Notes on modifying the SES program

Ewan Bennett

August 2022 (v1.1)

Abstract

Subway Environmental Simulation (SES) is a computer program for designing ventilation systems in rail tunnels. Those who use it heavily almost always end up writing their own preprocessors & postprocessors and modifying the SES code to add new capabilities.

This document describes the oddities I encountered and the traps that my colleagues and I fell into while using and extending SES over the years. It is an *aide-memoire*: a set of notes to remind myself of aspects of SES that I had studied in detail, then forgotten.

Contents

1 Introduction

SES (Subway Environmental Simulation) is a computer program for designing ventilation systems in rail tunnels. It was produced under the sponsorship of the US Department of Transportation (DoT) and released in the mid 1970s to whoever asked for it (mostly consulting engineers and railway tunnel operators; a list of early users can be found online in document UMTA-MA-06-0100-79-13). They received a three-ring binder and a reel of tape with the Fortran source code, which they could compile on the mainframe of their choice. There were a number of updates, with the last public version (4.1) being released in 2002 as a PC executable on CD with the source code and the SES v4.1 User Manual.

Occasional users of SES make do with reading text output files in US customary units formatted for 132-column fanfold printer paper.

Those who use it heavily almost always end up writing their own preprocessors & postprocessors and modifying the SES code to add new capabilities.

This document is intended for those who use it heavily. It describes the oddities I encountered and the traps that my colleagues and I fell into while using/extending SES over the last decade.

Although the source code of SES was provided to those who asked for it, it was not released under any software licence. So when I started my own tunnel ventilation software project in May 2020, I studiously avoided including any SES source code in my github repository (`github.com/ECB2020/Hobyah`). All that tiptoeing around changed in early 2021 when a github repository named OpenSES (`github.com/Open-SES/OpenSES`) was created. My congratulations to the engineers who convinced the US government to let them release the v4.1 source code under the 3-clause BSD licence and founded the OpenSES repository to develop it.

SES is written in Fortran IV (a.k.a. Fortran 66), which has a coding style that today's programmers gaze at with horror. Fortran IV is full of `GOTO` statements, arithmetic `IF` statements and `EQUIVALENCE` in `COMMON` blocks. Its variable names are limited to six characters.

Although the SES code is hard for modern-day programmers to follow, the program documentation is first class. The "Program dictionary of variables" file `Record.txt` is most excellent: when I'm staring at a line of code like

```
    GRASHA = DIACGV(ITYP)**3*(TGACCV(NUMV)-TAAVG)*AIR(TEMP,4)
```

it is good to know that a clear and complete explanation of what `DIACGV`, `ITYP`, `TGACCV`, `NUMV` and `TAAVG` are can be found in `Record.txt`. That file has saved my bacon on many occasions.

In late 2019 WSP (a firm of consulting engineers who absorbed the consulting engineers that oversaw the development of SES) convinced DoT to allow them to release their internal version. The new release (called SVS v6) has a number of updates: it is in SI units and has additional capabilities. Readers who are currently using SVS v6 may be wondering if any of the traps or oddities I found in SES v4.1 are present in SVS v6. I don't know, as I've never used it. However, I can see from the documentation that WSP have modified the input format to fix the one of the traps I describe (in form 7C) and one of the oddities (in form 9E).

Each of the sections below describes something notable that I encountered in SES v4.1 when writing converters for it or extending it to handle new features. A few sections in this document go deep into the source code. Where line numbers of named Fortran routines are given, the line numbers refer to the source code of SES version 4.1 which can be found on the SES v4.1 CDs sent out by the Volpe Transportation Centre or in the initial commit of files (April 2021) to the OpenSES github repository, `github.com/Open-SES/OpenSES`. Where sample SES filenames are given, the files can be found in my github repository (`github.com/ECB2020/Hobyah`).

One final point: everything in this document is my personal opinion, based on my interpretation of the SES v4.1 source code. There may well be errors and I welcome corrections.

2 Trains alongside jet fans

I expected that symmetric geometry would always lead to symmetric airflow. A few years ago I came across a case where that wasn't true and tracked down the cause.

Figure 1 shows the geometry of a (contrived) test file, `SES-oddity-jet-fans-1.ses`. The test file has two small single-track tunnels with identical properties that connect to a large cavern with an open shaft at its midpoint. Two identical trains (the green blocks) are stopped on the track with their ends just outside the cavern. There are jet fans of equal thrust in the two single-track tunnels, both blowing away from the cavern. The geometry, trains and jet fans are symmetrical. The airflow is also symmetrical, with air coming in through the shaft and splitting 50/50, as you would expect.

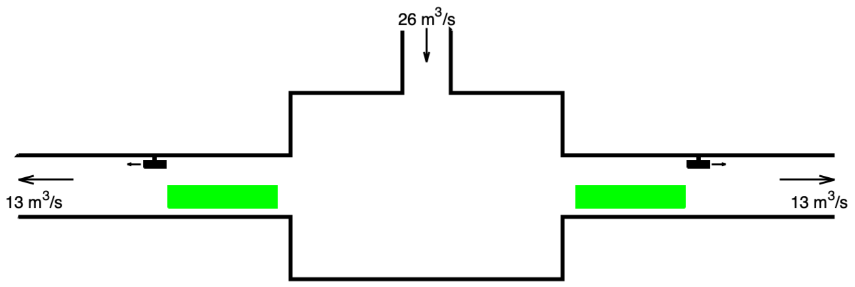

Figure 1: Airflow split in `SES-oddity-jet-fans-1.ses`

In a second test file (`SES-oddity-jet-fans-2.ses`) both trains are slightly closer to the shaft (Figure 2). Part of each train is now inside the crossover cavern. The geometry, trains and fans are still symmetrical. We would expect the airflow to still be symmetrical, but it isn't. What gives?

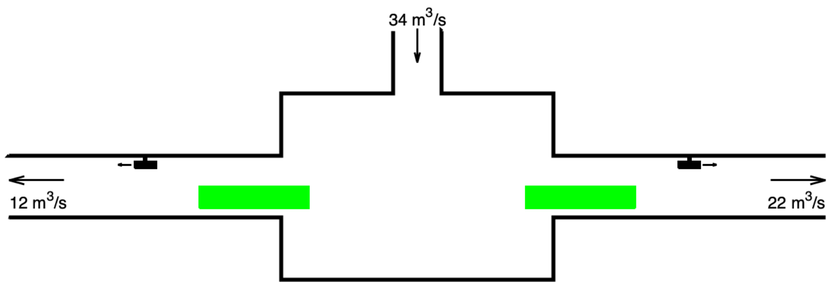

Figure 2: Airflow split in `SES-oddity-jet-fans-2.ses`

It took me a while to figure out this behaviour in the source code[1].

[1] In the source files `Omega2.for` and `Omega5.for`.

It turns out that jet fans are considered to be at the back boundary of the segment they are in. If there are trains crossing the back boundary then the jet fans in that segment are considered to be in the annulus around those trains.

And as we all know, jet fans in small tunnels in SES generate higher pressure rises than the same jet fans in large tunnels. Figure 3 is an updated version of Figure 2 showing the orientation of the segments and the jet fan icons moved to where SES places them (at the back ends of the segments).

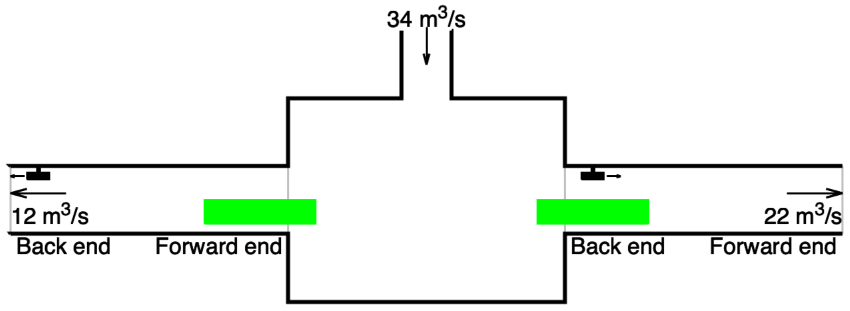

Figure 3: Orientation of segments

Figure 3 makes the cause of the difference clearer. The jet fan on the left is in the open tunnel and the jet fan on the right is in the annulus around a train. The jet fan in the small tunnel (the annulus area) generates a higher pressure rise than the jet fan in the open tunnel, so the flow is not symmetrical.

The pressure rise of jet fans is calculated in line 178 of routine `Omega5.for`. It reads

```
GAIN=QIFS(IFS)*(VIFS(IFS)-VNETRG)/ANETRG
```

That is equivalent to

$$\Delta P = Q_{jf}\frac{(v_{jf} - v_{ann})}{A_{ann}} \tag{1}$$

where

ΔP	$=$	dimensionless pressure rise (m^2/s^2),
Q_{jf}	$=$	jet fan volume flow $\times$ installation efficiency (m^3/s),
v_{jf}	$=$	jet fan air velocity (m/s),
v_{ann}	$=$	annulus air velocity at the back end of the segment (m/s),
A_{ann}	$=$	annulus area at the back end of the segment (m^2).

`VNETRG` and `ANETRG` are local (annulus around trains) variables in `Omega5.for`; their calculation can be seen in lines 124 to 153 of that routine.

I stumbled on this behaviour on a fee-earning project in which the jet fans seemed far too effective, and I took the time to investigate what was up. Eventually I arrived at the simplified test files in this example. If you have a file in which the jet fans seem to be too effective, it is probably due to this undocumented feature (the jet fans being alongside trains).

A devil's advocate could argue that the geometry is not truly symmetrical—the tunnel on the right should be reversed (forward end at the left, backward end at the right) to make it so. But there is nothing in the SES User Manual to suggest that segment orientation matters and nothing that states that jet fans are in the annulus area at the back end of the segment (as far as I can see).

3 Speed limits

This is a minor one.

Train routes have speed limits, that force a train to slow along a particular piece of track (e.g. on tight bends).

SES was programmed so that trains are free to accelerate as soon as the nose of the train clears the speed limit. That behaviour is easy to program but differs from how speed limits work in real railways (where trains only accelerate once the tail of the train clears a speed limit). This means that speed limits on railway alignment drawings should be adjusted before being put into SES, to add the length of the trains to the extent of the lowest speed limits.

Few users notice this odd behaviour in SES. Most junior ventilation engineers of limited railway experience tasked with building SES models won't know of it. They just put in the speed limits they get from the alignment drawings, allowing trains to accelerate earlier than they should.

This has real-world implications. If trains in SES are allowed to accelerate to full speed out of stations earlier than trains do in the real world, fatigue pressures on trackside infrastructure such as platform screens or trackside dampers will be overestimated. Which is not necessarily a bad thing: it may be wrong, but at least it is conservative.

It is possible to modify SES so that trains keep to the lowest speed limit that exists between their nose and tail (I added this to Aurecon's internal version of SES). It wasn't as easy as I thought it would be: I ended up iterating over all the track sections between the nose and the tail and taking the lowest speed limit. I also learned more about coasting in SES than I ever wanted to know.

4 True static thrust of jet fans

In SES v4.1, the static thrusts of jet fans are defined by a theoretical equation that uses volume flow Q, jet velocity U_o and standard air density $\rho = 1.2$ kg/m^3. The static thrust T of a jet fan is calculated by $T = \rho Q U_0$.

Choosing volume flow and jet velocity as inputs turned out to be unfortunate, although the programmers of SES were probably unaware of that in the 1970s. These days, most jet fan catalogues list a tested static thrust, an air velocity and a volume flow, all derived from the test methods specified in ISO 13350 (which was first published in 1999, although I think that drafts had been floating around since the 1980s).

The input required by SES v4.1 in form 7C is a volume flow and an air velocity.

The obvious thing for a naïve engineer with access to a modern-day jet fan catalogue is to take the volume flow and air velocity from the catalogue and put them into an SES input file. Those are the inputs the program asks for... they're in the fan supplier's catalogue... what could possibly go wrong?

Alas! If you put the volume flow and air velocity from a jet fan catalogue and let SES calculate the thrust from them, then the thrust it calculates is typically about 6% higher than the tested static thrust given in the catalogue.[2] Many engineers have fallen into the trap of just taking the volume flow and airspeed from a catalogue and putting them into an SES file, thus accidentally overstating the fan performance.

Some of these projects have involved hundreds of jet fans. Good luck telling your contractor client that they need to cost 212 jet fans instead of 200 late in the tender design phase. Not to mention the knock-on effects on the electrical design of adding another dozen jet fans and splitting them between enough redundant LV boards.

I see that WSP got burned by the same trap, as in SVS v6 the form 7C input has been changed to static thrust and jet velocity—a sensible change that eliminates the problem entirely.

For what it is worth, there are similar traps in IDA Tunnel and Motts Aero. In those programs the inputs are jet fan area A and jet speed U_0. I've lost count of how many times I've had to explain to my engineers that they can't calculate the fan area A from the fan diameter D (no matter how logical $A = \frac{\pi}{4}D^2$ may seem). The area of one jet fan in IDA/Aero must be back-calculated from $A = T/(\rho U_0^2)$, where T and U_0 are the static thrust and jet speed in the supplier's catalogue.

To my mind the only sensible way to define jet fans in a tunnel ventilation program is to set a static thrust, jet velocity, installation efficiency and a count of jet fans.

[2]Not always: I've seen some entries in fan catalogues where the theoretical static thrust was 10% lower than the tested thrust.

5 Train performance option 2

SES has three train performance options (internally it is assigned to a variable named `TPOPT`):

- implicit train speed calculation and implicit train heat emission calculation (TPOPT 1)
- explicit speed and implicit heat (TPOPT 2)
- explicit speed and explicit heat (TPOPT 3)

Options 1 and 3 work fine, as far as I can tell.

Option 2 has a big problem with the tractive effort required to climb hills and a smaller problem with the tractive effort required to negotiate curves.

In option 2, users are required to set the geometry of a route (gradients and track radii), an explicit speed-time curve (form 8E) and define the train performance (form 9). SES follows the speed-time curve in form 8E and uses the route geometry and train performance to calculate how much tractive effort is required to keep to those speeds. It feeds that tractive effort into the calculation of heat emission from the traction and braking systems.

Figure 4 shows the difference in tractive effort required—and waste traction heat emitted—by a train on the level and uphill in option 1 and option 2 calculations.

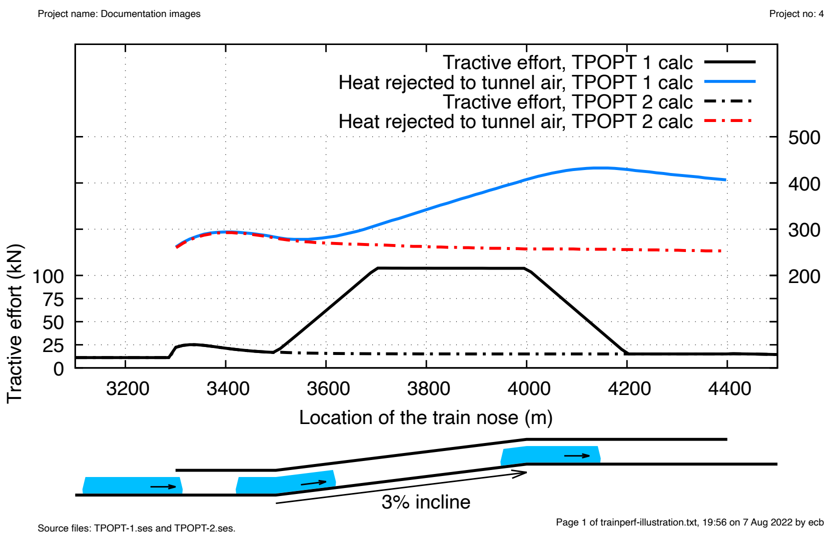

Figure 4: Comparison of tractive effort and heat rejected, `TPOPT 1` & `TPOPT 2`

On level track the train exerts the same tractive effort in both calculations. But when any part of the train in the option 1 calculation is on the incline, the tractive effort needed to raise the train rises drastically. In the option 1 calculation the traction system works hard to climb the incline and emits a lot of waste heat. In the option 2 calculation, the traction system doesn't have to work hard to climb the incline and emits much less heat.

To explain the cause of the difference, we'll trace what happens in an option 1 calculation and compare it to what happens in an option 2 calculation.

SES uses two routines (`Train.for` and `Locate.for`) to calculate the tractive effort required to climb hills and overcome curve resistance. The option 1 calculation and the option 2 calculation use them in slightly different ways that allow a problem to occur in the option 2 calculation.

5.1 Option 1 calculation

`Locate.for` does most of the work: it calculates the tractive effort required to climb hills (a variable named `RG`) and the tractive effort required to overcome curve resistance (variable `RC`). When set in `Locate.for`, `RG` and `RC` are in dimensionless units (`RG` is a fraction of train mass, `RC` is in pounds per short ton). Curve resistance `RC` rarely matters, but gradient resistance `RG` is a huge influence on the tractive effort and a correspondingly large influence on the heat emitted from the traction system.

Once the values of `RG` and `RC` return to `Train.for` in the option 1 calculation, `Train.for` calculates the train's gross mass. `RG` is then multiplied by train gross mass in pounds and `RC` is multiplied by train gross mass in tons. The relevant extracts of the code in `Train.for` are given below (preceded by their line numbers in that file).

```
485 C*****FIND TRAIN POSITION, AND ITS GRADE AND CURVE RESISTANCE
486       CALL LOCATE (NUMV,ITYP,IROUTE,RG,RC)
487 C*****FIND TRAIN TOTAL WEIGHT, GRADE AND CURVE RESISTANCES AND SUM OF
488 C*****GRADE, CURVE, AND ROLLING RESISTANCES
489       WTRN=WV(ITYP)+WPATV(NUMV)
490       RG=RG*WTRN
491       RC=RC*WTRN/TONLB

506       RSISTV(NUMV)=RSISTV(NUMV)+RG+RC+RM
```

Line 486 calls `Locate`, which returns the values of `RG` and `RC`. Line 489 calculates the mass of the train (`WTRN`) as the tare mass (`WV`) plus the mass of passengers aboard (`WPATV`). Line 490 converts `RG` from fraction of train mass to pounds of tractive effort required. Line 491 converts `RC` from pounds per ton to pounds of tractive effort required (`TONLB` is a constant declared in `Dses.for`, the number of pounds in a short ton).

Line 506 calculates the total resistance to the train's movement (`RSISTV`). It lumps together aerodynamic resistance, mechanical resistances, grade resistance and curve resistance (all in pounds). Everything is in the same units and the calculation is correct.

5.2 Option 2 calculation

The option 2 calculation in `Train.for` is shorter (and too short):

```
653 C*****FIND TRAIN POSITION, AND ITS GRADE AND CURVE RESISTANCE
654       CALL LOCATE (NUMV,ITYP,IROUTE,RG,RC)

672       RSISTV(NUMV)=RSISTV(NUMV)+RG+RC+RM
```

Line 654 gets the values of `RG` as a fraction of train mass and `RC` in pounds per ton. Line 672 adds the value of `RG` (still as a fraction of train mass) and the value of `RC` (still in pounds per ton) to the other resistances (which are in pounds).

None of the lines between 654 and 672 convert `RG` or `RC` into pounds. There ought to be something along the lines of

```
      RG=RG*WV(ITYP)
      RC=RC*WV(ITYP)/TONLB
```

after line 654 to convert `RG` and `RC` into pounds. And a note in the manual to say that when using train performance option 2, users should not set an average empty carriage mass in form 9E but instead set a representative carriage gross mass (tare mass plus a mass of passengers).

Not having those two crucial extra lines of code in the option 2 calculation means that the tractive effort needed to climb gradients is too small by a factor of `<train mass in pounds>`. Curve resistance is wrong by a factor of `<train mass in tons>`; but on an upgrade, the curve resistance is just a rounding error compared to gradient resistance.

I first encountered this while working on a diesel freight tunnel in 2014. We wanted to ensure that the heat emissions in SES were correct, so a colleague wrote a tractive effort calculation in a spreadsheet, using a constant train speed. I wrote an SES file with constant train speed in train performance option 2 to give us a side-by-side comparison. We knew something was up when we saw a four thousand tonne coal train waltzing up an incline with negligible tractive effort in SES.[3]

[3] We first plotted it from the results of a custom version of SES that I had just compiled, so I turned white. Fortunately for me we saw the same train waltz up the same incline in a vanilla v4.1 run. If it hadn't, I would probably never have lived it down.

6 Count of branched junctions

In form 1D the fifth entry is a count of branched junctions (the count of how many nodes there are where three or more sections meet).

It is hardly ever considered, but it can have a significant effect. As far as I can tell, the actual count of junctions doesn't matter, as long as it is one or more.

If it is one or more, SES will call a routine that calculates pressure losses at nodes. If it is zero or negative, SES will skip the calculation of node pressure losses and the change in total pressure across every node will be zero.

I came across it when a colleague asked for help with a file in which a Saccardo junction (an extension I had written) was having no effect on the airflow. On the surface, everything in the file seemed fine. Thoroughly puzzled, I started tracing the program flow. Eventually I found that the routine that calculated the pressure changes across junctions (`Omega3.for`) was not being called. By tracing the code logic, I figured out that the cause was that the counter of branched junctions in the input file's form 1D (`NBRJCT`) was zero. This caused lines 17–18 of `Qderiv.for` to skip an all-important call to `Omega3.for`. Without that call, the changes in total pressure across nodes were all set to zero.

The count of branched junctions was no doubt a useful switch to have in the 1970s when computing time on mainframes cost serious money. If you were doing a run in which you knew that all the pressure losses across junctions were zero you could tell SES to skip the calls to calculate junction pressures and reduce the cost of running your file on the mainframe.

Nowadays the cost of running files is negligible, so setting the count of branched junctions to zero is nothing but a trap for the unwary. You should never set the fifth entry in form 1D to zero.

SES does not warn you if you have branched junctions that ought to calculate pressure losses (junction types 1–6) but have set `NBRJCT` to zero. This seems like a bit of an oversight. Serious users should add such a check to their preprocessor or postprocessor, or (better still) disable the program's ability to skip the call to `Omega3.for` entirely and deprecate the fifth entry in form 1D.

7 (Pounds per ton) per (mph per second)

One of the entries in form 9E is the amount of tare train mass to treat as spinning masses (wheels, motors) that have a gyroscopic resistance to acceleration. Most modern train performance programs require an input as a percentage of tare mass, often defaulting to 10%. SES v4.1 requires it in units of (lb/ton)/(mph/sec), with a default value of 8.8 (lb/ton)/(mph/sec) if you set it to zero.

Many years ago I spent a long flight delving into the source code of `Garage.for` trying to figure out why the authors chose (lb/ton)/(mph/sec) as the units in the input and what those units might equate to in an SI units–US units conversion program.

The relevant variable in SES is `RRACC`. This is what `Record.txt` has to say about `RRACC`:

```
RRACC(ITYP) IS THE ROTATIONAL ACCELERATION RESISTANCE OF ROTATING
       PARTS.  THIS NUMBER IS THEN CONVERTED BY THE FOLLOWING FORMULA

       RRACC(ITYP) = (91.2 + RRACC(ITYP))/(91.2 * GRACC )
       NOTE 91.2 = ((2000 LBS/TON)*(5280 FT/MILE))/
                   ((32.174 FT/(SEC**2))*3600 SEC/HR)
       EXTERNAL (LBS/TON)/(MPH/SEC),
       INTERNAL (EQUIV. TRANSLATIONAL MASS, SLUGS)/(WEIGHT,LBS)
```

That last line is the key to what is going on. The input number is converted from (lb/ton)/(mph/sec) to a ratio of masses (slugs per pound).

The parameter `GRACC` in the calculation is the acceleration of gravity, set in `Dses.for` as 32.174 ft/s^2. A slug is a unit of mass equal to 32.174 pounds—a unit that no doubt made sense when it was defined, but which just looks weird to those of us accustomed to metric.[4]

The default input value in SES is 8.8 (lb/ton)/(mph/sec), resulting in SES calculating the internal value for `RRACC` as

$$\frac{91.2 + 8.8}{91.2 \times 32.174} = 0.03408\ slugs/lb. \tag{2}$$

So each lb of train tare mass is equivalent to a mass of 0.03408 slugs when you account for the rotating masses. We can convert the slugs/lb to lb/lb by multiplying by the number of pounds in a slug:

$$0.03408\ \ slugs/lb \times 32.174\ \ lb/slug = 1.0965\ \ lb/lb. \tag{3}$$

So 8.8 (lb/ton)/(mph/sec) is another way of saying that the default SES input value for rotational inertia in the train adds 9.65% to the train's tare mass. This is good news; it means that 8.8 (lb/ton)/(mph/sec) is consistent with the 10% tare mass I keep getting told to use by traction power engineers.

I see that WSP decided to do something about this in SVS v6, changing the input to an equivalent rotational mass per car (kg). Personally, I prefer to use units of `% tare mass` in SI input files (so that the rotational mass stays as 10% if a user changes the carriage tare mass). But it doesn't really matter.

[4] A bit like expressing the cooling power of chillers as tons of ice per day: it made perfect sense in the early 20th century, when people who owned cold stores bought ice by the ton and might be in the market for one of those newfangled chillers. Best to express cooling power in terms they understand.

8 Motor voltage

The last number in form 9F is the motor terminal voltage, which is stored in an array named `VOLTMV`. The number is read and checked in `Garage.for` but is never used anywhere else (you might think that motor voltage would be used in the motor resistive losses, but they are calculated as I^2R losses—motor voltage isn't involved).

The lines below from `Garage.for` are the only lines that use `VOLTMV` that I could find:

```
C-----   INPUT FORM 9F
      READ (IN,360) TITLE1,DUMY1,DUMY2,DUMY3,DUMY4,DUMY5,DUMY6,VOLTMV(I)

       WRITE (OUT,401) VOLTMV(I)
       CALL CHECKR (VOLTMV(I),100.0,1000.0,238)
```

All those lines do is read the number, print it to the output file and check if it is in the range 100–1000 V. It is not used anywhere else. So why is it there at all? Perhaps they included it in case they found a use for it later. But they never did, so the last number in form 9F can be set to anything between 100 and 1000 V without affecting the calculation or raising input error 238.

9 Heat from the traction system

One of the most confusing aspects of SES v4.1 is the calculation of train performance, particularly the heat rejected from the traction power system.

9.1 A two-finger salute to the laws of thermodynamics

The main oddity in the traction power calculation in SES v4.1 is that users are free to set up the inputs in form 9 such that the power going into the traction system (DC voltage × DC current) is less than the power delivered at the wheel-rail interface (tractive effort × train speed).

The losses in the traction power system are all applied to the power into it.

You can set the line voltage, line currents and the efficiency of choppers to (say) 92% and SES will duly turn 8% of the input power into heat. But if the input power is less than the power at the wheel-rail interface then your true traction efficiency will be too good. In one of its major failings, SES v4.1 doesn't warn you if your train delivers more power at the wheel-rail interface than it draws from the catenary.

Careful users usually end up doing their own calculations with the tractive effort, train speed, line currents, line voltage, motor currents, motor resistance and chopper efficiency. I got frustrated with figuring these calculations out in spreadsheets, losing the spreadsheets then having to figure it out again in new spreadsheets a few years later.

Eventually I added code to Aurecon's version of `Garage.for` that calculated the losses in the traction power system for me. It printed a new table of efficiencies after form 9I in the .PRN file.

That strategy worked: I no longer needed to relearn how the traction power calculation worked every few years. Instead I could adjust the inputs in form 9 and look at the table of efficiencies in my modified .PRN file until the efficiencies were what I was aiming for. My junior colleagues didn't have to learn it. . . though the smarter ones took the time to learn it anyway.

It worked so well for me that when I wrote my own SES converter under the Hobyah suite (`SESconv.py`) I included a similar table printed to the SI output file. The following is a sample of one of the simpler versions of that table (edited slightly to make it typeset properly in LaTeX). It is printed after the output for form 9I:

```
INTERNAL MOTOR RESISTANCE                            0.300   OHMS/MOTOR    FORM 9I
Hobyah freebie: a table of traction efficiencies
                         Power delivered at wheel-rail interface
Traction efficiency = -------------------------------------------
                   Power delivered at wheel-rail interface + loss in traction

Loss in traction is I^2 R losses in the motors + chopper inefficiency.  There are
12 motors and 6 powered cars, so:
   Loss in traction = 12 * (motor amps)^2 * (circuit resistance)
                     + 6 * (line voltage) * (line amps) * (1 - chopper efficiency)

           Tractive     Power at      Loss in       Efficiency      Chopper
 Speed      effort       wheels      traction      of traction    efficiency
(km/h)    (N/train)     (W/train)    (W/train)         (%)            (%)
   0.00     663976             0       611510           0.00         92.0
   8.05     663976       1484119       694310          68.13         92.0
  16.09     663976       2968239       777110          79.25         92.0
  32.19     663976       5936478       943430          86.29         92.0
  48.28     485265       6507981       354998          94.83         92.0
  80.47     290860       6501300       354998          94.82         92.0
```

There are three variants of the table:

- cam control
- chopper control with the switched resistances in form 9I set to zero.
- chopper control with non-zero switched resistances in form 9I (you should not use this. SES doesn't raise a fatal error, but I reckon it ought to).

The converter can be told to print additional debugging data by using the `-debug1` option when running from the command line. Anyone interested in checking the calculation in detail can find it in routine `WriteTrainEfficTable` in the `SESconv.py` source code on `github.com/ECB2020/Hobyah`.

No doubt WSP got sick of faffing around with their spreadsheets too. In SVS v6 they implemented train controller option 3.

I haven't used it yet. But as far as I can tell from the manual, option 3 ignores the voltages, currents and resistances entirely. It calculates the losses from an efficiency value (varying with speed) applied to the power delivered at the wheel–rail interface (see pages 154 & 155 of the SVS v6 User's Manual). Much better!

9.2 Line currents

The calculation of line currents in SES v4.1 is odd. The heart of the oddity is that the line current per powered car is limited to twice the motor current regardless of how many motors there are in a powered car. I am no expert in DC power traction systems but I think that the SES behaviour is down to how DC traction power systems in EMUs used to work: they usually had four motors per powered car (two per bogie). They ran all four motors in series at low speed and two pairs of motors in parallel at high speed (see Figures 8.2 and 8.3 of the SES v4.1 User's Manual).

Modern power trains with AC and variable speed drives (choppers) don't have that limitation. In the past I have tried to set up traction power calculations in SES in which the line currents were four times the motor currents, and those runs did not make sense because SES always limited the line current at higher speeds to no more than twice the motor current. These days I don't bother to make them match when using v4.1. Instead I set:

- chopper control,
- no more than two traction motors per powered car,
- high motor currents,
- zeros for the first four entries in form 9I
- lowest permissible motor resistance for the fifth entry in form 9I (0.001 ohms),
- line currents that are never more than twice the motor current,
- all the losses in the chopper efficiency.

I then adjust the line currents until the table of traction efficiencies in the printed output from `SESconv.py` matches the efficiencies I want.

10 Density effects in fire runs

In 1980 SES version 3 was released. Version 3 added a fire model (based on the mine ventilation software MFIRE). V3 had calculations of air density ratio in each line subsegment, air density ratios at each node and mean air density ratio in the subsegments of each vent segment.

The addition of air density changes meant that the programmers had to make decisions about how to handle the effects of density on fixed losses, jet fans and axial fans.

10.1 Jet fans

Jet fans ("impulse fan systems" in the code comments, with acronym `IFS`) are taken to be in cold air upwind of the fire (this is stated in the comments in `Omega5.for`):

```
C --  IFS IS ASSUMED TO USE OUTSIDE AIR OR BE UPSTREAM OF
C --  FIRE - HENCE NO DENSITY ADJUSTMENT
```

This assumption that a jet fan would always be passing cold air is something I find odd. It is not difficult to figure out the direction of airflow at the jet fan and use that to choose whether to derate it by the air density ratio in the first subsegment or by air density ratio in the node at the back end of the segment. The programmers of SES v3 took the time to derate axial/centrifugal fan characteristics in vent segments for density; it's weird they didn't do the same for jet fans at the back ends of line segments. Perhaps they ran out of time.

10.2 Axial/centrifugal fans

The manual states that fans in vent segments are derated according to the local air density. After I sussed out the undocumented behaviour that places jet fans at the back end of their line segments (see section 4 of this document) I began to wonder if something similar affected fans in vent segments. My line of thought was as follows:

- Each subsegment in a vent shaft carries air at a different density in a fire run.
- Are the fans in the first or last subsegment?
- Could the orientation of a vent segment affect the performance of its axial fan?

To test this I created input files that morphed into the contrived run in Figure 5.

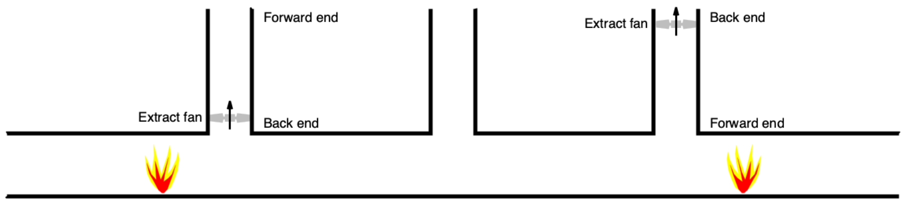

Figure 5: Extract shafts with different orientations

Figure 5 shows a symmetric geometry. It has two identical axial fans and two identical fires (the file the figure is based on is named `SES-oddity-axials-1.ses`). The only difference between the fan shaft on the left side and fan shaft on the right is that the shaft on the left is connected to the incident tunnel at the back end and the shaft on the right is connected at the forward end. For illustrative purposes the fan icons are shown at the back ends of the vent shafts.

The vent shafts are long and have many subsegments, each of which has its own air temperature and air density. If axial/centrifugal fans are at the back ends of their vent segments (in the same way that jet fans are at the back ends of their line segments) then the two fans in this run should be passing air at different densities and end up at different points on their fan characteristic.

When run, the air flow was symmetrical and the two shafts extracted the same amount of air, as shown in Figure 6.

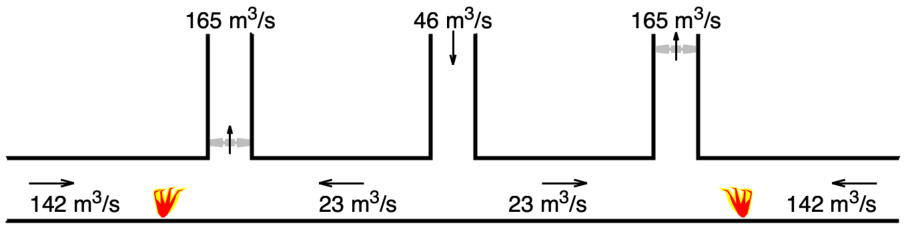

Figure 6: Airflow results

When I looked into it further, it became apparent that `Pinpnt.for` calculates the mean of the temperatures of all the subsegments in a vent shaft at each timestep. The loop in lines 107–132 of `Pinpnt.for` sums up the subsegment temperatures. Lines 134–139 divides by the count of subsegments to get the mean temperature, then divides the mean by external air temperature to yield a temperature ratio. Routine `Omega1.for` takes the inverse of that temperature ratio as a proxy for the density ratio of air passing through the fan and uses it to adjust the pressure rise that the fan in the vent segment can exert.

So the temperature derating of fans in vent segments is independent of the segment orientation. Whew!

Actually the amount of air extracted in the example run does differ: 164.63 m^3/s and 164.632 m^3/s. I tried this with a few different versions to check if it was a compiler-specific thing (the 2002 Windows executable, SES v4.1 compiled by gfortran with compiler optimisation off and on), but I always saw a slight difference. The difference went away when both shafts were oriented the same. It isn't obvious to me what the cause of the difference is and—to be honest—I can't be bothered to delve into it (Fortran 66 and all that). Who cares about a difference of 2 ℓ/s out of 165,000?

10.3 Fixed losses and friction

The input files for SES let the user set pressure loss factors at the back end and forward end of each segment, for forwards airflow and backwards airflow (four pressure loss factors per segment). These are stored in arrays `CFPLS`, `CFNLS`, `CBPLS` and `CBNLS`. The pressure loss factors for forwards airflow are added together and stored in `CEPS`. The pressure loss factors for backwards airflow are also added together and stored in `CENS`.

Adding the pressure loss factors at the back end and forward end of segments was a good move for a program that had constant density (SES versions 1 and 2). Not quite so attractive when the fire model was introduced in version 3.

Before I looked into the code, I had the horrible feeling that in fire runs the pressure losses in `CENS` and `CEPS` would be applied to the air velocity at the mean air density in the line segment (because they did that for the fans in vent segments). When I ran a few test cases and looked at the source code (`Omega4.for` and `Omega5.for`) I was happy to find that my gut feel was wrong.

`Omega4.for` and `Omega5.for` (the routines that adjust segment pressure losses in fire runs) do not use `CEPS` and `CENS`.

Instead, they calculate two density corrections in each segment (one at the back end, one at the forward end) and apply them to the four pressure loss factors that the user set at the back end and forward end (arrays `CFPLS`, `CFNLS`, `CBPLS` and `CBNLS`). Similar consideration is given to wall friction, which is calculated in each subsegment applying a density correction appropriate to that subsegment's temperature. In terms of density adjustment of fixed losses and friction in fire runs, SES seems to be pretty solid.

11 TAALS

`TAALS` is an array of real numbers that define the amplitude of the annual fluctuation of the air temperature in each line segment. It is used to determine the summary/environmental control zone (ECZ) temperatures in subsegments over the course of the year. It is declared in the file `DSHRHS`, a common block only used in the ground heat transfer calculation.

In routine `Dthts2.for`, every entry in `TAALS` is set to half the annual fluctuation of outside air temperature (`ANNAMP`, set in form 1F). It looks like it should be a first guess (because an array of values is set, one per segment).

I expected `TAALS` to have been updated during each ECZ estimate, with lower fluctations in segments far from atmosphere and higher fluctuations in segments near to atmosphere. That's behaviour that I've seen from calculations by other programs and test data.

But as far as I can see, none of the values in the `TAALS` array get updated during the run. The `TAALS` array is used in routines `Dthts2.for`, `Wlltm2.for` and `Julie.for` but it is never updated after being set to `0.5 * ANNAMP`. The following extracts are where I found it in the code:

```
Record.txt:
 1920   TAALS(ILS)  IS THE AMPLITUDE OF THE ANNUAL FLUCTUATION OF THE
 1921          AIR TEMPERATURE IN A LINE SEGMENT.  DEG F.  ( C,R ).

DSHRHS:
   22  C LINE SEGMENT VARIABLES  (LMLSEG)
   23        COMMON   /HSSHAR/  TAALS(LMLSEG)

Dthts2.for:
   98       DO 400 ISEG=1,NLS
  101            TAALS(ISEG) = .5*ANNAMP
  123   400 CONTINUE

Wlltm2.for:
   69        CALL JULIE(HYRAD,RPRIM,OMEG,THDFLS(1,ISEG),THDFLS(2,ISEG),
   70       1   THCNLS(1,ISEG),THCNLS(2,ISEG),HCNLSS(ISS),TAALS(ISEG),
   71       2   0.0,0.0,DUMY1,HYRAD,TANN,ISKIP)
   72        TWALLA(ISS) = TWALLA(ISS)+TAALS(ISEG)*COS(OMEG*DUMY1)*(1.-
EFANNL)
   73       1              + TANN*EFANNL

Julie.for (where TAALS(ISEG) is given the name AA):
    1        SUBROUTINE  JULIE(  R, RPRIM, W, AL1, AL2, K1, K2, HA, AA,
    2       1            TAVG, TDS, TIM, RADIUS, TWALL, ISKIP )

   21  C      AA    - AMPLITUDE OF THE ANNUAL TUNNEL AIR TEMPERATURE VARIATION
   22  C              ABOUT THE MEAN VALUE,  DEG F.
   ..
  158        T1BAR = HA*ZMC8*RARGMO/ZMC4*(AA*COS(TAU1+RARGTH+W*TIM)
  159       1+TM*COS(TAU1+RARGTH))+HA*ZMC1*RARGNO/ZMC7/ZMC2*(AA*COS(TAU2
  160       2+RARGPH+W*TIM)+TM*COS(TAU2+RARGPH))
  ...
  171        T2BAR = ZMC9*HA*RARGNO/ZNOX2*(AA*COS(TAU3+RARGPH+W*TIM)+TM*
  172        1COS(TAU3+RARGPH))
```

Why have an array of values that are set to `0.5 * ANNAMP` but are never updated? It is a puzzle.

Programmers had huge concerns about memory in the 1970s, so why did they not just create one new `REAL` variable, give it the value `0.5 * ANNAMP` and use that? Why create an entire array of `REAL`s, all with the same value?

It has all the hallmarks of allocating an array that they planned to update with each ECZ estimate, but didn't get around to programming in.

12 ECZ estimates that fail to converge

When I work on large, long-term projects I tend to build SES files that have all the air paths included as line or vent segments. If I need a particular air path to be closed (such as an escape cross-passage or a vent adit) I don't remove the section from the simulation file (too much work, and it caused my plotting programs to raise errors about nonexistent segments). Instead, I set a small area and high pressure loss factors (ζ) in those airflow paths to reduce the airflow through those air paths to a low level.[5]

I never used to adjust the perimeters to be more in tune with the smaller areas because I didn't think it mattered. That turned to have been a very bad idea when doing ECZ estimates.

In an SES thermodynamic calculation a vent segment with a small area (3 ft^2), a large perimeter and a low airflow can cause an ECZ estimate to fail. I don't understand why. But I've noticed that air paths with small areas, high perimeters and low airflow trigger some floating point voodoo in the ECZ estimate code that causes heat to drain out through the high perimeter and freeze the air in the air path.

Then the ECZ estimate blows up in my face. These days, when I want an air path to be closed I set the smallest area, highest ζ **and** small perimeter…just to be safe.

13 Restart run file names

One of the joys of working with a program designed to take input from punched cards is that lines of input are limited to 80 characters. This 80-character limit holds for the path name and file name of restart files, which in SES v4.1 can

[5]As an aside, I have seen a lot of "isolation dampers" so poorly set at commissioning stage that they leak a crazily large amount of air when in the closed position. Removing the air path of closed dampers doesn't seem right to me if you want to model real-world performance. There is always some leakage, and cowboy contractors never take the time to reduce it to a minimum.

be placed on the line after form 13. On quite a few occasions I've had to move restart files to folders higher up in the folder hierarchy to avoid exceeding the 80 character limit.

I've also noticed that I can reliably get SES v4.1 to crash with a Fortran runtime error (end of file) and a backtrace message by writing the name of the restart file on the last line or second-last line of the input file. There must be two blank lines after the line holding the file name. This seems to be because SES checks to see if the restart data is appended to the input file but doesn't trap EOF errors when it tries to read the restart data.

14 Zero length perimeters in form 3B

Hairsplitting time!

Form 3B allows the user to set up to eight pairs of perimeters/roughness heights, from which a mean weighted roughness height is calculated. SES goes about this calculation in an odd way in they eyes of modern programmers, but which I think would have made sense in the 1970s with punch cards. It is best to think about it in terms of how SES was originally written, so think of it thus: SES runs two loops, one when it reads the punch card of perimeters and another when it reads the punch card of roughnesses.

Lines 154 to 158 of `Lsins.for` are the first loop. Those lines adds up eight perimeters (blank entries are counted as zero values). Each time the loop encounters a non-zero value of perimeter it increments a general-purpose counter, `NUMY1`.

Lines 172 to 175 are the second loop. Those lines sum up the product of the first `NUMY1` perimeters and roughnesses (not the product of all eight).

The mean roughness is then calculated, using `NUMY1` values on the numerator and eight values on the denominator. The best way to illustrate the problem is to express the calculation of mean roughness height as

$$\bar{\varepsilon} = \frac{\sum_{i=1}^{i=\mathtt{NUMY1}} \varepsilon_i \times S_i}{\sum_{j=1}^{j=8} S_j} \tag{4}$$

where

S	=	a perimeter in the first line of form 7B,
ε	=	a roughness height in the second line of form 7B,
`NUMY1`	=	the count of non-zero perimeters in the first line of form 7B and
$\bar{\varepsilon}$	=	the mean roughness height.

This sums up all eight values of $S_j$in the denominator but only the first `NUMY1` values of $\varepsilon_i \times S_i$ in the numerator.

That allows a problem to occur when users set zero values of perimeter to the left of non-zero values of perimeter. Consider the following input of form 3B with four entries for perimeter (first line) and four entries for roughness (second line):

```
0.0         0.0         13.3        40.
0.12        0.15        0.1         0.33
```

I freely admit that this is a crazy way to set up your SES input file. But there is nothing to warn you that this is a bad idea, and I could see it being done in files generated by scripts with a library of surface roughnesses.

The first loop sums up eight perimeters (two zero values, two non-zero values and four empty fields that are treated as zero). When the first loop ends `NUMY1` is 2 (as two of the eight perimeters were non-zero). The second loop sums up the first `NUMY1` values of $\varepsilon \times S$: $0.0 \times 0.12 + 0.0 * 0.15 = 0.0$. SES writes the following to the output file:

```
SEGMENT                                                                  FORM 3B
 PERIMETERS           0.0     0.0                TOTAL PERIMETER        53.3  FT
ROUGHNESS                                        WEIGHTED AVERAGE
 LENGTHS           0.1200  0.1500                ROUGHNESS LENGTH     0.0000  FT
   HYDRAULIC DIAMETER                                          14.2   FT
   RELATIVE ROUGHNESS   ( E/D )                              0.00000
   FULLY TURBULENT FRICTION FACTOR (FROM AVERAGE ROUGHNESS)  0.0055
```

Now that is weird output. It ought to set off alarm bells for the following reasons:

- The printed total perimeter on the second line (`53.3 ft`) exceeds the sum of the printed individual perimeters (`0.0 ft`)
- The calculated mean roughness on the fourth line is zero even though the individual roughnesses are high.

If the eight numbers in form 3B are rearranged:

```
13.3        40.         0.0         0.0
0.1         0.33        0.12        0.15
```

we get a different result for the mean roughness and friction factor:

```
SEGMENT                                                                  FORM 3B
 PERIMETERS          13.3    40.0                TOTAL PERIMETER        53.3  FT
ROUGHNESS                                        WEIGHTED AVERAGE
 LENGTHS           0.1000  0.3300                ROUGHNESS LENGTH     0.2726  FT
   HYDRAULIC DIAMETER                                          14.2   FT
   RELATIVE ROUGHNESS   ( E/D )                              0.01925
   FULLY TURBULENT FRICTION FACTOR (FROM AVERAGE ROUGHNESS)  0.0473
```

The different outcome is all due to me having gamed the system. Someone wrote code in the 1970s that minimised CPU usage (a laudable goal) and made sense when humans prepared SES input on punchcards and could reject obviously stupid input like zero values of perimeter. So they wrote code that read eight perimeters and counted how many were non-zero. Then they read just that count of roughness heights instead of reading eight roughness heights (which took up costly, unnecessary CPU cycles in the 1970s).

To summarize: if you are foolish enough to use zero values of perimeter in form 3B then the order of the entries affects your calculated friction factor. Which is bad.

In the first arrangement we ended up with a segment with Darcy friction factor 0.0055. When we rearranged the order of the numbers in the input we got friction factor 0.0473, which is about nine times higher.

Many expert users reading this section will consider this an extreme example. I agree: nobody but an idiot would ever put a zero value of perimeter in form 3B to the left of a non-zero value of perimeter.

But in my experience of writing engineering software, idiot users are incredibly creative. Especially when we are writing automated scripts to generate SES input files and are not aware of thoroughly unlikely corner cases like this one.[6]

What to do about this? If you are writing an SES postprocessor you should check the total perimeter in the output of form 3B against the sum of the individual perimeters in form 3B and issue an informative error message if they don't match. If I may blow my own trumpet for a moment, look at error no. 4221 in `SESconv.py`:

```
> *Error* type 4221
> 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 204 in "SES-fault-4221-form-3B.PRN" are wrong.
> Faulty line of input (349th) is
>    PERIMETERS         17.2    8.0      0.0         TOTAL PERIMETER      53.3  FT
```

[6]Any ex-colleagues reading this need not worry that this is me casting aspersions. On many, many occasions the idiot user was me wearing my user hat instead of my programmer hat.

15 Mean roughness in form 3B

I spent some time delving into the code that processes form 3B to write the previous section, because 3B has always vaguely bothered me. After staring at it for a while I realized why. If you give SES more than one perimeter and one roughness, SES calculates a weighted mean roughness height then uses that mean roughness height to get the tunnel friction factor.

Think about the concept of the "*weighted mean roughness height*" of a vent duct for a moment.

I've always calculated the friction factors of the individual parts of perimeters and weighted each friction factor according to the length of perimeter it occupies. I'd like to be able to point to a source for why I do that; but I can't find one. It just makes sense to me to weight the friction factors.

Air ducts that have surfaces of different roughness happens all the time in tunnel ventilation design: I've worked on many projects in which the floors of smoke ducts in cut and cover tunnels were made from cast-insitu concrete (pretty smooth), the walls were made of secant piles (quite rough) and the ceilings were the undersides of T-beams (a standard item in structural engineering that is an incredibly rough surface as far as air flow is concerned).

Let's apply the concept of *reductio ad absurdum* to this. Consider flow between two infinite parallel plates (one rough, one smooth). You would expect a thick, high-friction boundary layer on the rough plate and a thin, low-friction boundary layer on the smooth plate. The friction factors of the two plates differ. The pressure gradient would depend on their mean friction factor.

You wouldn't expect the pressure gradient to depend on the friction factor calculated from the mean of the roughness heights of the two plates; that's just crazy.

There is a description of weighting the friction factors in section 6 of SES validation document UMTA-DC-06-0010-73-3 (1973), which was later used as the basis of Table 4.2 of the SES v4.1 User Manual. UMTA-DC-06-0010-73-3 talks about weighting friction factors according to the perimeter they extend over, but the User Manual talks about weighting roughness heights according to the perimeter they extend over (confusingly, Table 4.2 weights friction factors instead).

There's something wrong here, as lines 173 to 176 of `Lsins.for` definitely calculate a weighted mean roughness height. Nowadays I do all assessment of friction in a spreadsheet and set the input to SES as one perimeter and one roughness height—that way SES has no choice about how to interpret my input.

What's going on here? If I were to speculate I'd guess that a draft version of SES required form 3B to give pairs of perimeters and friction factors. At some point late in the program development someone ordered the programmers to make roughness height the input instead. They made the change in a way that was easy to program in, but which caused the program to make less sense in engineering terms. I've been there a lot, where some idiot ordered me to change something at the last moment and we ended up with a sub-optimal outcome. So I sympathize.

16 Five weird tricks to enlarge your arrays

Array sizes in SES are set at compile time and cannot be changed during the run (Fortran IV didn't allow arrays to be sized at runtime).

Anyone wanting to run a large model either has to split it in two or roll up their sleeves and start changing the `PARAMETER` statements at the top of `DSHARE`.

Editing the sizes of the `PARAMETER` statements in `DSHARE` and recompiling is how users increase the size of the SES arrays. Increasing them allows the program to handle models with more subsegments, segments, sections, nodes and train routes. But in a few cases, changing the size of one parameter means having to change the size of an apparently unrelated parameter.

Most of the relationships between the array sizes are documented in the comments in `DSHARE`, but I have been tripped up when changing a few that are not mentioned in `DSHARE`. Here they are.

16.1 LMTRRT and LMTBL2

`LMTRRT` is the count of train routes. If it is increased, the parameter `LMTBL2` should be increased too. The relationship isn't stated in `DSHARE`, instead it is in the comment on `LMTBL2` in `Record.txt` (it is complex, so you should study it if you are thinking about increasing the number of allowable train routes). For what it is worth, when I increased `LMTRRT` to 50 train routes, I ended up setting `LMTBL2` to 65,000 (up from 20 routes and 20,000 respectively in v4.1).

16.2 LMSS and LMSSTN

`LMSS` is the count of subsegments, `LMSSTN` is the count of subsegments plus the count of thermal nodes. It follows that `LMSSTN` ought to always be larger than `LMSS`.

16.3 LMSS and LMEQRM

`LMSS` is the count of subsegments, `LMEQRM` sets the size of the dynamic thermal response matrices (DTRMs) declared in `DSHRHS`. The higher the number of subsegments, the larger the DTRMs need to be. Don't overdo it; two of the DTRMs are 2D `REAL` arrays of size `LMEQRM` × `LMEQRM`, so it is easy to run out of memory on small machines if you are running a lot of SES simulations in parallel.

16.4 LMNODE and LMCOND

`LMNODE` is the count of nodes, `LMCOND` is the count of node pressure loss coefficients. They have a loose correlation, so if you increase the count of nodes you should probably also increase `LMCOND`. It may not be a problem; many nodes are type 7 nodes (which have no pressure loss coefficients and do not take up any space in `LMCOND`) but it's better to be safe than sorry.

16.5 LMLSS and Pinpnt.for

`LMLSS` the count of line subsegments (set to 1200 subsegments in v4.1). If it is increased, `Pinpnt.for` needs to be modified too. The relevant lines in `Pinpnt.for` are

```
55  C**** LINE SEGMENT VARIABLE (LMLSS)
56        DIMENSION  FLSS(LMLSS),AFSS(LMLSS), ABSS(LMLSS)
57        DATA  FLSS / 1200 * 0.0 /
```

Line 56 creates three new arrays of size `LMLSS`, one called `FLSS`. The `DATA` statement on line 57 fills the first 1200 entries in array `FLSS` with zero (when variables are created in Fortran they contain random data instead of being set to zero). If you increase `LMLSS` in `DSHARE`, you ought to change the count in line 57 to fill the whole of your larger array with zero.

I don't even know if it is necessary to fill `FLSS` with zeros before starting to calculate with it. But it may be easier to change "`DATA FLSS / 1200 * 0.0 /`" to "`DATA FLSS / LMLSS * 0.0 /`" so that you never have to think about it again when you change the size of `LMLSS`.

17 SI units conversion

Anyone who has to write programs to convert between US units and SI units has to make choices about what conversion factors to use in some circumstances. Most conversions are unambiguous, but a few are not. This section discusses the conversion factors that I consider ambiguous and explains how I came to choose the conversion factors that I use in my codes. The unambiguous conversions are discussed at length in the comments in `UScustomary.py` on `github.com/ECB2020/Hobyah`.

I'm putting this discussion of the ambiguous conversion factors here because when you write SI–US unit converters you run a risk of falling afoul of Finagle's Law. Explaining your choices up front is always sensible in these circumstances, because “explaining your choices up front” might end up being “getting your retaliation in first”.

17.1 Which BTU?

The BTU (British Thermal Unit) is a measure of energy. There are many definitions of the BTU, as anyone who looks up the conversion online can attest: thermochemical, US natural gas, IT plus a few others that Wikipedia has flagged with [*citation needed*]. The US National Institute of Standards and Technology (NIST) gives six slightly different definitions of the BTU in its publication NIST.SP.811 (US–SI conversion factors). Clearly NIST think that differentiating between these six different definitions is a useful thing for it to spend its time doing.

As far as I can see there is no One True definition of the BTU in SI units.

Anyone who has dug into the SES v4.1 source code wearing their “I need to figure out US–SI conversions” hat will quickly zero in on `Dses.for`, because most of the unit conversions are defined there. The conversion between power in watts and power in BTUs per second (BTU/s) is as follows (preceded by line numbers in `Dses.for`):

```
110    C*****CONVERSION FROM WATTS TO BTU/SEC - BTU/(WATT-SEC)
111          WTBTUS=9.4866E-04
```

SES defined the constant `WTBTUS` so that heat gains from purely electrical loads (traction losses, brake heat, train aircon) can be converted from watts into BTU/s for the air & wall temperature calculations. Line `111` means that SES v4.1 considers one watt to be the equivalent to 0.00094866 BTU/s. It follows that 1 BTU/s is 1054.1184 W and that 1 BTU is 1054.1184 J. For convenience we’ll call that number the “SES BTU”. 1 SES BTU is 1054.1184 Joules.

If a traction grid on a train is pumping out (say) 100 kW into the air in the tunnel (recall that the 100 kW is from a purely electrical calculation, so always done in watts), SES v4.1 turns it into heat gains of 94.866 SES BTU/s (341,517.6 SES BTU/hr) and uses that as the heat gain in its air temperature calculations.

How does SES v4.1’s internal definition of the BTU compare to current engineering practice? I’m no expert in BTU conversion factors, but the SES BTU seems weird to me. For starters, it is unique (as far as I can tell).

The two most common modern definitions of the BTU seem to be the International Tables (IT) BTU and the thermochemical BTU. The IT BTU is 1055.056 J (0.09% higher than the one used by SES v4.1) and the thermochemical BTU is 1054.35 J (0.02% higher). Using those two definitions, our notional 100 kW heat gain from the traction grid would turn into 94.781 IT BTU/s and 94.845 thermochemical BTU/s.

To me 94.866, 94.845 and 94.781 all seem close enough to not bother about, so for the last decade or so I've used the nonstandard definition of the BTU in line 111 of `Dses.for` when writing converters. I've continued to use it in `SESconv.py`, but I don't use it in the US–SI conversions for Hobyah output. Hobyah uses the IT BTU.

I've recently been playing around with WSP's conversion utility `ip2sip2ip.exe`. When the converter was written the programmers seem to have used the IT BTU: when I put a heat gain of 1,000,000 BTU/hr heat gains in form 9C `ip2sip2ip.exe` turned it into 2,930,711 watts. So 1 BTU must be 1055.05585 J in WSP's converter, which is as near as dammit the IT BTU.

17.2 Gravity

A relatively uncontroversial one. SES v4.1 defines the acceleration of gravity in `Dses.for` as 32.174 ft^2/s. The comments in the source code (`Dses.for`) identifies this coming from AMCA[7]:

```
C*****AMCA VALUE OF ACCELERATION CAUSED BY GRAVITY - FT/(SEC**2)
      GRACC=32.1740
```

32.174 ft^2/s, is equivalent to an SI value of 9.8066352 m/s^2. This is 1.6 parts per million lower than the official 1901 determined value of 9.80665 m/s^2, so I think we're good. Especially as gravity varies with latitude.

17.3 Density of mercury

SES v4.1 requires the user to put the atmospheric pressure into form 1F as inches of mercury. This means that a representative density of mercury needs to be selected if you are to convert your files between SI and US units. Density of mercury varies slightly with temperature, so you need to fix on one value of density if you want to convert an atmospheric pressure in Pa or kPa into the height of a mercury barometer.

Fortunately, SES v4.1 gives two definitions of standard atmospheric pressure in `Dses.for`:

```
C*****CONVERSION OF STANDARD ATMOSPHERE TO IN. HG. - (IN. HG.)/(S.A.)
      SAINHG=29.921
C*****CONVERSION OF STD. ATM. TO LBS/SQ IN - LBS/(SQ IN-S.A.)
      SAPSI=14.696
```

I think it is reasonable to assume from the code comments above that as far as SES is concerned, 29.921 inches on a mercury barometer means the air pressure

[7]AMCA = Air Movement and Control Association, a US body who write guidelines/standards for ventilation systems, often subtly different to the guidelines/standards written by SMACNA, ASHRAE, AIRAH and CIBSE.

is 14.696 psi. Pounds and inches have exact equivalents in SI units. A bit of factoring gives an expression for the density of mercury:

$$\rho_{\mathrm{Hg}} = \frac{lbs/in^2 \times kg/lb}{in.\ Hg \times (m/in.)^3} \tag{5}$$

$$= \frac{14.696 \times 0.45359237}{29.921 \times 0.0254^3} = 13595.26\ \mathrm{kg/m^3} \tag{6}$$

18 Zero values in forms 6B and 10

In form 6B and form 10, temperatures that are close to 0 °F cause SES to replace them with the corresponding outside air conditions from form 1F. SES does three slightly different tests for zero temperature values (two types in form 6B and one type in form 10).

The User Manual page for form 6 (page 12-46) states that if zero values are used for temperatures they will be replaced by the outside air conditions. This is why most of us usually set zero for all six temperatures in form 6B.

The User Manual page for form 10 (page 12-103) does not state that if zero values are used for a train's grid temperature it will be replaced by the outside dry-bulb temperature from form 1F (`TDBAMB`). That really should be added to the manual. If I'd known that before I spotted it in the source code, I'd have used zeros for all grid temperatures in form 10 for stationary trains in fire runs.