J. Feicht Updated 6/26/2019
Rev. kernalbt7.1.1

This *Mathematica* notebook generates the beamtube pressure profile using the btwaterleak methodology of particle diffusion. The pressure distribution is the solution to the set of n-coupled differential equations where “n” represents the number of sections.

Options include the pump arraignments, leaks, temperature profile, etc. A switch is included to force NDSolve to use the 4th order Runge-Kutta technique to solve the ODE’s, although doing this is not recommended. There are some asides in this version that discuss how to solve initial value problem, also setting up coupled differential equations. I’ve keep them in the code for future reference.

Any data input or changes to defined values should use *Mathematica* floating point notation, for example use “2.0”, or “2.” instead of symbolic notation “2”, doing this makes the code run much faster because the code interprets this as a number. Instances where integers are expected are noted.

Note: Deleting all code output before saving will greatly reduce file size.

Note: It is always a good idea to quit then restart the kernel after changing variables. Mathematica has a good (I’d say too good) memory. Avoid odd behavior by doing this every time.

Define tube geometry, pump speeds and pump locations. Warning mixed units!

In[1]:=
```
ClearAll (*See note on resetting the kernel, ClearAll etc. are not always throrough *)
```
Out[1]=
```
ClearAll
```

In[2]:=
```
ClearSystemCache
```
Out[2]=
```
ClearSystemCache
```

In[3]:= `Tubelength = 4000. * 100. (*300 meter converted to centimeter*)`

Out[3]= 400 000.

In[4]:= `Tubediameter = 1.2 * 100. (*nominal 10" diameter converted to centimeter*)`

Out[4]= 120.

In[5]:= $\text{Tubearea} = \frac{\pi\, \text{Tubediameter}^2}{4.}$; (*centimeter^2*)

In[6]:= `IonpumpS = 500. * 1000. (*500 liter/sec ion pump speed converted to cc/sec*);`

```
TempCryopumpS = 1000. * 6.5 × 10^3 (*speed cc/sec*)
(*these are the temporary TempCryopumpS*)

Nopump = 0.0(*speed cc/sec*)
 (*use this or code in 0.0 when temporary pumps are removed*);

Turbospeed = 1900. * 100. (*speed of turbo in cc/sec*);
```

Out[7]= 6.5×10^6

In[10]:= `ChamberSpeed = 500 * 1000(*speed cc/sec*) (*these are the end station speeds*)`

```
CryoTrapS = 1.0 × 10^5 * 100 (*these are the coaxial LN2 Cryotraps*)
```

Out[10]= 500 000

Out[11]= $1. \times 10^7$

Populate the Number of Ports (i.e. number of pumps)

In[12]:=
```
Numberofports = 2(*need to have at least 2 ports,
 one on each end, this value is the number of pumps too*)
```

Out[12]= 2

In[13]:= `Portpitch = Tubelength / (Numberofports - 1) (*centimeter*)`

Out[13]= 400 000.

Populate the pump speeds and locations (comment this section out if you want to directly input pump speeds)

In[14]:=
```
Do[Port[i] = {(i - 1) * Portpitch,
    Nopump}, {i, 1, Numberofports}]; (*set pump vector and speeds*)
```

In[15]:=
```
Do[Print[Port[i]], {i, 1, Numberofports}];
(*output is a vector (ordered pair) reported as the distance from "one end",
then what the pump speed is*)
```

{0., 0.}

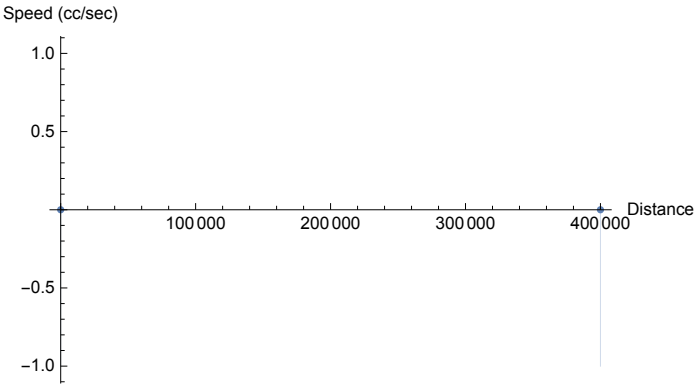
{400 000., 0.}

Plot pump speeds (cc/sec) along beamtube

In[16]:=
```
portspeedtable = Table[Port[j], {j, 1, Numberofports}];
(*first need to make a table of the indexed ordered pairs*)
```

In[17]:=
```
ListPlot[portspeedtable, {Joined → False},
 AxesLabel → {"Distance", "Speed (cc/sec)"}, Filling → Bottom]
```

Out[17]=



Define starting pressure, temperature and gas amu

In[18]:=
```
startpress = 5.0 × 10^-9 (*Starting pressure in Torr*)
```

Out[18]= $5. \times 10^{-9}$

In[19]:=
```
amu = 28. (*gas amu*)
```

Out[19]= 28.

In[20]:=
```
Tkelvin = 300. (*Kelvin*)
```

Out[20]= 300.

Define physical constants, conversion factors, variable declaration, etc.

In[21]:=
```
vt = 5.263 × 10^4 Sqrt[18/amu] Sqrt[Tkelvin/296] (*thermal velocity cm/sec*)
particlescm3 = 296./Tkelvin 3.0 × 10^16 Ptorr
particlescm2 = 1.0 × 10^15 monolayers
```

Out[21]= 42 482.

Out[22]= 2.96×10^{16} Ptorr

Out[23]= $1. \times 10^{15}$ monolayers

In[24]:=
```
pipeconductance = 3.81 * Tubediameter^3 / Tubelength * (Tkelvin / amu)^(1/2)
(*conductance for a long round pipe in liters/second*)
```

Out[24]= 53.8754

In[25]:=
```
timeconstant = IntegerPart[1.0 * (Tubearea * Tubelength) / (1000. * pipeconductance)]
(*get an estimate of the system time constant volume/(liter/sec)*)
```

Out[25]= 83 969

In[26]:=
```
timeconstant = 40 000
```

Out[26]= 40 000

Define how many sections to break the calculation up into, normally use 100 sections, higher values cost solve time but increase accuracy.

In[27]:=
```
Sections = 100(* this is the INTEGER number of calculation sections
  that the beamtube will be broken into, i.e. the number of coupled
  differential equations to solve, affects pump placement calc, etc.*)
```

Out[27]= 100

Establish which tube sections have vacuum pumps, then map pump location & speed to the “n” tube sections.

In[28]:=
```
(*You may ask why we need this section? The reason is that
 because the tube can be broken up into many sections(user defined),
the particular section where the placement of pumps, leaks,
etc. can change. This code puts the pumps in their correct section.*)
```

In[29]:=
```
Seclength = Tubelength / Sections (*length of each "calculation" section*)
```

Out[29]= 4000.

In[30]:=
```
Sections(*how many sections were requested*)
```

Out[30]= 100

In[31]:=
```
$MachineEpsilon
```

Out[31]= 2.22045×10^{-16}

In[32]:=
```
sectionrange[1] = {1, Interval[{0, Seclength}]}(*defines the first section*)
```

Out[32]= {1, Interval[{0, 4000.}]}

In[33]:=
```
sectionrange[Sections] =
 {Sections, Interval[{(Tubelength - Seclength + .0001), Tubelength}]}
 (*defines the last section*)
```

Out[33]=
```
{100, Interval[{396 000., 400 000.}]}
```

In[34]:=
```
Do[sectionrange[i] = {i, Interval[{((i - 1) * Seclength + .0001), (i * Seclength)}]},
 {i, 2, Sections - 1}] (*defines the intermediate sections*)
(*the 0.0001 is a suck-ass way to stop MemberIntervalQ from
  duplicating pumps by not allowing the intervals to touch,
there is probably a switch in the code to fix this but I havent found it yet*)
```

In[35]:=
```
Table[sectionrange[i], {i, 1, Sections}] ;(*take a look at the sections and their range*)
```

In[36]:=
```
Do[findport[i, n] = Boole[IntervalMemberQ[sectionrange[i][[2]], Port[n][[1]]]],
 {n, 1, Numberofports}, {i, 1, Sections}]
 (* find the section (i) that has the pump port(n)*)
```

In[37]:=
```
Do[tablel[j] = Table[findport[i, j], {i, 1, Sections}], {j, 1, Numberofports}]
(*intentionally misspelled*)
```

In[38]:=
```
portboole = Sum[tablel[i], {i, 1, Numberofports}] ;
(*make a boolean table of which section has the port*)
```

Check that the pump distribution is correct and no overlaps have occurred

In[39]:=
```
Total[portboole] == Numberofports (*these numbers must be equal/True
   (which means the number of ports and the boolean sum are the same)
  or there is an overlap ant that port is being counted twice*)
```

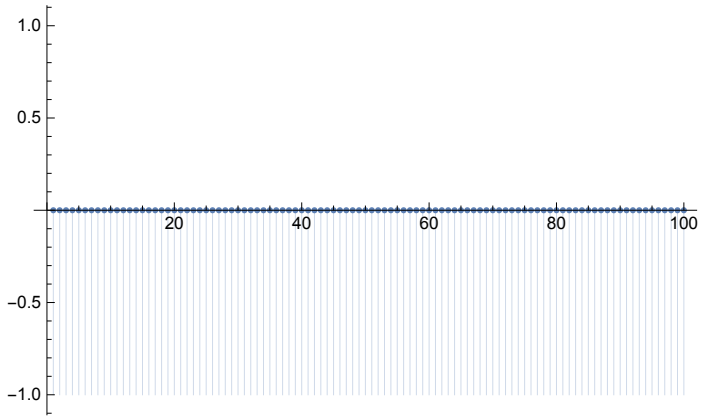
Out[39]=
```
True
```

Map the pump speeds to each port

In[40]:=
```
Speedy[x_] := Nopump * x  (*construct a function that multiplies by pump speed*)
Do[g = MapAt[Speedy, portboole, ;;], {n, 1, Sections}]; (*map the speed over the table*)
```

In[42]:=
```
ListPlot[g, Filling -> Bottom]
```

Out[42]=



In[43]:=
```
Do[speed[i] = g[[i]], {i, 1, Sections}] (*fyi, [[i]] is a compact notation for Part,
it picks out that element in the list. We need these to be individual values
 (unlike Pumps) so they can go into the ODE by section. Speed should be in cc/sec*)

(*>>>>>>>>>>Note this will overwrite any earlier speed
  definitions so make a choice how you want to do this<<<<<<<<<*)
```

In[44]:=
```
speed[1001] (*speed in cc/sec*)
```

Out[44]= speed[1001]

In[45]:=
```
speed[101] (*check if working correctly*)
```

Out[45]= speed[101]

In[46]:=
```
speed[1] = 0.0
```

Out[46]= 0.

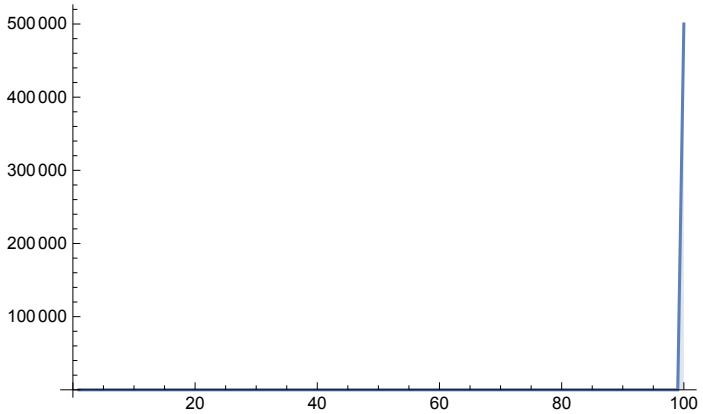
In[47]:=
```
speed[Sections] = IonpumpS
```

Out[47]= 500 000.

In[48]:=
```
speeds = Table[speed[i], {i, 1
     , Sections}] ;(*allows a quick look to be sure it's working*)
```

In[49]:=
```
ListPlot[speeds, Filling → Bottom, Joined → True]
(*speed plot versus section number in cc/sec*)
```

Out[49]=



Setup beamtube leak(s). This section can be developed further as required. A similar section with programmed temperature also needs to be developed.

In[50]:=
```
(*Leaklocation=2 10^5  (*centimeters from vertex*);*)
```

In[51]:=
```
(*Sectionnumber= Leaklocation/Tubelength *Sections; (*finds tube section with leak*)*)
```

In[52]:=
```
(*Leakrate=1 10^-5 (1000)  (*leak rate in torr-liter/sec converted to torr-cc/sec*);*)
```

In[53]:=
```
(*Leakstartday=500;*)
```

In[54]:=
```
(*Leakstopday=501;*)
```

In[55]:=
```
Do[qlk[i] = 0.0, {i, 1, Sections}] (*sets a leak rate in each section,
qlk is an variable in each of the differential equations*)
```

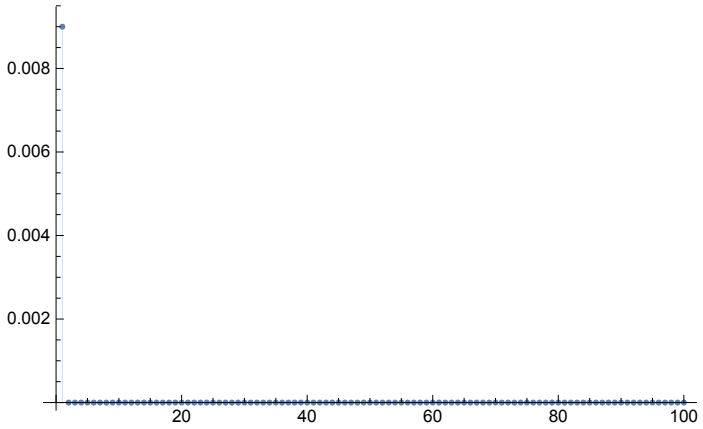
In[56]:=
```
qlk[1] = 9.0 × 10^-3 (* Shows way to short circuit
   the previous Do loop and set the value of qlk[1] to do testing*)
```

Out[56]=
```
0.009
```

In[57]:=
```
Leaks = Table[qlk[i], {i, 1
    , Sections}];
```

In[58]:=
```
ListPlot[Leaks, Filling → Bottom] (*speed plot versus section number in cc/sec*)
```

Out[58]=



Set time step size

In[59]:=
```
timestep = recordstep * 24 * 3600 / 1440 (*timesttep in seconds >>>>>>>
     1440 is not remarkable other than it gives 60 second intervals*)
```

Out[59]=
```
60 recordstep
```

In[60]:=
```
RKstep = 1 (*RungaKutta step size*);
```

Load solver

In[61]:=
```
Needs["DifferentialEquations`NDSolveUtilities`"];
```

Set up the derivative coefficients

In[62]:=
```
aa = 2. * vt * .5 * Tubediameter / (3.0 * Seclength^2)   (* units are cm/sec * cm/cm^2 = 1/sec *)
bb = 4. / Tubediameter   (* units are 1/cm *)
dd = 1. / (π (.5 Tubediameter)^2 Seclength)   (* units are 1/cm^3 *)
```

Out[62]= 0.106205

Out[63]= 0.0333333

Out[64]= 2.21049×10^{-8}

Get the 4th order Runge-Kutta coefficients (it is better to let the code choose the solver, but we can force using RK if we want)

In[65]:=
```
(*the following are the R-K 4th order coefficients*)
```

In[66]:=
```
crkamat = {{1/2}, {0, 1/2}, {0, 0, 1}};
crkbvec = {1/6, 1/3, 1/3, 1/6};
crkcvec = {1/2, 1/2, 1};
ClassicalRungeKuttaCoefficients[4, p_] := N[{crkamat, crkbvec, crkcvec}, p];
```

Set the outgassing coefficients of each section.

In[69]:=
```
Do[ajj_n = 3.0 × 10^-11, {n, 1, Sections}]; (*outgassing per section in torr-cm^3/(cm^2-sec),
typical values for unbaked stainless steel are on
  the order of 6x10^-9 Torr-liter/sec-cm^2 or since 1000 cc/liter = 6x10^-6 Torr-cc/sec-cm^2 *)
```

In[70]:=
```
(*ajj_1=1.0 10^-5*)
```

Set up the n-coupled differential equations to get pressure profile.

In[71]:=
```
diffeqns = Table[{p_(i+1)'[t] == aa (p_i[t] + p_(i+2)[t] - 2 p_(i+1)[t]) + bb ajj_(i+1) +
      dd (qlk[i + 1] - speed[i + 1] * p_(i+1)[t]), p_(i+1)[0] == startpress}, {i, 1, Sections - 2}];
(*this generates the ODE's for sections n+1 to n-1 of the n beam tube sections*)
```

In[72]:=
```
firstsection =
 {p_1'[t] == aa (p_2[t] - p_1[t]) + bb ajj_1 + dd (qlk[1] - speed[1] * p_1[t]), p_1[0] == startpress}
 (*this generates the ODE for section 1 of the n beam tube sections*)
```

Out[72]= $\{p_1'[t] == 1.99944 \times 10^{-10} + 0.106205 (-p_1[t] + p_2[t]),\ p_1[0] == 5. \times 10^{-9}\}$

In[73]:=
```
lastsection = {p_Sections'[t] == aa (p_Sections-1[t] - p_Sections[t]) + bb ajj_Sections +
    dd (qlk[Sections] - speed[Sections] * p_Sections[t]), p_Sections[0] == startpress}
 (*this generates the ODE for the last section of the n beam tube sections*)
 (* note that btwaterleak has a section 101,
 this needs more investigation, may be required here too*)
```

Out[73]=
$$\left\{p_{100}'[t] == 1.\times 10^{-12} + 2.21049\times 10^{-8}\left(0. - 500\,000.\, p_{100}[t]\right) + 0.106205\left(p_{99}[t] - p_{100}[t]\right),\ p_{100}[0] == 5.\times 10^{-9}\right\}$$

In[74]:=
```
temp = Prepend[diffeqns, firstsection]; (* add all the sections to ODE list*)
diffeqns = Append[temp, lastsection];
```

In[76]:=
```
vars2 = Table[p_i[t], {i, 1, Sections}]; (*define the variables*)
```

Solve the coupled ODE's. Can force solver to use 4th order RK if desired (see notation). There is a subtlety here, the time that the solution runs over only needs to be the settling time of the system, not how long you want to simulate the pumpdown. At sound speeds this is a relatively short time. If you want to simulate a water desorption rate it needs to be wrapped around the ODE solution with a Do statement or similar. For example the temperature, outgassing rate, or leak rate could all be parameterized around the ODE solution.

In[77]:=
```
btw = NDSolve[diffeqns, vars2, {t, 1, timeconstant}] ;
(*let Mathematica choose the solver*)
```

In[78]:=
```
(*btwRK=NDSolve[diffeqns,vars2,{t,1,timeconstant},Method→{"ExplicitRungeKutta",
    "DifferenceOrder"→4,"Coefficients"→ClassicalRungeKuttaCoefficients},
  StartingStepSize→.01, MaxSteps→1000000]*) (*force using Runge Kutta method*)
```

Plot the beamtube pressure distribution. Example of pumpdown with constant outgassing rate/section

---

In[79]:=
```
(*ListPlot3D[Table[vars2/.First[btw],{t,1,timeconstant}],
 AxesLabel->{"Section Number","Time","Pressure"}]*)
```

In[80]:=
```
(*ListPlot3D[Log10[Table[vars2/.First[btw],{t,1,timeconstant}]]]  (*try a log plot*)*)
```

In[81]:=
```
(*ListPlot3D[Table[vars2/.First[btwRK],{t,1,timeconstant,10}],
 AxesLabel->{"Section Number","Time","Pressure"}]*)
```

In[82]:=
```
(*ListPlot3D[Log[Table[vars2/.First[btwRK],{t,1,timeconstant,10}]]]
(*try a log plot*)*)
```

Some ways to access the data

In[83]:=
```
secpresvstime = Table[vars2 /. First[btw], {t, 1, timeconstant}];
(*these are the solutions, this can be a really big file,
it contains a value for every time set from 1 to timeconstant,
for each section (typically 100 sections)*)
```

In[84]:=
```
First[secpresvstime] (*take a look at first file elements. This is the
 pressure in each section at start. You should see a lot of initial conditions*)
```

Out[84]=
$$\{5.18991\times10^{-9}, 5.01067\times10^{-9}, 5.00135\times10^{-9}, 5.00101\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9},$$
$$5.001\times10^{-9}, 5.001\times10^{-9}, 5.001\times10^{-9}, 5.0009\times10^{-9}, 4.99832\times10^{-9}, 4.94881\times10^{-9}\}$$

In[85]:=
```
Last[secpresvstime](*take a look at last file elements*)
```

Out[85]=
$$\{1.66504\times10^{-7}, 1.6464\times10^{-7}, 1.62787\times10^{-7}, 1.60943\times10^{-7}, 1.59109\times10^{-7}, 1.57285\times10^{-7},$$
$$1.55471\times10^{-7}, 1.53667\times10^{-7}, 1.51872\times10^{-7}, 1.50087\times10^{-7}, 1.48311\times10^{-7}, 1.46546\times10^{-7},$$
$$1.4479\times10^{-7}, 1.43043\times10^{-7}, 1.41306\times10^{-7}, 1.39578\times10^{-7}, 1.3786\times10^{-7}, 1.36151\times10^{-7},$$
$$1.34451\times10^{-7}, 1.3276\times10^{-7}, 1.31079\times10^{-7}, 1.29406\times10^{-7}, 1.27742\times10^{-7}, 1.26088\times10^{-7},$$
$$1.24441\times10^{-7}, 1.22804\times10^{-7}, 1.21175\times10^{-7}, 1.19555\times10^{-7}, 1.17942\times10^{-7}, 1.16338\times10^{-7},$$
$$1.14743\times10^{-7}, 1.13155\times10^{-7}, 1.11575\times10^{-7}, 1.10002\times10^{-7}, 1.08438\times10^{-7}, 1.0688\times10^{-7},$$
$$1.0533\times10^{-7}, 1.03788\times10^{-7}, 1.02252\times10^{-7}, 1.00723\times10^{-7}, 9.92014\times10^{-8}, 9.7686\times10^{-8},$$
$$9.6177\times10^{-8}, 9.46744\times10^{-8}, 9.31778\times10^{-8}, 9.16872\times10^{-8}, 9.02024\times10^{-8}, 8.87232\times10^{-8},$$
$$8.72495\times10^{-8}, 8.57809\times10^{-8}, 8.43175\times10^{-8}, 8.2859\times10^{-8}, 8.14051\times10^{-8}, 7.99558\times10^{-8},$$
$$7.85108\times10^{-8}, 7.70699\times10^{-8}, 7.56329\times10^{-8}, 7.41997\times10^{-8}, 7.27699\times10^{-8}, 7.13435\times10^{-8},$$
$$6.99202\times10^{-8}, 6.84997\times10^{-8}, 6.7082\times10^{-8}, 6.56667\times10^{-8}, 6.42536\times10^{-8}, 6.28426\times10^{-8},$$
$$6.14333\times10^{-8}, 6.00257\times10^{-8}, 5.86193\times10^{-8}, 5.72141\times10^{-8}, 5.58097\times10^{-8}, 5.4406\times10^{-8},$$
$$5.30027\times10^{-8}, 5.15996\times10^{-8}, 5.01964\times10^{-8}, 4.87929\times10^{-8}, 4.73888\times10^{-8},$$
$$4.59839\times10^{-8}, 4.4578\times10^{-8}, 4.31708\times10^{-8}, 4.1762\times10^{-8}, 4.03515\times10^{-8}, 3.89389\times10^{-8},$$
$$3.75239\times10^{-8}, 3.61064\times10^{-8}, 3.46861\times10^{-8}, 3.32627\times10^{-8}, 3.1836\times10^{-8}, 3.04057\times10^{-8},$$
$$2.89715\times10^{-8}, 2.75331\times10^{-8}, 2.60904\times10^{-8}, 2.46431\times10^{-8}, 2.31908\times10^{-8},$$
$$2.17333\times10^{-8}, 2.02704\times10^{-8}, 1.88017\times10^{-8}, 1.7327\times10^{-8}, 1.58461\times10^{-8}, 1.43587\times10^{-8}\}$$

In[86]:=
```
secpresvstime[[timeconstant]];
(*this is another way to get the last set of data in the file, or at any time interval*)
```

In[87]:=
```
finalpressures = Table[secpresvstime[[i]][[Sections]];, {i, 1, timeconstant}];
(*pull out the last sections pressure vs time at time=timeconstant*)
```

In[88]:=
```
secpresvstime[[timeconstant]][[Sections]]
(*pull out the final pressure in the last section at time=timeconstant*)
```
Out[88]= 1.43587×10^{-8}

In[89]:=
```
secpresvstime[[timeconstant]][[1]]
(*pull out the final pressure in the first section at time=timeconstant*)
```
Out[89]= 1.66504×10^{-7}

In[90]:=
```
sec1pvst = Table[secpresvstime[[i]][[1]], {i, 1, timeconstant}];
 (*get the pressure vs time history of section 1*);
seclastpvst = Table[secpresvstime[[i]][[Sections]], {i, 1, timeconstant}];
(*get the pressure vs time history of the last section *);
```

In[92]:=
```
ListLogPlot[{sec1pvst, seclastpvst},
  Joined → True, PlotLegends → {"First Section", "Last Section"}];
```

In[93]:=
```
secpresvstime[[timeconstant]][[1]] (* final pressure in the 1st section,
change the 1 to whatever section you want to query*)
```
Out[93]= 1.66504×10^{-7}

In[94]:=
```
secpresvstime[[timeconstant]][[Sections]]
(* final pressure in the nth or last section*)
```
Out[94]= 1.43587×10^{-8}

Following section is used to calculate conductance. You need to set one end as a source (means set a leak or “high” ajj), the other end as a sink (pump). Watch out for pressure range, stay out of viscous range.

In[95]:=
```
deltaP = secpresvstime[[timeconstant]][[1]] -
  secpresvstime[[timeconstant]][[Sections]] (*grab the pressure at section 1 (source),
  subtract the pressure at last section (sink)*)
```
Out[95]= 1.52145×10^{-7}

In[96]:=
$$\text{ajjconductance} = \frac{\pi\,\text{Tubediameter} * \text{Seclength}\left(\text{ajj}_1\right)}{\text{deltaP}} * 0.001$$

(* $\text{cm}^2 \text{x} \frac{\text{torr cm}^3}{\text{cm}^2\text{sec}}$ x $\frac{1}{\text{torr}}$ x $\frac{.001\ \text{liter}}{\text{cm}^3} = \frac{\text{liter}}{\text{second}}$ >>>> Use this if setting ajj as source*)

Out[96]= 0.297341

In[97]:=
$$\text{leaksourceC} = \frac{\text{qlk}[1]}{\text{deltaP}} * .001$$
(*use this if setting a leak rate as source
(note qlk is in Torr-cc/sec, hence the 0.001 conversion. Units are liter/sec*)

Out[97]= 59.1541

In[98]:=
```
pipeconductance
(*this is what the handbook formula gives as the conductance in liter/sec,
uses amu and temperature corrections. Use for comparison to calculation above*)
```

Out[98]= 53.8754

In[99]:=
```
condformulasimple =
 12.1 * 120.0^3 / (4.0 × 10^5)  (*this is the simple formula for conductance of the beam tube,
  120 cm diameter and 4 km long, units are liter/second*)
```

Out[99]= 52.272

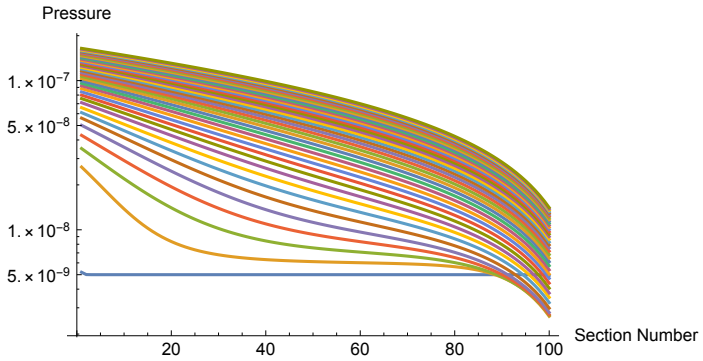
Plot data. Several ideas on how to plot pieces or all of the data.

In[100]:=
```
(*p8=LogPlot[p1[t]/.First[btw],{t,1,timeconstant},PlotRange→All,GridLines→Automatic];
(*this allows you to plot the pressure in a section pn*)*)
```

In[101]:=
```
(*p50=LogPlot[p50[t]/.First[btw],{t,1,timeconstant},
   PlotRange→All,GridLines→Automatic,PlotStyle→{Red,Dashed}] ;
 (*this allows you to plot the pressure in each section pn*)*)
```

In[102]:=
```
ListLogPlot[Table[vars2 /. First[btw], {t, 1, timeconstant, 1000}],
 AxesLabel -> {"Section Number", "Pressure"}, Joined → True]
 (*this allows you to plot the pressure evolution over a time range in each section pn*)
```

Out[102]=



In[103]:=
```
ListLogPlot[{Last[secpresvstime], First[secpresvstime]}, Joined → True]
```

Out[103]=