obvious where the duplication sprang from. Here is

```

```

> the full list of times:
>   0.0, 900.0, 1800.0, 2700.0, 3600.0, 3600.0,
>   3960.0, 4320.0, 4680.0, 5040.0, 5400.0, 5760.0,
>   6120.0, 6480.0 and 6840.0.
> Faulty line of input (47th) is
>       times startstopcount(0, 3600, 5) + range(3600, 7200, 360)

> *Error* type 2625 ****
> Came across unusual input in "fault-2625-ProcessSchedule.txt".
> In route "trainpassage", train schedule "once"
> there are 1 trains that start at the route origin
> with a negative train speed, meaning that they exit as
> soon as they spawn. You probably forgot to include a
> "downstart" keyword to make the trains spawn on the
> down side of the tunnel complex. The run will proceed
> but you may want to look into this.

> *Error* type 2661 ****
> Came across faulty input in "fault-2661-CheckForPortals.txt".
> In route "trainpassage" the tunnel "patchway" was
> given as one of the tunnels in a list of
> tunnels. But that tunnel has no nodes connecting
> it to other tunnels, so it cannot appear in a
> route with other tunnels. Please either edit
> the "begin tunnels" block in the route to remove
> the tunnel, or edit the definition of the tunnel
> ends to add a node or two.
> Faulty lines of input (42nd, 24th and 28th) are
>   begin tunnels
>     back    0    portal 0    Patchway-Old
>     fwd     1140   portal 0

> *Error* type 2681 ****
> In the file named "fault-2681-ProcessClones.txt"
> the name of a set of cloned tunnels did not
> have a "*" character in it, meaning that its
> tunnels could not be generated with numbers
> in place of the "*". Please add one, e.g. "XP*".
> Faulty line of input (37th) is
>   begin tunnelclones XP

> *Error* type 2682 ****
> In the file named "fault-2682-ProcessClones.txt"
> a set of cloned tunnels generated a tunnel
> name ("XP1") that clashed with the
> name of another tunnel. Please edit the
> file to resolve the name clash
> Faulty lines of input (36th, 36th and 42nd) are
>   begin tunnel XP1
>   begin tunnelclones XP*
>     numbering "range(1, 17.1, 1)"    XP01 to XP17

> *Error* type 2683 ****
> In the file named "fault-2683-ProcessClones.txt",

```

```

> the back end of a set of cloned tunnels used
> the name of a sectype ("bored") that was not
> valid. Please either use a valid name or
> use the word "sectypes" and include a line
> defining a list of names of sectypes. For
> what it's worth, here are the names of the
> sectype(s) you've set:
>   c+c1, c+c2, closed and open.
> Faulty line of input (38th) is
>   back 0 node XP*wb bored      zeta_bf:=2.4 zeta_fb:=2.4

> *Error* type 2684 ****
> In the file named "fault-2684-ProcessClones.txt",
> the back end of a set of cloned tunnels indicated
> that the sectypes for the tunnels were set by
> a "sectypes" line of entry, but there was no
> such line of entry. Please either add one or
> change the name at the back end of the tunnel
> to be the name of a valid sectype.
> Faulty line of input (39th) is
>   back 0 node XP*wb sectypes      zeta_bf:=2.4 zeta_fb:=2.4

> *Error* type 2685 ****
> In the file named "fault-2685-ProcessClones.txt",
> the list of sectypes for some cloned tunnels
> was not the same length as the list of tunnel
> numbers. Please ensure they are the same length
> (the lengths are 15 and 17 respectively).
> Faulty lines of input (38th and 37th) are
>   sectypes "[closed]*12 + [open] + [closed]*2"
>   numbering "range(1, 17.1, 1)"  XP01 to XP17

> *Error* type 2686 ****
> In the file named "fault-2686-ProcessClones.txt",
> a "sectypes" line of entry in a set of cloned
> tunnels contained one invalid sectype name,
> "partial". Please edit the list so that all
> the entries in it are the names of valid
> sectypes. For what it's worth, here are the
> names of the sectype(s) you've set:
>   c+c1, c+c2, closed and open.
> Faulty line of input (38th) is
>   sectypes "[closed]*12 + [partial] + [closed]*4"

> *Error* type 2687 ****
> In the file named "fault-2687-ProcessClones.txt",
> a "sectypes" line of entry in a set of cloned
> tunnels contained 2 invalid sectype names:
>   partial and close2.
> Please edit the list so that all the entries
> in the list are the names of valid sectypes.
> For what it's worth, here are the names of
> the sectype(s) you've set:
>   c+c1, c+c2, closed and open.

```

```

> Faulty line of input (38th) is
>     sectypes "[closed]*12 + [partial] + [close2]*4"

> *Error* type 2701 *****
> In the file named "fault-2701-CheckAsterisks.txt"
> the name of a set of cloned tunnels had too
> many groups of "*" characters in it. Please
> reduce it to one group, e.g. "XP**ab".
> Faulty line of input (36th) is
>     begin tunnelclones XP**ab**

> *Error* type 2721 *****
> In the file named "fault-2721-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block had
> no line with the phrase
>     "calculate with blockage correction term"
> or the phrase
>     "calculate without blockage correction term".
> on it.
> Please edit the file to add one or the other.
> There was a line starting with "calculate", but
> the remaining words weren't what is required.
> Faulty line of input (33rd) is
>     Calculate with pi = 3

> *Error* type 2722 *****
> In the file named "fault-2722-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block had
> two specifications of the same vehicle type
> ("hgv"). Please edit the file to remove
> one or change its name.
> Faulty lines of input (36th and 37th) are
>     vehicle      HGV    12.0   1.2    3  10  2
>     vehicle      HGV    11.7   1.4    3  10  2.5 20  2

> *Error* type 2723 *****
> In the file named "fault-2723-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block had
> a vehicle type named "tot", which is a
> reserved word. Please edit the file to remove
> one or change its name.
> Faulty line of input (33rd) is
>     vehicle      Tot    11.7   1.4    3  10  2.5 20  2

> *Error* type 2724 *****
> In the file named "fault-2724-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block has
> a line of speed-gradient pairs for a type of
> vehicle that does not exist ("nonesuch").
> The only valid vehicle types are
>     car_p, car_d, lcv_p, lcv_d and hgv.
> Faulty line of input (36th) is
>     speedgradpairs  nonesuch  120 0.04  90 0.08  80 0.11

```

```

> *Error* type 2725 ****
> In the file named "fault-2725-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block has
> a line of speed-gradient pairs for the
> "hgv" vehicle type, but there is
> no data on it.
> Please add one or more speed-gradient pairs,
> e.g. "hgv 120 0.04 80 0.06".
> N.B. This means "this vehicle type can travel
> at 120 km/h on upslopes that are equal to or
> less 4% and at 80 km/h on upslopes that are
> over 4% and equal to or less than 6%".
> Faulty line of input (36th) is
>     speedgradpairs HGV

> *Error* type 2726 ****
> In the file named "fault-2726-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block has
> a line of speed limits for the "hgv" vehicle
> type, with pairs of numbers (speed limit
> and the gradient it applies up to. One of
> the gradients ("4.0") is too high: gradients
> in "traffictypes" blocks are fractional
> (0 to 1), with 0 being a flat tunnel and 1
> being a vertical shaft. If you gave the
> gradients as percentages, please divide
> them all by 100 to turn them into fractions.
> Otherwise, please modify the value (and any
> others over 1) to be between 0 and 1.
> Faulty line of input (37th) is
>     speedgradpairs HGV 120 4 100 5 80 6

> *Error* type 2727 ****
> In the file named "fault-2727-ProcessVehTypes.txt"
> the (road vehicle) "traffictypes" block has
> a line of speed limits for the "hgv"
> vehicle type. One of the gradients is over
> 11%, which is practically unheard of in a
> road tunnel. Are you sure that these
> gradients are in fractions (0-1) and not
> in percentages (0 to 100)? Note that if
> you are trying to model a mine decline
> tunnel as a road tunnel, you're probably
> out of luck, as the standard method of
> estimating emissions in road tunnels
> only goes up to 6%. If you are rolling
> your own custom emissions and genuinely
> have emissions tables that apply at 12.2%,
> then your best bet is to edit the source
> code of Hobyah.py to temporarily comment
> out the block of code that raises error
> message 2727 and the line of entry with
> "return(None)" on it (the "return(None)"
> is what causes the run to stop).

```

```
> Faulty line of input (37th) is
>     speedgradpairs HGV 120 0.04 40 0.122

> *Error* type 2741 ****
> In the file named "fault-2741-ProcessTraffic1.txt",
> there was a "trafficsteady" block (a block that
> puts traffic into routes) without there being a
> "traffictypes" block to define the properties of
> the traffic.
> Please edit the file to add a "traffictypes" block
> or remove all the blocks that put traffic in routes.

> *Error* type 2742 ****
> In the file named "fault-2742-ProcessTraffic1.txt",
> the "trafficsteady westbound" block
> tried to put traffic into a route that does not
> exist, "wb-nonexistent".
> Please edit the file to either add a route with
> that name, remove the route name from the block,
> or correct the name of the route. For what it's
> worth, these are the valid route names:
>     ebbranch, ebmain and wb.
> Faulty line of input (35th) is
>     routes      WB-nonexistent

> *Error* type 2743 ****
> In the file named "fault-2743-ProcessTraffic1.txt",
> there is a "trafficsteady" block that puts
> traffic into the same route, "ebmain" twice.
> Please edit the file so that each route name
> appears only once in all the "trafficsteady"
> blocks.
> Faulty line of input (33rd) is
>     routes EBmain    EBmain

> *Error* type 2744 ****
> In the file named "fault-2744-ProcessTraffic1.txt",
> there are two "trafficsteady" blocks that put
> traffic into the same route, "ebmain". The
> first trafficsteady block is "Eastbound" and
> the other is "eastbound2".
> Please edit the file so that each route name
> appears only once.
> Relevant lines of input (32nd and 43rd) are
>     begin trafficsteady Eastbound
>     begin trafficsteady Eastbound2

> *Error* type 2745 ****
> In the file named "fault-2745-ProcessTraffic1.txt",
> the "trafficsteady eastbound" block set
> traffic numbers in the following route:
>     ebmain.
> Please edit the file to add lines of "standstill"
> or "moving" for this route.
```

```

> Relevant line of input (33rd) is
> begin trafficsteady Eastbound

> *Error* type 2761 ****
> In the file named "fault-2761-ProcessJFtypes.txt"
> the "jetfantypes" block had two jet fan types
> with the same name, ("rev1").
> Please edit the file to remove one or change
> its name.
> Faulty lines of input (23rd and 23rd) are
> reversible REV1 1050 0.7 29.8 1050 0.7 -29.8 jetlength:=0.1
> reversible REV1 1050 0.7 29.8 1050 0.7 -29.8 jetlength:=0.1

> *Error* type 2841 ****
> In the file named "fault-2841-TrafficInTunnels.txt",
> the trafficsteady block named "eastbound"
> sets moving traffic and stationary traffic in
> the same tunnel, "eastbound1".
> This is not allowed when traffic is put into
> routes in the same "traffictypes" block, as
> the routes share the same lanes.
> If you need to put both stationary and moving
> traffic into two routes that pass through the
> same tunnel, put them into different blocks,
> as the program then assumes that their lanes
> are segregated, much like the eastbound and
> westbound lanes are segregated in single-tube
> bidirectional tunnels.
> Please edit the file to correct the clash.
> In trafficsteady block "eastbound":
> * The following route puts stationary
>   traffic in the tunnel: ebmain
> * The following route puts moving traffic
>   in the tunnel: ebranch
> Relevant line of input (34th) is
> begin trafficsteady Eastbound

> *Error* type 2861 ****
> In the file named "fault-2861-DoRoutesMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set stationary traffic in
> the same tunnel ("eastbound1").
> The "standstill" keywords gave different values
> for the density of stationary traffic, which
> is not allowed when they are in the same
> "traffictypes" block because they share the
> same lanes. If you need to put blocks of
> stationary traffic of different base densities
> in the same tunnel, put them into different
> "traffictypes" blocks.
> The first route set density 165 veh/lane-km,
> equivalent to 202.218 PCU/lane-km.
> The second route set density 164.999 veh/lane-km,
> equivalent to 219.4317 PCU/lane-km.

```

```

> As an aside, why do you think you need to have
> different base densities in different routes?
> Much safer to use one figure and play around
> with PCU values.
> Note that the densities don't need to be an
> exact match: a mismatch of 0.005 PCU/lane-km
> is accepted.
> Please edit the file to correct the clash.
> Faulty lines of input (45th and 46th) are
>     standstill ebmain 165 veh/lane-km up_ptl down_ptl
>     standstill ebbranch 164.999 veh/lane-km up_ptl down_ptl

> *Error* type 2862 ****
> In the file named "fault-2862-DoRoutesMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set counts of lanes in the
> same tunnel ("eastbound1").
> Route "ebmain" has the following count of
> lanes along the length of this tunnel:
>     Route chainages:    10000 | 12080
>     Tunnel distances:   10020 | 12100
>     Lane at that point: 3 lanes | 3 lanes
> Route "ebbranch" has the following count of
> lanes along the length of this tunnel:
>     Route chainages:    900 | 1000 | 1540 | 2980
>     Tunnel distances:   10020 | 10120 | 10660 | 12100
>     Lane at that point: 2 lanes | 2 lanes | 2 lanes to 3 | 3 lanes
> Route "ebbranch" sets a mismatched count of lanes
> at chainage 1000 m (distance 10120 in the tunnel).
> The mismatch is 2 lanes in "ebbranch" vs 3 lanes in
> "ebmain".
> The corresponding chainage in route "ebmain" is
> 10100 m (the error may be in that route instead).
> Please edit the file to correct the clash by
> editing the count of lanes or the chainages at
> which they change in one or both routes.
> Relevant lines of input (131st and 109th) are
>     begin lanes
>     begin lanes

> *Error* type 2881 ****
> In the file named "fault-2881-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in
> the same tunnel ("eastbound1").
> The "moving" keywords gave different values
> for the traffic speed, which is not allowed
> when they are in the same "traffictypes"
> block. If you need to put traffic flows at
> different speeds in the same tunnel (this
> only happens when the traffic flows are in
> segregated lanes), then put the traffic into
> different "trafficsteady" blocks.
> Please edit the file to correct the clash.

```

```

> Faulty lines of input (41st and 42nd) are
>     moving ebmain 70
>     moving ebbranch 80

> *Error* type 2882 ****
> In the file named "fault-2882a-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound" had
> two routes that set moving traffic in the
> same tunnel ("eastbound1"). The two routes
> have mismatched speed limits in that tunnel.
> Mismatches are not allowed when two routes are
> in the same trafficsteady block; they share
> the same lanes, so their speed limits in the
> tunnel must match.
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11220 | 11920
>     Stop distance: 10320 | 11220 | 11920 | 12100
>     Up chainage: 10000 | 10300 | 11200 | 11900
>     Down chainage: 10300 | 11200 | 11900 | 12080
>     Gradient: -0.04 | -0.005 | 0.005 | 0.03
>     Speed limit: 75 | 75 | 75 | 75
>     Cause of limit: speeds | speeds | speeds | speeds
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 11040 | 12040
>     Stop distance: 11040 | 12040 | 12100
>     Up chainage: 900 | 1920 | 2920
>     Down chainage: 1920 | 2920 | 2980
>     Gradient: 0 | 0 | 0
>     Speed limit: 58 | 62 | 62
>     Cause of limit: speeds | speeds | speeds
> Please study the speed limit and gradient
> transcripts above, then edit the file to
> correct the clash by editing the speed
> limits or the gradients in one or both
> routes.
> Relevant lines of input (125th and 103rd) are
>     begin speedlimits
>     begin speedlimits

> *Error* type 2882 ****
> In the file named "fault-2882b-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound" had
> two routes that set moving traffic in the
> same tunnel ("eastbound1"). The two routes
> have mismatched speed limits in that tunnel.
> The mismatches may spring from speed limits
> set in the routes or speed limits imposed by
> the inability of the "hgv" vehicle type to
> climb steep upgrades.
> Mismatches are not allowed when two routes are
> in the same trafficsteady block; they share
> the same lanes, so their speed limits in the

```

```

> tunnel must match.
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>   Start distance: 10020 | 10320 | 11020 | 11520 | 11820
>   Stop distance: 10320 | 11020 | 11520 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11000 | 11500 | 11800
>     Down chainage: 10300 | 11000 | 11500 | 11800 | 12080
>       Gradient: -0.04 | -0.005 | -0.005 | -0.005 | 0.01
>       Speed limit: 80 | 80 | 100 | 100 | 100
>     Cause of limit: speeds | speeds | speeds | speeds | speeds
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>   Start distance: 10020 | 10320 | 11020 | 11520 | 11820
>   Stop distance: 10320 | 11020 | 11520 | 11820 | 12100
>     Up chainage: 9500 | 9800 | 10500 | 11000 | 11300
>     Down chainage: 9800 | 10500 | 11000 | 11300 | 11580
>       Gradient: -0.04 | -0.005 | -0.005 | -0.005 | 0.03
>       Speed limit: 80 | 80 | 100 | 100 | 90
>     Cause of limit: speeds | speeds | speeds | speeds | gradient
> Please study the speed limit and gradient
> transcripts above, then edit the file to
> correct the clash by editing the speed
> limits or the gradients in one or both
> routes.
> Relevant lines of input (132nd, 103rd, 95th and 123rd) are
>   begin speedlimits
>   begin speedlimits
>   begin gradients fractions
>   begin gradients fractions

> *Error* type 2883 ****
> In the file named "fault-2883a-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). One route set
> speed limits in the tunnel and the other did
> not.
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>   Start distance: 10020 | 10320 | 11220 | 11920
>   Stop distance: 10320 | 11220 | 11920 | 12100
>     Up chainage: 10000 | 10300 | 11200 | 11900
>     Down chainage: 10300 | 11200 | 11900 | 12080
>       Gradient: -0.04 | -0.005 | 0.005 | 0.03
>       Speed limit: 75 | 75 | 75 | 75
>     Cause of limit: speeds | speeds | speeds | speeds
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>   Start distance: 10020
>   Stop distance: 12100
>     Up chainage: 900
>     Down chainage: 2980
>       Gradient: 0
>     Speed limit: 80

```

```

> Cause of limit: speeds
> Route "ebmain" sets a speed limit of 75.0 km/h
> at chainage 10300 m. Route "ebbranch" wants
> 80.0 km/h along the length of the tunnel.
> Please edit the file to correct the clash by
> editing the speed limits in the route or by
> adding speed limits to the "ebbranch" route.
> Relevant lines of input (104th and 122nd) are
>     begin speedlimits
>     begin gradients fractions

> *Error* type 2883 ****
> In the file named "fault-2883b-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). One route set
> speed limits in the tunnel and the other did
> not.
> The mismatches may spring from speed limits
> set in the routes or speed limits imposed by
> the inability of the "hgv" vehicle type to
> climb steep upgrades.
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>     Gradient: -0.04 | 0.03 | 0.03
>     Speed limit: 100 | 90 | 90
>     Cause of limit: speeds | gradient | gradient
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>     Gradient: -0.04 | 0.01 | 0.03
>     Speed limit: 100 | 100 | 90
>     Cause of limit: speeds | speeds | gradient
> Route "ebmain" sets a speed limit of 90.0 km/h
> at chainage 11800 m. Route "ebbranch" wants
> 100.0 km/h along the length of the tunnel.
> Please edit the file to correct the clash by
> editing the speed limits in the route or by
> adding speed limits to the "ebbranch" route.
> Relevant lines of input (104th, 122nd and 122nd) are
>     begin speedlimits
>     begin gradients fractions
>     begin gradients fractions

> *Error* type 2884 ****
> In the file named "fault-2884a-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"

```

```

> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). Neither route set
> speed limits in the tunnel, but speed limits
> were imposed by a vehicle type needing to
> slow down on one or both routes' gradients:
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>         Gradient: -0.04 | 0.06 | 0.03
>         Speed limit: 100 | 90 | 90
>     Cause of limit: speeds | gradient | gradient
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>         Gradient: -0.04 | 0.01 | 0.03
>         Speed limit: 100 | 100 | 90
>     Cause of limit: speeds | speeds | gradient
> Please edit the file to correct the clash by
> making the gradients in the routes match.
> Relevant lines of input (96th and 117th) are
>     begin gradients fractions
>     begin gradients fractions

> *Error* type 2884 ****
> In the file named "fault-2884b-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). Neither route set
> speed limits in the tunnel, but speed limits
> were imposed by a vehicle type needing to
> slow down on one or both routes' gradients:
> Route "ebmain" has no speed limits or
> gradients in this tunnel and tried to put
> traffic at 100 km/h into the tunnel.
> Route "ebbranch" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>         Gradient: -0.04 | 0.01 | 0.03
>         Speed limit: 100 | 100 | 90
>     Cause of limit: speeds | speeds | gradient
> Please edit the file to correct the clash by
> making the gradients in the routes match.
> Relevant lines of input (91st and 110th) are
>     begin route EBmain
>     begin gradients fractions

```

```

> *Error* type 2884 ****
> In the file named "fault-2884c-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). Neither route set
> speed limits in the tunnel, but speed limits
> were imposed by a vehicle type needing to
> slow down on one or both routes' gradients:
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>     Gradient: -0.04 | 0.01 | 0.03
>     Speed limit: 100 | 100 | 90
>     Cause of limit: speeds | speeds | gradient
> Route "ebbranch" has no speed limits or
> gradients in this tunnel and tried to put
> traffic at 100 km/h into the tunnel.
> Please edit the file to correct the clash by
> making the gradients in the routes match.
> Relevant lines of input (97th and 112th) are
>     begin gradients fractions
>     begin route EBbranch

> *Error* type 2884 ****
> In the file named "fault-2884d-DoSpeedsMatch.txt",
> the trafficsteady block named "eastbound"
> had two routes that set moving traffic in the
> same tunnel ("eastbound1"). Neither route set
> speed limits in the tunnel, but speed limits
> were imposed by a vehicle type needing to
> slow down on one or both routes' gradients:
> Route "ebmain" has the following speed
> limits along the length of this tunnel:
>     Start distance: 10020 | 10320 | 11820
>     Stop distance: 10320 | 11820 | 12100
>     Up chainage: 10000 | 10300 | 11800
>     Down chainage: 10300 | 11800 | 12080
>     Gradient: -0.04 | 0.01 | 0.03
>     Speed limit: -90 | -100 | -100
>     Cause of limit: gradient | speeds | speeds
> Route "ebbranch" has no speed limits or
> gradients in this tunnel and tried to put
> traffic at -100 km/h into the tunnel.
> Please edit the file to correct the clash by
> making the gradients in the routes match.
> Relevant lines of input (98th and 113th) are
>     begin gradients fractions
>     begin route EBbranch

> *Error* type 2901 ****

```

```

> Came across faulty input in "fault-2901a-CheckRouteChs.txt".
> Route "wb" had a block setting lane counts.
> Two changes of lane were closer than 10 m:
>   chainage 11000 and chainage 11009
> This is not allowed because it makes it hard
> to catch differences between routes in road
> tunnels that are in the same "trafficsteady"
> block.
> Please edit the file to ensure that these
> chainages (and all others in this block) are
> at least ten metres apart.
> Faulty lines of input (45th and 46th) are
>   2           11000
>   3           11009

> *Error* type 2901 ****
> Came across faulty input in "fault-2901b-CheckRouteChs.txt".
> Route "wb" had a block setting lane counts.
> Two changes of lane were closer than 10 m:
>   chainage 11000 and chainage 11009
> This is not allowed because it makes it hard
> to catch differences between routes in road
> tunnels that are in the same "trafficsteady"
> block.
> Please edit the file to ensure that these
> chainages (and all others in this block) are
> at least ten metres apart.
> Faulty line of input (44th) is
>   2 11000  3 11009  2 13000

> *Error* type 2921 ****
> In the file named "fault-2921a-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev1") with a thrust in forwards mode (99 N)
> that's outside the usual range of 100 to 2300 N.
> Concerning line of input (24th) is
>   reversible      REV1    99  0.7  29.8   1050  0.7  -29.8

> *Error* type 2921 ****
> In the file named "fault-2921b-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev2") with a thrust in forwards mode (2301 N)
> that's outside the usual range of 100 to 2300 N.
> Concerning line of input (25th) is
>   reversible      rev2  2301  0.7  29.8   1050  0.7  -29.8

> *Error* type 2922 ****
> In the file named "fault-2922a-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev1") with an installation efficiency in forwards
> mode (0.1) that's outside the usual range of 0.2 to 1.
> Concerning line of input (24th) is
>   reversible      REV1  1080  0.1  29.8   1050  0.7  -29.8

```

```

> *Error* type 2922 ****
> In the file named "fault-2922b-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev2") with an installation efficiency in forwards
> mode (1.05) that's outside the usual range of 0.2 to 1.
> Concerning line of input (25th) is
>     reversible      rev2  1080 1.05  29.8   1050  0.7  -29.8

> *Error* type 2923 ****
> In the file named "fault-2923a-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev1") with a jet velocity in forwards mode (19.9 m/s)
> that's outside the usual range of 20 to 40 m/s.
> Concerning line of input (24th) is
>     reversible      REV1  1080  0.7  19.9   1050  0.7  -29.8

> *Error* type 2923 ****
> In the file named "fault-2923b-CheckJFThrust.txt"
> the "jetfantypes" block had a jet fan type (called
> "rev2") with a jet velocity in forwards mode (40.1 m/s)
> that's outside the usual range of 20 to 40 m/s.
> Concerning line of input (25th) is
>     reversible      rev2  1080  0.7  40.1   1050  0.7  -29.8

> *Error* type 2941 ****
> In the file named "fault-2941a-GetJFBlocks.txt"
> tunnel "first" had a bank of jet fans of type
> "uni" at 11000 m .
> There are no definitions of jet fan types in
> this file. Please either remove the bank of
> jet fans or add a "jetfantypes" block and
> define the "uni" type.
> Faulty line of input (25th) is
>     jetfans1  11000  EB_JF1  2  uni  -1

> *Error* type 2941 ****
> In the file named "fault-2941b-GetJFBlocks.txt"
> tunnel "first" had a bank of jet fans of type
> "uni" at 11000 m .
> There are no definitions of jet fan types in
> this file. Please either remove the bank of
> jet fans or add an entry in the "jetfantypes"
> block for the "uni" type.
> Faulty line of input (29th) is
>     jetfans1  11000  EB_JF1  2  uni  -1

> *Error* type 2942 ****
> In the file named "fault-2942-GetJFBlocks.txt"
> tunnel "first" had a bank of jet fans of type
> "rev2" at 11000 m .
> Alas, there is no jet fan type with this name.
> Please edit the file to remove the bank of jet
> fans, change the name of the jet fan type or add
> the jet fan type to the "jetfantypes" block.

```

```

> For what it is worth, the following jet fan type(s)
> exist:
>   uni.
> Faulty line of input (31st) is
>   jetfans1  11000  EB_JF1  2  rev2 -1

> *Error* type 2943 *****
> In the file named "fault-2943-GetJFBlocks.txt"
> tunnel "first" had a bank of jet fans of type
> "uni" at 11000 m .
> Jet fans of type "uni" are unidirectional and
> cannot run in reverse, but the bank of fans tries
> to run them in reverse (the speed fraction in the
> "jetfans1" line is -1.2).
> Please edit the file to change the type of jet
> fan in the bank or turn the fans off.
> Note that if you genuinely want unidirectional
> jet fans blow towards the back end of a tunnel,
> define a reversible jet fan and put in zero for
> the installation efficiency in forwards mode.
> Clashing lines of input (25th and 32nd) are
>   unidirectional uni  1100  0.7  30                               URTW-112-29
>   jetfans1  11000  EB_JF1  2  uni -1.2 start:=20 stop:=1000

> *Error* type 2944 *****
> In the file named "fault-2944-GetJFBlocks.txt"
> tunnel "first" had a bank of jet fans that
> had a fractional number of jet fans operating
> (2.1).
> This is only allowed if you have the line
>
>   jetfancounts nonintegers
>
> in the "settings" block. Please either only use
> integer counts of jet fans in this file or add
> that line to the "settings" block.
> Faulty line of input (41st) is
>   jetfans1  11000  EB_JF1  2.1  rev  1.0

> *Error* type 2945 *****
> In the file named "fault-2945a-GetJFBlocks.txt",
> tunnel "first" had a bank of jet fans
> (called eb_jf1) running in forwards mode
> in which the jet plume extended outside the
> tunnel. The tunnel extends from
> 10000 to 12000.
> The plume from the jet fans extends from
> 11953 to 12053.
> 53 m of the jet fan plume is outside the
> tunnel, as depicted in this ASCII art:
>
>   back end          fwd end
>   <-----extent of tunnel----->
>   [==]>>>>>>>>>>>>>>>>>
```



```

> Please edit the file to adjust the location of
> the jet fan outlet so that it does not coincide
> with a boundary between segments. That is not
> allowed because there may be between two and
> six tunnel air velocities at boundaries and it
> is unclear which should be used.
>
> Note that the type of jet fan (called "rev")
> has a length of 20 m, which you should factor
> into whether to move the bank of jet fans or
> alter the length of the jet fan type in the
> "jetfanatypes" block.
> Faulty line of input (33rd) is
>   jetfans1 11920 EB_JF1 2 rev -1.0 start:=20 stop:=1000

> *Error* type 2981 ****
> The 1st of "fault-2981-ProcessFilesLoop.txt"
> (a filesloop) had no definition of which files to
> run the loop over - please add one. Valid definitions
> are:
> * a line with "all files" (without quotes) between
>   the "begin filesloop" line and the first "begin
>   graph" or "begin image" line.
> * a "begin excluded...end excluded" block between
>   the "begin filesloop" line and the first "begin
>   graph" or "begin image" line. Put the nicknames
>   to NOT plot at inside the block. All the other
>   nicknames in the "begin files" block will be used.
> * a "begin nicknames...end nicknames" block between
>   the "begin filesloop" line and the first "begin
>   graph" or "begin image" line. Put the nicknames
>   to plot at inside the block.
> Relevant line of input (29th) is
>   begin filesloop # ignore #

> *Error* type 2982 ****
> The 1st loop of "fault-2982-ProcessFilesLoop.txt"
> (a filesloop) had more than one definition of
> which files to run the loop over - please use
> just one of:
>   "all files",
>   a "begin exclude...end exclude" block, or
>   a "begin nicknames...end nicknames" block.
> Conflicting lines of input (30th and 31st) are
>   all files
>   begin exclude

> *Error* type 2983 ****
> The 1st loop of "fault-2983-ProcessFilesLoop.txt"
> had a "begin exclude" entry that referred
> to an invalid nickname, file10".
> Please change it to a valid nickname or
> remove it. For what it's worth, here are all
> the valid nickname(s):

```

```
>   file1 and file2.  
> Faulty line of input (32nd) is  
>   file10  
  
> *Error* type 1027 ****  
> Came across a number entry that was a word that  
> represents infinity ("inf") in "fault-1027-CheckNanInf.txt".  
> Infinity is not a valid input. Please change it  
> entry to a non-infinite number.  
> Faulty line of input (19th) is  
>   C+C1   103   55   inf  
  
> *Error* type 1028 ****  
> Came across a number entry that was IEEE 754's  
> Not-a-Number value in "fault-1028-CheckNanInf.txt".  
> Please change the entry to a number.  
> Faulty line of input (19th) is  
>   C+C1   103   55   nan  
  
> *Error* type 1041 ****  
> Came across a faulty line of input in  
> "fault-1041a-ProcessOptionals.txt".  
> The line contained ":=" (which signifies an  
> optional entry) but there was no key before  
> the :=. Please add one.  
> Faulty line of input (9th) is  
>   := optional frictiontype Fanning  
  
> *Error* type 1041 ****  
> Came across a faulty line of input in  
> "fault-1041b-ProcessOptionals.txt".  
> The line contained ":=" (which signifies an  
> optional entry) but there was no key before  
> the :=. Please add one.  
> Note that this error can be triggered when  
> when there are instances of := (marking the  
> optional entries) but not enough keys. For  
> what it's worth, here are the words taken  
> from the line as optional entries:  
>   option1 := option2  
> Faulty line of input (11th) is  
>   option1 := option2 := option3 frictiontype Fanning  
  
> *Error* type 1041 ****  
> Came across a faulty line of input in  
> "fault-1041c-ProcessOptionals.txt".  
> The line contained ":=" (which signifies an  
> optional entry) but there was no key before  
> the :=. Please add one.  
> Note that this error can be triggered when  
> when there are instances of := (marking the  
> optional entries) but not enough keys. For  
> what it's worth, here are the words taken  
> from the line as optional entries:
```

```

>     option1 := dark
> Faulty line of input (10th) is
>     rho_atm 1.2 option1 := dark := light

> *Error* type 1042 ****
> Came across a faulty line of input in
> "fault-1042-ProcessOptionals.txt".
> The line contained ":" (which signifies an
> optional entry) but there was no value after
> the :=. Please add one.
> Faulty line of input (9th) is
>     frictiontype Fanning optional :=

> *Error* type 1043 ****
> Came across a faulty line of input in
> "fault-1043-ProcessOptionals.txt".
> The line contained two optional entries
> with the same key (option1). One set it
> it to "foo", the other set it to
> "bar". Please pick one.
> Faulty line of input (8th) is
>     frictiontype Fanning option1 := foo option1 := bar

> *Error* type 1044 ****
> Came across a faulty line of input in
> "fault-1044-ProcessOptionals.txt".
> The line contained nothing but optional entries.
> Note that this error can be triggered when
> when there are instances of := (marking the
> optional entries) but not enough keys. For
> what it's worth, here are the words taken
> from the line as optional entries:
>     option1 := dark
>     option2 := light
> Faulty line of input (13th) is
>     option1 := dark option2 := light

> *Error* type 1061 ****
> Came across a faulty line of input in
> "fault-1061a-WordOrPhrase.txt".
> The line contained an optional argument
> that started with a quote character ("") but
> had nothing else on the line, not even
> another "".
> Faulty line of input (21st) is
>     back 1000 portal 0 duct area:=""

> *Error* type 1061 ****
> Came across a faulty line of input in
> "fault-1061b-WordOrPhrase.txt".
> The line contained an optional argument
> that started with a quote character ('') but
> had nothing else on the line, not even
> another '.

```

```
> Faulty line of input (21st) is
>     back 1000    portal 0    duct      area  :=  '
> *Error* type 1062 *****
> Came across a faulty line of input in
> "fault-1062-WordOrPhrase.txt".
> The line contained an optional argument
> that started with a quote character ("") but
> had no matching quote character at the end
> of the expression. Please add one.
> Faulty line of input (21st) is
>     back 1000    portal 0    area:="    duct

> *Error* type 1063 *****
> Came across a faulty line of input in
> "fault-1063-WordOrPhrase.txt".
> The line contained an optional argument
> that started with a quote character ("") but
> there was nothing between it and the closing
> quote character. Please add something.
> Faulty line of input (21st) is
>     back 1000    portal 0    duct      area:="  "

> *Error* type 3001 *****
> In the file named "fault-3001a-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting traffic in all the routes in
> the block. Please edit the file to resolve the
> the clash by removing one of the entries.
> Faulty lines of input (40th and 41st) are
>     standstill    allroutes 165 veh/lane-km 10000 13000
>     moving        allroutes 90

> *Error* type 3001 *****
> In the file named "fault-3001b-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting traffic in all the routes in
> the block. Please edit the file to resolve the
> the clash by removing one of the entries.
> Faulty lines of input (41st and 42nd) are
>     moving        allroutes 90
>     standstill   allroutes 165 veh/lane-km 10000 13000

> *Error* type 3001 *****
> In the file named "fault-3001c-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting traffic in all the routes in
> the block. Please edit the file to resolve the
> the clash by removing one of the entries.
> Faulty lines of input (41st and 42nd) are
>     moving        allroutes 90
>     moving        allroutes 90

> *Error* type 3001 *****
```

```
> In the file named "fault-3001d-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting traffic in all the routes in
> the block. Please edit the file to resolve the
> the clash by removing one of the entries.
> Faulty lines of input (41st and 42nd) are
>   standstill allroutes 165 veh/lane-km 10000 13000
>   standstill allroutes 165 veh/lane-km 10000 13000

> *Error* type 3002 ****
> In the file named "fault-3002a-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> an entry setting traffic in all routes and an
> entry setting traffic in route ebmain.
> If you have "standstill allroutes" entry you
> can't set traffic in all routes and then set
> traffic routes. Please edit the file to resolve
> the clash by removing one of the entries.
> Faulty lines of input (40th and 41st) are
>   standstill allroutes 165 PCU/lane-km 10000 13000
>   standstill ebmain 165 PCU/lane-km 10000 13000

> *Error* type 3002 ****
> In the file named "fault-3002b-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> an entry setting traffic in all routes and an
> entry setting traffic in route ebmain.
> If you have "standstill allroutes" entry you
> can't set traffic in all routes and then set
> traffic routes. Please edit the file to resolve
> the clash by removing one of the entries.
> Faulty lines of input (40th and 41st) are
>   standstill allroutes 165 PCU/lane-km 10000 13000
>   moving ebmain 50

> *Error* type 3003 ****
> In the file named "fault-3003-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting moving traffic twice
> in the same route (ebbranch).
> Please edit the file to resolve the clash
> by removing one of the entries.
> Faulty lines of input (42nd and 43rd) are
>   moving ebbranch 90
>   moving ebbranch 90

> *Error* type 3004 ****
> In the file named "fault-3004-TrafficClash.txt",
> the trafficsteady "eastbound" block had
> two entries setting stationary and moving
> traffic in the same route (ebbranch).
> Please edit the file to resolve the clash
> by removing one of the entries.
> Faulty lines of input (40th and 42nd) are
```

```

> standstill ebbranch 165 veh/lane-km 10000 13000
> moving ebbranch 90

> *Error* type 3021 ****
> In the file named "fault-3021a-MovingSpeeds.txt",
> the "trafficsteady eastbound" block had a
> speed over 120 km/h (it was 161.0 km/h).
> Please edit the file so that the speed is at
> or below 120 km/h or add the optional
> argument "hoons := allowed" (without quotes) to
> the line of input.
> Faulty line of input (43rd) is
>     moving EBbranch 161

> *Error* type 3021 ****
> In the file named "fault-3021b-MovingSpeeds.txt",
> the "trafficsteady eastbound" block had a
> speed over 120 km/h (it was 74.57 mph/120.0 km/h).
> Please edit the file so that the speed is at
> or below 74.56 mph (120 km/h) or add the optional
> argument "hoons := allowed" (without quotes) to
> the line of input.
> Faulty line of input (41st) is
>     moving EBbranch 74.57

> *Error* type 3022 ****
> In the file named "fault-3022-MovingSpeeds.txt",
> the "trafficsteady eastbound" block had a
> speed near zero instead of a "standstill" keyword
> giving the density of stationary traffic. Please
> comment out the "moving" line of entry and add a
> "standstill" line of entry starting with a density
> of traffic and the units it is given in, e.g.
>     standstill ebmain 165 PCU/lane-km
>     standstill ebmain 130 veh/lane-km
>     standstill ebmain 265 PCU/lane-mile
> or
>     standstill ebmain 209 veh/lane-mile
> You can use other numbers if you can justify them
> to your project's reviewers: 165 PCU/lane-km
> is just the value recommended by PIARC.
> Note that the rest of the line must include the
> route chainage that traffic starts at and the
> route that chainage traffic stops at, e.g.
>     standstill ebmain 165 PCU/lane-km    10000 12860
> Faulty line of input (41st) is
>     moving EBmain 0.1 # km/h

> *Error* type 3041 ****
> In the file named "fault-3041a-StationaryDists.txt",
> the "trafficsteady eastbound" block had
> stationary traffic with a traffic density above
> 165.0 pcu/lane-km (it was 167.0 pcu/lane-km).
> Please edit the file so that the density is at

```

```

> or below 165.0 pcu/lane-km.
> Alternatively, add the optional argument
> "highdensity := allowed" to the line.
> Faulty line of input (37th) is
>     standstill EBbranch 167 PCU/lane-km    10000 13000

> *Error* type 3041 ****
> In the file named "fault-3041b-StationaryDists.txt.txt",
> the "trafficsteady eastbound" block had
> stationary traffic with a traffic density above
> 265.54 veh/lane-mile (it was 267.0 veh/lane-mile).
> Please edit the file so that the density is at
> or below 265.54 veh/lane-mile (165.0 veh/lane-km).
> Alternatively, add the optional argument
> "highdensity := allowed" to the line.
> Faulty line of input (41st) is
>     standstill EBbranch 267 veh/lane-mile    10000 13000

> *Error* type 3042 ****
> In the file named "fault-3042-StationaryDists.txt",
> the "trafficsteady eastbound" block tried to
> limit the extent of traffic in route "ebbranch",
> but the extent of the traffic is too short to
> matter (less than 0.1 m).
> Please edit the file to either remove the line
> or adjust the chainages.
> Faulty line of input (40th) is
>     standstill ebbranch 165 veh/lane-km 10000 10000

> *Error* type 3043 ****
> In the file named "fault-3043-StationaryDists.txt",
> the "trafficsteady "eastbound" block tried to
> set traffic in route "ebmain" but the
> route definition does not have a count of
> lanes for it. Stationary traffic cannot be
> set without knowing lane counts.
> Please edit the file to add an entry for the
> count of lanes to the definition of route
> "ebmain".
> Relevant line of input (98th) is
>     begin route EBmain

> *Error* type 3061 ****
> The file named "fault-3061a-ProcessSESData.txt" did not
> set a suitable value for atmospheric air pressure
> in form 1F of one of its "SESDATA" blocks.
> Suitable values are either an atmospheric
> pressure in Pascals or the phrase "p_atm"
> (without double quotes) to use the atmospheric
> pressure used in the method of characteristics
> calculation (101325.0 Pa).
> Faulty line of input (194th) is
>     form1F 25.      19.      p_am      23.5      18.73      28.78      20.17      4.6

```

```

> *Error* type 3061 ****
> The file named "fault-3061b-ProcessSESData.txt" did not
> set a suitable value for atmospheric air pressure
> in form 1F of one of its "SESdata" blocks.
> Suitable values are either an atmospheric
> pressure in inches of mercury or the phrase "p_atm"
> (without double quotes) to use the atmospheric
> pressure used in the method of characteristics
> calculation (29.921 in. Hg).
> Faulty line of input (167th) is
>     form1F 25.      19.      p_am      23.5      18.73      28.78      20.17      4.6

> *Error* type 3062 ****
> The file named "fault-3062-ProcessSESData.txt" had
> conflicting SES input. The tunnel named
> "westbound1" had an "SESpragmat" optional
> entry that told Hobyah to get the SES stack
> heights from the gradients in route "nonexistent".
> Unfortunately that route does not exist.
> The following routes exist:
>     ebmain, ebbbranch and wbmain.
> Please edit the file to fix the conflict,
> either by removing the optional argument in
> the "SESpragmat" line or adding a definition
> for route "nonexistent".
> Problematic line of input (53rd) is
>     SESSpragmat 301 line route:= nonexistent

> *Error* type 3063 ****
> The file named "fault-3063-ProcessSESData.txt" had an
> SES fire definition (form 4) in which the fire
> location was not in the tunnels of the route.
> The fire was located at chainage 9900 m and the
> tunnels in the route start at 10000 m and stop
> at 12860 m.
> Please edit the fire definition to place the
> fire inside the tunnels on the route.
> Faulty line of input (205th) is
>     4A_locn    wbmain    9900    A2W design fire

> *Error* type 3064 ****
> The file named "fault-3064-ProcessSESData.txt" had an
> SES fire definition (form 4) in which the fire
> location was outside the tunnel that you want
> to put it in (tunnel "eastbound1"). The fire
> was located at distance 12200 m.
> The back end of the tunnel is at 10020 m and the
> forward end is at 12100 m.
> Please edit the fire definition to place the
> fire inside the tunnel or change to another
> tunnel.
> Faulty line of input (207th) is
>     4A_locn    eastbound1    12200    A2W design fire

```

```
> *Error* type 3065 ****
> The file named "fault-3065-ProcessSESData.txt" had
> an SES fire definition (form 4) in which the fire
> location was placed in tunnel "xp1").
> That tunnel was turned into a vent segment in
> the SES input file, as per the instruction in the
> "SESpragmat" keyword. You can't put fires into
> SES vent segments, which SES will complain about.
> Please either edit the fire definition to place
> it in a tunnel that maps to an SES line segment
> or tell Hobyah to turn the tunnel into a line
> segment.
> Conflicting lines of input (64th and 204th) are
>     SESSpragmat 901 vent # When writing SES files, number the segments
>     4A_locn  XP1 15 Fire in a cross-passage

> *Error* type 3066 ****
> The file named "fault-3066-ProcessSESData.txt" had an
> incomplete SES fire definition (form 4). You
> included an entry starting with "4b_heat" to
> set a fire size, start time and stop time but
> did not include an entry starting with "4A_locn"
> to set the location and description of the fire.
> Please either add one or remove the line with
> "4B_heat" from the SESdata block.
> Relevant line of input (186th) is
> begin SESdata

> *Error* type 3067 ****
> The file named "fault-3067-ProcessSESData.txt" had an
> incomplete SES fire definition (form 4). You
> included an entry starting with "4A_locn" to
> set the location and description but did not
> include an entry starting with "4b_heat" to
> set the fire size, start time and stop time.
> Please either add one or remove the line with
> "4A_locn" from the SESdata block.
> Relevant line of input (185th) is
> begin SESdata

> *Error* type 3068 ****
> The file named "fault-3068-ProcessSESData.txt" had an
> incomplete SES fire definition (form 4). You
> included an entry starting with "4c_flames" to
> set the radiative heat transfer properties of
> the fire but did not include an entry starting
> with "4A_locn" to set the location and description
> of the fire. Please either add one or remove the
> line with "4C_flames" from the SESdata block.
> Relevant line of input (186th) is
> begin SESdata

> *Error* type 3069 ****
> The file named "fault-3069-ProcessSESData.txt" had an
```

```

> incomplete SES fire definition (form 4). You
> included an entries starting with "4A_locn" and
> "4B_heat" to set the location, description, heat
> release rate and start and stop times but did not
> include an entry starting with "4c_flames" to
> set the radiative heat transfer properties of
> the fire.
> Please either add one or remove the lines with
> "4A_locn" and "4B_heat" from the SESdata block.
> Relevant line of input (186th) is
>   begin SESdata

> *Error* type 3070 ****
> The file named "fault-3070-ProcessSESData.txt" had an
> incorrect SES jet fan definition (form 7C).
> You tried to create a jet fan definition for
> SES from a unidirectional jet fan in Hobyah
> (the jet fan type called "jftype1").
> You asked for the properties of that jet fan
> type when running it in reverse, but properties
> of a unidirectional jet fan running in reverse
> are not defined. Please edit the SES jet fan
> to correct the error.
> Faulty line of input (203rd) is
>   form7C1 JFtype1 reverse 15 1800

> *Error* type 3071 ****
> The file named "fault-3071-ProcessSESData.txt" had an
> invalid SES axial fan definition (forms 7A
> and 7B). You tried to copy over data from
> Hobyah fan characteristic definition named
> "fan1". Unfortunately it uses a datasource
> to define its characteristic, not the four
> pairs of flow and pressure that SES needs
> in its input files.
> Please edit the file to define one or more
> fan characteristics using the "form_7B" way
> of defining flow/pressure points and use
> those characteristics when exporting fan
> characteristics to SES.
> Faulty lines of input (204th and 182nd) are
>   form7AB fan1
>   Datasource fan_curve2 Q P_tot direction:= reverse

> *Error* type 3072 ****
> The file named "fault-3072-ProcessSESData.txt" had
> conflicting SES input. The tunnel named
> "xp1" had an "SESpragmat" optional
> entry that told Hobyah to get the SES stack
> heights from the gradients in route "wemain".
> Unfortunately that tunnel does not run through
> this route (it is not in any routes).
> Please edit the file to fix the conflict,
> either by removing the optional argument in

```

```
> the "SESpragmat" line or adding a definition
> for route "wbmain".
> Problematic line of input (64th) is
>     SESpragmat 901 vent route:= wbmain

> *Error* type 3073 ****
> The file named "fault-3073-ProcessSESData.txt" had
> conflicting SES input. The tunnel named
> "eastbound1" had an "SESpragmat" optional
> entry that told Hobyah to get the SES stack
> heights from the gradients in route "wbmain".
> Unfortunately that tunnel does not run through
> this route, it pass through the following routes:
>     ebmain and ebbranch.
> Please edit the file to fix the conflict,
> either by removing the optional argument in
> the "SESpragmat" line or adding a definition
> for route "wbmain".
> Problematic line of input (184th) is
>     begin SESdata

> *Error* type 3074 ****
> The file named "fault-3074-ProcessSESData.txt" had
> conflicting SES input. The tunnel named
> "eastbound1" passes through the following
> routes:
>     ebmain and ebbranch.
> Hobyah cannot figure out which route to take the
> stack heights from when it builds the SES file.
> Please edit the file to fix the conflict by
> adding the optional argument "route" to the
> "SESpragmat" entry in the tunnel definition
> and choosing one of the above routes.
> Problematic line of input (34th) is
>     SESpragmat 101 line

> *Error* type 3075 ****
> The file named "fault-3075-ProcessSESData.txt" had
> conflicting SES input. The segments in
> tunnel "eastbound1" were converted
> into SES vent segments in the "SESdata"
> block but this tunnel is in route "ebmain".
> SES vent segments cannot be put into routes.
> Please edit the file to fix the conflict,
> either by changing the SES segment type
> to line segments or removing the tunnel
> from all routes.

> Conflicting lines of input (208th and 35th) are
>     form8 EBmain orientation:=reverse
>     SESpragmat 101 vent route:= ebmain

> *Error* type 3076 ****
> Cannot generate an SES file from "fault-3076-ProcessSESData.txt"
```

```
> because the file has 2 independent tunnel
> networks. SES can only calculate one network
> of tunnels. If you try to run an SES file
> based on this Hobyah file, SES will raise error
> number 128 (sections are not all connected).
> Please either split the tunnel networks into
> different Hobyah input files or connect the
> networks together so that SES can treat them
> as one network.

> *Error* type 3121 ****
> Came across faulty input in "fault-3121a-Process8CLists.txt".
> In route "wb" there was a block of lane
> counts, one of which was less than one (0.5).
> Please edit the file to set the count of lanes
> to be integers of one or more.
> Faulty line of input (44th) is
>          0.5           12880

> *Error* type 3121 ****
> Came across faulty input in "fault-3121b-Process8CLists.txt".
> In route "wb" there was a block of track
> radii, one of which was less than -math.inf (-5000.0).
> Please edit the file to set the track radii
> to be real numbers between -math.inf and
> +math.inf.
> Faulty line of input (44th) is
>          -5000           12880

> *Error* type 3121 ****
> Came across faulty input in "fault-3121c-Process8CLists.txt".
> In route "wb" there was a block of energy
> sectors, one of which was less than zero (-1.0).
> Please edit the file to set the energy sectors
> to be integer values zero or above.
> Faulty line of input (45th) is
>          -1           12880

> *Error* type 3121 ****
> Came across faulty input in "fault-3121d-Process8CLists.txt".
> In route "wb" there was a block of coasting
> switches, one of which was less than zero (-1.0).
> Please edit the file to set the coasting switches
> to be integer values zero or one.
> Faulty line of input (45th) is
>          -1           12880

> *Error* type 3121 ****
> Came across faulty input in "fault-3121e-Process8CLists.txt".
> In route "wb" there was a block of regen
> fractions, one of which was less than zero (-0.3).
> Please edit the file to set the regen fractions
> to be real numbers between zero and one.
> Faulty line of input (44th) is
```

```
>          -0.3      12880

> *Error* type 3122 ****
> Came across faulty input in "fault-3122a-Process8CLists.txt".
> In route "wb" there was a block of coasting
> switches, one of which was more than one (2.0).
> Please edit the file to set the coasting switches
> to be integer values zero or one.
> Faulty line of input (44th) is
>          2      12880

> *Error* type 3122 ****
> Came across faulty input in "fault-3122b-Process8CLists.txt".
> In route "wb" there was a block of regen
> fractions, one of which was more than one (1.3).
> Please edit the file to set the regen fractions
> to be real numbers between zero and one.
> Faulty line of input (44th) is
>          1.3      12880

> *Error* type 3123 ****
> Came across faulty input in "fault-3123a-Process8CLists.txt".
> In route "wb" there was a block of lane
> counts, one of which was not an integer (1.5).
> Please edit the file to set the count of lanes
> to integers.
> Faulty line of input (44th) is
>          1.5      12880

> *Error* type 3123 ****
> Came across faulty input in "fault-3123b-Process8CLists.txt".
> In route "wb" there was a block of energy
> sectors, one of which was not an integer (1.5).
> Please edit the file to set the energy sectors
> to integers.
> Faulty line of input (44th) is
>          1.5      12880

> *Error* type 3123 ****
> Came across faulty input in "fault-3123c-Process8CLists.txt".
> In route "wb" there was a block of coasting
> switches, one of which was not an integer (0.7).
> Please edit the file to set the coasting switches
> to integers.
> Faulty line of input (44th) is
>          0.7      12880

> *Error* type 3124 ****
> Came across faulty input in "fault-3124-Process8CLists.txt".
> In route "wb" there was a block of lane
> counts. The last count ended at chainage
> 12700.0, which is before the down daylight
> portal of the tunnel complex.
> This means traffic disappears in the middle
```

```

> of the tunnel, which is bad. Please edit
> the file to set lanes all the way to the
> tunnel end (it's at 12880.0).
> Faulty line of input (46th) is
>          2          12700

> *Error* type 5002 *****
> A binary file has a version string (the
> first thing in the binary file) that was
> not a string. The .sbn file was probably
> not created by SESconv.py.
> The version string should have been some-
> thing like "SESconv.py binary version 12",
> but it was of class 'tuple' and contained:
>   ("("This is the tuple's 1st element", 'This is the 2nd')")
> The faulty file is "S-5002a-version-wrong-type.sbn".

> *Error* type 5002 *****
> A binary file has a version string (the
> first thing in the binary file) that was
> not a string. The .sbn file was probably
> not created by SESconv.py.
> The version string should have been some-
> thing like "SESconv.py binary version 12",
> but it was of class 'tuple' and started
> with:
>   ("("This is the tuple's 1st element", 'This is the 2nd', 'T...")
> The faulty file is "S-5002b-version-wrong-type.sbn".

> *Error* type 5003 *****
> A binary file has a version string (the
> first thing in the binary file) that
> looks like a Hobyah.py version string,
> not an SESconv.py version string.
> The version string should have been some-
> thing like "SESconv.py binary version 12",
> but it was
>   "Hobyah.py binary version 6"
> Someone probably renamed a Hobyah.py
> ".hbn" file to end with ".sbn" instead.
> The faulty file is "S-5003-renamed-hbn-file.sbn".

> *Error* type 5004 *****
> A binary file has a version string that
> didn't match the required form (the words
> "SESconv.py binary version XX") where XX is
> an integer. Instead it was
>   "Little dog Turpie barks so that I cannot sleep nor slumber"
> The faulty file is "S-5004-version-wrong-str.sbn".
> It was not created by SESconv.py, it just
> happens to be a pickle file that started
> with a string.

> *Error* type 5005 *****

```

```
> A binary file has a version number that is
> too old for this program to process correctly.
> The file is of binary version 2 and this
> program can only handle files of version 12.
> Please reprocess the SES output file with a
> version of SESconv.py that generates binary
> files of version 12.
> The faulty file is "S-5005-version-too-old.sbn".

> *Error* type 5006 ****
> A binary file has a version number that is
> too new for this program to process correctly.
> The file is of binary version 9999 and this
> program can only handle files of version 12.
> Please update your version of this program
> and its dependent modules.
> The faulty file is "S-5006-version-too-new.sbn".

> *Error* type 5022 ****
> The binary file "S-5022-ended-early.sbn"
> ended unexpectedly. Please check whether
> it was generated from SESconv.py.
> The pickle.load() function failed while
> reading the transient data.

> *Error* type 5023 ****
> The binary file "S-5023-misc-unpickling-error.sbn"
> failed during a pickle.read() action.
> The most common cause of this error is a
> new version of numpy or pandas being fed
> array data written by an earlier version
> of numpy or pandas, in which case the way
> to solve the problem is to run SESconv.py
> on the SES output file again.
> It failed while reading the fixed data
> with the following traceback message:
>   'utf-8' codec can't decode byte 0x94 in position 6: invalid start byte

> *Error* type 5024 ****
> The binary file "S-5024-unexpected-contents.sbn"
> has a block of data of unexpected length
> for the transient data.
> Expected 76 items but read 4 instead.

> *Error* type 5041 ****
> Came across an invalid plot type in "fault-5041-CheckProperty.txt".
> The plot type was "flowrate", to be plotted
> from an SES output file. Unfortunately that
> property is not valid for SES files, you can
> only plot the following:
>   dp, dp_indiv, qses, vses, wall_temp, db, w,
>   sens, lat, sens_pul, lat_pul, annulus, qcold,
>   qwarm, vcold, vwarm, route, type, chainage,
>   time, speed, qtrains, accel, aerodrag, cd,
```

```

> te, motoramps, lineamps, flywh_rpm, acceltemp,
> deceltemp, pwr_all, heat2air, svseffic, mode,
> pwr_aux, pwr_prop, pwr_regen, pwr_flywh, pwr_accel,
> pwr_decel, pwr_mech, heat_adm, heat_sens,
> heat_lat, effic1, svsregen1, dp_up, area_up,
> dp_skin, dp_down, area_down, trains, fires,
> jetfans, elevations, gradients, stacks, stackgrads,
> speedlimits, radius, sectors, coasting, svsregen2,
> sections, segments, subsegs, sublength, area,
> fireseg, perimeter, roughness, darcy, fanning,
> wetted, meandb_am, meandb_pm, meandb_off,
> meanw_am, meanw_pm, meanw_off, wallt_am, wallt_pm,
> wallt_used, misc_sens, misc_lat, steady_sens,
> steady_lat, ground_sens, airex_sens, airex_lat,
> hvac_sens, hvac_lat, hvac_tot, pipetemp, pipe_ht,
> -dp, -qses, -vses, -qcold, -qwarm, -vcold
> and -vwarm.
> Please edit the file to correct it.
> Faulty line of input (27th) is
>         transient flowrate myfile 101-2m

> *Error* type 5043 *****
> Came across an invalid plot type in "fault-5043-CheckProperty.txt".
> The plot type was "speed", to be plotted
> on a profile plot.
> Unfortunately train speeds can only be
> plotted using the transient keyword. If you
> want the X-axis values to be chainage instead
> of time, add the "xaxis:=chainage" optional
> argument.
> If you want to plot train speed against
> chainage, please adjust your line to read
> as follows:
>         transient speed myfile train1@0 xaxis:=chainage
> Faulty line of input (28th) is
>         profile speed myfile train1@0

> *Error* type 5045 *****
> Came across an invalid plot type in "fault-5045-CheckProperty.txt".
> The plot type was "DP", to be plotted
> from an SES output file. Unfortunately the
> supplementary print option (form 1C) was
> set to 1 so these results were not in the
> printed output.
> That option needs to be set to 3 or 5.
> Please either rerun SES file "SES-041-fire-1.ses"
> with the option set correctly and reprocess its
> .PRN file with SESconv.py, or remove the curve.
> Faulty line of input (28th) is
>         transient DP myfile 101-2m

> *Error* type 5065 *****
> Tried to plot a curve in "fault-5065-CheckAt.txt",
> but found that first part of the location

```

```
> identifier "shoe1@200.0" was not an allowed
> word, it was "shoe1". The only words allowed
> are:
>     route and train.
> Please edit the file to correct it.
> Faulty line of input (28th) is
>         transient qcold myfile shoe1@200

> *Error* type 5066 ****
> Tried to plot a curve in "fault-5066-CheckAt.txt",
> but found that first part of the location
> identifier "route@200.0" only contained the
> word "route" without a route number before
> the "@". Please edit the file to add a
> route number.
> Faulty line of input (28th) is
>         transient qcold myfile route@200

> *Error* type 5067 ****
> Tried to plot a curve in "fault-5067-CheckAt.txt",
> but found that first part of the location
> identifier "routeXX@200.0" did not have a
> route number after it, it had "XX"
> instead. Please edit the file and change
> it to a suitable route number.
> Faulty line of input (28th) is
>         transient qcold myfile routeXX@200

> *Error* type 5068 ****
> Tried to plot a curve in "fault-5068-CheckAt.txt",
> but found that first part of the location
> identifier "route5ud@200.0" had extra and
> unnecessary detail (a subpoint identifier).
> Please remove "ud" from just before the "@".
> Faulty line of input (28th) is
>         transient qcold myfile route5ud@200

> *Error* type 5069 ****
> Tried to plot a curve in "fault-5069-CheckAt.txt",
> but found that first part of the location
> identifier "route1xy@200.0" tried to plot at a
> subpoint identifier that does not exist, "x".
> The only acceptable sub-point identifiers are
> "u" (up end), "m", (midpoint) and "d" (down
> end) either alone or in any combination.
> Please remove "x" from the line (and check
> for any other unacceptable letters).
> Faulty line of input (27th) is
>         profile qcold myfile route1xy@200

> *Error* type 5081 ****
> Tried to plot a curve in "fault-5081-TransientAt.txt",
> but found that first part of the location
> identifier "train2020-2.0" did not have a
```

```

> valid train number after it, it had "202"
> instead. Valid entries are in the range
> 0 to 99.
> Please edit the file to correct it.
> Faulty line of input (28th) is
>         transient qcold myfile train202@-2

> *Error* type 5084 ****
> Tried to plot a curve in "fault-5084-TransientAt.txt",
> at a distance along route 1 but the
> location in the route (-9999.0) is below
> the route origin (3281.93).
> Please edit the file to correct it.
> Faulty line of input (28th) is
>         transient qcold myfile route1@-9999

> *Error* type 5085 ****
> Tried to plot a curve in "fault-5085-TransientAt.txt",
> at a distance along route 1 but the
> location in the route (99999.0) is above
> the end of the route (8205.64).
> Please edit the file to correct it.
> Faulty line of input (28th) is
>         transient qcold myfile route1@99999

> *Error* type 5087 ****
> Tried to plot a curve in "fault-5087-TransientAt.txt",
> but found that the segment identifier "-101"
> was zero or negative. Please edit the file
> to give a positive segment number.

> Faulty line of input (29th) is
>         transient qses myfile -101

> *Error* type 5088 ****
> Tried to plot a curve in "fault-5088-TransientAt.txt",
> but found that the location identifier "-27-10"
> was not a valid segment number (e.g. 123),
> segment-subseg combination (e.g. 104-5,
> 104-5b, 104-5m or 104-5f), or section
> number (e.g. sec204). Please edit the
> file to give a valid SES location term.

> Faulty line of input (36th) is
>         transient qcold myfile -27-10

> *Error* type 5089 ****
> Tried to plot a curve in "fault-5089-TransientAt.txt",
> but found that the location identifier "sec101"
> was too vague: it defined a section number for
> a plot, but a segment number was needed.
> Please edit the file to give a valid segment
> number here e.g., 101.
> Faulty line of input (28th) is

```

```
> transient qses myfile sec101

> *Error* type 5091 ****
> Tried to plot a curve in "fault-5091-TransientAt.txt",
> but found that the location identifier "sec101"
> was too vague: it defined a section number for
> a plot, but a segment and subsegment number was
> needed. Please edit the file to give a valid
> segment and subsegment number e.g., 304-12.
> Faulty line of input (28th) is
>         transient DB myfile sec101

> *Error* type 5092 ****
> Tried to plot a curve in "fault-5092-TransientAt.txt",
> but found that the location identifier "101"
> was too vague: it defined a segment number for
> a plot, but a segment and subsegment number was
> needed. Please edit the file to give a valid
> segment and subsegment number e.g., 304-12.
> Faulty line of input (28th) is
>         transient DB myfile 101

> *Error* type 5094 ****
> Tried to plot a curve in "fault-5094-TransientAt.txt",
> but found that the location identifier "sec101"
> was too vague: it defined a section number for
> a plot, but a segment number, subsegment number
> and a location inside the subsegment (back end,
> midpoint or forward end) were needed (known in
> Hobyah as a subpoint location).
> Please edit the file to give a valid subpoint
> location such as 304-12b, 304-12f or 304-12m
> (these are the back end, forward end and midpoint
> of the 12th subsegment of segment 504 respectively).
> Note that you can use "304-12" and the plot will
> default to the midpoint of the segment. That
> can lead to trouble in some circumstances so
> please use it sparingly.
> Faulty line of input (29th) is
>         transient qcold myfile sec101

> *Error* type 5095 ****
> Tried to plot a curve in "fault-5095-TransientAt.txt",
> but found that the location identifier "101"
> was too vague: it defined a segment number for
> a plot, but a segment number, subsegment number
> and a location inside the subsegment (back end,
> midpoint or forward end) were needed (known in
> Hobyah as a subpoint location).
> Please edit the file to give a valid subpoint
> location such as 304-12b, 304-12f or 304-12m
> (these are the back end, forward end and midpoint
> of the 12th subsegment of segment 504 respectively).
> Note that you can use "304-12" and the plot will
```

```
> default to the midpoint of the segment. That
> can lead to trouble in some circumstances so
> please use it sparingly.
> Faulty line of input (29th) is
>         transient qcold myfile 101

> *Error* type 5101 *****
> Tried to plot a curve in "fault-5101-CheckSegNumber.txt",
> but found that the identifier "91-2b"
> referenced a segment (91) that is not in
> SES file "SES-041-fire-1.ses".
> Please edit the file to correct this.
> For what it is worth this is a list of
> the segment number(s) used in that file:
>   54, 101, 102, 103, 104, 105, 106, 107, 108,
>   120, 121, 122, 123, 124, 125, 126, 202, 203,
>   204, 205, 206, 207, 208, 220, 221, 222, 223,
>   224, 225, 226, 301, 302, 303, 304, 305, 306,
>   307, 308, 311, 312, 313, 314, 315, 316, 317,
>   318, 332, 333, 334, 335, 342, 343, 344, 345,
>   352, 353, 354, 355, 361, 362, 363, 364, 365,
>   371, 372, 373, 831, 832, 833, 834, 835, 836,
>   837, 838, 931, 932, 933, 934, 935 and 936.
> Faulty line of input (27th) is
>         transient qses myfile 91-2b

> *Error* type 5121 *****
> Tried to plot a curve in "fault-5121a-CheckSubNumber.txt",
> but found that the identifier "101-0"
> referenced a subsegment (0) that does not
> exist in SES file "SES-041-fire-1.ses".
> Segment 101 has subsegments numbered from
> 1 to 10.
> Please edit the file to correct the
> subsegment number.
> Faulty line of input (27th) is
>         transient db SESfile 101-0

> *Error* type 5121 *****
> Tried to plot a curve in "fault-5121b-CheckSubNumber.txt",
> but found that the identifier "101-112"
> referenced a subsegment (112) that does not
> exist in SES file "SES-041-fire-1.ses".
> Segment 101 has subsegments numbered from
> 1 to 10.
> Please edit the file to correct the
> subsegment number.
> Faulty line of input (27th) is
>         transient db SESfile 101-112

> *Error* type 5141 *****
> Tried to plot a curve in "fault-5141-CheckSecNumber.txt",
> but found that the identifier "sec87"
> referenced a section (87) that is not in
```

```
> SES file "SES-043-fire-3.ses".
> Please edit the file to correct this.
> For what it is worth this is a list of the
> section numbers used in that file:
>   101, 102, 103, 104, 105, 106, 107, 108, 120,
>   121, 122, 123, 124, 125, 126, 202, 203, 204,
>   205, 206, 207, 208, 220, 221, 222, 223, 224,
>   225, 226, 301, 302, 303, 304, 305, 306, 307,
>   308, 311, 312, 313, 314, 315, 316, 317, 318,
>   332, 333, 334, 335, 342, 343, 344, 345, 352,
>   353, 354, 355, 361, 362, 363, 364, 365, 371,
>   372, 373, 831, 832, 833, 834, 835, 836, 837,
>   838, 931, 932, 933, 934, 935 and 936.
> Faulty line of input (27th) is
>           transient DP SESfile sec87

> *Error* type 5161 ****
> Tried to plot a curve in "fault-5161-CheckRouteNumber.txt",
> but found that the identifier "route5@200.0"
> referenced an SES route (5) in SES
> file "SES-056-no-routes.ses".
> That SES file does not have routes in it, because
> its train simulation option is not 1, 2 or 3.
> Please edit "fault-5161-CheckRouteNumber.txt" to
> correct the curve definition.
> Faulty line of input (27th) is
>           transient qcold myfile route5@200

> *Error* type 5162 ****
> Tried to plot a curve in "fault-5162-CheckRouteNumber.txt",
> but found that the identifier "route5@200.0"
> referenced an SES route (5) in SES
> file "SES-041-fire-1.ses".
> That route does not exist in the SES file,
> it only has routes 1 to 2.
> Please edit "fault-5162-CheckRouteNumber.txt" to
> correct the curve definition.
> Faulty line of input (28th) is
>           transient qcold myfile route5@200

> *Error* type 5241 ****
> Tried to plot a curve in "fault-5241-CheckZoneStuff.txt",
> but found that the location identifier "zone"
> did not give a zone number. Please edit the
> file to add a zone number (the file has 3 zones).
> Faulty line of input (27th) is
>           transient airex_sens myfile zone

> *Error* type 5242 ****
> Tried to plot a curve in "fault-5242-CheckZoneStuff.txt",
> but found that the location identifier "zonedud"
> did not have a zone number in it, it had "dud"
> instead. Please edit the file and change
> it to a suitable zone number for this file
```

```

> (1 to 3).
> Faulty line of input (28th) is
>         transient airex_sens myfile zonedud

> *Error* type 5243 ****
> Tried to plot a curve in "fault-5243a-CheckZoneStuff.txt",
> but found that the identifier "zone-2"
> referenced zone -2 in the SES file
>   "SES-005-impl-impl-5.ses".
> That zone does not exist in the SES file,
> it only has zones 1 to 3.
> Please edit "fault-5243a-CheckZoneStuff.txt" to
> correct the curve definition.
> Faulty line of input (28th) is
>         transient airex_sens myfile zone-2

> *Error* type 5243 ****
> Tried to plot a curve in "fault-5243b-CheckZoneStuff.txt",
> but found that the identifier "zone4"
> referenced zone 4 in the SES file
>   "SES-005-impl-impl-5.ses".
> That zone does not exist in the SES file,
> it only has zones 1 to 3.
> Please edit "fault-5243b-CheckZoneStuff.txt" to
> correct the curve definition.
> Faulty line of input (28th) is
>         transient airex_sens myfile zone4

> *Error* type 5244 ****
> Tried to plot a curve in "fault-5244a-CheckZoneStuff.txt",
> but found that the identifier "zone2"
> referenced zone 2 in SES file
>   "SES-005-impl-impl-5.ses".
> That zone exists in the SES file, but is not a
> controlled zone (type 1), it is an uncontrolled
> zone (type 2). Please edit "fault-5244a-CheckZoneStuff.txt"
> to correct the curve definition so that it plots
> in a controlled zone or remove the curve definition.
> Faulty line of input (28th) is
>         transient airex_sens myfile zone2

> *Error* type 5244 ****
> Tried to plot a curve in "fault-5244b-CheckZoneStuff.txt",
> but found that the identifier "zone3"
> referenced zone 3 in SES file
>   "SES-005-impl-impl-5.ses".
> That zone exists in the SES file, but is not a
> controlled zone (type 1), it is a non-inertial
> zone (type 3). Please edit "fault-5244b-CheckZoneStuff.txt"
> to correct the curve definition so that it plots
> in a controlled zone or remove the curve definition.
> Faulty line of input (28th) is
>         transient airex_sens myfile zone3

```

```

> *Error* type 5245 ****
> Tried to plot a curve in "fault-5245-CheckZoneStuff.txt",
> in zone 1 in SES
> file "SES-005-impl-impl-5.ses".
> That zone exists in the SES file, but the
> property you want to plot is not a valid
> property to plot across an entire controlled
> zone. The only valid properties are:
> misc_sens, misc_lat, steady_sens, steady_lat,
> ground_sens, airex_sens, airex_lat, hvac_sens,
> hvac_lat and hvac_tot.
> Please edit "fault-5245-CheckZoneStuff.txt" to
> correct the curve definition.
> Faulty line of input (28th) is
>         transient meandb_AM myfile zone1

> *Error* type 5261 ****
> Tried to plot a pressure profile in "fault-5261-RouteAt.txt",
> but found that the optional argument "datum"
> (which gives a segment number to take a
> pressure from to offset the pressure profile
> by) gave a dud segment number (762).
> Please edit the file to give a valid segment
> number or remove the optional entry.
> Faulty line of input (29th) is
>         profile DP myfile route1@600 datum := 762

> *Error* type 5262 ****
> Tried to plot a curve in "fault-5262-RouteAt.txt",
> but found that the location identifier was
> invalid. It was "route1" whereas it should
> be something like "route4@120".
> Please edit the file to correct it.
> Faulty line of input (27th) is
>         profile qcold myfile route1

> *Error* type 6001 ****
> Skipping "fault-6001-ProcessPlots.txt", because
> you do not have permission to write to
> its gnuplot file "fault-6001-ProcessPlots.plt" in
> the "ancillaries" folder.

> *Error* type 6021 ****
> Found an invalid keyword in a graph
> block in "fault-6021-ProcessGraph.txt".
> The keyword is "Dud". Please edit
> the file to correct it. For what it
> is worth, the valid keywords are
> begin, bmargin, btmargins, fadata, icons,
> lmargin, lrmargins, margins, profile, property,
> rmargin, tmargin, transient, userdata, verbatim,
> waterfall, x2label, x2range, xlabel, xrange,
> y2label, y2range, ylabel and yrange.
> Faulty line of input (20th) is

```

```

> Dud line of entry

> *Error* type 6041 *****
> Found a valid keyword in a graph after
> the start of the curve definitions in
> "fault-6041-ProcessCurves.txt".
> Unfortunately this is not allowed, as
> it is too complex to process (the curves
> are all lumped into one gnuplot "plot"
> command). Please edit the file to either
> remove it or move it before the 37th line
> of the input file.
> Faulty line of input (38th) is
>      ylabel "Volume flow" # This line should cause an error

> *Error* type 6043 *****
> There was a curve definition for user-defined
> data in "fault-6043-ProcessCurves.txt", but there
> were no blocks of user-defined data and not
> a block of .csv filenames in the file.
> Faulty line of input (37th) is
>      userdata file1 2 3

> *Error* type 6044 *****
> Found an invalid nickname ("block0") for a
> user-defined curve in "fault-6044a-ProcessCurves.txt".
> Please edit the file to correct it. For what
> it is worth these are the names of the blocks
> of user data in the file:
>   block1, block2 and block3.
> And these are the nicknames of the .csv files
> mentioned in the file:
>   1976_g_3a.
> Faulty line of input (44th) is
>      userdata block0 2 3

> *Error* type 6044 *****
> Found an invalid nickname ("block0") for a
> user-defined curve in "fault-6044b-ProcessCurves.txt".
> Please edit the file to correct it. For what it
> is worth these are the nicknames of the blocks
> of user data in the file:
>   block1, block2 and block3.
> Faulty line of input (40th) is
>      userdata block0 2 3

> *Error* type 6044 *****
> Found an invalid nickname ("block0") for a
> user-defined curve in "fault-6044c-ProcessCurves.txt".
> Please edit the file to correct it. For what it
> is worth these are the nicknames of the .csv files
> mentioned in the file:
>   1976_g_3a.
> Faulty line of input (28th) is

```

```

>         userdata block0  2  3

> *Error* type 6046 ****
> Skipping "fault-6046-ProcessCurves.txt", because
> you do not have permission to write to
> one of its data files, "fault-6046-ProcessCurves-p1-g1-c1.txt" in
> the "ancillaries" folder.

> *Error* type 6061 ****
> Found the invalid nickname "calc" in
> a curve in "fault-6061-GetFileData.txt".
> You may have deleted the binary file and
> be running in plot mode instead of in
> calc mode.
> Please change the "runtype plot" option
> in the settings block to "runtype calc"
> to regenerate the binary file.
> Faulty line of input (29th) is
>         transient vcold calc Heysen@490      # Not a valid nickname

> *Error* type 6062 ****
> Found the invalid nickname "dudfile" in
> a curve in "fault-6062-GetFileData.txt".
> Please edit the file to correct it.
> Valid nicknames in this file are as
> follows:
>     validfile.
> Faulty line of input (28th) is
>         transient vcold dudfile Heysen@490      # Not a valid nickname

> *Error* type 6063 ****
> Found an invalid binary file in "fault-6063-GetFileData.txt".
> The binary file "S-6063-dud-producer.sbn" is
> referenced by the nickname "weirdprog" but is
> not of a suitable type; it is of type "SES -1".
> The only valid types come from these programs and
> versions:
>     Hobyah 1, SES 4.10, OpenSES 4.2, OpenSES 4.3,
>     offline-SES 204.2, offline-SES 204.3, offline-SES 204.4,
>     offline-SES 204.5, offline-SES 204.6 and SVS 6.6.2.
> Please remove the line or edit it to use a valid
> type of binary file.
> Faulty line of input (28th) is
>         transient vvarm weirdprog 101-1m

> *Error* type 6064 ****
> Came across a file nickname that was not valid in
> in the plots block of "fault-6064-GetFileData.txt".
> "*name" is a special nickname that can only be
> used inside "filesloops" blocks. You probably
> copied and pasted the graph from a loop definition
> into a page definition. Please change all the
> instances of "*name" outside of "filesloop"
> blocks to valid nicknames. The following are

```

```

> the valid nickname(s) in this file:
>   SESfile.
> Faulty line of input (27th) is
>       transient vwarn *name 101-1m

> *Error* type 6121 *****
> Tried to plot a curve in "fault-6121-CheckAts.txt",
> but found that the location identifier "@120"
> started with the symbol "@", meaning that no entity
> to plot at (tunnel, route, train etc.) was given.
> Please edit the file to correct it.

> Faulty line of input (28th) is
>       transient qcold myfile @120

> *Error* type 6122 *****
> Tried to plot a curve in "fault-6122-CheckAts.txt",
> but found that the location identifier "59300@"
> ended with the symbol "@", meaning that the second
> part of the identifier (distance along an entity or
> time to plot at) was not given.
> Please edit the file to correct it.

> Faulty line of input (27th) is
>       transient DB myfile 59300@

> *Error* type 6123 *****
> Tried to plot a curve in "fault-6123-CheckAts.txt",
> but found that the location identifier "route1@593@00"
> contained more than one "@" symbol.
> Please edit the file to correct it.

> Faulty line of input (27th) is
>       transient DB myfile route1@593@00

> *Error* type 6141 *****
> A curve definition for the user-defined data
> "sample_data" in "fault-6141-PickUserData.txt" was
> given a number (0) for the column to use
> for the X axis but the number is below 1.
> The block of data has 5 columns of data,
> so a number between 1 and 5 is needed.
> Please edit the file to correct it.

> Faulty line of input (39th) is
>       userdata sample_data    0    1

> *Error* type 6142 *****
> A curve definition for the user-defined data
> "sample_data" in "fault-6142-PickUserData.txt" was
> given a number (5) for the column to use
> for the Y axis but it is too high as there are
> only 4 columns in that block.
> Please edit the file to correct it.

```

```
> Faulty line of input (39th) is
>         userdata sample_data    1    5

> *Error* type 6143 *****
> A curve definition for the user-defined data
> "sample_data" in "fault-6143-PickUserData.txt" was
> given the column name "temp1" to use for
> the Y axis but no column with that name was in
> that particular block. Please edit the file to
> correct it. For what it is worth the data block
> contained the following column names:
>     time, y1, y2, y3 and #tr_index.
> Faulty line of input (33rd) is
>         userdata sample_data time temp1

> *Error* type 6161 *****
> Found a problem in a line of entry in a graph
> block in "fault-6161-CheckGnuplotName.txt".
> The file named '''file-with-both-quotes''.png'''
> exists but gnuplot cannot be told to plot it
> because the file name contains both double and
> single quote characters (" and ').
> Please rename the image file and change the line
> in the file.
> Faulty line of input (21st) is
>         filename file-with-both-quotes''.png

> *Error* type 6162 *****
> Found a problem in a line of entry in a graph
> block in "fault-6162-CheckGnuplotName.txt".
> The file named "file-with-quote'.png"
> exists but gnuplot cannot be told to plot it
> because the file name contains a single quote
> character and the file path contains a double
> quote character. The file path is
> /Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/../../"myfolder/
> Please rename the image file or the image path
> so that there is only " or ' (not both) and
> change the line in the file.
> Faulty line of input (22nd) is
>         filename ../../"myfolder/file-with-quote'.png

> *Error* type 6163 *****
> Found a problem in a line of entry in a graph
> block in "fault-6163-CheckGnuplotName.txt".
> The file named 'file-with-double-quote''.png'
> exists but gnuplot cannot be told to plot it
> because the file name contains a double quote
> character and the file path contains a single
> quote character. The file path is
> /Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/../../'myfolder/
> Please rename the image file or the image path
> so that there is only " or ' (not both) and
```

```
> change the line in the file.  
> Faulty line of input (22nd) is  
>     filename  ../../'myfolder/file-with-double-quote".png  
  
> *Error* type 6164 ****  
> Found a problem in a line of entry in a graph  
> block in "fault-6164-CheckGnuplotName.txt".  
> The file named 'image-name-OK.png'  
> exists but gnuplot cannot be told to plot it  
> because the path name contains both double and  
> single quote characters (" and '). The path  
> is /Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/../../'myfolder"/.  
> Please rename the path file and change the line  
> in the file.  
> Faulty line of input (21st) is  
>     filename  ../../'myfolder"/image-name-OK.png  
  
> *Error* type 6181 ****  
> Found conflicting lines of entry in an image  
> block in "fault-6181-ProcessImage.txt".  
> There was a "width" keyword setting the width  
> of an image and a "height" keyword setting the  
> height. Only one of these is permitted, due  
> to the way gnuplot handles image. Please  
> edit the file to remove one or the other.  
> Faulty lines of input (23rd and 24th) are  
>     height 0.1  
>     width 0.2  
  
> *Error* type 6182 ****  
> Found an invalid image block "fault-6182-ProcessImage.txt".  
> There was no entry defining the width or height  
> of an image. Please edit the file to add one  
> or the other, something along the lines of  
>     width 0.1      # fraction of page width> or  
>     height 0.14   # fraction of page height  
> The image block began at the 19th line of  
> the file:  
> begin image  
  
> *Error* type 6183 ****  
> Found an invalid line of entry in an image  
> block in "fault-6183-ProcessImage.txt".  
> There was an "filename" keyword that was not  
> followed by an image file name. Please  
> edit the file to correct this.  
> Faulty line of input (23rd) is  
>     filename  
  
> *Error* type 6184 ****  
> Found conflicting lines of entry in a graph  
> block in "fault-6184-ProcessImage.txt".  
> There was a "leftbase" keyword defining where to set  
> out an image from and a "rightbase" keyword to set it
```

```
> a second time. Only one of these is permitted, due to
> the way that gnuplot handles images.
> Please edit the file to remove one or the other.
> Faulty lines of input (22nd and 23rd) are
>   leftbase 0.5 0.5
>   rightbase 0.6 0.5

> *Error* type 6185 ****
> Found conflicting lines of entry in a graph
> block in "fault-6185-ProcessImage.txt".
> There was a "rightbase" keyword defining where to set
> out an image from and a "height" keyword. Unfortunately,
> due to the way that gnuplot handles images you can only
> pair a "rightbase" keyword with the "width" keyword.
> Please edit the file to either use the "width" keyword
> or use a "leftbase", "leftmid" or "lefttop" keyword
> instead of "rightbase".
> Faulty lines of input (20th and 21st) are
>   rightbase 0.5 0.5
>   height 0.2

> *Error* type 6186 ****
> Found conflicting lines of entry in a graph
> block in "fault-6186-ProcessImage.txt".
> There was a "leftmid" keyword to set where to set out
> an image from and a "width" keyword. Unfortunately,
> due to the way that gnuplot handles images you can only
> pair a "leftmid" keyword with the "height" keyword.
> Please edit the file to either use the "height" keyword
> or use a "leftbase", "midbase" or "rightbase" keyword
> instead of "leftmid".
> Faulty lines of input (20th and 21st) are
>   leftmid 0.5 0.5
>   width 0.2

> *Error* type 6187 ****
> Found an invalid line of entry in a graph
> block in "fault-6187-ProcessImage.txt".
> The name of the file after an "image" keyword
> did not end in a filename extension ("*.png",
> "*.jpg", "tif" or some such). Please edit
> the file to correct this.
> Faulty line of input (24th) is
>   filename 1-running-right

> *Error* type 6188 ****
> Found an invalid line of entry in a graph
> block in "fault-6188-ProcessImage.txt".
> The image file "unreadable-image.png"
> exists but cannot be read. Please either
> change the file's permissions or remove the
> block that tries to plot the image.
> Faulty line of input (23rd) is
>   filename unreadable-image.png
```

```

> *Error* type 6189 ****
> Found an invalid line of entry in a graph
> block in "fault-6189a-ProcessImage.txt".
> The image file "nonexistent-image.png"
> does not exist in the directory named
> "/Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/".
> Please edit the line of entry to point to
> an image that exists.
> Faulty line of input (24th) is
>     filename    nonexistent-image.png

> *Error* type 6189 ****
> Found an invalid line of entry in a graph
> block in "fault-6189b-ProcessImage.txt".
> The image file "nonexistent-image.png"
> does not exist in the directory named
> "/Users/ecb/Documents/sandboxes/Hobyah/test-files/raise-Hobyah-errors/../".
> Please edit the line of entry to point to
> an image that exists.
> Faulty line of input (25th) is
>     filename    ../nonexistent-image.png

> *Error* type 6190 ****
> Found a keyword in an invalid location in
> an image block in "fault-6190-ProcessImage.txt".
> The keyword is "verbatim", and it appears
> after the line with the "filename" keyword
> on it.
> The "filename" keyword marks the point in
> an image block that creates the "plot"
> keyword in gnuplot. Anything after it
> is ignored by gnuplot.
> Please edit the file to either remove the
> line(s) or move them to before the line
> with "filename" on it (line 23).

> Faulty lines of input (23rd and 24th) are
>     filename    ../../docfig-7-Ops-part.png
>     verbatim set title "image"

> *Error* type 6191 ****
> Found an invalid keyword in an image
> block in "fault-6191-ProcessImage.txt".
> The keyword is "xlabel". Please edit
> the file to correct it. For what it
> is worth, the valid keywords are
> begin, filename, height, leftbase, leftmid,
> lefttop, midbase, rightbase, verbatim and width.
> Faulty line of input (23rd) is
>     xlabel "graph keywords are not permitted here"

> *Error* type 7002 ****
> A binary file has a version string (the

```

```

> first thing in the binary file) that was
> not a string. The .hbn file was probably
> not created by Hobyah.py.
> The version string should have been some-
> thing like "Hobyah.py binary version 12",
> but it was of class 'tuple' and contained:
>   ("This is the tuple's 1st element", 'This is the 2nd')
> The faulty file is "H-7002a-version-wrong-type.hbn".

> *Error* type 7002 ****
> A binary file has a version string (the
> first thing in the binary file) that was
> not a string. The .hbn file was probably
> not created by Hobyah.py.
> The version string should have been some-
> thing like "Hobyah.py binary version 12",
> but it was of class 'tuple' and started
> with:
>   ("This is the tuple's 1st element", 'This is the 2nd', 'T...'
> The faulty file is "H-7002b-version-wrong-type.hbn".

> *Error* type 7003 ****
> A binary file has a version string (the
> first thing in the binary file) that
> looks like a Hobyah.py version string,
> not an SESconv.py version string.
> The version string should have been some-
> thing like "Hobyah.py binary version 5",
> but it was
>   "SESconv.py binary version 9"
> Someone probably renamed an SESconv.py
> ".sbn" file to end with ".hbn" instead.
> The faulty file is "H-7003-renamed-sbn-file.hbn".

> *Error* type 7004 ****
> A binary file has a version string that
> didn't match the required form (the words
> "Hobyah.py binary version XX" where XX is
> an integer. Instead it was
>   "Little dog Turpie barks so that I cannot sleep nor slumber"
> The faulty file is "H-7004-version-wrong-str.hbn".
> It was not created by Hobyah.py, it just
> happens to be a pickle file that started
> with a string.

> *Error* type 7005 ****
> A binary file has a version number that is
> too old for this program to process correctly.
> The file is of binary version 1 and this
> program can only handle files of version 7.
> Please rerun the file with a version of
> Hobyah.py that generates binary files of
> version 7.
> The faulty file is "H-7005-version-too-low.hbn".

```

```
> *Error* type 7006 ****
> A binary file has a version number that is
> too new for this program to process correctly.
> The file is of binary version 9999 and this
> program can only handle files of version 7.
> Please update your version of this program
> and its dependent modules.
> The faulty file is "H-7006-version-too-high.hbn".

> *Error* type 7021 ****
> The binary file "H-7021-not-pickle-file.hbn"
> is not a Python pickle file. Please check
> whether it was generated from Hobyah.py.
> The pickle.load() function failed while
> reading the binary file version.

> *Error* type 7022 ****
> The binary file "H-7022-raise-EOF.hbn"
> ended unexpectedly. Please check whether
> it was generated from Hobyah.py.
> The pickle.load() function failed while
> reading the transient data.

> *Error* type 7021 ****
> The binary file "H-7023-misc-unpickling-error.hbn"
> is not a Python pickle file. Please check
> whether it was generated from Hobyah.py.
> The pickle.load() function failed while
> reading the fixed data.

> *Error* type 7024 ****
> The binary file "H-7024-ends-early.hbn"
> has a block of data of unexpected length
> for the fixed data.
> Expected 27 items but read 5 instead.

> *Error* type 7041 ****
> Came across an invalid plot type in "fault-7041-CheckProperty.txt".
> The plot type was "volflwo", to be plotted
> from a Hobyah output file. Unfortunately that
> property is not valid for Hobyah, you can
> only plot the following:
>   accel, area, area_d, atkinson, celerity, coasting,
>   d_h, darcy, density, down_ch, dutypoint, elevations,
>   fanchar, fanchar-cursed, fanning, gradients,
>   lanes, massflow, pdiff, perimeter, pstat,
>   pstatabs, ptot, ptotabs, r_bf, r_fb, radius,
>   regenfrac, roughness, sectors, speed, speedlimits,
>   speedms, system, system-cursed, tot_dens,
>   tot_flow, up_ch, velocity, volflow, zeta_bf,
>   zeta_fb, -massflow, -pdiff, -pstat, -ptot,
>   -velocity and -volflow.
> Please edit the file to correct it.
```

```
> Faulty line of input (35th) is
>         transient volflwo calc Mainline1@11000

> *Error* type 7042 ****
> Came across an invalid plot type in "fault-7042-CheckProperty.txt".
> The plot type was "_t7042", to be plotted from
> file "fault-7042-CheckProperty.txt".
> Unfortunately the solver used to generate that
> binary file was "moc2", which cannot produce
> this plot type. The following solver(s) can
> produce it:
>     moc4.
> Please edit the file to remove the curve or
> rerun the file with a different solver.
> Faulty line of input (43rd) is
>         transient _t7042 calc Mainline1@11000

> *Error* type 7043 ****
> Came across an invalid plot type in "fault-7043a-CheckProperty.txt".
> The plot type was "R_bf", to be plotted
> on a fandata plot.
> Unfortunately that plot type can only be
> plotted on the following curve types:
>     transient.
> The only plot types that can be plotted on
> fandata plots are:
>     dutypoint, fanchar, fanchar-cursed, system
>     and system-cursed.
> Please edit the file to adjust or remove
> the entry.
> Faulty line of input (37th) is
>         fandata R_bf calc Mainline1@11000

> *Error* type 7043 ****
> Came across an invalid plot type in "fault-7043b-CheckProperty.txt".
> The plot type was "system", to be plotted
> on a transient plot.
> Unfortunately that plot type can only be
> plotted on the following curve types:
>     fandata.
> The only plot types that can be plotted on
> transient plots are:
>     accel, area_d, celerity, density, down_ch,
>     massflow, pdiff, pstat, pstatabs, ptot, ptotabs,
>     r_bf, r_fb, speed, speedms, tot_dens, tot_flow,
>     up_ch, velocity, volflow, zeta_bf, zeta_fb,
>     -massflow, -pdiff, -pstat, -ptot, -velocity
>     and -volflow.
> Please edit the file to adjust or remove
> the entry.
> Faulty line of input (46th) is
>         transient system calc Mainline1@11000 # Should fail

> *Error* type 7065 ****
```

```

> Tried to plot a curve in "fault-7065-_CheckAt.txt",
> but found that first part of the location
> identifier "airportwest@290.0" was not an allowed
> word, it was "airportwest". The only words allowed
> are:
>   airport_west and first.
> Please edit the file to correct it. N.B. If
> you have change the name of something, you
> may need to rerun the calculation.
> Faulty line of input (64th) is
>           Fandata fanchar calc airportwest@290

> *Error* type 7066 ****
> Tried to plot a curve in "fault-7066-_CheckAt.txt",
> but found that first part of the location
> identifier "train@0.0" only contained the
> word "train" without a train number before
> the "@". Please edit the file to add a
> train number.
> Faulty line of input (46th) is
>           transient speed calc train@0.0

> *Error* type 7067 ****
> Tried to plot a curve in "fault-7067-_CheckAt.txt",
> at train 0. Hobyah runs (unlike SES runs) do
> not have a train zero. Please edit the file
> and change the entry to be a train number of
> one or above.
> Faulty line of input (46th) is
>           transient speed calc train0@0.0

> *Error* type 7068 ****
> Tried to plot a curve in "fault-7068-_CheckAt.txt",
> but found that first part of the location
> identifier "train5.2@0.0" did not have a
> train number after it, it had "5.2"
> instead. Please edit the file and change
> it to an integer train number.
> Faulty line of input (46th) is
>           transient speed calc train5.2@0.0

> *Error* type 7081 ****
> Tried to plot a curve in "fault-7081a-TransientAt.txt",
> at a point in the tunnel but the property
> being plotted (pdiff) can only be plotted
> at fans and dampers but there were no fans
> or dampers in the calculation.
> Faulty line of input (38th) is
>           transient pdiff calc first@11000

> *Error* type 7081 ****
> Tried to plot a curve in "fault-7081b-TransientAt.txt",
> at a point in the tunnel but the property
> being plotted (pdiff) can only be plotted

```

```
> at fans and dampers. There were no fans in
> the calculation but "pdiff" may be plotted
> at the following damper(s):
>     fid2.
> Faulty line of input (53rd) is
>         transient pdiff      calc 204@11000

> *Error* type 7081 ****
> Tried to plot a curve in "fault-7081c-TransientAt.txt",
> at a point in the tunnel but the property
> being plotted (pdiff) can only be plotted
> at fans and dampers. There were no dampers
> in the calculation but "pdiff" may be plotted
> at the following fan(s):
>     fan_1.
> Faulty line of input (71st) is
>         transient pdiff      calc 204@11000

> *Error* type 7081 ****
> Tried to plot a curve in "fault-7081d-TransientAt.txt",
> at a point in the tunnel but the property
> being plotted (pdiff) can only be plotted
> at fans and dampers. "Pdiff" may be plotted
> at the following fans/dampers:
>     fid2 and fan_1.
> Faulty line of input (79th) is
>         transient pdiff      calc 204@11000

> *Error* type 7082 ****
> Tried to plot a curve in "fault-7082-TransientAt.txt",
> at a point in route 204 but the property
> being plotted (R_bf) can only be plotted
> at dampers. Please edit the file to
> correct the location or remove the curve
> definition.
> Faulty line of input (45th) is
>         transient R_bf      calc 204@11000

> *Error* type 7083 ****
> Tried to plot a curve in "fault-7083-TransientAt.txt",
> at distance 11000.0 along the tunnel named
> "first".
> That tunnel starts at 10000.0 and ends
> at 10330.0. Please edit the file to
> correct the location or remove the curve
> definition.
> Faulty line of input (36th) is
>         transient ptot      calc first@11000

> *Error* type 7085 ****
> Tried to plot a curve in "fault-7085-TransientAt.txt",
> at a "fan2" type fan, but the property being
> plotted (Ptot) cannot be plotted at that
> type of fan. The only properties that can be
```

```

> be plotted are:
>   dutypoint, fanchar, fanchar-cursed, system
>   and system-cursed.
> If you want to plot a property at one of the
> fan's two gridpoints, plot in the tunnel the
> fan is in; it's in tunnel "ventshaft" and
> the line defining the fan is:
>   fan2    10080 Fan_1 my_char fanseq time speed
> Faulty line of input (68th) is
>       transient Ptot calc Fan_1@0

> *Error* type 7087 *****
> Tried to plot a curve in "fault-7087-TransientAt.txt",
> at a train, but the property (ptot) cannot
> be plotted at trains.
> The only properties that can be plotted at
> trains are:
>   accel, down_ch, speed, speedms and up_ch.
> Faulty line of input (45th) is
>       transient ptot calc train1@0.0

> *Error* type 7121 *****
> Tried to plot a curve in "fault-7121a-FindSegment.txt",
> at chainage 430.0 in route "through".
> Unfortunately that location is at a node
> (or a join in a tunnel) where two or more
> segments meet. It is unclear which side
> of the node you want to plot at.
> Please edit the file to move the plot
> point a short distance (say 0.1 m) to
> make it clearer which side you want.
> Faulty line of input (58th) is
>       transient ptot calc THROUGH@430

> *Error* type 7121 *****
> Tried to plot a curve in "fault-7121b-FindSegment.txt",
> at distance 10200.0 in tunnel "first".
> Unfortunately that location is at a change
> of sectype in the tunnel. It is unclear
> which side of the change you want to plot
> at.
> Please edit the file to move the plot
> point a short distance (say 0.1 m) to
> make it clearer which side you want.
> Faulty line of input (38th) is
>       transient ptot calc first@10200

> *Error* type 7141 *****
> Tried to plot a curve in "fault-7141a-FanAt.txt",
> with data of a fan named "fid_1" in this file.
> Unfortunately there is no fan with that name
> in the file. For what it is worth these are
> the valid fan names:
>   fan_1.

```

```
> Faulty line of input (76th) is
>         fadata    system    calc  FID_1@20

> *Error* type 7141 ****
> Tried to plot a curve in "fault-7141b-FanAt.txt",
> with data of a fan named "fid_1" in Hobyah
> run "fault-7141a-FanAt.txt".
> Unfortunately there is no fan with that name
> in the file. For what it is worth these are
> the valid fan names:
>     fan_1.
> Faulty line of input (79th) is
>         fadata    system    file1  FID_1@20

> *Error* type 7142 ****
> Tried to plot a curve in "fault-7142-FanAt.txt",
> giving the static pressure properties of fan
> "fan_1", but the fan characteristic definition
> didn't include the fan diameter.
> Please add a diameter keyword so that the fan's
> dynamic pressure can be calculated, then
> subtracted from the fan total pressure.
> The fan's characteristic is "my_char" and is
> defined on the 63rd line of the Hobyah input
> file named "fault-7142-FanAt.txt".
> Are you sure you want to plot fan static
> pressure and not fan total pressure? Fan
> static pressure sucks.
> Problematic line of input (78th) is
>         fadata    system-cursed    calc  fan_1@20

> *Error* type 1201 ****
> Found too few entries in a number specifier in
> CheckRangeAndSI (there should be four). It
> occurred while processing the file
> "fault-1201a-CheckRangeAndSI.txt".
> The 1st entry for keyword "1201a" was "5.2"
> The faulty descriptor was "float      +      null".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1201a-CheckRangeAndSI.txt".

> *Error* type 1201 ****
> Found too few entries in a number specifier in
```

```
> CheckRangeAndSI (there should be four). It
> occurred while processing the file
> "fault-1201b-CheckRangeAndSI.txt".
> The 2nd entry for keyword "1201b" was "1.2 4"
> The faulty descriptor was "float      +      null".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1201b-CheckRangeAndSI.txt".

> *Error* type 1202 *****
> Found an invalid definition in the source
> code. It was
>   "flaot      +      null      a number (testblock)".
> instead of a tuple of valid words or a string
> starting with "int", "float" or "#name".
> The block being processed at the time was
> "testblock1".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1202a-ProcessBlock.txt".

> *Error* type 1202 *****
> Found an invalid definition in the source
> code. It was
>   "('any_word tin + null  an integer (testblock)',)'".
> instead of a tuple of valid words or a string
> starting with "int", "float" or "#name".
> The block being processed at the time was
> "testblock2".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
```

```
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1202b-ProcessBlock.txt".

> *Error* type 1203 ****
> Found an invalid range testing rule in CheckRangeAndSI
> while processing "fault-1203a-CheckRangeAndSI.txt".
> The 1st entry for keyword "1203a" was "1"
> The faulty descriptor was "float      ayn      null      any number (testblock)".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1203a-CheckRangeAndSI.txt".

> *Error* type 1203 ****
> Found an invalid range testing rule in CheckRangeAndSI
> while processing "fault-1203b-CheckRangeAndSI.txt".
> The 2nd entry for keyword "1203b" was "1.2"
> The faulty descriptor was "float ayn null any number (testblock)".
>
> Something unexpected occurred during your run
> and was caught by a sanity check. There is
> a brief description of what happened above.
>
> This is not the usual "here is what happened
> and this is how you might go about fixing it"
> error message. Because you can't fix it - this
> error can only be sorted out by patching the
> program.
>
> Please raise a bug report and attach the input
> file "fault-1203b-CheckRangeAndSI.txt".

> *Error* type 8003 ****
> Skipping "SES-fault-8003-nonexistent.PRN" in folder
> "/Users/ecb/Documents/sandboxes/Hobyah/files-SES/raise-SESconv-errors/"
> because it is not in that folder.

> *Error* type 8001 ****
> Skipping "SES-fault-8001-wrong-extension.PRt", because it
```

```
> doesn't end with one of the extensions
> ".PRN", ".OUT" or ".TMP".

> *Error* type 8002 ****
> Skipping "SES-fault-8002-no-file-access.PRN" in folder
> "/Users/ecb/Documents/sandboxes/Hobyah/files-SES/raise-SESconv-errors/"
> because you do not have permission to read it.

> *Error* type 8005 ****
> Skipping "SES-fault-8005-locked-log-file.PRN", because you
> do not have permission to write to its logfile.

> *Error* type 8006 ****
> Skipping "SES-fault-8006-locked-txt-file.PRN", because you
> do not have permission to write to its output file.

> *Error* type 8007 ****
> Skipping "SES-fault-8007-locked-bin-file.PRN", because you
> do not have permission to write to its binary file.

> *Error* type 8022 ****
> Failed to find a footer line in "SES-fault-8022-no-footer.PRN".
> Are you sure this is an SES output file?

> *Error* type 8023 ****
> Please check if the file "SES-fault-8023a-dud-header.PRN".
> came from SES. A line was found that looks like a
> header but with the text in the wrong place.
> Valid headers have "SES VER" in the 9th to 15th characters,
> your line has it in the 8th to 14th characters.
> Faulty line of input (1st) is
>
>      SES VER  4.10          SES file for raising a fault in SESconv.py.

> *Error* type 8023 ****
> Please check if the file "SES-fault-8023b-dud-header.PRN".
> came from SES. A line was found that looks like a
> header but with the text in the wrong place.
> Valid headers have "PAGE:" in the 113th to 117th characters,
> your line has it in the 112th to 116th characters.
> Faulty line of input (1st) is
>
>      SES VER 4.10          SES file for raising a fault in SESconv.py.

> *Error* type 8024 ****
> Failed to find lines marking "SES-fault-8024-unprocessable.PRN"
> as an SES output file (no header, couldn't find
> all three of the following lines near the start
> of the file:
>      "SUBWAY ENVIRONMENT SIMULATION"
>      "SIMULATION OF"
>      "VERSION 4.10"
> Are you sure this is an SES output file?
```

```

> *Error* type 8025 ****
> Ran out of lines of input while reading comments in "SES-fault-8025-dud-form-1B.PRN".
> It looks like your SES output file is corrupted.
> The line for form 1B, the one that should start with
>      "
>          DESIGN TIME"
> is absent. Try rerunning the file or checking the
> contents of the PRN file.

> *Error* type 8026 ****
> Came across an oddity in form 1B in the file "SES-fault-8026-dud-form-1B.PRN".
> This SES processor spots form 1B by looking out for the
> text "DESIGN TIME" (in all caps) at a particular place
> in a line of comment. It looks like one of your lines
> of comment happened to meet that criterion. Please edit
> input file to avoid this (easiest thing is to change
> your entry of "DESIGN TIME" to lower case) and edit the
> comments in the SES input file to stop it happening
> again. Either that, or the output file is corrupted.

> Faulty line of input (28th) is
>          DESIGN TIME 1700 HRS    JULY      2023 extra text

> *Error* type 8027 ****
> Failed to find a valid clock time in form 1B in file "SES-fault-8027-dud-day.PRN".
> The clock time was supposed to be something like "1700"
> but was actually "17xx".
> Please edit the file to correct the corrupted entry.
> Faulty line of input (24th) is
>          DESIGN TIME 17xx HRS    JULY      2023

> *Error* type 8028 ****
> Failed to find a valid month in form 1B in file "SES-fault-8028-dud-month.PRN".
> The text giving the month should have been something
> like "FEBRUARY" but was actually "OCTEMBER".
> Please edit the file to correct the corrupted entry.
> Faulty line of input (24th) is
>          DESIGN TIME 1700 HRS    OCTEMBER  2023

> *Error* type 8029 ****
> Failed to find a valid year in form 1B in file "SES-fault-8029-dud-year.PRN".
> The year was supposed to be something like "2028"
> but was actually "10,000".
> Please edit the file to correct the corrupted entry.
> Faulty line of input (24th) is
>          DESIGN TIME 1700 HRS    OCTOBER   10,000

> *Error* type 8030 ****
> Ran out of lines of input while skipping lines in "SES-fault-8030-no-form-1C.PRN".
> It looks like your SES output file is corrupted, as the
> line for form 1C was absent. Try rerunning the file or
> checking the contents of the PRN file.

> *Error* type 8031 ****
> There is a fatal problem with the input of "SES-fault-8031-impl-expl.PRN".

```

> The run uses train performance option 2 (explicit train
 > speed and implicit brake/traction heat calculation). In
 > SES version 4.1 the tractive effort calculation of train
 > performance option 2 is too low by a factor of several
 > hundred. This is a bug in SES v4.1 routine Train.for.
 >
 > Details of the bug are as follows:
 >
 > SES calculates the tractive effort required to overcome
 > the gradient resistance (variable name RG, tractive effort
 > needed to climb a hill) and the curve resistance (variable
 > name RC, friction at the flange-rail interface). It does
 > so by a call to Locate.for.
 > Locate.for returns RG as a fraction of train mass and RC
 > as lbs of flange drag on curves per short ton of train mass.
 >
 > When using train performance option 1, SES v4.1 works
 > properly. It gets RG and RC from Locate.for as fractions
 > of train mass (see line 185 of Train.for). It multiplies
 > RG by train mass (line 193 of Train.for) and it multiplies
 > RC by "train mass/2000" (line 194). These factors put RG
 > and RC into units suitable for the tractive effort
 > calculation.
 >
 > But when using train performance option 2, SES v4.1 does
 > not behave properly. It gets RG and RC from Locate.for as
 > fractions of train mass (at line 693) but does not multiply
 > by the relevant factors. Instead, it jumps to label 1100
 > and starts assuming that RG and RC are already in suitable
 > units.
 >
 > I came across the bug a few years ago when doing a freight
 > rail tunnel. We could see this 4,000 tonne train racing up
 > a 1.6% incline with negligible tractive effort when we used
 > train performance option 2.
 > A bit of investigation showed that SES v4.1 exhibited the
 > same behaviour, so we went looking for the cause and found
 > the bug in Train.for in the SES v4.1 distribution.
 >
 > Your best bet is to switch to implicit train performance
 > or (if you're up to the challenge) you can calculate the
 > heat rejection yourself and use train performance option 3.

> *Error* type 8032 *****
 > Ugh, something went horribly wrong with input
 > file "SES-fault-8032-offline-SES.PRN".
 > It looks like your offline-SES output file is
 > corrupted, the input file version number is
 > not a number, it was "204.xx". This usually
 > happens when you edit the .PRN file.

> *Error* type 8033 *****
 > There is a problem with the SES file "SES-fault-8033-no-segments.PRN".
 > The run has no line segments or vent segments, which

```

> means that this run is intended to calculate traction
> power performance in the open air.
> SESconv.py does not handle runs that only calculate
> traction power performance in the open air. It could
> handle them, but no-one who needs to do serious train
> performance calculations should be doing them in SES
> now that far more capable programs like OpenTrack or
> TRAIN are available.

> *Error* type 8041 ****
> Ugh, ran out of lines of input to read!
>
> This usually occurs when SESconv.py tries
> to process an SES file that triggered an
> untrapped Fortran runtime error. SES
> traps most of those, but not all.
>
> The first thing to do is check the end of
> the SES output file for clues about what
> went wrong. Here are the last ten lines of
> the output file (possibly truncated):
>                               INPUT VERIFICATION OF GENERAL DATA
>                               ...
>                               TRAIN PERFORMANCE OPTION      ...
>                               TEMPERATURE / HUMIDITY SIMULATION OPTION ...
>                               HUMIDITY DISPLAY OPTION        ...
>                               ENVIRONMENTAL CONTROL LOAD EVALUATION OPTION ...
>                               HEAT SINK SUMMARY PRINT OPTION ...
>                               SUPPLEMENTARY OUTPUT OPTION ...
>                               ALLOWABLE SIMULATION ERRORS ...
>                               ALLOWABLE INPUT ERRORS ...
>
> If those ten lines give no clues as to what
> happened, try running SES again. But run
> it in a terminal window (not by drag and
> drop) so that you can read the output after
> the SES run ends. Check the output to see
> see if a runtime error occurred.
> A typical runtime error looks like:
>     At line 61 of file Simq1.for
>     Fortran runtime error: Index '488670' of array...
>     Error termination. Backtrace:
>     #0 0x1026bd357
>     #1 0x1026bde17
>     #2 0x1026be1d3...
>
> Please only raise a report about a possible
> bug in SESconv.py if it looks like there was
> no Fortran runtime error.
> *Error* 8041 ****
> *Ugh, ran out of lines of input! See above for details. *

> *Error* type 8061 ****
> Seeking a string slice beyond the end of the line. Details are:

```

```

> Line:                      TRAIN PERFORMANCE OPTION 1      IMPLICIT
> Slice:-----
> Line length is 67, slice is 78:83.

> *Error* type 8062 (it may be an error, it may be OK).
> Sliced a line to get a number but found that there
> was a possibly related character before it. Details are:
> Line:    AMBIENT AIR DENSITY                               000000000.07281   LB
> Slice:-----

> *Error* type 8063 *****
> The contents of the slice was blank. Details are:
> Line:                      TRAIN PERFORMANCE OPTION           IMPLICIT
> Slice:-----


> *Error* type 8064 (it may be an error, it may be OK).
> The last character in the slice was whitespace. Details are:
> Line: AMBIENT AIR DRY-BULB TEMPERATURE                   85.0     DEG
> Slice:-----


> *Error* type 8065 (it may be an error, it may be OK).
> Sliced a line to get a number but found that there
> was a possibly related character after it. Details are:
> Line:                      TRAIN PERFORMANCE OPTION           11     IMPLICIT
> Slice:-----


> *Error* type 8066
> The Fortran format field is too small for the number. Details are:
> Line: ***** 101 -101      (SPECIAL TUNNEL)           losses at back end
> Slice:~~~~~_
> Returning a zero value instead.

> *Error* type 8067 *****
> A slice of a line did not contain a valid real number. Details are:
> Line:                      TRAIN PERFORMANCE OPTION           xxx     IMPLICIT
> Slice:-----


> *Error* type 8068 *****
> A slice of a line did not contain a valid integer. Details are:
> Line:                      TRAIN PERFORMANCE OPTION           1.2     IMPLICIT
> Slice:-----


> *Error* type 8069 *****
> Tried to convert a line and its unit but the unit is not on the line.
> This is likely a failure to consider an SES option, please raise a bug report.
> Details are: Form 1F, external WB.
> The intended conversion is from DEG F to deg C.
> Line: AMBIENT AIR WET-BULB TEMPERATURE                  146.944   K
> Slice:-----
```

```

> *Error* type 8101 ****
> Came across a faulty line in form 2A while reading "SES-fault-8101-wrong-entry-count.PRN".
> It doesn't have 5 entries in it, it has 7.
> Faulty line of input (214th) is
>   Extra words          101          101          102          1
>
> *Error* type 8121 ****
> Came across a duplicate segment no. in form 3A while reading
> "SES-fault-8121-duplicate-segments.PRN".
> There is already a segment 101, it is in section 101.

> Faulty line of input (334th) is
>   INPUT VERIFICATION FOR LINE SEGMENT 102 -101           Tunnel 101 for the second time

> *Error* type 8141 ****
> Came across a line of text that didn't have the expected US units text on it
> while reading form 3B in "SES-fault-8141-wrong-unit.PRN".
> Expected to find " FT".

> Faulty line of input (288th) is
>   LENGTHS          0.2000  0.2000           ROUGHNESS

> *Error* type 8161 ****
> Found an obscure mistake in the input of "SES-fault-8161-zero-for-branches.PRN"
> that invalidates its calculation. You have one or
> more junctions of type 1-6 in your file, which means
> that SES ought to call a routine that calculates the
> pressure losses across junctions (Omega3.for). But
> the count of branched junctions in form 1D is zero, so
> SES skips making that call and the junction pressure
> losses are all zero.
>
> What was the point of coding it that way? Back in the
> 1970s and 1980s CPU time was expensive. It was useful
> to have the ability to pay less for runs with branched
> junctions that the user knew were zero pressure loss.
> Nowadays the feature is just a forgotten trap that SES
> and SVS don't warn about that the unwary can fall into.
>
> The way to solve the problem is to set the count of
> branched junctions in form 1D to any positive number
> (1 is suggested) and rerun "SES-fault-8161-zero-for-branches".

> *Error* type 8181 ****
> Found a line of runtime data that wasn't what was
> expected. It should have been the first line of
> detailed print data for segment 101 but the text
> didn't match (000).
> Faulty line of input (1136th) is
>   5000.0  101 -000      (SPECIAL TUNNEL)           losses at back end

> *Error* type 8182 ****
> Found a line of runtime data that wasn't what was
> expected. It should have been the first line of

```

```

> abbreviated print data for segment 101 but the
> text didn't match (000).
> Faulty line of input (869th) is
>   101 -000          0.          0.

> *Error* type 8183 *****
> Found a line of runtime data that wasn't what was
> expected. It should have been the first line of
> abbreviated print data for segment 52 but the
> text didn't match (000).
> Faulty line of input (3742nd) is
>   2 -000      -43.327    -2.471    31.556    54.111

> *Error* type 8221 *****
> Came across an instance of form 3B in which there
> were zero values of perimeter appearing before non-
> zero values of perimeter. Details are:
>   There were 3 perimeters printed to the output file.
>   There were 4 perimeters in the input file.
> Due to a bug in SES some of the roughnesses have
> been ignored and the total perimeter and friction
> factor of segment 52 in "SES-fault-8221-form-3B.PRN" are wrong.
> Faulty line of input (349th) is
>   PERIMETERS      13.3     20.0      0.0          TOTAL PERIMETER

> *Error* type 8261 *****
> Found a mistake in the input of "SES-fault-8261-fire-in-non-fire-segment.PRN"
> that invalidates its calculation. You have a fire
> simulation option and have put a fire into segment
> 101, but segment 101 is not a fire segment so the
> walls will not warm up. This is not the way to do
> SES fire simulations properly; please review your
> run.

> *Error* type 8034 *****
> Failed to find a line with exactly two words
> (the word "VERSION" and an SVS version number
> like "6.6.2"). Instead, there was the following:
>   VERSION  6.6.2 extras

> *Error* type 8035 *****
> SVS file "SES-fault-8035-SVS-newer-SVS-version.OUT" has
> a version number ("6.7.1") that this script is
> not able to process yet.
> Please raise a request to get the new version
> of SVS supported.

> *Error* type 8036 *****
> SVS file "SES-fault-8036-SVS.OUT" ended before the
> file's SVS version number was found.

> *Error* type 8201 *****
> "SES-fault-8201.OUT" looks like an OpenSES
> output file but it has a version that is not

```

```
> recognised. This converter can only handle
> OpenSES versions 4.2 and 4.3, this one has the
> version 4.201 on the following line:
>   OpenSES, 4.201 , 14 July 2021
> Please raise a bug report so that the program
> can be updated to deal with the new version.

> *Error* type 8202 ****
> "SES-fault-8202.OUT" does not seem
> to be an OpenSES output file, as a line giving
> the OpenSES version number and ending in the year
> of issue could not be found, even though a line
> resembling part of the OpenSES licence was found.
> Are you sure this file came from OpenSES?

> *Error* type 8203 ****
> "SES-fault-8203.OUT" does not seem
> to be an OpenSES output file, as a line giving
> the run date and time could not be found, even
> though lines resembling part of the OpenSES
> licence and the version number were found.
> Are you sure this file came from OpenSES?

> *Error* type 8009 ****
> File "SES-fault-8009-no-restart-data.TMP" was not able to
> read its restart file. Either the restart file didn't
> exist or no restart file name was given in the input
> file.
```

Index

.txt files, 63
`<trafficname>.dens` plot type, 244
`<trafficname>.flow` plot type, 243
*, 90
. , 90
:=, 64
@, 90
#, 64
`3temperatures` keyword, 147
`4A_locn` keyword, 148
`4B_heat` keyword, 148
`4C_flames` keyword, 149

steady_sens, 310
wallT_AM, 306
wallT_PM, 307
wallT_used, 305
effic1 plot type, 295
elevations plot type, 209
fan plot types
 fanchar, 232
 fanchar-cursed, 234
 pdiff, 200
 system, 233
 system-cursed, 235
fanchar plot type, 232
fanchar-cursed plot type, 234
Fanning plot type, 229
file names, 65
filename keyword, 146
filetypes
 .txt, 63
fires icons, 240
fireseg plot type, 223
float optional entry, 175
flywh_rpm plot type, 279
font setting, 157
footer setting, 82
form1B keyword, 146
form1F keyword, 146
form1G keyword, 147
form3F keyword, 148
form7AB keyword, 149
form7C1 keyword, 149
form7C2 keyword, 150, 152
form8 keyword, 151
frictionapprox setting, 79
frictiontype setting, 78
gamma setting, 85
Gnuplot
 customising the terminal, 333
gradients plot type, 211
graph block, 161
graphs, annotating, 166
ground_sens plot type, 312
header setting, 83
heat2air plot type, 283
heat_ADM plot type, 292
heat_lat plot type, 294
heat_sens plot type, 293
HGV_dens plot type, 244
HGV_flow plot type, 243
Hobyah
 origin, 321
pronunciation, 20
the program, 18
HVAC_lat plot type, 316
HVAC_sens plot type, 315
HVAC_total plot type, 317
ignore page, 155
image
 leftbase keyword, 180
 leftmid keyword, 180
 leftright keyword, 180
 midbase keyword, 180
 topbase keyword, 180
image example, 179
image keyword, 179
leftbase keyword, 179
leftmid keyword, 179
leftright keyword, 179
midbase keyword, 179
topbase keyword, 179
images setting, 84
indenting, 64
input files, 63
jet fans icons, 240
keytextscale setting, 84
lanes plot type, 214
lat plot type, 266
lat_pul plot type, 266
LCV_dens plot type, 244
LCV_flow plot type, 243
length optional entry, 176
licence, software, 17
line length, 63
lineamps plot type, 278
lines
 blank, 63
 case-sensitivity, 63
 line length, 63
linewidth setting, 157
lists
 addition, 92
 combining, 95
 complex, 93
 constants in, 92
 multiplication, 92
 range() function, 93
 ranges, 93
 startstepcount() function, 94
 startstopcount() function, 94
lmargin setting, 164
lrmargin setting, 164

mandatory blocks
 plots block, 42
 settings block, 42
margins setting, 164
massflow plot type, 206
max_vel setting, 87
meanDB_AM plot type, 299
meanDB_off plot type, 301
meanDB_PM plot type, 300
meanW_AM plot type, 302
meanW_off plot type, 304
meanW_PM plot type, 303
misc_lat plot type, 309
misc_sens plot type, 308
mode plot type, 284
motoramps plot type, 277

namecheck optional entry, 180
NaN, 100, 273, 329

offline-SES, 20
orientation keyword, 157
orientation setting, 157

P_atm setting, 81
page block, 159
page orientation, 157
page sizes
 ANSI, 156
 custom, 156
 ISO, 156
pagesize keyword, 156
pagesize setting, 156
pdiff plot type, 200
perimeter plot type, 225
phrase, 91
 complex lists, 93
 like a Python list, 91
 list, 91
 list addition, 92
 list multiplication, 92
 quotes, 91
 range() function, 93
 ranges, 93
 startstepcount() function, 94
 startstopcount() function, 94
pipe_ht plot type, 319
pipetemp plot type, 318
plot settings, 155, 156
plot types
 <trafficname>.dens, 244
 <trafficname>.flow, 243
 accel, 269
 acceltemp, 280

aerodrag, 274
airex_lat, 314
airex_sens, 313
annulus, 267
area, 224
area_d, 236
car_dens, 244
car_flow, 243
Cd (train drag factor), 275
celerity, 198
chainage, 273
coasting, 217
Darcy, 228
DB (dry-bulb temperature), 260
deceltemp, 281
density, 199
DP (section pressure change), 258
DP_indiv (section pressure
 change), 259
effic1, 295
elevations, 209
fanchar, 232
fanchar-cursed, 234
Fanning, 229
fires icons, 240
fireseg, 223
flywh_rpm, 279
gradients, 211
ground_sens, 312
heat2air, 283
heat ADM, 292
heat_lat, 294
heat_sens, 293
HGV_dens, 244
HGV_flow, 243
HVAC.lat, 316
HVAC.sens, 315
HVAC.total, 317
jet fans icons, 240
lanes, 214
lat, 266
lat_pul, 266
LCV_dens, 244
LCV_flow, 243
lineamps, 278
massflow, 206
meanDB_AM, 299
meanDB_off, 301
meanDB_PM, 300
meanW_AM, 302
meanW_off, 304
meanW_PM, 303
misc_lat, 309
misc_sens, 308

mode, 284
motoramps, 277
Overview, 195
pdiff, 200
perimeter, 225
pipepipe_ht, 319
pipetemp, 318
pstat, 201
pstatabs, 203
ptot, 202
ptotabs, 204
pwr_accel, 289
pwr_all, 282
pwr_aux, 285
pwr_decel, 290
pwr_flywh, 288
pwr_mech, 291
pwr_prop, 286
pwr_regen, 287
qcold, 250
qSES, 249
qtrains, 251
qwarm, 252
R_bf, 239
R_fb, 240
radius, 215
roughness, 226
route, 271
sections, 219
sectors, 216
segments, 220
sens, 264
sens_pul, 264
speed, 270
speedlimits, 213
stackgrads, 212
stacks, 210
steady_lat, 311
steady_sens, 310
sublength, 222
subsegs, 221
svseffic, 297
svsregen1, 298
svsregen2, 218
system, 233
system-cursed, 235
TE (tractive effort) keyword, 276
time, 273
trains icons, 240
type, 272
vcold, 254
velocity, 197
volflow, 205
vSES, 253
vwarm, 255
W (humidity ratio), 263
wall_temp, 262
wallT_AM, 306
wallT_PM, 307
wallT_used, 305
wetted, 230
zeta_bf, 237
zeta_fb, 238
plotnames setting, 83
plots block
 fileloop block, 155
 page block, 155
 timeloop block, 155
plots block, 155
 plotting icons (trains, jet fans, fires), 240
plotunits setting, 157
pngtrim setting, 158
profile optional entry, 175
Project description, 80
Project name, 80
Project number, 80

pstatabs plot type, 203
ptot plot type, 202
ptotabs plot type, 204
pwr_accel plot type, 289
pwr_all plot type, 282
pwr_aux plot type, 285
pwr_decel plot type, 290
pwr_flywh plot type, 288
pwr_mech plot type, 291
pwr_prop plot type, 286
pwr_regen plot type, 287

QA data for graphs, 25
QA settings, 80
qcold plot type, 250
qcold, qwarm and qSES comparison, 255
qSES plot type, 249
qtrains plot type, 251
qwarm plot type, 252

R_bf plot type, 239
R_fb plot type, 240
radii block, 127
radius plot type, 215
range() function
 stop value absent from, 94
range() function, 93
ranges, 93
 combining, 95

ratio optional entry, 180
regenfraction block, 129
reserved symbols
 *, 90
 ,, 90
 :=, 64
 @, 90
 #, 64
rho_atm setting, 81
rise_time setting, 86
rotate optional entry, 180
roughness plot type, 226
route plot type, 271
route plot types
 airex_lat, 314
 airex_sens, 313
 annulus, 267
 area, 224
 car_dens, 244
 car_flow, 243
 celerity, 198
 coasting, 217
 Darcy, 228
 DB (dry-bulb temperature), 260
 density, 199
 DP (section pressure change), 258
 DP_indiv (section pressure
 change), 259
 elevations, 209
 Fanning, 229
 fires icons, 240
 fireseg, 223
 gradients, 211
 ground_sens, 312
 HGV_dens, 244
 HGV_flow, 243
 HVAC_lat, 316
 HVAC_sens, 315
 HVAC_total, 317
 jet_fans icons, 240
 lanes, 214
 lat, 266
 lat_pul, 266
 LCV_dens, 244
 LCV_flow, 243
 meanDB_AM, 299
 meanDB_off, 301
 meanDB_PM, 300
 meanW_AM, 302
 meanW_off, 304
 meanW_PM, 303
 misc_lat, 309
 misc_sens, 308
 perimeter, 225
 pipe_ht, 319
 pipetemp, 318
 qcold, 250
 qSES, 249
 qwarm, 252
 radius, 215
 roughness, 226
 sections, 219
 sectors, 216
 segments, 220
 sens, 264
 sens_pul, 264
 speedlimits, 213
 stackgrads, 212
 stacks, 210
 steady_lat, 311
 steady_sens, 310
 sublength, 222
 subsegs, 221
 svsregen2, 218
 trains icons, 240
 vcold, 254
 velocity, 197
 vSES, 253
 vwarm, 255
 W (humidity ratio), 263
 wall_temp, 262
 wallT_AM, 306
 wallT_PM, 307
 wallT_used, 305
 wetted, 230
run time, 82
runtype setting, 77

section plot types
 DP (section pressure change), 258
 DP_indiv (section pressure
 change), 259
 sections plot type, 219
 sectors plot type, 216
 sectors block, 127
 same sectype, 109
segment plot types
 DP_indiv (section pressure
 change), 259
 DP (section pressure change), 258
 segments plot type, 220
 sens plot type, 264
 sens_pul, 264
 SES train numbering, 328
 SESdata block, 47, 115, 144
 SESpragmat keyword, 115
 settings block, 76
 software licence, 17

solver setting, 85
speed plot type, 270
speedlimits plot type, 213
splitpages setting, 158
stackgrads plot type, 212
elevations plot type, 210
startstepcount() function, 94
startstopcount() function, 94
startstopcount() function
 startstopcount() function
 stop value present, 94
steady_lat plot type, 311
steady_sens plot type, 310
sublength plot type, 222
subsegs plot type, 221
SVS plot types
 svseffic, 297
 svsregen1, 298
 svsregen2, 218
svseffic plot type, 297
svsregen1 plot type, 298
svsregen2 plot type, 218
system plot type, 233
system-cursed plot type, 235

target keyword, 145
TE (tractive effort) keyword, 276
time
 accuracy, 331
 rounding, 331
time plot type, 273
time_accuracy setting, 87
time step, 82
rmargins setting, 164
tmargins setting, 164
Tractive effort
 plotting, 276
train plot types
 accel, 269
 acceltemp, 280
 aerodrag, 274
 Cd (train drag factor), 275
 chainage, 273
 deceltemp, 281
 effic1, 295
 flywh_rpm, 279
 heat2air, 283
 heat ADM, 292
 heat_lat, 294
 heat_sens, 293
 lineamps, 278
 mode, 284
 motoramps, 277
 pwr_accel, 289

pwr_all, 282
pwr_aux, 285
pwr_decel, 290
pwr_flywh, 288
pwr_mech, 291
pwr_prop, 286
pwr_regen, 287
qtrains, 251
route, 271
speed, 270
svseffic, 297
svsregen1, 298
TE (tractive effort), 276
time, 273
type, 272
trains icons, 240
transient plot types
 accel, 269
 acceltemp, 280
 aerodrag, 274
 airex_lat, 314
 airex_sens, 313
 annulus, 267
 Cd, 275
 chainage, 273
 DB (dry-bulb temperature), 260
 deceltemp, 281
 DP (section pressure change), 258
 DP_indiv (section pressure
 change), 259
 effic1, 295
 flywh_rpm, 279
 ground_sens, 312
 heat2air, 283
 heat ADM, 292
 heat_lat, 294
 heat_sens, 293
 HVAC_lat, 316
 HVAC_sens, 315
 HVAC_total, 317
 lat, 266
 lat_pul, 266
 lineamps, 278
 meanDB_AM, 299
 meanDB_off, 301
 meanDB_PM, 300
 meanW_AM, 302
 meanW_off, 304
 meanW_PM, 303
 misc_lat, 309
 misc_sens, 308
 mode, 284
 motoramps, 277
 pipe_ht, 319

pipetemp, 318
pwr_all, 282
pwr_accel, 289
pwr_aux, 285
pwr_decel, 290
pwr_flywh, 288
pwr_mech, 291
pwr_prop, 286
pwr_regen, 287
qcold, 250
qSES, 249
qtrains, 251
qwarm, 252
route, 271
sens, 264
sens_pul, 264
speed, 270
steady_lat, 311
steady_sens, 310
svseffic, 297
svsregen1, 298
TE (tractive effort), 276
time, 273
type, 272
vcold, 254
vSES, 253
vwarm, 255
W (humidity ratio), 263
wall_temp, 262
wallT_AM, 306
wallT_PM, 307
wallT_used, 305
tunnel plot types
 <trafficname>_dens, 244
 <trafficname>_flow, 243
area, 224
celerity, 198
Darcy, 228
density, 199
Fanning, 229
massflow, 206
perimeter, 225
pstat, 201
pstatabs, 203
ptot, 202
ptotabs, 204
roughness, 226
velocity, 197
volflow, 205
type plot type, 272
units setting, 80
units, SI and US, 19
vcold plot type, 254
vcold, vwarm and vSES comparison,
 255
velocity plot type, 197
verbatim
 verbatim block, 165
 verbatim keyword, 167
 centre & center, 166, 167
version
 version setting, 77
volflow plot type, 205
vSES plot type, 253
vwarm plot type, 255
W (humidity ratio) plot type, 263
wall_temp plot type, 262
wallT_AM plot type, 306
wallT_PM plot type, 307
wallT_used plot type, 305
wetted plot type, 230
zeta_bf plot type, 237
zeta_fb plot type, 238