Updates for generalization

Key idea: set the code up for the “general problem”, e.g. 2 steam chests, each with up to 8 valves each.

1) increase number of valves to 8 (later make this a list of arbitrary length)

2) change knzScalefunc

Background and Theory

Reference figures

```
In[ ]:= SetDirectory[NotebookDirectory[] ];
        fig01 = Import["HPT_Measurements_ModelParameters.png"];
        fig02 = Import["plotGovNoz4Regions.png"];
```

System of Interest

The system of interest has two flow elements in series: the governor valve and the 1st stage nozzle.

In[ ]:=

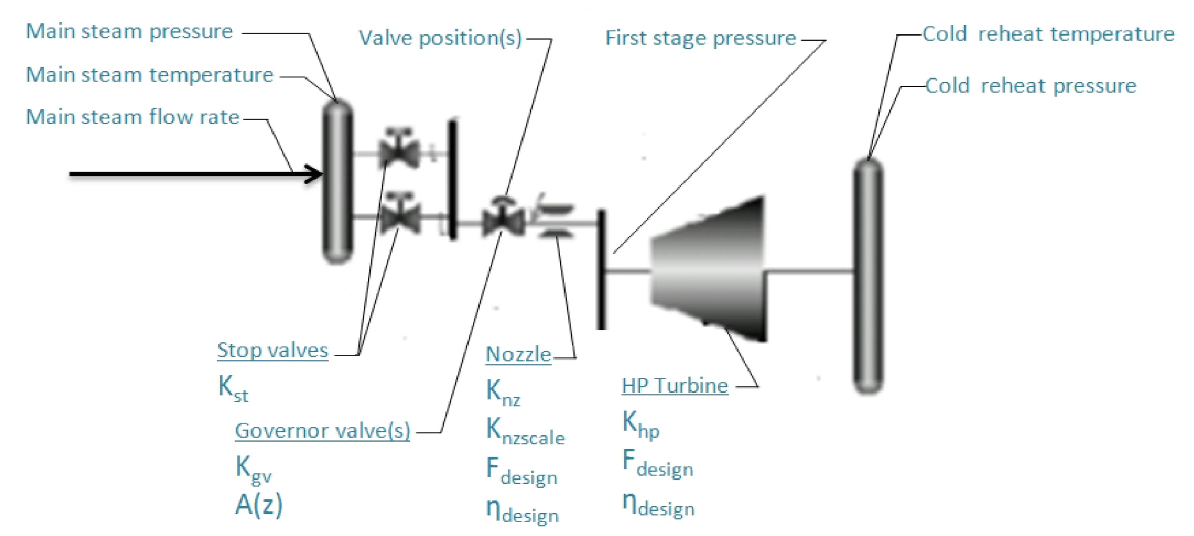


Figure 1: High Pressure Turbine data and model parameters

Out[ ]=

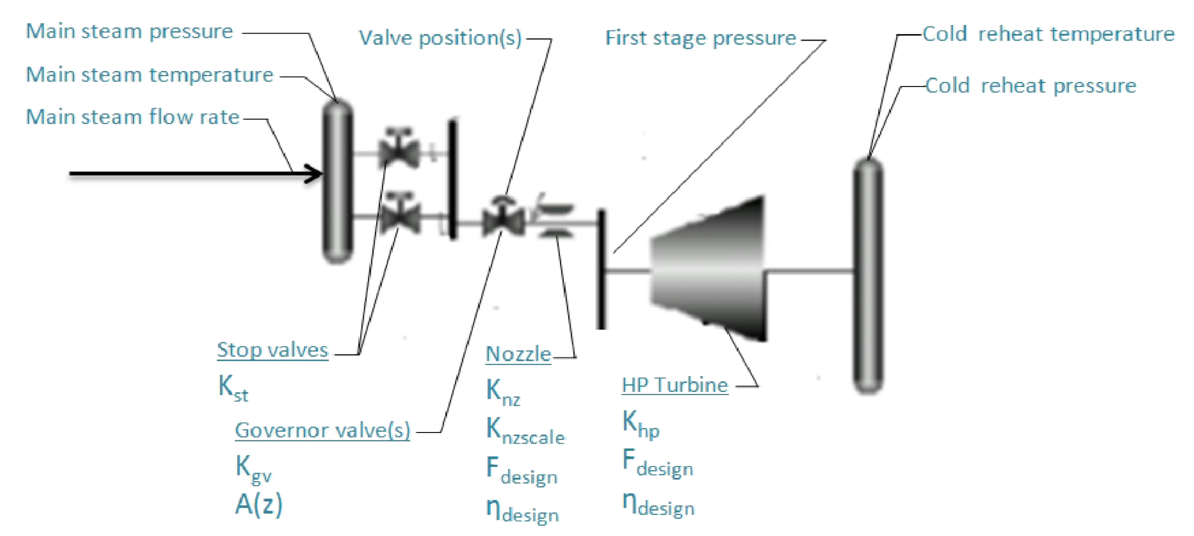
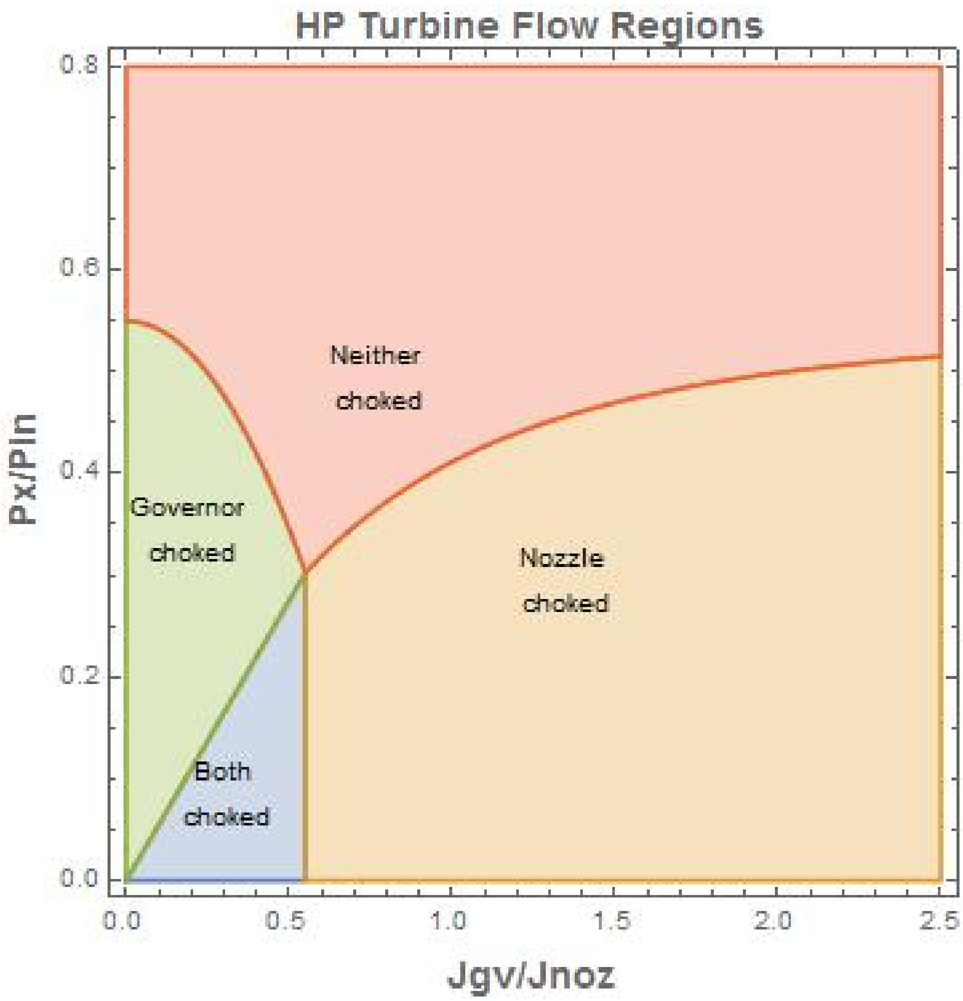


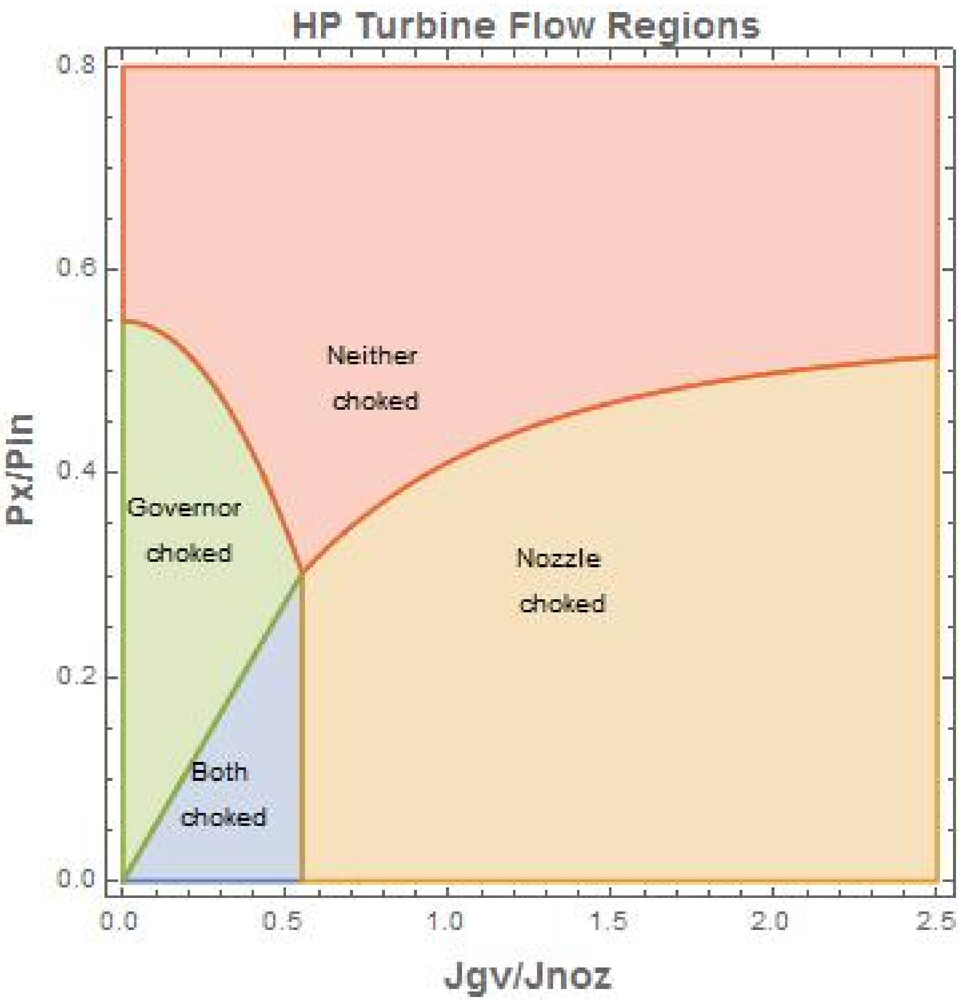
Figure 1: High Pressure Turbine data and model parameters

For each governor valve and nozzle pair, there are four possible flow regimes. The sketch below shows how these four regimes are related to the operating conditions. The x - axis is the ratio of the flow conductances for the governor valve and the nozzle. The y - axis is the ratio of the first stage pressure (after the nozzle) to the steam chest pressure (after the stop valve). For example, if the flow conductance for the governor valve is small, then the governor valve will be choked.

In[◦]:=


HP Turbine Flow Regions
Px/Pin
Jgv/Jnoz
0.0
0.2
0.4
0.6
0.8
0.0
0.5
1.0
1.5
2.0
2.5
Neither choked
Governor choked
Nozzle choked
Both choked

Out[◦]=



Fundamental Equations

We start with an equation relating flowrate and pressure drop,

$$\text{flow} = J \sqrt{\rho \, \text{Min}[\Delta P, \ (1 - \alpha) \ \text{Pin}]} \tag{1}$$

Where flow is the flow rate in units of mass per time,

J is a flow conductance,

ρ is the fluid density at the inlet conditions,

ΔP is the difference in pressure across the flow element,

α is the critical pressure ratio, typically 0.55 for steam systems,

Pin is the inlet pressure.

We initially analyze the governor valves and the 1st stage nozzles. For the governor valve the analysis uses the pressure after the stop valve and the “bowl” pressure, i.e. the intermediate pressure between the governor valve and the nozzle. For the 1st stage nozzle the analysis uses, the “bowl” pressure and the 1st stage pressure.

The flow is choked when the pressure drop is greater than (1-α) Pin.

Because the system of interest has two flow elements in series, the governor valve(s) and the 1st stage nozzle(s), there are four possible cases to consider:

case 1: neither the governor or the 1st stage nozzle is choked
case 2: the governor is choked and the nozzle is not choked
case 3: the governor is not choked and the nozzle is choked
case 4: both the governor and the nozzle are choked.

To analyze this system, we assume that the density between the governor valves and the 1st stage nozzle, referred to as the bowl pressure, ρ_b, is given by inlet density, ρ_{in},multiplied by the pressure ratio across the governor valve.

$$\rho_{\mathrm{b}} = \rho_{\mathrm{in}} \frac{\mathrm{P_b}}{\mathrm{P_{in}}} \tag{2}$$

Using equation (1) and (2), below are the choked and unchoked forms of the Bernoulli equation for the governor valve and the nozzle.

$$\begin{aligned}
\mathrm{fgovnc} &= \mathrm{jgov\ Sqrt}[\rho_{\mathrm{in}}\ (\mathrm{Pin} - \mathrm{Pb})] \\
\mathrm{fgovch} &= \mathrm{jgov\ Sqrt}[\rho_{\mathrm{in}}\ (1 - \alpha)\ \mathrm{Pin}] \\
\mathrm{fnoznc} &= \mathrm{jnoz\ Sqrt}[\rho_{\mathrm{in}}\ (\mathrm{Pb} / \mathrm{Pin})\ (\mathrm{Pb} - \mathrm{Px})] \\
\mathrm{fnozch} &= \mathrm{jnoz\ Sqrt}[\rho_{\mathrm{in}}\ (\mathrm{Pb} / \mathrm{Pin})\ (1 - \alpha)\ \mathrm{Pb}]
\end{aligned} \tag{3}$$

Define normalized flow conductance for the governor valve, jr, by dividing each equation by the flow conductance of the nozzle and the square root of the inlet pressure. Define normalized pressures (pbr and pxr for the bowl and exit pressure) by dividing by the inlet pressure. This yields the following:

$$\begin{aligned}
\mathrm{fgovnc} &= \mathrm{jr\ Sqrt}[\rho_{\mathrm{in}}\ (1 - \mathrm{pbr})]\ \mathrm{Pin\ Jnoz} \\
\mathrm{fgovch} &= \mathrm{jr\ Sqrt}[\rho_{\mathrm{in}}\ (1 - \alpha)]\ \mathrm{Pin\ Jnoz} \\
\mathrm{fnoznc} &= \mathrm{Sqrt}[\rho_{\mathrm{in}}\ (\mathrm{pbr})\ (\mathrm{pbr} - \mathrm{pxr})]\ \mathrm{Pin\ Jnoz} \\
\mathrm{fnozch} &= \mathrm{Sqrt}[\rho_{\mathrm{in}}\ (\mathrm{pbr})\ (1 - \alpha)\ \mathrm{pbr}]\ \mathrm{Pin\ Jnoz}
\end{aligned} \tag{4}$$

Regions of applicability

The governor choked equation is applicable when the pressure difference across the governor valve, Pin - Pb, is greater than the critical limit, (1-α) Pin.

$$\mathrm{Pin} - \mathrm{Pb} \geq (1 - \alpha)\ \mathrm{Pin}. \tag{5}$$

Dividing by Pin, to work with normalized pressures, and solving for pbr yields the boundary for when the flow through the governor choked equation is applicable.

$$\mathrm{pbr} \leq \alpha \tag{6}$$

Similarly, for the nozzle, the choked equation is applicable when the pressure difference across the nozzle, Pb - Px, is greater than (1- α) Pb.

$$\mathrm{Pb} - \mathrm{Px} \geq (1 - \alpha)\ \mathrm{Pb} \tag{7}$$

Dividing by Pin, to work with normalized pressures, and solving for pbr yields the boundary when the nozzle choked equation is applicable

$$\mathrm{pbr} \geq \frac{\mathrm{pxr}}{\alpha} \tag{8}$$

Next steps

We can now use equations (4), (6), and (8) to describe the bowl pressure, pbr, as a function of jr, α, and pxr.

Utilities, steam tables, and unit conversions

Matrix functions

In[ ]:=
```
colAppend[mat1_, mat2_] /; (Length@Dimensions@mat1 > 1 && Length@Dimensions@mat2 > 1) :=
 Join[mat1, mat2, 2]

colAppend[mat1_, col1_, pos_: -1] /;
   (Length@Dimensions@mat1 > 1 && Length@Dimensions@col1 == 1) :=
 Insert[mat1 // Transpose, col1, pos] // Transpose

colAppend[col1_, col2_] /; (Length@Dimensions@col1 == 1 && Length@Dimensions@col2 == 1) :=
 Transpose[{col1, col2}]

colAppend[col1_, mat1_, pos_: 1] /;
   (Length@Dimensions@col1 == 1 && Length@Dimensions@mat1 > 1) :=
 Insert[mat1 // Transpose, col1, pos] // Transpose

colDropLast[mat1_, pos_: -1] /; (Length@Dimensions@mat1 > 1) :=
 Module[{temp = mat1}, temp[[All, pos]] = Sequence[];
  temp]

colDelete[mat1_, pos_: 1] /; (Length@Dimensions@mat1 > 1) :=
 Module[{temp = mat1}, temp[[All, pos]] = Sequence[];
  temp]
```

Units

Unit definitions

```
In[ ]:= unitsSus =  "BritishThermalUnitsIT" / "Pounds" / "DegreesFahrenheit";
        unitsHus = "BritishThermalUnitsIT" / "Pounds";
        unitsTus = "DegreesFahrenheit";
        unitsRus = "Pounds" / "Feet"^3;
        unitsPus = "psi";
        unitsVus =  "Feet"^3 / "Pounds";
        unitsListus = {unitsSus, unitsHus, unitsTus, unitsRus, unitsPus, unitsVus};

In[ ]:= unitsSsi = "Kilojoules" / "Kilograms" / "Kelvins";
        unitsHsi = "Kilojoules" / "Kilograms";
        unitsTsi = "Celcius";
        unitsRsi = "Kilograms" / "Meters"^3;
        unitsPsi = "Kilopascal";
        unitsVsi = "Meters"^3 / "Kilograms";
        unitsListsi = {unitsSsi, unitsHsi, unitsTsi, unitsRsi, unitsPsi, unitsVsi};

In[ ]:= unitsList = Flatten[{unitsListus, unitsListsi}];

In[ ]:= Partition[Quantity[300, #] & /@ unitsList, Length@unitsList / 2] // TableForm
```

```
••• Quantity: Unable to interpret unit specification Celcius.

••• Quantity: Unable to interpret unit specification Kilopascal.
```

```
Out[ ]//TableForm=
     300 BTUIT/(lb °F)    300 BTUIT/lb    300 °F                   300 lb/ft3    300 lbf/in2

     300 kJ/(kg K)        300 kJ/kg       Quantity[300, Celcius]   300 kg/m3     Quantity[300, K
```

Misc unit conversions

```
In[ ]:= hSI[h_Quantity] := QuantityMagnitude[h];
        rhoSI[rho_Quantity] := QuantityMagnitude[rho]
        rhoEng[rho_Quantity] := QuantityMagnitude[rho, unitsRus]

        qmPsi[pQ_Quantity] := QuantityMagnitude[pQ, unitsPsi]
        qmPus[pQ_Quantity] := QuantityMagnitude[pQ, unitsPus]

        qmTsi[tQ_Quantity] := QuantityMagnitude[tQ, unitsTsi]
        qmTus[tQ_Quantity] := QuantityMagnitude[tQ, unitsTus]

        qmSsi[sQ_Quantity] := QuantityMagnitude[sQ, unitsSsi]
        qmSus[sQ_Quantity] := QuantityMagnitude[sQ, unitsSus]

        qmHsi[hQ_Quantity] := QuantityMagnitude[hQ, unitsHsi]
        qmHus[hQ_Quantity] := QuantityMagnitude[hQ, unitsHus]

        qmRsi[rQ_Quantity] := QuantityMagnitude[rQ, unitsRsi]
        qmRus[rQ_Quantity] := QuantityMagnitude[rQ, unitsRus]

        qmVsi[vQ_Quantity] := QuantityMagnitude[vQ, unitsVsi]
        qmVus[vQ_Quantity] := QuantityMagnitude[vQ, unitsVus]
```

Steam Tables, Part IIB: Interpolating Functions with us units

Our goal is to have functions that are very fast for determining steam and water properties. To do this, we will use a few variations of Interpolation and FunctionInterpolation.

http://reference.wolfram.com/language/ref/message/FunctionInterpolation/ncvb.html

File details

```
In[ ]:= SetDirectory@NotebookDirectory[]
Out[ ]= C:\Users\win10\Desktop\BowenU1_GovValves
```

Saturated boundaries (9 functions: Tsat(p), Hsatl(p), Hsatv(p), Ssatl(p), Ssatv(p), Rsatl(p), Rsatv(p), Vsatl(p), Vsatv(p))

Functions to find vales at saturated conditions.

a) Tsat(p)

b) Hsatl(p), Hsatv(p)

c) Ssatl(p), Hsatv(p)

d) Rsatl(p), Rsatv(p)

e) Vsatl(p), Vsatv(p)

f) i2stmPHsatQ[]

g) i2stmPSsatQ[]

```
In[ ]:= i2stmEps = 0.001;
```

Critical T and P

```
In[ ]:= istmPcritval = First@Flatten@Import[".\\i2stmData\\i2stmPcritical.csv"];
        istmTcritval = First@Flatten@Import[".\\i2stmData\\i2stmTcritical.csv"];
```

Saturation line Tsat = Tsat(p)

Testing indicated a third order fit matched results to less than 0.00005

```
In[ ]:= pTsatdata = Import[".\\i2stmData\\i2stmPTsatdata.csv"];
        intstmTsatatP = Interpolation[pTsatdata, InterpolationOrder → 3];
```

Properties of saturated liquid and vapor

Retrieve values along the saturation line.

```
In[ ]:= (*enthalpy*)
        pHsatvList = Import[".\\i2stmData\\i2stmPHsatvdata.csv"];
        pHsatlList = Import[".\\i2stmData\\i2stmPHsatldata.csv"];
        (*entropy*)
        pSsatvList = Import[".\\i2stmData\\i2stmPSsatvdata.csv"];
        pSsatlList = Import[".\\i2stmData\\i2stmPSsatldata.csv"];
        (*density*)
        pRsatvList = Import[".\\i2stmData\\i2stmPRsatvdata.csv"];
        pRsatlList = Import[".\\i2stmData\\i2stmPRsatldata.csv"];
        (*specific volume*)
        pVsatvList = Import[".\\i2stmData\\i2stmPVsatvdata.csv"];
        pVsatlList = Import[".\\i2stmData\\i2stmPVsatldata.csv"];
```

Make interpolating functions using the above tales of values and the Interpolation function.

```
In[ ]:= (*enthalpy*)
        intstmHsatvatP = Interpolation[pHsatvList, InterpolationOrder → 1];
        intstmHsatlatP = Interpolation[pHsatlList, InterpolationOrder → 1];
        (*entropy*)
        intstmSsatvatP = Interpolation[pSsatvList, InterpolationOrder → 1];
        intstmSsatlatP = Interpolation[pSsatlList, InterpolationOrder → 1];
        (*density*)
        intstmRsatvatP = Interpolation[pRsatvList, InterpolationOrder → 1];
        intstmRsatlatP = Interpolation[pRsatlList, InterpolationOrder → 1];
        (*entropy*)
        intstmVsatvatP = Interpolation[pVsatvList, InterpolationOrder → 1];
        intstmVsatlatP = Interpolation[pVsatlList, InterpolationOrder → 1];
```

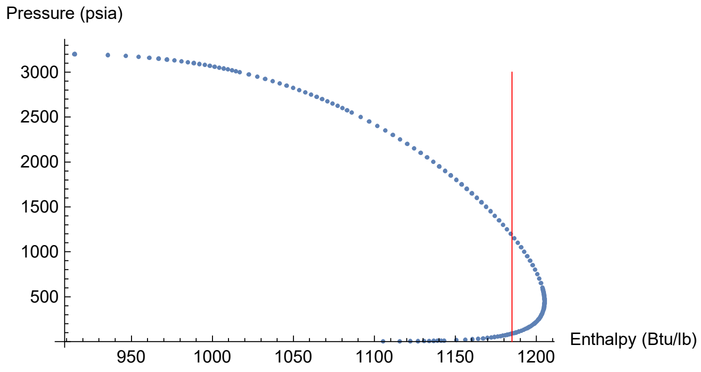
Points for plotting

```
In[]:= flip2DList[list_] := Transpose[{list[[All, 2]], list[[All, 1]]}]
       hsatvPlist = flip2DList[pHsatvList];
       hsatlPlist = flip2DList[pHsatlList];
       ssatvPlist = flip2DList[pSsatvList];
       ssatlPlist = flip2DList[pSsatlList];
```

Note that these are not a function, because more than one pressure has the same enthalpy. So a vertical line from a 1185 Btu/lb will intersect the graph twice.

```
In[]:= pHsatvList;
       ListPlot[hsatvPlist
        , PlotRange → All
        , AxesLabel → {Style["Enthalpy (Btu/lb)"], Style["Pressure (psia)"] }
        , Epilog → {Red, Line[{{1185, 1}, {1185, 3000}}] }
       ]
```

Out[]=



Queries to check for saturation

Given a pressure and temperature, are the conditions sufficiently close to saturation?

```
In[ ]:= intstmPTsatQ[pPsia_?NumericQ, tF_?NumericQ] := Module[{},
          Which[
           pPsia > istmPcritval, False,
           Abs[tF - intstmTsatatP[pPsia]] > 0.001, False,
           True, True
          ]
         ]

        intstmPHsatQ[pPsia_?NumericQ, hBtulb_?NumericQ] := Module[{},
          Which[
           pPsia > istmPcritval, False,
           hBtulb > intstmHsatvatP[pPsia], False,
           hBtulb < intstmHsatlatP[pPsia], False,
           True, True
          ]
         ]

        intstmPSsatQ[pPsia_?NumericQ, sBtulbF_?NumericQ] := Module[{},
          Which[
           pPsia > istmPcritval, False,
           sBtulbF > intstmSsatvatP[pPsia], False,
           sBtulbF < intstmSsatlatP[pPsia], False,
           True, True
          ]
         ]
```

Vapor fraction: Xph, Xps

```
In[ ]:= intstmXatPH[pPsia_?NumericQ, hBtulb_?NumericQ] := Module[{hsatl, hsatv},
          hsatv = intstmHsatvatP[pPsia];
          hsatl = intstmHsatlatP[pPsia];
          Which[
           pPsia > istmPcritval, 1.00,
           hBtulb > hsatv, 1.00,
           hBtulb < hsatl, 0.00,
           True, (hBtulb - hsatl) / (hsatv - hsatl)
          ]
         ]

        intstmXatPS[pPsia_?NumericQ, sBtulbF_?NumericQ] := Module[{ssatl, ssatv},
          ssatv = intstmSsatvatP[pPsia];
          ssatl = intstmSsatlatP[pPsia];
          Which[
           pPsia > istmPcritval, 1.00,
           sBtulbF > ssatv, 1.00,
           sBtulbF < ssatl, 0.00,
           True, (sBtulbF - ssatl) / (ssatv - ssatl)
          ]
         ]
```

PT functions: (Hpt, Spt, Rpt, Vpt)

These functions must check if the given temperature is near the saturated conditions. If so, then return saturated liquid (by convention).

Supercritical pressure conditions (Hsupcrit(p,t), Ssupcrit(p,t), Rsupcrit(p,t), Vsupcrit(p,t))

```
In[ ]:= (*enthalpy*)
        ptHsupcritVals = Import[".\\i2stmData\\i2stmPTHsupercritdata.csv"];
        ptHsupcrit = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptHsupcritVals;
        (*entropy*)
        ptSsupcritVals = Import[".\\i2stmData\\i2stmPTSsupercritdata.csv"];
        ptSsupcrit = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSsupcritVals;
        (*density*)
        ptRsupcritVals = Import[".\\i2stmData\\i2stmPTRsupercritdata.csv"];
        ptRsupcrit = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptRsupcritVals;
        (*specific volume*)
        ptVsupcritVals = Import[".\\i2stmData\\i2stmPTVsupercritdata.csv"];
        ptVsupcrit = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptVsupcritVals;
```

```
In[]:= (*interplating functions*)
intstmHatPTsupcrit = Interpolation[ptHsupcrit, InterpolationOrder → 1];
intstmSatPTsupcrit = Interpolation[ptSsupcrit, InterpolationOrder → 1];
intstmRatPTsupcrit = Interpolation[ptRsupcrit, InterpolationOrder → 1];
intstmVatPTsupcrit = Interpolation[ptVsupcrit, InterpolationOrder → 1];
```

Subcritical pressure and superheated temperature (Hsh(p,t), Ssh(p,t), Rsh(p,t), Vsh(p,t))

```
In[]:= (*enthalpy*)
ptSHHlist02Vals = Import[".\\i2stmData\\i2stmPTHsubcritVapordata.csv"];
ptSHHlist02 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSHHlist02Vals;
(*entropy*)
ptSHSlist02Vals = Import[".\\i2stmData\\i2stmPTSsubcritVapordata.csv"];
ptSHSlist02 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSHSlist02Vals;
(*density*)
ptSHRlist02Vals = Import[".\\i2stmData\\i2stmPTRsubcritVapordata.csv"];
ptSHRlist02 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSHRlist02Vals;
(*specific volume*)
ptSHVlist02Vals = Import[".\\i2stmData\\i2stmPTVsubcritVapordata.csv"];
ptSHVlist02 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSHVlist02Vals;
```

Make interpolating functions

```
In[]:= intstmHatPTsh = Interpolation[ptSHHlist02, InterpolationOrder → 1];
intstmSatPTsh = Interpolation[ptSHSlist02, InterpolationOrder → 1];
intstmRatPTsh = Interpolation[ptSHRlist02, InterpolationOrder → 1];
intstmVatPTsh = Interpolation[ptSHVlist02, InterpolationOrder → 1];
```

Subcritical pressure and subcooled temperature (Hsubcool(p,t), Ssubcool(p,t), Rsubcool(p,t), Vsubcool(p,t))

```
In[]:= (*enthalpy*)
ptHlist01Vals = Import[".\\i2stmData\\i2stmPTHsubcritLiquiddata.csv"];
ptHlist01 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptHlist01Vals;
(*entropy*)
ptSlist01Vals = Import[".\\i2stmData\\i2stmPTSsubcritLiquiddata.csv"];
ptSlist01 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptSlist01Vals;
(*density*)
ptRlist01Vals = Import[".\\i2stmData\\i2stmPTRsubcritLiquiddata.csv"];
ptRlist01 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptRlist01Vals;
(*specific volume*)
ptVlist01Vals = Import[".\\i2stmData\\i2stmPTVsubcritLiquiddata.csv"];
ptVlist01 = {{#[[1]], #[[2]]}, #[[3]]} & /@ ptVlist01Vals;
```

```
In[ ]:= intstmHatPTliq = Interpolation[ptHlist01, InterpolationOrder → 1];
        intstmSatPTliq = Interpolation[ptSlist01, InterpolationOrder → 1];
        intstmRatPTliq = Interpolation[ptRlist01, InterpolationOrder → 1];
        intstmVatPTliq = Interpolation[ptVlist01, InterpolationOrder → 1];
```

Combined results

```
In[ ]:= i2stmHatPT[pPsia_?NumericQ, tF_?NumericQ] := Module[{},
          Which[
           pPsia ≥ istmPcritval, intstmHatPTsupcrit[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) > 0.001, intstmHatPTsh[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) < -0.001, intstmHatPTliq[pPsia, tF],
           True, intstmHsatlatP[pPsia]
          ]
         ]

        i2stmSatPT[pPsia_?NumericQ, tF_?NumericQ] := Module[{},
          Which[
           pPsia ≥ istmPcritval, intstmSatPTsupcrit[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) > i2stmEps, intstmSatPTsh[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) < -i2stmEps, intstmSatPTliq[pPsia, tF],
           True, intstmSsatlatP[pPsia]
          ]
         ]

        i2stmRatPT[pPsia_?NumericQ, tF_?NumericQ] := Module[{},
          Which[
           pPsia ≥ istmPcritval, intstmRatPTsupcrit[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) > i2stmEps, intstmRatPTsh[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) < -i2stmEps, intstmRatPTliq[pPsia, tF],
           True, intstmRsatlatP[pPsia]
          ]
         ]

        i2stmVatPT[pPsia_?NumericQ, tF_?NumericQ] := Module[{},
          Which[
           pPsia ≥ istmPcritval, intstmVatPTsupcrit[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) > i2stmEps, intstmVatPTsh[pPsia, tF],
           (tF - intstmTsatatP[pPsia]) < -i2stmEps, intstmVatPTliq[pPsia, tF],
           True, intstmVsatlatP[pPsia]
          ]
         ]
```

Saturated conditions (Hpx(p,x), Spx(p,x), Rpx(p,x), Vpx(p,x) and Xph(p,h),

Xps(p,s)

```
In[ ]:= i2stmHatPx[pPsia_?NumericQ, xIn_?NumericQ] := Module[{x},
          x = Min[1, Max[0, xIn]];
          Which[
           pPsia ≥ istmPcritval, -99,
           True, (1 - x) intstmHsatlatP[pPsia] + x intstmHsatvatP[pPsia]
          ]
         ]

        i2stmSatPx[pPsia_?NumericQ, xIn_?NumericQ] := Module[{x},
          x = Min[1, Max[0, xIn]];
          Which[
           pPsia ≥ istmPcritval, 0,
           True, (1 - x) intstmSsatlatP[pPsia] + x intstmSsatvatP[pPsia]
          ]
         ]

        i2stmRatPx[pPsia_?NumericQ, xIn_?NumericQ] := Module[{x},
          x = Min[1, Max[0, xIn]];
          Which[
           pPsia ≥ istmPcritval, 0,
           True, (1 - x) intstmRsatlatP[pPsia] + x intstmRsatvatP[pPsia]
          ]
         ]

        i2stmVatPx[pPsia_?NumericQ, xIn_?NumericQ] := Module[{x},
          x = Min[1, Max[0, xIn]];
          Which[
           pPsia ≥ istmPcritval, 0,
           True, (1 - x) intstmVsatlatP[pPsia] + x intstmVsatvatP[pPsia]
          ]
         ]

        i2stmXatPH[pPsia_?NumericQ, hBtulb_?NumericQ] := Module[{},
          Which[
           pPsia ≥ istmPcritval, 1,
           hBtulb ≥  intstmHsatvatP[pPsia] , 1,
           hBtulb ≤  intstmHsatlatP[pPsia] , 0,
           True,
           (hBtulb - intstmHsatlatP[pPsia] ) / ( intstmHsatvatP[pPsia] - intstmHsatlatP[pPsia])
          ]
         ]

        i2stmXatPS[pPsia_?NumericQ, sBtulbF_?NumericQ] := Module[{},
          Which[
      pPsia ≥ istmPcritval, 1,
      sBtulbF ≥  intstmSsatvatP[pPsia] , 1,
      sBtulbF ≤  intstmSsatlatP[pPsia] , 0,
      True,
      (sBtulbF - intstmSsatlatP[pPsia] ) / ( intstmSsatvatP[pPsia] - intstmSsatlatP[pPsia])
     ]
    ]
```

PH functions: Tph, Sph, Rph, Vph

To find T, given ph, use find root to find t s.t. h = h(p,t)
T find S and R: given ph, find T, then use pt to find required value.

In[ ]:=

```
In[ ]:= Remove[intstmTatPH];
    intstmTatPH[pPsia_?NumericQ, hBtuLb_?NumericQ,
      tminF_?NumericQ, tmaxF_?NumericQ] := Module[{ tx, r},
      r = FindRoot[ hBtuLb == i2stmHatPT[pPsia, tx], {tx, tminF, tmaxF}, AccuracyGoal → 3];
      tx /. r
     ]
    intstmTatPH[pPsia_?NumericQ, hBtuLb_?NumericQ] := Module[{ tx, r},
      Which[
       pPsia > istmPcritval, intstmTatPH[pPsia, hBtuLb, 40, 1200],
       hBtuLb > intstmHsatvatP[pPsia],
       intstmTatPH[pPsia, hBtuLb,  intstmTsatatP[pPsia], 1200],
       (intstmHsatvatP[pPsia] ≥ hBtuLb && hBtuLb ≥  intstmHsatlatP[pPsia]),
       intstmTsatatP[pPsia],
       hBtuLb < intstmHsatvatP[pPsia],
       intstmTatPH[pPsia, hBtuLb,  40, intstmTsatatP[pPsia]],
       True, intstmTatPH[pPsia, hBtuLb, 40, 1200]
      ]
     ]

    intstmSatPH[pPsia_?NumericQ, hBtuLb_?NumericQ] := Module[{},
      Which[
       hBtuLb > i2stmHatPT[pPsia, 1200], -99,
       hBtuLb < i2stmHatPT[pPsia, 40], -98,
       pPsia > istmPcritval, i2stmSatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]],
       (intstmHsatvatP[pPsia] ≥ hBtuLb && hBtuLb ≥  intstmHsatlatP[pPsia]),
       i2stmSatPx[pPsia, i2stmXatPH[pPsia, hBtuLb]],
       hBtuLb > intstmHsatvatP[pPsia],  i2stmSatPT[pPsia,
        intstmTatPH[pPsia, hBtuLb, intstmTsatatP[pPsia] + 0.01, 1200]],
       hBtuLb < intstmHsatlatP[pPsia],  i2stmSatPT[pPsia,
        intstmTatPH[pPsia, hBtuLb, 40, intstmTsatatP[pPsia] - 0.01]],
       True, i2stmSatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]]
      ]
```

```
 ]

intstmRatPH[pPsia_?NumericQ, hBtuLb_?NumericQ] := Module[{},
  Which[
   hBtuLb > i2stmHatPT[pPsia, 1200], -99,
   hBtuLb < i2stmHatPT[pPsia, 40], -98,
   pPsia > istmPcritval, i2stmRatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]],
   (intstmHsatvatP[pPsia] ≥ hBtuLb && hBtuLb ≥ intstmHsatlatP[pPsia]),
   i2stmRatPx[pPsia, i2stmXatPH[pPsia, hBtuLb]],
   hBtuLb > intstmHsatvatP[pPsia], i2stmRatPT[pPsia,
    intstmTatPH[pPsia, hBtuLb, intstmTsatatP[pPsia] + 0.01, 1200]],
   hBtuLb < intstmHsatlatP[pPsia], i2stmRatPT[pPsia,
    intstmTatPH[pPsia, hBtuLb, 40, intstmTsatatP[pPsia] - 0.01]],
   True, i2stmRatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]]
  ]
 ]

intstmVatPH[pPsia_?NumericQ, hBtuLb_?NumericQ] := Module[{},
  Which[
   hBtuLb > i2stmHatPT[pPsia, 1200], -99,
   hBtuLb < i2stmHatPT[pPsia, 40], -98,
   pPsia > istmPcritval, i2stmVatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]],
   (intstmHsatvatP[pPsia] ≥ hBtuLb && hBtuLb ≥ intstmHsatlatP[pPsia]),
   i2stmVatPx[pPsia, i2stmXatPH[pPsia, hBtuLb]],
   hBtuLb > intstmHsatvatP[pPsia], i2stmVatPT[pPsia,
    intstmTatPH[pPsia, hBtuLb, intstmTsatatP[pPsia] + 0.01, 1200]],
   hBtuLb < intstmHsatlatP[pPsia], i2stmVatPT[pPsia,
    intstmTatPH[pPsia, hBtuLb, 40, intstmTsatatP[pPsia] - 0.01]],
   True, i2stmVatPT[pPsia, intstmTatPH[pPsia, hBtuLb, 40, 1200]]
  ]
 ]
```

PS functions: Hps

We develop interpolation functions for the following regions:
a) supercritical (P > Pcrit)
b) sub-critical liquid (s < Ssatl)
c) sub-critical vapor (s > Ssatl)
d) sub-critical mixture (s >= Ssatl && s <= Ssatv)

First make a set of functions using FindRoot.
Use this to create a set of points that can be used for an interpolating function

Our strategy is to start with a master grid resulting from combining 3 sets of pressure entropy values:
a) regularly spaced grid over the entire p-s range of interest

b) closely spaced grid near the saturation line

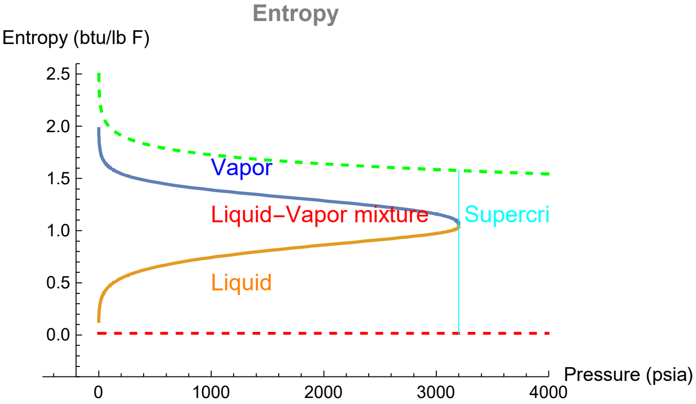
c) cloely spaced grid near the sub-scritical to supercritical boundary

Then we check

First review the range of entropy values

```
In[ ]:= plotSsat = Plot[{intstmSsatvatP[p], intstmSsatlatP[p] }, {p, 1, 3200.1}
    , PlotRange → {{-500, 4000}, {-0.4, 2.6}}
    , PlotLabel → Style["Entropy", 12, Gray, Bold]
    , AxesLabel →
     {Style["Pressure (psia)", 10, Black], Style["Entropy (btu/lb F)", 10, Black ]}
    , AxesOrigin → {-200, -0.4}
    , Epilog → {Cyan,
      Line[{{3200.11, i2stmSatPT[3200, 40.1]}, {3200.11, i2stmSatPT[3200., 1200.]}} ]
      , Text[Style["Supercritical", 12, Cyan], {3250, 1.0}, {-1, -1}]
      , Text[Style["Vapor", 12, Blue], {1000, 1.45}, {-1, -1}]
      , Text[Style["Liquid-Vapor mixture", 12, Red], {1000, 1.0}, {-1, -1}]
      , Text[Style["Liquid", 12, Orange], {1000, 0.35}, {-1, -1}]
     }
   ];
plotSbounds = Plot[{i2stmSatPT[p, 40], i2stmSatPT[p, 1200]}, {p, 1, 4000},
    PlotStyle → {{Red, Dashed}, {Green, Dashed}}
    , PlotRange → {{-500, 4000}, {-0.4, 2.6}}

   ];
Show[{plotSsat, plotSbounds}]
```

Out[ ]=



Tps (using FindRoot)

```
In[ ]:= Remove[intstmTatPS];
        intstmTatPS[pPsia_?NumericQ, sBtuLbF_?NumericQ,
          tminF_?NumericQ, tmaxF_?NumericQ] := Module[{tx, r},
          r = FindRoot[ sBtuLbF == i2stmSatPT[pPsia, tx], {tx, tminF, tmaxF}, AccuracyGoal → 3];
          tx /. r
         ]
        intstmTatPS[pPsia_?NumericQ, sBtuLbF_?NumericQ] := Module[{tx, r},
          Which[
           pPsia > istmPcritval, intstmTatPS[pPsia, sBtuLbF, 40, 1200],
           sBtuLbF > intstmSsatvatP[pPsia],
           intstmTatPS[pPsia, sBtuLbF, intstmTsatatP[pPsia], 1200],
           (intstmSsatvatP[pPsia] ≥ sBtuLbF && sBtuLbF ≥ intstmSsatlatP[pPsia]),
           intstmTsatatP[pPsia],
           sBtuLbF < intstmSsatvatP[pPsia],
           intstmTatPS[pPsia, sBtuLbF, 40, intstmTsatatP[pPsia]],
           True, intstmTatPS[pPsia, sBtuLbF, 40, 1200]
          ]
         ]
```

Hps (using Tps or Xps, if saturated)

```
In[ ]:= intstmHatPS[pPsia_?NumericQ, sBtuLbF_?NumericQ] := Module[{},
          Which[
           sBtuLbF > i2stmSatPT[pPsia, 1200], -99,
           sBtuLbF < i2stmSatPT[pPsia, 40], -98,
           pPsia > istmPcritval, i2stmHatPT[pPsia, intstmTatPS[pPsia, sBtuLbF, 40, 1200]],
           (intstmSsatvatP[pPsia] ≥ sBtuLbF && sBtuLbF ≥ intstmSsatlatP[pPsia]),
           i2stmHatPx[pPsia, i2stmXatPS[pPsia, sBtuLbF]],
           sBtuLbF > intstmSsatvatP[pPsia], i2stmHatPT[pPsia,
            intstmTatPS[pPsia, sBtuLbF, intstmTsatatP[pPsia] + 0.01, 1200]],
           sBtuLbF < intstmSsatlatP[pPsia], i2stmHatPT[pPsia,
            intstmTatPS[pPsia, sBtuLbF, 40, intstmTsatatP[pPsia] - 0.01]],
           True, i2stmHatPT[pPsia, intstmTatPS[pPsia, sBtuLbF, 40, 1200]]
          ]
         ]
```

Equations for Flow and pressure drop

The following criteria are applicable for the physical systems of interest. The governor valve will always be slightly open, so jr is greater than zero. The critical pressure ratio is 0.55, so the parameter α is between 0 and 1, and the 1st stage pressure is above absolute vacuum.

```
In[ ]:= solAssumptions = {jr >  0 && 1 > α > 0 && pxr > 0};
```

Equation summary

Governor valve equations, where we have divided both sides by Jnoz and $P_{\text{in}}^{1/2}$

```
In[ ]:= fgovnc = jr Sqrt[ ρin (1 - pbr)];
        fgovch = jr Sqrt[ ρin (1 - α) ];
        pbrGovLim = α ; (*if the ratio of the bown pressure to inlet pressure is less than α,
        the governor will be choked*)
```

Nozzle equations

```
In[ ]:= fnoznc =  Sqrt[ρin (pbr) (pbr - pxr)];
        (*earlier versions of Dynsim omit the
         pressure correction for the nozzle inlet density*)
        fnozncDyn =  Sqrt[ρin (pbr - pxr)];
        fnozch  = Sqrt[ρin (pbr) (1 - α) pbr];
        pbrNovLim = pxr / α ;(* condition when the nozzle is choked,
        depends on both bowl pressure and outlet pressure (1st stage pressure) *)
```

Case 1: Neither choked

Determine the analytic solutions for the bowl pressure, using the equations for the governor and nozzle not choked. There are two possible solutions. The Mathematica Refine routine is used to find the solution that is physically meaningful.

```
In[ ]:= solnc = Solve[fgovnc == fnoznc, pbr]
        Refine[(pbr /. #) >  0, solAssumptions] &  /@ solnc
        sol2nc = pbr /. solnc[[2, 1]] // FullSimplify;
```

Out[ ]=
$$\left\{\left\{\text{pbr}\to\frac{1}{2}\left(-\text{jr}^2+\text{pxr}-\sqrt{4\,\text{jr}^2+\text{jr}^4-2\,\text{jr}^2\,\text{pxr}+\text{pxr}^2}\right)\right\},\left\{\text{pbr}\to\frac{1}{2}\left(-\text{jr}^2+\text{pxr}+\sqrt{4\,\text{jr}^2+\text{jr}^4-2\,\text{jr}^2\,\text{pxr}+\text{pxr}^2}\right)\right\}\right\}$$

```
Out[ ]= {False, True}
```

Dynsim Equations

Prior to Dynsim 5.3, Dynsim-P has an error. For the neither choked case, Dynsim does not account for a difference in the density in the inlet and bowl locations.

```
In[ ]:= fNoznc == fnozncDyn // TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{fNoznc} = \sqrt{\rho_{\text{in}}\,(\text{pbr} - \text{pxr})}$$

In[ ]:=
```
solncDyn = pbr /. Solve[fgovnc == fnozncDyn, pbr][[1]];
pbr == solncDyn // TraditionalForm
```

Out[ ]//TraditionalForm=

$$\text{pbr} = \frac{\text{jr}^2 + \text{pxr}}{\text{jr}^2 + 1}$$

Multiplying both side by Pin yields

$$\text{pbr} * \text{Pin} = \left(\frac{\text{jr}^2 + \text{pxr}}{\text{jr}^2 + 1}\right) \text{Pin}$$

$$\frac{\text{Pb}}{\text{Pin}} \text{Pin} = \left(\frac{\text{jr}^2 + \frac{\text{P1st}}{\text{Pin}}}{\text{jr}^2 + 1}\right) \text{Pin} \tag{9}$$

$$\text{Pb} = \left(\frac{\text{P1st} + \text{jr}^2 \text{Pin}}{\text{jr}^2 + 1}\right)$$

The result above is identical the equation used in the neither choked branch of the Dynsim code, shown below.

```
            {
                // Neither valve nor nozzle is choked
                //arg1 = -Jrsq;
                //arg2 = Jrsq - Pfsr;
                //arg3 = 1.0;
                //Pbowl[i]  = nhi->P * quadr_(&arg1, &arg2, &arg3);
                Pbowl[i] = (nhx->P + Jrsq * nhi->P) / (1.0 + Jrsq);
                arg1 = (float)(Ri*(nhi->P - Pbowl[i]));
                arg2 = 0.0;
                // changed to prevent cross-threading
                //Fgv[i] = gv->J * Sqr(&arg1, &arg2);
                Fgv[i] = gv->J.getFVNS() * Sqr(&arg1, &arg2);
             }
```

Case II: Governor valve choked and Nozzle not choked

Following the same steps as Case I, we use the flow equations when the governor is choked and the nozzle is not choked.

In[ ]:=
```
solgv = Solve[fgovch == fnoznc, pbr];
Refine[(pbr /. #) > 0, solAssumptions] & /@ solgv
sol2gv = pbr /. solgv[[2, 1]];
```

Out[ ]=
```
{False, True}
```

Dynsim Equations

Here we derive the equations used in the Dynsim code and compare them with the result above.

$$\begin{aligned} \text{fgovch} &= \text{jgov Sqrt}[\rho_{\text{in}}\,(1-\alpha)\,\text{Pin}] \\ \text{fnoznc} &= \text{jnoz Sqrt}[\rho_{\text{in}}\,(\text{Pb}/\text{Pin})\,(\text{Pb}-\text{Px})] \end{aligned} \tag{10}$$

Equate both flows

$$\text{jgov Sqrt}[\rho_{\text{in}}\ (1-\alpha)\ \text{Pin}] = \text{jnoz Sqrt}[\rho_{\text{in}}\ (\text{Pb}/\text{Pin})\ (\text{Pb}-\text{Px})] \tag{11}$$

Divide by jnoz and square both sides

$$\left(\frac{\text{jgov}}{\text{jnoz}}\right)^2 \rho_{\text{in}}\ (1-\alpha)\ \text{Pin} = \rho_{\text{in}}\ (\text{Pb}/\text{Pin})\ (\text{Pb}-\text{Px}) \tag{12}$$

Divide by Pin, note that Pb/Pin = pbr and Px/Pin = pxr. Cancel ρ_{in} which appears on both sides

$$\begin{gathered}\left(\frac{\text{jgov}}{\text{jnoz}}\right)^2 \rho_{\text{in}}\ (1-\alpha)\ \frac{\text{Pin}}{\text{Pin}} = \rho_{\text{in}}\ (\text{Pb}/\text{Pin})\ \left(\frac{\text{Pb}}{\text{Pin}}-\frac{\text{Px}}{\text{Pin}}\right)\\ \left(\frac{\text{jgov}}{\text{jnoz}}\right)^2 (1-\alpha) = (\text{pbr})\ (\text{pbr}-\text{pxr})\end{gathered} \tag{13}$$

Collect terms and put the equation in standard form. Note that α is 0.55 for steam.

$$\begin{gathered}\text{pbr}^2 + \text{pxr pbr} - \left(\frac{\text{jgov}}{\text{jnoz}}\right)^2 (1-\alpha) = 0\\ \text{pbr}^2 + \text{pxr pbr} - 0.45\left(\frac{\text{jgov}}{\text{jnoz}}\right)^2 = 0\end{gathered} \tag{14}$$

This result is consistent with the Dynsim dsst_TurGovNozFlow code. In the routine flowComp the PbVlv routine corresponds to the the bowl pressure if only the governor valve is choked.
The Dynsim code uses the expression: arg1 + arg2 X + arg3 X^2 = 0. The coefficients in the equation above and the code below match.

```
Jrsq  = Jred*Jred;
            arg1 = (float)-0.45*Jrsq;
            arg2 = -Pfsr;
            arg3 = 1.0;
            PbVlv = (float)(nhi->P *

#ifdef SUN
                             quadr_(
#endif // end SUN
#ifdef _MSC_VER
                             QUADR(
#endif // end _MSC_VER
                                    &arg1, &arg2, &arg3) );
```

We can check that the Dynsim equation and the equation derived previously are identical. First generate a solution from the Dynsim equation

```
In[ ]:= solGovckAllDyn  = Solve[ pbr^2 - pxr pbr - (1 - α) jr^2 == 0, pbr];
        solGovckDynPos = Refine[(pbr /. #) >  0, solAssumptions] & /@ solGovckAllDyn;
        solGovckDyn = pbr /. First@Pick[Flatten@solGovckAllDyn, solGovckDynPos ]
```

Out[ ]= $$\frac{1}{2}\left(\text{pxr}+\sqrt{4\,\text{jr}^2+\text{pxr}^2-4\,\text{jr}^2\,\alpha}\right)$$

Note the solution we developed for case 2, when the governor was chosen. These equations clearly match.

```
In[ ]:= sol2gv
```

Out[ ]= $$\frac{1}{2}\left(\text{pxr}+\sqrt{4\,\text{jr}^2+\text{pxr}^2-4\,\text{jr}^2\,\alpha}\right)$$

We can also ask Mathematica to test if the equations are identical. (trivial now, but useful for more complicated problems)

```
In[ ]:= sol2gv == solGovckDyn // FullSimplify
Out[ ]= True
```

Case III: Governor not choked and Nozzle choked

Following the same steps as Case I, we use the flow equations when the governor is not-choked and the nozzle is choked.

```
In[ ]:= solNoz  =  Solve[fgovnc  == fnozch, pbr];
        solNozPos  = Refine[(pbr /. #) >  0, solAssumptions] & /@ solNoz
        pbr /. First@Pick[ Flatten@solNoz, solNozPos]
        sol2Noz  = pbr /. solNoz[[1, 1]];
Out[ ]= {True, False}
```

Out[ ]= $$\frac{\text{jr}^2-\text{jr}\sqrt{4+\text{jr}^2-4\,\alpha}}{2\,(-1+\alpha)}$$

Dynsim Equations

Below we derive the equations used in the Dynsim code

$$\begin{aligned}\text{fgovnc} &= \text{jgov Sqrt}[\rho_{\text{in}}\,(\text{Pin}-\text{Pb})]\\ \text{fnozch} &= \text{jnoz Sqrt}[\rho_{\text{in}}\,(\text{Pb}/\text{Pin})\,(1-\alpha)\,\text{Pb}]\end{aligned} \qquad (15)$$

equating these yields

$$\text{jgov Sqrt}[\rho_{\text{in}}\,(\text{Pin}-\text{Pb})] = \text{jnoz Sqrt}[\rho_{\text{in}}\,(\text{Pb}/\text{Pin})\,(1-\alpha)\,\text{Pb}] \qquad (16)$$

dividing by jnoz, squaring both sides yields

$$\frac{\text{jgov}}{\text{jnoz}}\text{Sqrt}[(\text{Pin}-\text{Pb})] = \text{Sqrt}[(\text{Pb}/\text{Pin})\,(1-\alpha)\,\text{Pb}] \qquad (17)$$

square both sides and divide by Pin

$$\mathtt{jr}^2\ (1-\mathtt{pbr}) = \mathtt{pbr}\ (1-\alpha)\ \mathtt{pbr} \qquad (18)$$

collecting terms and putting this in normal form.

$$(1-\alpha)\ \mathtt{pbr}^2 + \mathtt{jr}^2\ \mathtt{pbr} - \mathtt{jr}^2 = 0$$

$$\mathtt{pbr}^2 + \frac{\mathtt{jr}^2}{(1-\alpha)}\ \mathtt{pbr} - \frac{\mathtt{jr}^2}{(1-\alpha)} = 0 \qquad (19)$$

$$\mathtt{pbr}^2 + \frac{\mathtt{jr}^2}{0.45}\ \mathtt{pbr} - \frac{\mathtt{jr}^2}{0.45} = 0$$

One can compare the equation above with the Dynsim code from dsst_TurGovNozFlow::flowComp(). A copy of the code is below. Note that the coefficients are the same, for example arg1 = -jr^2/0.45

```
            arg1 = static_cast<float>(-Jrsq/0.45);
            arg2 = static_cast<float>(Jrsq/0.45);
            arg3 = (float)1.0;

            PbNoz = (float)(nhi->P *
#ifdef SUN
                             quadr_(
#endif // end SUN
#ifdef _MSC_VER
                             QUADR(
#endif // end _MSC_VER
                                    &arg1, &arg2, &arg3));
```

We can check that the Dynsim equation and the equation derived previously are identical. First generate a solution from the Dynsim equation

```
In[ ]:= solNozckAllDyn  = Solve[ pbr^2 + jr^2 / (1 - α) pbr - jr^2 / (1 - α)  == 0, pbr];
        solNozDynPos = Refine[ (pbr /. #) >  0, solAssumptions] & /@ solNozckAllDyn;
        solNozckDyn = pbr /. First@Pick[Flatten@solNozckAllDyn, solNozDynPos ]
```

Out[ ]=
$$\frac{1}{2}\left(-\frac{\mathtt{jr}^2}{1-\alpha}-\frac{\mathtt{jr}\sqrt{4+\mathtt{jr}^2-4\alpha}}{-1+\alpha}\right)$$

Then test if the equations are identical.

```
In[ ]:= sol2Noz == solNozckDyn // FullSimplify

Out[ ]= True
```

Case IV: Governor and Nozzle both choked

```
In[ ]:= solgvnoz =  Solve[fgovch == fnozch, pbr];
        Refine[ (pbr /. #) >  0, solAssumptions] &  /@ solgvnoz
        sol2gvnoz = pbr /. solgvnoz[[2, 1]];

Out[ ]= {False, True}
```

Region plot for four flow regimes (2-D and 3-D plots)

We will derive criteria to determine which of the four operating states (not choked, governor choked, nozzle choked, or both choked) is applicable to a given set of operating data. We start with the neither choked cases. Then look for the limits of when the system just becomes choked. We will present our results as a graph of the area of the jr-pxr space, and indicate which regions of the jr-pxr space correspond to neither choked, governor choked, nozzle choked, and both choked. This indicates that the operating region can be determined given Pin, P1st stage, and the governor and nozzle flow coefficients

Formatting details

```
In[ ]:= rgbPbSurf = Orange;
        rgbGvckSurf = RGBColor[0.10, 0.25, 1.0];
        rgbNzckSurf = Green;
```

Neither choked

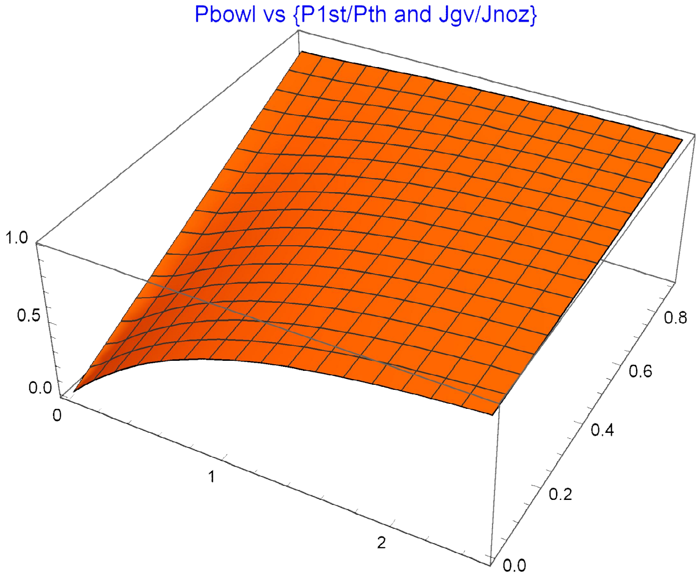
```
In[ ]:= pbr == sol2nc // DisplayForm
```

Out[ ]//DisplayForm=

$$\text{pbr} == \frac{1}{2}\left(-\text{jr}^2+\text{pxr}+\sqrt{\text{jr}^4-2\,\text{jr}^2\,(-2+\text{pxr})+\text{pxr}^2}\right)$$

```
In[ ]:= Plot3D[ sol2nc, {jr, 0, 2.5}, {pxr, 0, 0.9}
         , PlotLabel → Style["Pbowl vs {P1st/Pth and Jgv/Jnoz}", Blue, 12]
         , PlotStyle → {rgbPbSurf}
        ]
```

Out[ ]=



The z-axis, from 0 to 0.1, is the bowl pressure, downstream of the governor valves.

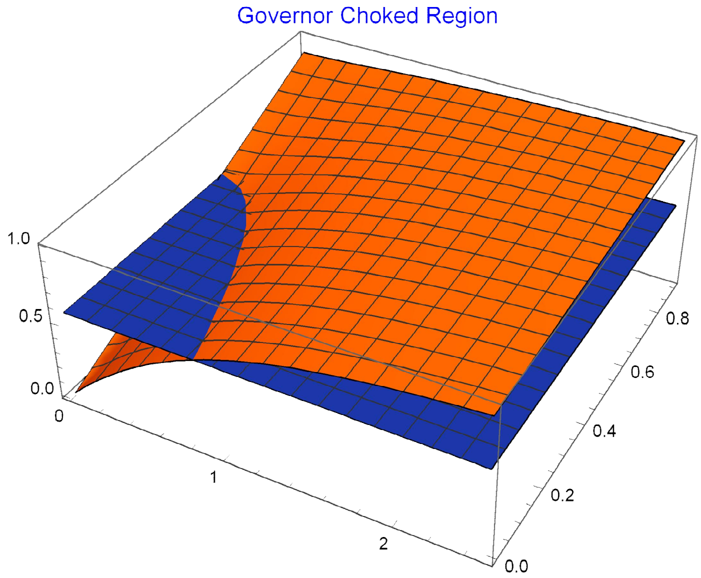
The x-axis is the ratio of Jgv/Jnoz, from 0 to 2.5.
The y-axis is the ratio of 1st stage pressure to throttle pressure, from 0 to 0.9

Governor choked

The governor is choked when pbr < 0.55, i.e. if the bowl pressure is too far below the inlet pressure than the governor valve will choke. The limit of pbr of 0.55 is shown as a blue horizontal surface in the plot below. The governor valves are choked whenever bowl pressure is below the blue surface on the 3-D graph below.

```
In[ ]:= Plot3D[{ sol2nc, 0.55}, {jr, 0, 2.5}, {pxr, 0, 0.9},
         PlotLabel → Style["Governor Choked Region", Blue, 12]
         , PlotStyle → {rgbPbSurf, rgbGvckSurf}
        ]
```

Out[ ]=



We can solve for the equation corresponding to the intersection of the orange and blue surfaces. The equation defining the orange surace gives pbr as a function of two variables, jr and pxr. To find the equation where the nozzle just begins to choke, set the value of the equation to the choke limit, 0.55, and then use the Mathematica Solve routine to solve for one of the two variables. For example we can solve for pxr as a function of all other parameters. We do this twice below. The first time with a numeric value for the pressure ratio limit and the second time with a symbol for the pressur ratio limit.

```
In[ ]:= pxrGVck = pxr /. Flatten@Solve[0.55 == sol2nc, pxr]
        pxrGVckTxt = pxr /. Flatten@Solve[α == sol2nc, pxr] // FullSimplify
```

Out[ ]= $0.55 - 0.818182\, jr^2$

Out[ ]= $\frac{jr^2\,(-1+\alpha)}{\alpha}+\alpha$

Below are similar results, solving for the parameter jr.

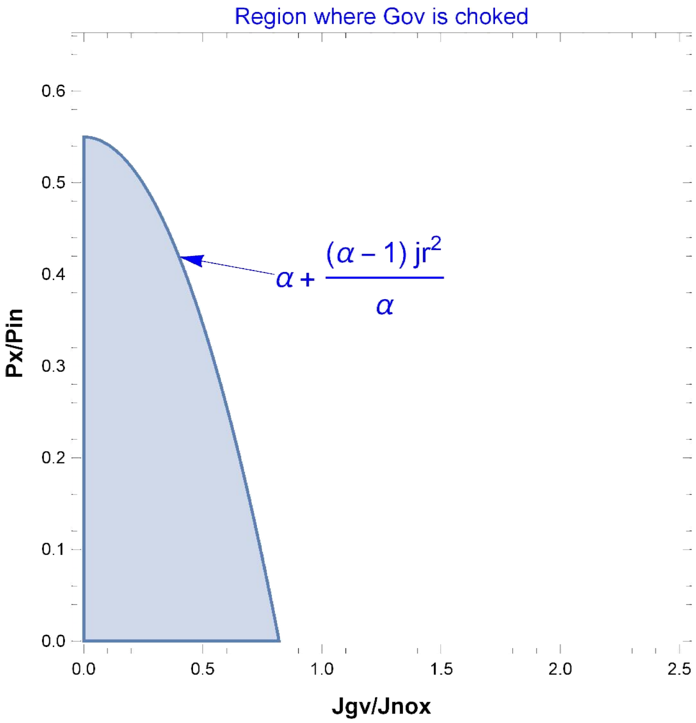
In[ ]:=
```
jrGvck = jr /. Last@Flatten@Solve[0.55 ⩵ sol2nc, jr]
```

Out[ ]= $\sqrt{0.672222 - 1.22222\ pxr}$

We can also make a plot of the region where the governor is choked.

In[ ]:=
```
RegionPlot[sol2nc < 0.55, {jr, 0, 2.5}, {pxr, 0, 0.65}
 , Frame → True
 , FrameLabel → {Style["Jgv/Jnox", Bold, 12], Style["Px/Pin", Bold, 12]}
 , PlotLabel → Style["Region where Gov is choked", Blue, 12]
 , Epilog → {Blue, Text[Style[pxrGVckTxt, 15 ], {0.8, 0.4}, {-1, 0}]
   , Arrow[{{0.8, 0.4}, {0.4, pxrGVck /. {jr → 0.4}}}]
  }]
```

Out[ ]=



Nozzle only choked

The nozzle will choke when the pressure ratio across the nozzle is be greater 0.55, which corresponds to a bowl pressure than pxr/0.55. In the introduction we wrote down this equation and review it below.

In[ ]:=
```
pbrNovLim
pbrNovLim /. α → 0.55
```

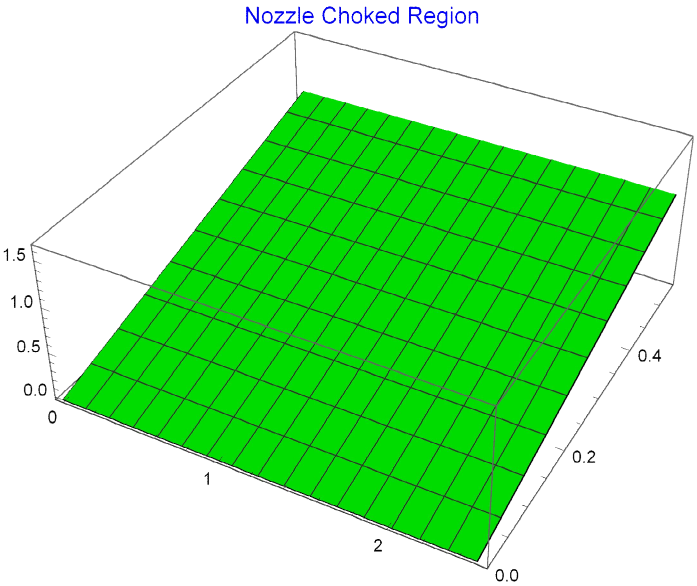
Out[ ]= $\frac{pxr}{\alpha}$

Out[ ]= 1.81818 pxr

This equation describes a surface. As the exit pressure increases the limiting bowl pressure also increases. For example, if the 1st stage stage pressure is 0.275 (in normalized coordinates), then the bowl pressure must be below 0.5, if the bowl pressure is higher than this limit the nozzle will choke.

```
In[ ]:= Plot3D[{ pbrNovLim /. α → 0.55}, {jr, 0, 2.5},
         {pxr, 0, 0.9}, PlotLabel → Style["Nozzle Choked Region", Blue, 12]
         , PlotRange → {Full, {0, 0.55}, Automatic }
         , PlotStyle → {rgbNzckSurf}
        ]
```

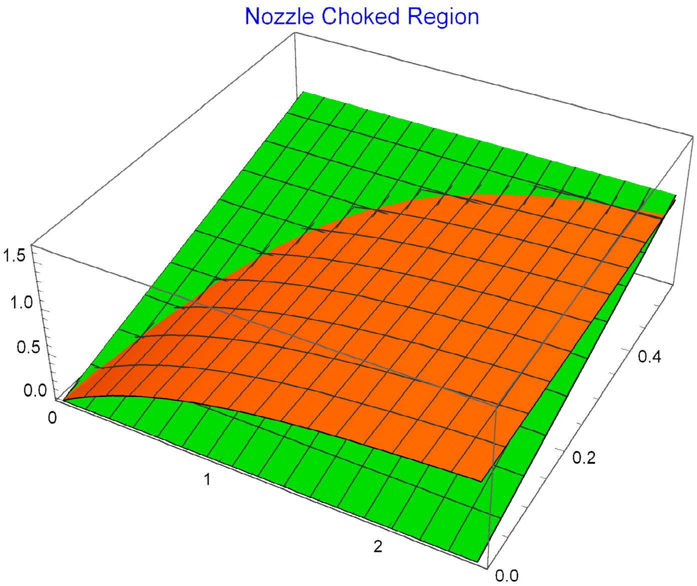
Out[ ]=



Looking for the intersection of the nothing choked surface (orange surface) and the nozzle choked surface (greeen surface). Whenver the bowl pressure is **above** this surface, the nozzle is choked.

```
In[]:= Plot3D[{ sol2nc, pbrNovLim /. α → 0.55}, {jr, 0, 2.5},
        {pxr, 0, 0.9}, PlotLabel → Style["Nozzle Choked Region", Blue, 12]
        , PlotRange → {Full, {0, 0.55}, Automatic }
        , PlotStyle → {rgbPbSurf, rgbNzckSurf}
       ]
```

Out[]=



Below derive analytic equations for when the nozzle is choked.

```
In[]:= sNzckpxr = Solve[(pbrNovLim == sol2nc), pxr] // FullSimplify
       sNzckjr = Solve[(pbrNovLim == sol2nc), jr ] // FullSimplify
```

Out[]=
$$\left\{\left\{\text{pxr}\to\frac{\text{jr}\left(\text{jr}-\sqrt{4+\text{jr}^2-4\,\alpha}\right)\alpha}{2\,(-1+\alpha)}\right\},\left\{\text{pxr}\to\frac{\text{jr}\left(\text{jr}+\sqrt{4+\text{jr}^2-4\,\alpha}\right)\alpha}{2\,(-1+\alpha)}\right\}\right\}$$

Out[]=
$$\left\{\left\{\text{jr}\to-\frac{\mathbb{i}\,\text{pxr}\sqrt{\frac{-1+\alpha}{\alpha}}}{\sqrt{-\text{pxr}+\alpha}}\right\},\left\{\text{jr}\to\frac{\mathbb{i}\,\text{pxr}\sqrt{\frac{-1+\alpha}{\alpha}}}{\sqrt{-\text{pxr}+\alpha}}\right\}\right\}$$

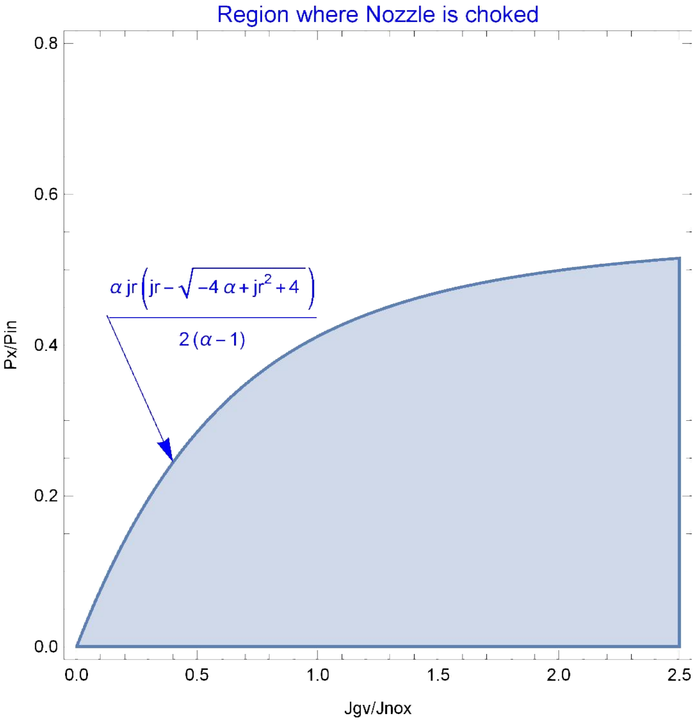
```
In[]:= pxrNZckTxt = pxr /. sNzckpxr[[1]] // FullSimplify[#, (α < 1) && ({jr, α } ∈ Reals) ] &
       pxrNZck = pxrNZckTxt /. α → 0.55
```

Out[]=
$$\frac{\text{jr}\left(\text{jr}-\sqrt{4+\text{jr}^2-4\,\alpha}\right)\alpha}{2\,(-1+\alpha)}$$

Out[]=
$$-0.611111\,\text{jr}\left(\text{jr}-\sqrt{1.8+\text{jr}^2}\right)$$

```
In[◦]:= RegionPlot[sol2nc ≥  pbrNovLim /. α → 0.55, {jr, 0, 2.5}, {pxr, 0, 0.8}
         , Frame → True
         , FrameLabel → {"Jgv/Jnox", "Px/Pin"}
         , PlotLabel → Style["Region where Nozzle is choked", Blue, 12]
         , Epilog → {Blue, Text[Style[pxrNZckTxt, 10 ], {0.125, 0.45}, {-1, 0}]
           , Arrow[{{0.125, 0.44}, {0.4, pxrNZck /. {jr → 0.4}}}]
          }]
```

Out[◦]=



Both choked

The governor is choked anytime the bowl pressure is less than 0.55 of the inlet pressure, pbr ⩽ 0.55

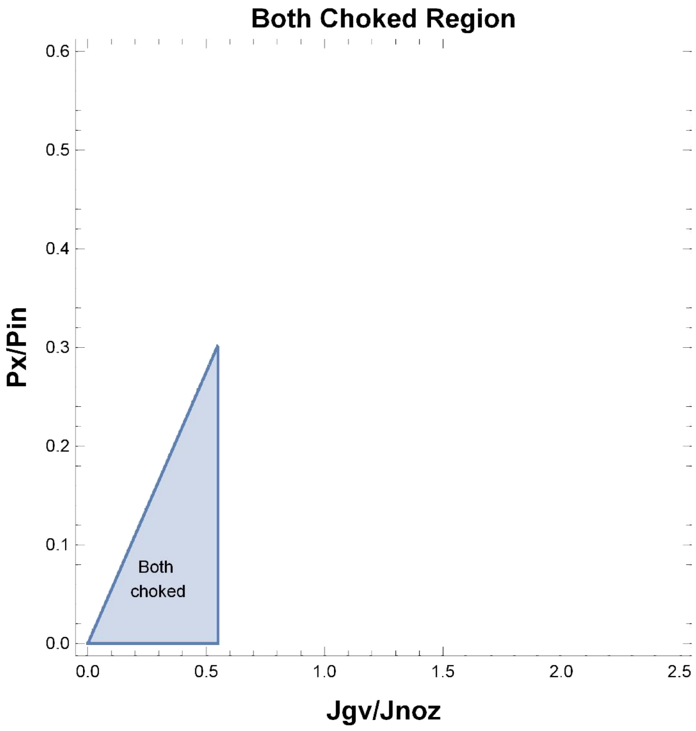
The nozzle is choked anytime the bowl pressure is greater than outlet pressure (1st stage pressure) divided by 0.55, pbr ≥ pxr/0.55.

Equating the flow conditions when both are choked produces a relationship that pbr = jr.

pbr < 0.55

In[ ]:=
```
RegionPlot[ {jr ≤ 0.55 && pxr ≤ 0.55 jr}, {jr, 0, 2.5}, {pxr, 0, 0.6}
 , MaxRecursion → 10
 , Frame → True
 , GridLines → None
 , FrameLabel → {Style["Jgv/Jnoz", Bold, 14], Style["Px/Pin", Bold, 14]}
 , PlotLabel → Style["Both Choked Region", Bold, 14]
 , Epilog → {Text[Style["Both\n choked"], {0.16, 0.04}, {-1, -1}]}
]
```

Out[ ]=



In[ ]:=

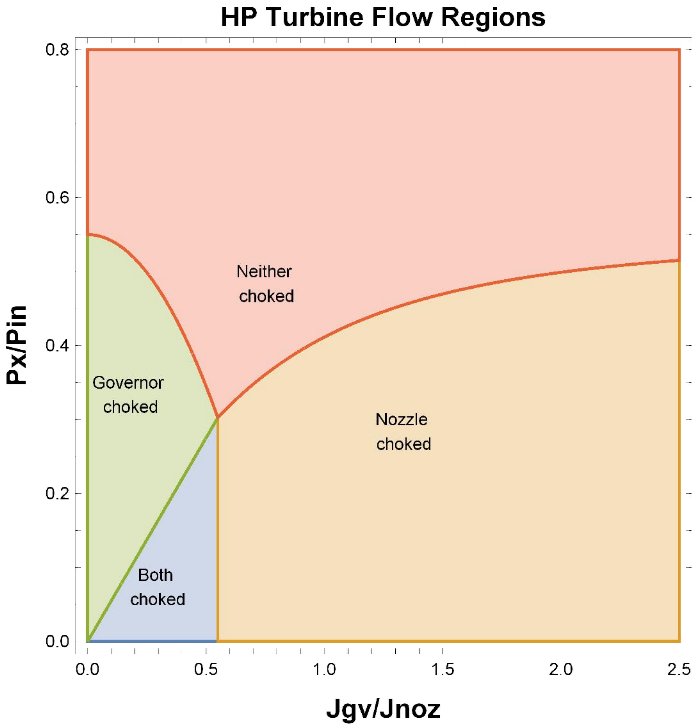
Combining the results

Define a function which returns the region for given values of jr and pxr. Let outputs be 0, 1, 2, or 3. 0 indicates neither choked, 1 indicates gov choked, 2 indicates nozzle choked, and 3 indicates both choked.

```
In[ ]:= fChokeStatus[jrIn_Real, pxrIn_Real] :=
         Piecewise[{ {3, jrIn ≤ 0.55 && pxrIn ≤ 0.55 jrIn}
           , {2, (pxrIn < pxrNZck  /.  jr → jrIn) && jrIn > 0.55}
           , {1, (pxrIn < pxrGVck /. jr → jrIn) && pxrIn > 0.55 jrIn}
          }
          , 0
         ]

        fChokeStatusDyn[jrIn_Real, pxrIn_Real] := Module[{pbVlvck, pbNozck},
          pbVlvck  = solGovckDyn /. {jr → jrIn, pxr → pxrIn, α → 0.55};
          pbNozck = solNozckDyn /. {jr → jrIn, pxr → pxrIn, α → 0.55};
           Piecewise[{ {3, jrIn ≤ 0.55 && pxrIn / 0.55 ≤   jrIn}
             , {1, (pbVlvck ≤  0.55) && pxrIn >  (0.55 pbVlvck) }
             , {2, (0.55 pbNozck ≥  pxrIn ) && (pbNozck > 0.55) }

            }
            , 0
           ]
         ]
```

```
In[]:= plotGovNoz4Regions = RegionPlot[{fChokeStatus[jr, pxr] == 3
          , fChokeStatus[jr, pxr] == 2
          , fChokeStatus[jr, pxr] == 1
          , fChokeStatus[jr, pxr] == 0}, {jr, 0, 2.5}, {pxr, 0, 0.8}
        , MaxRecursion → 10
        , Frame → True
        , GridLines → None
        , FrameLabel → {Style["Jgv/Jnoz", Bold, 14], Style["Px/Pin", Bold, 14]}
        , PlotLabel → Style["HP Turbine Flow Regions", Bold, 14]
        , Epilog → {Text[Style["Both\n choked"], {0.16, 0.04}, {-1, -1}]
          , Text[Style["Governor\n choked"], {0.02, 0.3}, {-1, -1}]
          , Text[Style["Nozzle\n choked"], {1.2, 0.25}, {-1, -1}]
          , Text[Style["Neither\n choked"], {0.62, 0.45}, {-1, -1}]
         }
      ]
```

Out[]=



```
In[]:= (*Export["plotGovNoz4Regions.png",plotGovNoz4Regions];*)
```

Dynsim Equations

We repeat the analysis above using the very same equations found in the Dynsim code. We expect the results to be the same.

```
In[ ]:= fChokeStatusDyn[jrIn_Real, pxrIn_Real] := Module[{pbVlvck, pbNozck},
          pbVlvck  = solGovckDyn /. {jr → jrIn, pxr → pxrIn, α → 0.55};
          pbNozck = solNozckDyn /. {jr → jrIn, pxr → pxrIn, α → 0.55};
           Piecewise[{ {3, jrIn ≤ 0.55 && pxrIn / 0.55 ≤  jrIn}
             , {1, (pbVlvck ≤ 0.55) && pxrIn > (0.55 pbVlvck) }
             , {2, (0.55 pbNozck ≥ pxrIn ) && (pbNozck > 0.55) }

            }
            , 0
          ]
        ]
```

```
In[ ]:= (*commented out: because this calculated takes a few minutes*)
        (*plotGovNoz4RegionsDyn = RegionPlot[{fChokeStatusDyn[jr, pxr] == 3
           ,fChokeStatusDyn[jr, pxr] == 2
           ,fChokeStatusDyn[jr, pxr] == 1
           ,fChokeStatusDyn[jr, pxr] == 0}, {jr, 0, 2.5}, {pxr,0,0.8}
          ,MaxRecursion→ 10
          ,Frame→ True
          ,GridLines→ None
          ,FrameLabel→ {Style["Jgv/Jnoz", Bold, 14],Style["Px/Pin", Bold, 14]}
          ,PlotLabel→ Style["HP Turbine Flow Regions (Dynsim Eqs)", Bold, 8]
          ,Epilog→ {Text[Style["Both\n choked"],{0.16,0.04},{-1,-1}]
            ,Text[Style["Governor\n choked"],{0.02,0.3},{-1,-1}]
            ,Text[Style["Nozzle\n choked"],{1.2,0.25},{-1,-1}]
            ,Text[Style["Neither\n choked"],{0.62,0.45},{-1,-1}]
           }
         ]*)
```

Manipulate I: Flow regimes (not choked, gov choked, nozzle choked, both choked)

nto a single graph.

Code

```
In[1]:= m1 = Manipulate[
           parms = {pxr → pratIn, α → 0.55};
           If[(pbrNovLim /. parms) > (pbrGovLim /. parms),
            {
             jr1 = NSolve[{(sol2gv /.  parms) == (pbrGovLim /. parms), jr > 0}, jr][[1]];
             jr2 = NSolve[{(sol2Noz /. parms) == (pbrNovLim /. parms), jr > 0}, jr][[1]];

             p1 = {sol2gv /.  parms, jr ≤ (jr /. jr1)};
```

```
    p2 = {sol2nc  /. parms, jr ≤  (jr /. jr2)};
    p3 = { sol2Noz /. parms, jr > (jr /. jr2)} ;

    colorfunc = Piecewise[ { {Red, # ≤  jr /. jr1 },
         {Blue , # ≤   jr /. jr2 },
         { Green, # >  jr /. jr2} }] &;

    epi = {Red
       , Line[{{0, pbrGovLim /. parms}, {2.5, pbrGovLim /. parms}}]
       , Text["Governor Choked", {2.05, pbrGovLim /. parms}, {1, 1}]
       , Arrow[{{2.05, pbrGovLim /. parms}, {2.05, 0.0}}]
       , Green
       , Line[{{0, pbrNovLim /. parms}, {2.5, pbrNovLim /. parms}}]
       , Text["Nozzle Choked", {0.52, pbrNovLim /. parms}, {-1, -1}]
       , Arrow[{{0.45, pbrNovLim /. parms}, {0.45, 1.0}}]
       , Blue, Text["Neither Choked", {1.20, (pbrGovLim + pbrNovLim) / 2 /. parms}]
      };

   }
   , {
    jr1 = NSolve[{(sol2gv /.  parms) == (pbrNovLim /. parms), jr > 0}, jr][[1]];
    jr2 = NSolve[{(sol2gvnoz /. parms) == (pbrGovLim /. parms), jr > 0}, jr][[1]];

    p1 = {sol2gv /.  parms, jr ≤ (jr /. jr1)};
    p2 = {sol2gvnoz  /. parms, jr ≤  (jr /. jr2)};
    p3 = { sol2Noz /. parms, jr > (jr /. jr2)} ;

    colorfunc = Piecewise[ { {Red, # ≤  jr /. jr1 },
         {Orange , # ≤   jr /. jr2 },
         { Green, # >  jr /. jr2} }] &;

    epi = {Red
       , Line[{{0, pbrGovLim /. parms}, {2.5, pbrGovLim /. parms}}]
       , Text["Governor Choked", {2.05, pbrGovLim /. parms}, {1, 1}]
       , Arrow[{{2.05, pbrGovLim /. parms}, {2.05, 0.0}}]
       , Green
       , Line[{{0, pbrNovLim /. parms}, {2.5, pbrNovLim /. parms}}]
       , Text["Nozzle Choked", {0.52, pbrNovLim /. parms}, {-1, -1}]
       , Arrow[{{0.46, pbrNovLim /. parms}, {0.46, 1.0}}]
       , Orange, Text["Both Governor and Nozzle Choked",
        {1.40, (pbrGovLim + pbrNovLim) / 2 /. parms}]
      };
   }
  ];

 pwnc = Piecewise[{p1, p2, p3}];
```

```
  epi2 = Flatten[{epi
      , {Black, PointSize[Medium], Point[{jr, pwnc} /. jr1]}
      , {Black, PointSize[Medium], Point[{jr, pwnc} /. jr2]}
     },
     1];
  Plot[
   pwnc, {jr, 0.01, 2.5}
   , ColorFunction → colorfunc
   , ColorFunctionScaling → False
   , Frame → True
   , FrameLabel → {"Jgov/Jnoz", "Pb/Pin"}
   , GridLines → Automatic
   , Epilog → epi2
   , PlotRange → {{0, 2.5}, {0.0, 1.2}}],
  {{pratIn, 0.37, "Px/Pi"}, 0.01, 1 - 0.5501, Appearance → "Labeled"}
  , Delimiter
  , Style["System Summary", Bold]
  , {{bc, 1, " "}, {1 → g}, ControlType → RadioButton}
  , Delimiter
  , {{pTh, 1500, "Pi"}, 1000, 2500, 50, Appearance → "Labeled"}
  , {{knoz, 10, "Knoz"}, 1, 20, 0.5, Appearance → "Labeled"}
  , ControlPlacement → {Left, Left, Left}
  , Initialization :→ (
     (*governor equations *)
     fgovnc = jr Sqrt[ ρin (1 - pbr)];
     fgovch = jr Sqrt[ ρin (1 - α) ];
     pbrGovLim = α ;
     (*nozzle equations *)
     fnoznc =  Sqrt[ρin (pbr) (pbr - pxr)];
     fnozch  = Sqrt[ρin (pbr) (1 - α) pbr];
     pbrNovLim = pxr / α;
     (*neither choked *)
     solnc = Solve[fgovnc == fnoznc, pbr];
     sol2nc = pbr /. solnc[[2, 1]];
     (*governor choked *)
     solgv = Solve[fgovch == fnoznc, pbr];
     sol2gv = pbr /. solgv[[2, 1]];
     (*nozzle choked*)
     solNoz =  Solve[fgovnc == fnozch, pbr];
     sol2Noz = pbr /. solNoz[[1, 1]];
     (*both governor and nozzle choked*)
     solgvnoz =  Solve[fgovch == fnozch, pbr];
     sol2gvnoz = pbr /. solgvnoz[[2, 1]];
     g = Show[fig01, ImageSize → Medium];
    )
 ];
```

Manipulate -- Flow Regions

Manipulate II: Flow Regions and state plot

Predicted flow rates

We can now create a function to compute the flow rate through the governor valve and nozzle. Given the boundary pressures and the flow coefficients, the equations above allow one to determine which flow regime is applicable (neither choked, governor choked, nozzle choked, or both choked). Then use the appropriate equation to find the bowl pressure. Then use the appropriate equation to determine the flow rate. For error checking two flow rates are computed: one from the governor valve flow equation and one from the nozzle flow equation. These flow rates should match.

We design a few different interfaces. One pair of options allows the user to enter data either in a reduced format (e.g. the pressure ratio and the flow coefficient ratio) or in engineering units. The other pair of options allows the user to enter the values as a list or as individual values. In all cases the rptVals parameter allows the user to specify which parameters are reported. If the keyword All is provided, then all computed parameters are reported, also a list of indices can be provided. If a single index is in the list, then the function can be used to plot results, for example.

Flow rate equations

Basic equation: calcGovFlowEng (calculated governor flow in engineering units, for a single valve and nozzle)

This function takes operating parameter and flow coefficients to compute the flow rate through the nozzle and governor.
Inputs to the function are in “Engineering units”, pressures in psia, density in lb/ft3, flow and coefficients in lb/sec / sqrt(psi lb/tt^3).
Outputs are also reported in engineering units, with flow rates in lb/sec

Inputs: Inlet Pressure (psia)
        Inlet Density (lb/ft^3)
        Governor flow coefficient (lb/sec / sqrt (psi lb/ft^3)
        Nozzle flow coefficient
        First stage pressure (psia)
        iDynErr (if True, use pre-Dynsim 5.3.1 calculation methods)
        rptVals (which values to report)

Outputs: Flow status (0: nothing choked, 1: governor choked; 2: nozzle choked, 3: both choked)

flow coefficient ratio
pressure ratio
flow rate (per chosen nozzle equation) lb/sec
flow rate (per chosen nozzle equation) lb/sec
bowl pressure (psia)

In[ ]:=

```
In[ ]:= Clear[calcGovFlowEng]
        calcGovFlowEng[ pInVal_?NumericQ , rhoVal_?NumericQ , jGVval_?NumericQ ,
          jNozVal_?NumericQ , pFirstVal_?NumericQ , iDynErr_: False, rptVals_ : All] := Module[
          {valList
           , jrVal
           , pxrVal
           , pbrVal
           , pbVal
           , fgvVal
           , fnzVal
           , qVal
           , ckStatus
           , results},
          (*step 1: determine chock status*)
          jrVal = jGVval / jNozVal;
          pxrVal = pFirstVal / pInVal;
          valList = {jr → jrVal, pxr →  pxrVal, pbr → pbrVal, ρin → rhoVal, α → 0.55};
          ckStatus = If[ iDynErr, fChokeStatusDyn[jr /. valList, pxr /. valList],
            fChokeStatus[jr /. valList, pxr /. valList] ];
          (*step 2: use appropriate flow relationships*)
          Which[
           ckStatus == 0,
           {(*neither choked*)
            pbrVal = If[ iDynErr, solncDyn /. valList, sol2nc /. valList];
            pbVal = pbrVal pInVal;
            fgvVal = jNozVal Sqrt[pInVal] fgovnc /. valList ;
            fnzVal = jNozVal Sqrt[pInVal] If[ iDynErr, fnozncDyn /. valList, fnoznc /. valList];
           },
           ckStatus == 1,
           {(*governor choked*)
            pbrVal  = sol2gv /. valList;
            pbVal = pbrVal pInVal ;
            fgvVal = jNozVal Sqrt[pInVal] fgovch /. valList ;
            fnzVal = jNozVal Sqrt[pInVal] fnoznc /. valList;
           }
           ,
           ckStatus == 2,
           {(*nozzle choked*)
            pbrVal  = sol2Noz /. valList;
```

```
      pbVal = pbrVal pInVal ;
      fgvVal = jNozVal Sqrt[pInVal] fgovnc /. valList ;
      fnzVal = jNozVal Sqrt[pInVal] fnozch /. valList;
     }
     ,
     ckStatus == 3,
     {(*Both gov and nozzle choked*)
      pbrVal = sol2gvnoz /. valList;
      pbVal = pbrVal pInVal ;
      fgvVal = jNozVal Sqrt[pInVal] fgovch /. valList ;
      fnzVal = jNozVal Sqrt[pInVal] fnozch /. valList;
     }
    ];
    (*step 3: report results*)
    results = {ckStatus, jrVal, pxrVal, fgvVal, fnzVal,
      pbrVal, fgvVal / (rhoVal pbrVal), pInVal , pbVal, pFirstVal } ;
    If[ Length@rptVals == 1
     , results[[rptVals]][[1]]
     , results[[rptVals]] ]
   ]
```

note results location

```
In[ ]:= iFlowEngCKstatus = 1;
iFlowEngJr = 2;
iFlowEngPxr = 3;
iFlowEngFgv = 4;
iFlowEngFnz = 5;
iFlowEngPbowlRatio = 6;
iFlowEngQflow = 7;
iFlowEngPgv = 8;
iFlowEngPbowl = 9;
iFlowEngPfs = 10;
```

Additional interfaces

It may be helpful to pass parameters as a list.

```
In[ ]:= calcGovFlowEng[perfList_List, rptVals_: All] := Module[{
           pInVal
           , rhoVal
           , jGVval
           , jNozVal
           , pFirstVal
           , iDynErr},
          {pInVal, rhoVal, jGVval, jNozVal, pFirstVal, iDynErr} = perfList;
          calcGovFlowEng[pInVal, rhoVal, jGVval, jNozVal, pFirstVal, iDynErr, rptVals]
         ]
```

Tests

for reporting

```
In[ ]:= strGovFlowEngfHeadings =
          {None, {"ck", "Jg/Jnz", "P1st/Pin", "Fgv \n(lb/sec)", "Fnz \n(lb/sec)"
            , "Pbowl/Pin", "Vol flow \n(ft^3/sec)" ,
            "Pgv \n (psia)",  "Pbowl \n(psia)", "Pfs \n(psia)"}};
```

some simple tests.

```
In[ ]:= {calcGovFlowEng[3500., 5.5, 0.25, 10., 2500., False, All]
          , calcGovFlowEng[3500., 5.5, 0.25, 10., 1600., False, All]
          , calcGovFlowEng[3500., 5.5, 5.00, 10., 1600.,  False, All]
          , calcGovFlowEng[3500., 5.5, 18.0, 10., 1600.,  False, All]
          , calcGovFlowEng[3500., 5.5, 50.0, 10., 1600.,  False, All]
          , calcGovFlowEng[3500., 5.5, 100., 10., 1600.,  False, All]} //
         TableForm[#, TableHeadings → strGovFlowEngfHeadings] &

Out[ ]//TableForm=
        ck     Jg/Jnz     P1st/Pin     Fgv          Fnz          Pbowl/Pin     Vol flow       Pgv         Pl
                                       (lb/sec)     (lb/sec)                   (ft^3/sec)      (psia)      (
        0      0.025      0.714286     18.5324      18.5324      0.714535      4.71569        3500.       2
        1      0.025      0.457143     23.2681      23.2681      0.457757      9.24195        3500.       1
        0      0.5        0.457143     430.898      430.898      0.614186      127.559        3500.       2
        2      1.8        0.457143     828.335      828.335      0.889989      169.223        3500.       3
        2      5.         0.457143     914.55       914.55       0.98262       169.223        3500.       3
        2      10.        0.457143     926.575      926.575      0.99554       169.223        3500.       3
```

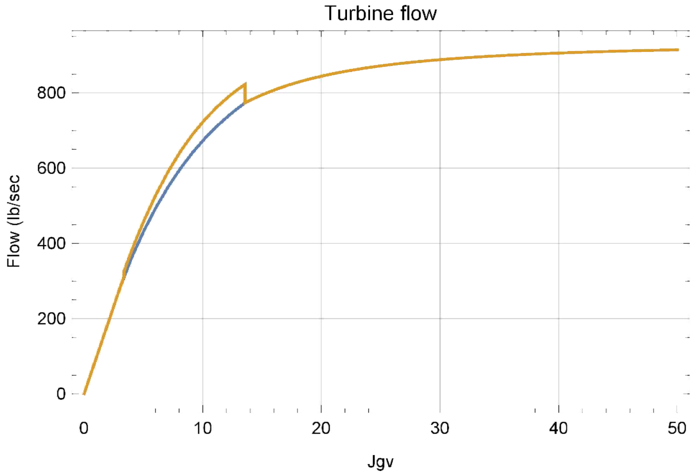
```
In[ ]:= {calcGovFlowEng[3500., 5.5, 0.25, 10., 2500., True, All]
          , calcGovFlowEng[3500., 5.5, 0.25, 10., 1600., True, All]
          , calcGovFlowEng[3500., 5.5, 5.00, 10., 1600., True, All]
          , calcGovFlowEng[3500., 5.5, 18.0, 10., 1600., True, All]
          , calcGovFlowEng[3500., 5.5, 50.0, 10., 1600., True, All]
          , calcGovFlowEng[3500., 5.5, 100., 10., 1600., True, All]} //
         TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
```

```
Out[ ]//TableForm=
ck    Jg/Jnz    P1st/Pin    Fgv          Fnz          Pbowl/Pin    Vol flow      Pgv       Pb
                            (lb/sec)     (lb/sec)                  (ft^3/sec)    (psia)    (p
0     0.025     0.714286    18.5347      18.5347      0.714464     4.71675       3500.     25
1     0.025     0.457143    23.2681      23.2681      0.457757     9.24195       3500.     16
0     0.5       0.457143    457.165      457.165      0.565714     146.931       3500.     19
2     1.8       0.457143    828.335      828.335      0.889989     169.223       3500.     3
2     5.        0.457143    914.55       914.55       0.98262      169.223       3500.     34
2     10.       0.457143    926.575      926.575      0.99554      169.223       3500.     34
```

The gold line is with the original Dynsim-P equations. The blue line is with the corrected equations. Note the significant discontinuity of the gold line.

```
In[ ]:= Plot[ {calcGovFlowEng[3500., 5.5, jGV, 10., 1600., False, All][[4]],
          calcGovFlowEng[3500., 5.5, jGV, 10., 1600., True, All][[4]]}
        , {jGV, 0.01, 50}
        , PlotLabel → Style["Turbine flow"]
        , PlotRange → All
        , Frame → True, FrameLabel → {Style["Jgv"], Style["Flow (lb/sec"]}
        , GridLines → Automatic
       ]
```

Out[ ]=



some simple tests with the list interface

```
In[ ]:= pData00 = {3500., 5.5, 0.25, 10., 2500., False};
        pData01 = {3500., 5.5, 0.25, 10., 1600., False};
        pData02 = {3500., 5.5, 5.00, 10., 1600., False};
        pData03 = {3500., 5.5, 18.0, 10., 1600., False};
        {calcGovFlowEng[pData00]
          , calcGovFlowEng[pData01]
          , calcGovFlowEng[pData02]
          , calcGovFlowEng[pData03]
         } // TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
```

```
Out[ ]//TableForm=
        ck    Jg/Jnz    P1st/Pin    Fgv         Fnz         Pbowl/Pin    Vol flow      Pgv        Pl
                                    (lb/sec)    (lb/sec)                 (ft^3/sec)     (psia)    (|
        0     0.025     0.714286    18.5324     18.5324     0.714535     4.71569       3500.      2!
        1     0.025     0.457143    23.2681     23.2681     0.457757     9.24195       3500.      1(
        0     0.5       0.457143    430.898     430.898     0.614186     127.559       3500.      2:
        2     1.8       0.457143    828.335     828.335     0.889989     169.223       3500.      3:
```

```
In[ ]:= {calcGovFlowEng[pData00, All]
          , calcGovFlowEng[pData01, All]
          , calcGovFlowEng[pData02, All]
          , calcGovFlowEng[pData03, All]
         } // TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
```

```
Out[ ]//TableForm=
        ck    Jg/Jnz    P1st/Pin    Fgv         Fnz         Pbowl/Pin    Vol flow      Pgv        Pl
                                    (lb/sec)    (lb/sec)                 (ft^3/sec)     (psia)    (|
        0     0.025     0.714286    18.5324     18.5324     0.714535     4.71569       3500.      2!
        1     0.025     0.457143    23.2681     23.2681     0.457757     9.24195       3500.      1(
        0     0.5       0.457143    430.898     430.898     0.614186     127.559       3500.      2:
        2     1.8       0.457143    828.335     828.335     0.889989     169.223       3500.      3:
```

Comparison with Dynsim

```
In[ ]:= dynPoints01 = { pIn → 3502.61, rhoIn → 4.84729,
          jGvIn → 55.5792, jNozIn → 7.72594 × 1.06445, p1stIn → 2554.31}
        dynPoints02 = { pIn → 3502.61, rhoIn → 4.84729, jGvIn → 3.57696,
          jNozIn → 7.72594 × 1.06445, p1stIn → 2554.31}

Out[ ]= {pIn → 3502.61, rhoIn → 4.84729, jGvIn → 55.5792, jNozIn → 8.22388, p1stIn → 2554.31}

Out[ ]= {pIn → 3502.61, rhoIn → 4.84729, jGvIn → 3.57696, jNozIn → 8.22388, p1stIn → 2554.31}
```

```
In[ ]:= perf01 = {pIn, rhoIn, jGvIn, jNozIn, p1stIn} /. dynPoints01;
        perf02 = {pIn, rhoIn, jGvIn, jNozIn, p1stIn} /. dynPoints02;
        r01f = calcGovFlowEng[Append[perf01, False]];
        r02f = calcGovFlowEng[Append[perf02, False]];
        r01t = calcGovFlowEng[Append[perf01, True]];
        r02t = calcGovFlowEng[Append[perf02, True]];
        rall = {r01f, r01t, r02f, r02t};
        rall // TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
```

```
Out[ ]//TableForm=
        ck     Jg/Jnz      P1st/Pin    Fgv         Fnz         Pbowl/Pin    Vol flow     Pgv
                                       (lb/sec)    (lb/sec)                 (ft^3/sec)    (psia)
        0      6.75827     0.729259    550.005     550.005     0.994232     114.125      3502.61
        0      6.75827     0.729259    551.564     551.564     0.994199     114.452      3502.61
        0      0.434948    0.729259    217.616     217.616     0.781998     57.4097      3502.61
        0      0.434948    0.729259    222.389     222.389     0.77233      59.4034      3502.61
```

```
In[ ]:=
```

Nozzle Exit conditions

Operating Data for Bowen Unit 1

Valve Trim and Nozzle Scale Factor for Bowen Unit 3

Below are the valve trim curves in the model today (2016 November)

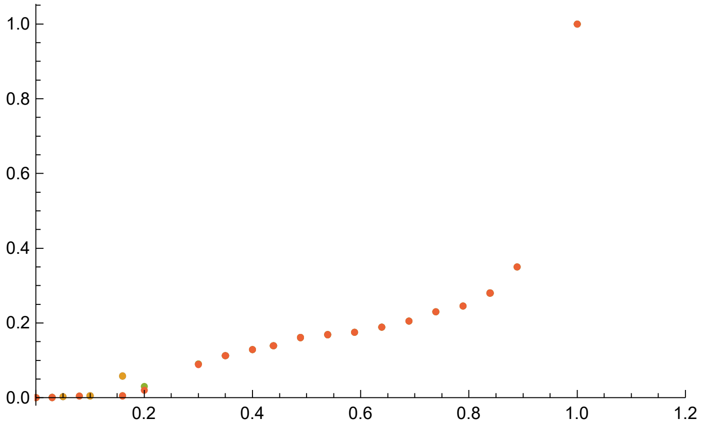
Trim data prep (from model)

```
In[ ]:= tabVCV4a = {{-1, 0}, {0, 0.0}, {0.03, 0.0006}, {0.08, 0.0045}, {0.16, 0.0050}
           , {0.2, 0.02}, {0.3, 0.089}, {0.35, 0.1126}, {0.4, 0.129}
           , {0.4388, 0.139}, {0.4888, 0.161}, {0.5388, 0.1687}
           , {0.5888, 0.175}, {0.6388, 0.189}, {0.689, 0.205}, {0.739, 0.23}
           , {0.789, 0.2456}, {0.839, 0.28}, {0.889, 0.35}, {1.0, 1.0}, {2, 1.0} };
        tabVCV3a = {{-1, 0}, {0, 0.0}, {0.03, 0.0008}, {0.08, 0.0045}, {0.16, 0.0050}
           , {0.2, 0.03}, {0.3, 0.09}, {0.35, 0.1126}, {0.4, 0.129}
           , {0.4388, 0.139}, {0.4888, 0.161}, {0.5388, 0.1687}
           , {0.5888, 0.175}, {0.6388, 0.189}, {0.689, 0.205}, {0.739, 0.23}
           , {0.789, 0.2456}, {0.839, 0.28}, {0.889, 0.35}, {1.0, 1.0}, {2, 1.0} };
        tabVCV2a = {{-1, 0}, {0, 0.0}, {0.03, 0.0007}, {0.05, 0.003}, {0.1, 0.0050}
           , {0.16, 0.058}, {0.3, 0.09}, {0.35, 0.1126}, {0.4, 0.129}
           , {0.4388, 0.139}, {0.4888, 0.161}, {0.5388, 0.1687}
           , {0.5888, 0.175}, {0.6388, 0.189}, {0.689, 0.205}, {0.739, 0.23}
           , {0.789, 0.2456}, {0.839, 0.28}, {0.889, 0.35}, {1.0, 1.0}, {2, 1.0} };
        tabVCV1a = {{-1, 0}, {0, 0.0}, {0.03, 0.0007}, {0.05, 0.003}, {0.1, 0.0050}
           , {0.16, 0.058}, {0.3, 0.09}, {0.35, 0.1126}, {0.4, 0.129}
           , {0.4388, 0.139}, {0.4888, 0.161}, {0.5388, 0.1687}
           , {0.5888, 0.175}, {0.6388, 0.189}, {0.689, 0.205}, {0.739, 0.23}
           , {0.789, 0.2456}, {0.839, 0.28}, {0.889, 0.35}, {1.0, 1.0} , {2, 1.0}};
        Transpose[{tabVCV1a, tabVCV2a , tabVCV3a, tabVCV4a}] // TableForm
        ListPlot[{tabVCV1a, tabVCV2a , tabVCV3a, tabVCV4a}, PlotRange → {{0, 1.2}, All}]
```

Out[ ]//TableForm=

```
-1       -1       -1       -1
0        0        0        0
0        0        0        0
0.       0.       0.       0.
0.03     0.03     0.03     0.03
0.0007   0.0007   0.0008   0.0006
0.05     0.05     0.08     0.08
0.003    0.003    0.0045   0.0045
0.1      0.1      0.16     0.16
0.005    0.005    0.005    0.005
0.16     0.16     0.2      0.2
0.058    0.058    0.03     0.02
0.3      0.3      0.3      0.3
0.09     0.09     0.09     0.089
0.35     0.35     0.35     0.35
0.1126   0.1126   0.1126   0.1126
0.4      0.4      0.4      0.4
0.129    0.129    0.129    0.129
0.4388   0.4388   0.4388   0.4388
0.139    0.139    0.139    0.139
0.4888   0.4888   0.4888   0.4888
0.161    0.161    0.161    0.161
0.5388   0.5388   0.5388   0.5388
0.1687   0.1687   0.1687   0.1687
0.5888   0.5888   0.5888   0.5888
0.175    0.175    0.175    0.175
0.6388   0.6388   0.6388   0.6388
0.189    0.189    0.189    0.189
0.689    0.689    0.689    0.689
0.205    0.205    0.205    0.205
0.739    0.739    0.739    0.739
0.23     0.23     0.23     0.23
0.789    0.789    0.789    0.789
0.2456   0.2456   0.2456   0.2456
0.839    0.839    0.839    0.839
0.28     0.28     0.28     0.28
0.889    0.889    0.889    0.889
0.35     0.35     0.35     0.35
1.       1.       1.       1.
1.       1.       1.       1.
2        2        2        2
1.       1.       1.       1.
```

Out[=]=

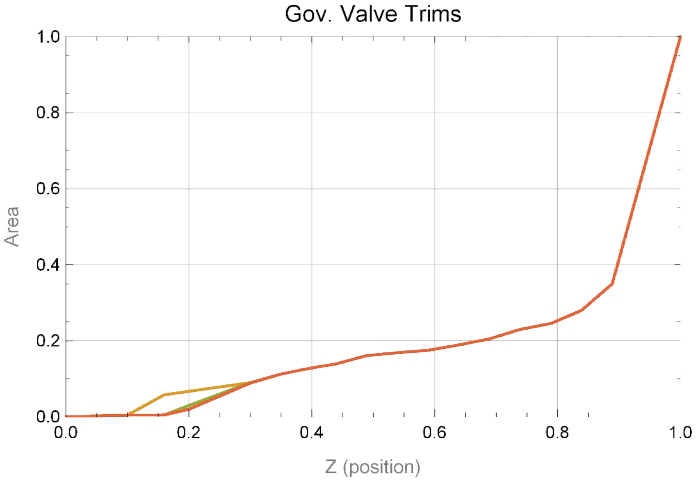


Valve Trim Interpolating function

In[]:=

```
aVCV4afunc = Interpolation[tabVCV4a, InterpolationOrder → 1];
aVCV3afunc = Interpolation[tabVCV3a, InterpolationOrder → 1];
aVCV2afunc = Interpolation[tabVCV2a, InterpolationOrder → 1];
aVCV1afunc = Interpolation[tabVCV1a, InterpolationOrder → 1];

aVCV5afunc = Interpolation[tabVCV4a, InterpolationOrder → 1];
aVCV6afunc = Interpolation[tabVCV3a, InterpolationOrder → 1];
aVCV7afunc = Interpolation[tabVCV2a, InterpolationOrder → 1];
aVCV8afunc = Interpolation[tabVCV1a, InterpolationOrder → 1];

pArea = Plot[{aVCV1afunc[z], aVCV2afunc[z], aVCV3afunc[z], aVCV4afunc[z]}, {z, 0, 1}
  , PlotRange → {{0, 1}, {0, 1}}
  , Frame → True, GridLines → Automatic
  , PlotLabel → Style["Gov. Valve Trims", Black, 12]
  , FrameLabel → {Style["Z (position)", Gray, 10], Style["Area", Gray, 10]}
 ]
```

Out[=]=

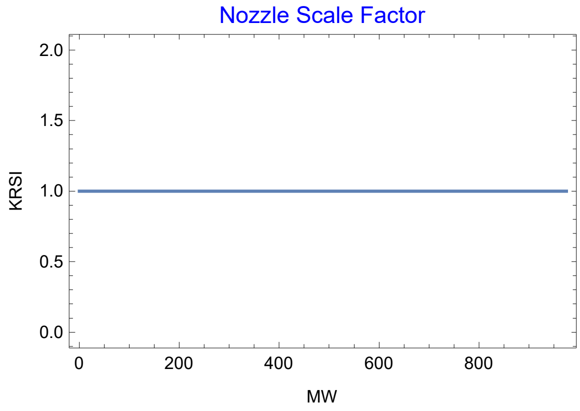


Nozzle scale factor

A scale factor of the nozzle flow coefficient was used for Bowen Unit 3, it may be required for other projects as well.
To start we keep the scale factors constant.

```
In[ ]:= knzScaleX = {0., 500., 700., 935.};
knzScaleY = {1.0, 1.0, 1.0, 1.0};
knzScalefunc =
  Interpolation[Transpose[{knzScaleX, knzScaleY}], InterpolationOrder → 1];
pKNzfunc = Plot[knzScalefunc[mw], {mw, 0, 975}, Frame → True,
  FrameLabel → {Style["MW"], Style["KRSI"]},
  PlotLabel → Style["Nozzle Scale Factor", 12, Blue], ImageSize → {300, 300}]
```

Out[ ]=



Net Flow calculation for Bowen Unit 1

System Performance functions, definition

We first write a function that builds on the calcGovFlowEng function developed earlier. Our new function will to call calcGovFlowEng for each of the governor valves in the system of interest. This function compute the flow rates and bowl pressure given operating data and 3 flow coefficients.

Performance function for parallel governor nozzle systems

Note: Bowen uses FlowEngDyn, because it is preDynsim 5.3.1

```
In[ ]:= Remove[calcBowenGovNozEng];
calcBowenGovNozEng[dataPerf_List, ksvIn_Real, kgvIn_Real, knzIn_Real,
  iDynErr_, gvTrim_List, iknzs_Integer : 1, rptList_: All] := Module[{
```

```
 a1, a2, a3, a4, a5, a6, a7, a8
 , knzScaleA, nstopA, dpStopA, fDataA,  rDataA
 , knzScaleB, nstopB, dpStopB, fDataB,  rDataB
 , p1, p2, p3, p4, p5, p6, p7, p8
 , r1, r2, r3, r4, r5, r6, r7, r8
 , f1, f2, f3, f4, f5, f6, f7, f8
 , ftot},
(*lookup area for each valve position*)
a1 = gvTrim[[1]]@(dataPerf[[iZ1]] / 100);
a2 = gvTrim[[2]]@(dataPerf[[iZ2]] / 100);
a3 = gvTrim[[3]]@(dataPerf[[iZ3]] / 100);
a4 = gvTrim[[4]]@(dataPerf[[iZ4]] / 100);
a5 = gvTrim[[5]]@(dataPerf[[iZ5]] / 100);
a6 = gvTrim[[6]]@(dataPerf[[iZ6]] / 100);
a7 = gvTrim[[7]]@(dataPerf[[iZ7]] / 100);
a8 = gvTrim[[8]]@(dataPerf[[iZ8]] / 100);

(*----------------------------*)
(*steam chest A*)
nstopA = Max[1., Total@(If[# > 0.001, 1., 0.] & /@ {a1, a2, a3, a4})];
(*TO DO: make this a loop*)
(*compute DP across the stop valve, assuming measured steam flow*)
fDataA = dataPerf[[iFlow]] 1000. / 3600.;
(*convert from klb/hr to lb/sec*)
rDataA = dataPerf[[iRth]];
dpStopA = (fDataA / (nstopA ksvIn))^2 / rDataA;
(*Bowen Unit 3.s4m file added this correction --
 should change knzScalefunc to a function of flow rate *)
 knzScaleA = If[ iknzs < 1, 1, knzScalefunc@dataPerf[[iMW]] ];
(*prepare data for each governor valve*)
p1 = {(dataPerf[[iPth]] - dpStopA)
   , dataPerf[[iRth]]
   , kgvIn a1
   , knzIn knzScaleA
   , dataPerf[[iPfs]]
   , iDynErr};
p2 = p1;
p3 = p1;
p4 = p1;
p2[[3]] = kgvIn a2;
p3[[3]] = kgvIn a3;
p4[[3]] = kgvIn a4;
(*compute results for each governor valve*)
r1 = calcGovFlowEng[p1];
r2 = calcGovFlowEng[p2];
r3 = calcGovFlowEng[p3];
```

```
  r4 = calcGovFlowEng[p4];
  (*-----------------------------*)
  (*steam chest B *)
  nstopB = Max[1., Total@(If[# > 0.001, 1., 0.] & /@ {a5, a6, a7, a8})];
  (*TO DO: make this a loop*)
  (*compute DP across the stop valve, assuming measured steam flow*)
  fDataB = dataPerf[[iFlow]] 1000. / 3600.;
  (*convert from klb/hr to lb/sec*)
  rDataB = dataPerf[[iRth]];
  dpStopB = (fDataB / (nstopB ksvIn) ) ^2 / rDataB;
  (*Bowen Unit 3.s4m file added this correction --
   should change knzScalefunc to a function of flow rate *)
   knzScaleB = If[ iknzs < 1, 1, knzScalefunc@dataPerf[[iMW]] ];
  (*prepare data for each governor valve*)
  p5  = {(dataPerf[[iPth]] - dpStopB)
    , dataPerf[[iRth]]
    , kgvIn a5
    , knzIn knzScaleB
    , dataPerf[[iPfs]]
    , iDynErr} ;
  p6 = p5;
  p7 = p5;
  p8 = p5;
  p6[[3]] = kgvIn a6;
  p7[[3]] = kgvIn a7;
  p8[[3]] = kgvIn a8;
  (*compute results for each governor valve*)
  r5 = calcGovFlowEng[p5];
  r6 = calcGovFlowEng[p6];
  r7 = calcGovFlowEng[p7];
  r8 = calcGovFlowEng[p8];
  (*---------------------------------------*)
  (*summarize results, with a row for each pair of governor valves and nozzles*)
  {r1, r2, r3, r4, r5, r6, r7, r8}
 ]

(*modParms = {ksv→ (61.2), kgv→ 55.58, knz→ (7.72594)};*)
gvFuncs00 = {aVCV1afunc, aVCV2afunc, aVCV3afunc, aVCV4afunc, aVCV5afunc,
   aVCV6afunc, aVCV7afunc, aVCV8afunc}; (*original gv funcs in model*)
calcBowenGovNozEng[hptData1[[3]] , 61.2, 55.58, 7.72594, True, gvFuncs00 ] //
 TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
calcBowenGovNozEng[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 1] //
 TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
calcBowenGovNozEng[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 0] //
 TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
```

Out[ ]//TableForm=

```
ck   Jg/Jnz    P1st/Pin    Fgv        Fnz        Pbowl/Pin   Vol flow     Pgv
                           (lb/sec)   (lb/sec)               (ft^3/sec)    (psia)
---------------------------------------------------------------------------------
2    6.24541   0.0197694   175.971    175.971    0.988722    137.165      908.858
2    6.98729   0.0197694   176.368    176.368    0.990949    137.165      908.858
2    6.76171   0.0197694   176.261    176.261    0.990347    137.165      908.858
2    7.03035   0.0197694   176.387    176.387    0.991058    137.165      908.858
2    6.90474   0.0197694   176.33     176.33     0.990735    137.165      908.858
2    6.96      0.0197694   176.355    176.355    0.990879    137.165      908.858
2    6.92805   0.0197694   176.341    176.341    0.990796    137.165      908.858
2    7.0634    0.0197694   176.402    176.402    0.99114     137.165      908.858
```

Out[ ]//TableForm=

```
ck   Jg/Jnz    P1st/Pin    Fgv        Fnz        Pbowl/Pin   Vol flow     Pgv
                           (lb/sec)   (lb/sec)               (ft^3/sec)    (psia)
---------------------------------------------------------------------------------
2    6.24541   0.0197694   175.971    175.971    0.988722    137.165      908.858
2    6.98729   0.0197694   176.368    176.368    0.990949    137.165      908.858
2    6.76171   0.0197694   176.261    176.261    0.990347    137.165      908.858
2    7.03035   0.0197694   176.387    176.387    0.991058    137.165      908.858
2    6.90474   0.0197694   176.33     176.33     0.990735    137.165      908.858
2    6.96      0.0197694   176.355    176.355    0.990879    137.165      908.858
2    6.92805   0.0197694   176.341    176.341    0.990796    137.165      908.858
2    7.0634    0.0197694   176.402    176.402    0.99114     137.165      908.858
```

Out[ ]//TableForm=

```
ck   Jg/Jnz    P1st/Pin    Fgv        Fnz        Pbowl/Pin   Vol flow     Pgv
                           (lb/sec)   (lb/sec)               (ft^3/sec)    (psia)
---------------------------------------------------------------------------------
2    6.24541   0.0197694   175.971    175.971    0.988722    137.165      908.858
2    6.98729   0.0197694   176.368    176.368    0.990949    137.165      908.858
2    6.76171   0.0197694   176.261    176.261    0.990347    137.165      908.858
2    7.03035   0.0197694   176.387    176.387    0.991058    137.165      908.858
2    6.90474   0.0197694   176.33     176.33     0.990735    137.165      908.858
2    6.96      0.0197694   176.355    176.355    0.990879    137.165      908.858
2    6.92805   0.0197694   176.341    176.341    0.990796    137.165      908.858
2    7.0634    0.0197694   176.402    176.402    0.99114     137.165      908.858
```

In[ ]:=

```
iBowenChokeStatus = 1;
iBowenJgJnz = 2;
iBowenPfsPin = 3;
iBowenFgv = 4;
iBowenFnz = 5;
iBowenPbowlPin = 6;
iBowenVolFlow = 7;
iBowenPgv = 8;
iBowenPbowl = 9;
iBowenPfs = 10;
```

Functions for Manipulate analysis

Summary functions

```
In[∘]:= Remove[calcBowenGovNozEngFlowTotal]
        calcBowenGovNozEngFlowTotal[dataPerf_List, ksvIn_Real,
          kgvIn_Real, knzIn_Real, iDynErr_, gvTrim_List, iknzs_Integer : 1] :=
         Module[
          {rAll, flowList},
          rAll = calcBowenGovNozEng[dataPerf, ksvIn, kgvIn, knzIn, iDynErr, gvTrim, iknzs];
          flowList = #[[4]] & /@ rAll;
          Total[flowList]
         ]
```

```
In[∘]:= rTest = calcBowenGovNozEng[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 1]
        rTest // TableForm
        #[[4]] & /@ rTest

Out[∘]= {{2, 6.24541, 0.0197694, 175.971, 175.971, 0.988722, 137.165, 908.858, 898.607, 17.9676},
         {2, 6.98729, 0.0197694, 176.368, 176.368, 0.990949, 137.165, 908.858, 900.631, 17.9676},
         {2, 6.76171, 0.0197694, 176.261, 176.261, 0.990347, 137.165, 908.858, 900.084, 17.9676},
         {2, 7.03035, 0.0197694, 176.387, 176.387, 0.991058, 137.165, 908.858, 900.73, 17.9676},
         {2, 6.90474, 0.0197694, 176.33, 176.33, 0.990735, 137.165, 908.858, 900.437, 17.9676},
         {2, 6.96, 0.0197694, 176.355, 176.355, 0.990879, 137.165, 908.858, 900.568, 17.9676},
         {2, 6.92805, 0.0197694, 176.341, 176.341, 0.990796, 137.165, 908.858, 900.493, 17.9676},
         {2, 7.0634, 0.0197694, 176.402, 176.402, 0.99114, 137.165, 908.858, 900.805, 17.9676}}

Out[∘]//TableForm=
        2    6.24541    0.0197694    175.971    175.971    0.988722    137.165    908.858    898.60
        2    6.98729    0.0197694    176.368    176.368    0.990949    137.165    908.858    900.63
        2    6.76171    0.0197694    176.261    176.261    0.990347    137.165    908.858    900.08
        2    7.03035    0.0197694    176.387    176.387    0.991058    137.165    908.858    900.73
        2    6.90474    0.0197694    176.33     176.33     0.990735    137.165    908.858    900.43
        2    6.96       0.0197694    176.355    176.355    0.990879    137.165    908.858    900.56
        2    6.92805    0.0197694    176.341    176.341    0.990796    137.165    908.858    900.49
        2    7.0634     0.0197694    176.402    176.402    0.99114     137.165    908.858    900.80

Out[∘]= {175.971, 176.368, 176.261, 176.387, 176.33, 176.355, 176.341, 176.402}
```

```
In[∘]:= calcBowenGovNozEngFlowTotal[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 1]

Out[∘]= 1410.41
```

```
In[∘]:= {calcBowenGovNozEngFlowTotal[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 1]
         , calcBowenGovNozEngFlowTotal[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00, 0]
         , calcBowenGovNozEngFlowTotal[hptData1[[3]] , 61.2, 55.58, 7.72594 ,  True, gvFuncs00]
         , calcBowenGovNozEngFlowTotal[hptData1[[3]] , 61.2, 55.58, 7.72594 , True, gvFuncs00]}


Out[∘]= {1410.41, 1410.41, 1410.41, 1410.41}
```

MoreTesting

```
In[]:= gvPointsTest =
          {{-2, 0}, {0.0005, 0.022}, {0.0315, 0.012}, {0.0625, 0.01}, {0.0955, 0.007},
           {0.142, 0.018}, {0.171, 0.029}, {0.2075, 0.049}, {0.2365, 0.065}, {0.259, 0.09},
           {0.278, 0.113}, {0.3005, 0.141}, {0.319, 0.096}, {0.338, 0.082},
           {0.36075, 0.0852}, {0.39, 0.102}, {0.421, 0.127}, {0.443, 0.129}, {0.4985, 0.21},
           {0.576, 0.227}, {0.689, 0.2015}, {0.839, 0.28}, {0.889, 0.35}, {1, 1}};
       gvFuncTest} = Interpolation[gvPointsTest, InterpolationOrder → 1];
    Plot[ gvFuncTest[z], {z, 0, 1}
     , Epilog → {Red, PointSize[0.02], Point[gvPointsTest]}
     , AxesLabel → {Style["Position", 10], Style["Area", 10]}]
```

```
In[]:= gvFuncListTest = {gvFuncTest, gvFuncTest, gvFuncTest,
       gvFuncTest, gvFuncTest, gvFuncTest, gvFuncTest, gvFuncTest};
    Length@ gvFuncListTest

Out[]= 8
```

```
In[]:= tTest = 37;
    hptData1[[tTest ;; tTest + 1]] // TableForm[#, TableHeadings → {None, strCol}] &
    rTest = calcBowenGovNozEng[hptData1[[tTest]] , 61.2, 47., 8.4, True, gvFuncs00, 0] ;
    rTest // TableForm[#, TableHeadings → strGovFlowEngfHeadings] &
    {hptData1[[tTest, 3]], 3.6 Total@rTest[[All, 5]],
     3.6 Total@rTest[[All, 5]] - hptData1[[tTest, 3]]}
```

```
Out[]//TableForm=
t (min)     MW          F (klb/hr)    T (F)       Pth         Pfs         Pcrh        Tcrh        :
36          1.08507     82.8891       833.954     906.552     58.8213     16.4713     408.71      :
37          1.0489      81.3636       834.52      908.118     57.6556     16.4713     408.71      :
```

```
Out[]//TableForm=
ck     Jg/Jnz        P1st/Pin      Fgv           Fnz           Pbowl/Pin     Vol flow        Pgv
                                   (lb/sec)      (lb/sec)                    (ft^3/sec)      (psia)
1      0.109035      0.0648852     20.7802       20.7802       0.112458      146.439         906.545
1      0.108609      0.0648852     20.6989       20.6989       0.112196      146.206         906.545
1      0.026487      0.0648852     5.04794       5.04794       0.0694321     57.6169         906.545
1      0.0264        0.0648852     5.03137       5.03137       0.0694041     57.451          906.545
1      0.026515      0.0648852     5.05328       5.05328       0.0694411     57.6704         906.545
1      0.0264713     0.0648852     5.04496       5.04496       0.069427      57.5871         906.545
1      0.10523       0.0648852     20.055        20.055        0.110131      144.314         906.545
1      0.101896      0.0648852     19.4195       19.4195       0.108105      142.361         906.545
```

```
Out[]= {82.8891, 364.072, 281.183}
```

Plotting functions using calculated results

```
In[]:=
    rgbPlant = RGBColor[0, 0.2, 0.8];
    rgbModel = RGBColor[1, 0.7, 0];
```

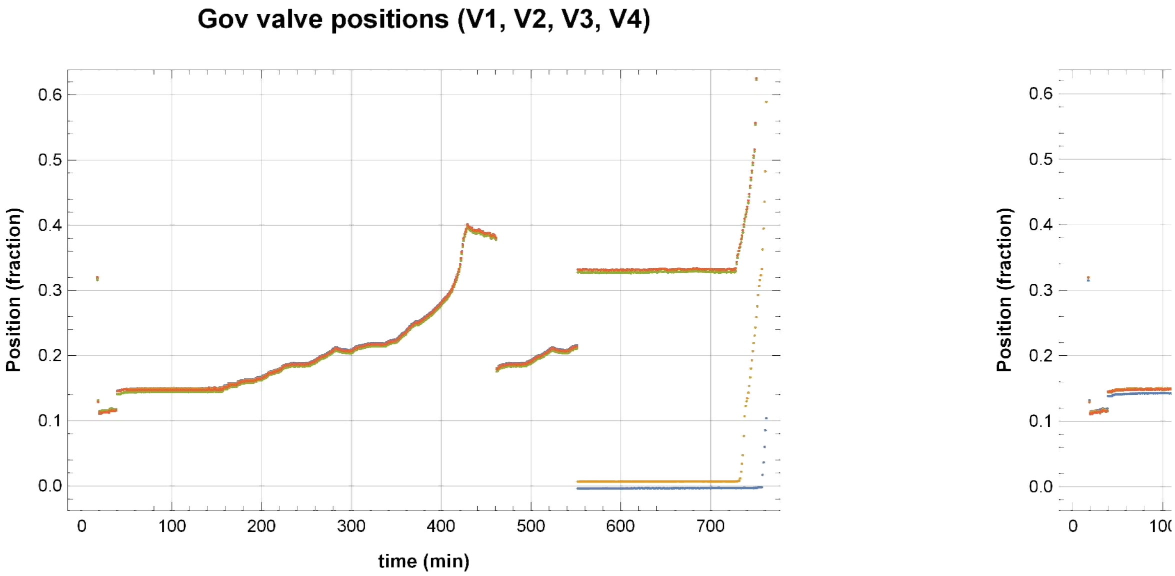
```
rgbGv1 = RGBColor[0.25, 0.25, 1];
rgbGv2 = RGBColor[0.25, 1, 1];
rgbGv3 = RGBColor[1, 0.5, 0.5];
rgbGv4 = RGBColor[0.5, 1, 0.5];

kpLabel = 14;
kfLabel = 10;
pImageSize = {300, 300};

plotGovValveTrim[fOrig_, fNew_, ptTable_, iSize_: pImageSize] :=
  Plot[{fOrig[x], fNew[x]}, {x, -0.025, 1.025}
   , PlotRange → {{-0.1, 1.1}, {-0.1, 1.1} }
   , Epilog → {Text[ptTable, {0.05, 0.9}, {-1, +1}]}
   , Frame → True
   , FrameLabel → {Style["Valve position (fraction)", kfLabel, Bold]
     , Style["Valve area (fraction)", kfLabel, Bold]}
   , PlotLabel → Style["Gov valve trim", kpLabel, Bold]
   , GridLines → Automatic
   , ImageSize → iSize
   , ImageMargins → 0];

plotGovValveTrim2Trims[funcs_List, ptTable_, iSize_: pImageSize] :=
  Plot[Through[funcs [x]] , {x, -0.025, 1.025}
   , PlotRange → {{-0.1, 1.1}, {-0.1, 1.1} }
   , Epilog → {Text[ptTable, {0.05, 0.9}, {-1, +1}]}
   , Frame → True
   , FrameLabel → {Style["Valve position (fraction)", kfLabel, Bold]
     , Style["Valve area (fraction)", kfLabel, Bold]}
   , PlotLabel → Style["Gov valve trim", kpLabel, Bold]
   , GridLines → Automatic
   , ImageSize → iSize
   , ImageMargins → 0];

plotBowenPlantModelFlow[dataPlant_List, dataModel_List] :=
 ListPlot[{dataPlant, dataModel}
  , PlotRange → {All, {2500, 7500}}
  , PlotStyle → {{rgbPlant}, {rgbModel}}
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Flow (klb/hr)", kfLabel, Bold]}
  , PlotLabel → Style["Flow to HP Turbine", kpLabel, Bold]
  , ImageSize → {300, 300}
 ]

plotBowenPlantModelFlowDiff[dataDiff_List] :=
 ListPlot[{dataDiff}
```

```
  , PlotRange → {All, {-4000, 4000}}
  , PlotStyle → {rgbModel}
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Flow Diff (klb/hr)", kfLabel, Bold]}
  , PlotLabel → Style["Flow Difference (Model - Data)", kpLabel, Bold]
  , ImageSize → {300, 300}
 ]

plotBowenModelPR[dataModel_List] :=
 ListPlot[{dataModel}
  , PlotRange → {All, {0, 1}}
  , PlotStyle → {{rgbModel}}
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Pressure Ratio ", kfLabel, Bold]}
  , PlotLabel → Style["Pressure Ratio: Psv / P1st", kpLabel, Bold]
  , ImageSize → {300, 400}
 ]

plotBowenModelK[dataModel_List] :=
 ListPlot[{dataModel}
  , PlotRange → {All, All}
  , PlotStyle → {{rgbModel}}
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["$\frac{lb / sec}{\sqrt{psi\ \frac{lb}{ft^3}}}$ ", kfLabel, Bold]}
  , PlotLabel → Style["HP Turbine J", kpLabel, Bold]
  , ImageSize → {300, 400}
 ]
plotBowenModelChokeStatus[dataModel_List, timeRange_: All] :=
 ListPlot[dataModel
  , PlotRange → {timeRange, {-1, 4}}
  , PlotStyle → {{rgbGv1}
    , {rgbGv2}
    , {rgbGv3}
    , {rgbGv4}
   }
  , AxesOrigin → {Automatic, -1}
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Choke status", kfLabel, Bold]}
  , PlotLabel → Style["Governor Choke status", kpLabel, Bold]
  ,
  Epilog → {Black, Text[Style["Choke Status: 0 Not choked, 1 GV, 2 Noz, 3 GV & Noz", 9],
     Scaled[{0.02, 3.5 / 4.}], {-1, 0} ]
    , rgbGv1, Line[{Scaled[{0.05, 3.2 / 4.}], Scaled[{0.10, 3.2 / 4}]}],
    Text[Style["GV 1", 9, rgbGv1], Scaled[{0.12, 3.2 / 4.}], {-1, 0}]
    , rgbGv2, Line[{Scaled[{0.25, 3.2 / 4.}], Scaled[{0.30, 3.2 / 4}]}],
    Text[Style["GV 2", 9, rgbGv2], Scaled[{0.32, 3.2 / 4.}], {-1, 0}]
    , rgbGv3, Line[{Scaled[{0.45, 3.2 / 4.}], Scaled[{0.50, 3.2 / 4}]}],
    Text[Style["GV 3", 9, rgbGv3], Scaled[{0.52, 3.2 / 4.}], {-1, 0}]
    , rgbGv4, Line[{Scaled[{0.65, 3.2 / 4.}], Scaled[{0.70, 3.2 / 4}]}],
    Text[Style["GV 4", 9, rgbGv4], Scaled[{0.72, 3.2 / 4.}], {-1, 0}]
   }
  , ImageSize → {300, 300}
 ]

plotBowenModelFlowGVi[dataModel_List, timeRange_: All] :=
 ListPlot[dataModel
  , PlotRange → {timeRange, {0, 3500}}
  , PlotStyle → {{rgbGv1}
    , {rgbGv2}
    , {rgbGv3}
    , {rgbGv4}
   }
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Flow (klb/hr)", kfLabel, Bold]}
  , PlotLabel → Style["Flow to each GV", kpLabel, Bold]
  , Epilog → {
    , rgbGv1, Line[{Scaled[{0.05, 3.5 / 4.}], Scaled[{0.10, 3.5 / 4}]}],
    Text[Style["GV 1", 9, rgbGv1], Scaled[{0.12, 3.5 / 4.}], {-1, 0}]
    , rgbGv2, Line[{Scaled[{0.25, 3.5 / 4.}], Scaled[{0.30, 3.5 / 4}]}],
    Text[Style["GV 2", 9, rgbGv2], Scaled[{0.32, 3.5 / 4.}], {-1, 0}]
    , rgbGv3, Line[{Scaled[{0.45, 3.5 / 4.}], Scaled[{0.50, 3.5 / 4}]}],
    Text[Style["GV 3", 9, rgbGv3], Scaled[{0.52, 3.5 / 4.}], {-1, 0}]
    , rgbGv4, Line[{Scaled[{0.65, 3.5 / 4.}], Scaled[{0.70, 3.5 / 4}]}],
    Text[Style["GV 4", 9, rgbGv4], Scaled[{0.72, 3.5 / 4.}], {-1, 0}]
   }
  , ImageSize → {300, 300}
 ]

plotBowenModelFlowJRi[dataModel_List, timeRange_: All] :=
 ListPlot[dataModel
  , PlotRange → {timeRange, {0, 10}}
  , PlotStyle → {{rgbGv1}
    , {rgbGv2}
    , {rgbGv3}
    , {rgbGv4}
   }
  , Frame → True
  , FrameLabel → {Style["time (min)", kfLabel, Bold]
    , Style["Ratio (-)", kfLabel, Bold]}
  , PlotLabel → Style["Flow Coefficient Ratio: Jgv / Jnoz", kpLabel, Bold]
  , Epilog → {
    rgbGv1, Line[{Scaled[{0.05, 3.5 / 4.}], Scaled[{0.10, 3.5 / 4}]}]
    , Text[Style["GV 1", 9, rgbGv1], Scaled[{0.12, 3.5 / 4.}], {-1, 0}]
    , rgbGv2, Line[{Scaled[{0.25, 3.5 / 4.}], Scaled[{0.30, 3.5 / 4}]}]
    , Text[Style["GV 2", 9, rgbGv2], Scaled[{0.32, 3.5 / 4.}], {-1, 0}]
    , rgbGv3, Line[{Scaled[{0.45, 3.5 / 4.}], Scaled[{0.50, 3.5 / 4}]}]
    , Text[Style["GV 3", 9, rgbGv3], Scaled[{0.52, 3.5 / 4.}], {-1, 0}]
    , rgbGv4, Line[{Scaled[{0.65, 3.5 / 4.}], Scaled[{0.70, 3.5 / 4}]}]
    , Text[Style["GV 4", 9, rgbGv4], Scaled[{0.72, 3.5 / 4.}], {-1, 0}]
   }
  , ImageSize → {300, 300}
 ]
```

Plotting function using historian results

```
In[ ]:= ptszAllData1 =
          Transpose@{hptData1[[All, 1]], hptData1[[All, #]] / 100.} & /@ {iZ1, iZ2, iZ3, iZ4};
        pzAllData1 = ListPlot[ptszAllData1
            , Frame → True
            , FrameLabel → {Style["time (min)", kfLabel, Bold]
              , Style["Position (fraction)", kfLabel, Bold]}
            , PlotLabel → Style["Gov valve positions", kpLabel, Bold]
            , GridLines → Automatic
          ];

        ptszGVDataA =
          Transpose@{hptData1[[All, 1]], hptData1[[All, #]] / 100.} & /@ {iZ1, iZ2, iZ3, iZ4};
        pzGVDataA = ListPlot[ptszGVDataA
            , Frame → True
            , FrameLabel → {Style["time (min)", kfLabel, Bold]
              , Style["Position (fraction)", kfLabel, Bold]}
            , PlotLabel → Style["Gov valve positions (V1, V2, V3, V4)", kpLabel, Bold]
            , GridLines → Automatic
          ];

        ptszGVDataB =
          Transpose@{hptData1[[All, 1]], hptData1[[All, #]] / 100.} & /@ {iZ5, iZ6, iZ7, iZ8};
        pzGVDataB = ListPlot[ptszGVDataB
            , Frame → True
            , FrameLabel → {Style["time (min)", kfLabel, Bold]
              , Style["Position (fraction)", kfLabel, Bold]}
            , PlotLabel → Style["Gov valve positions (V5, V6, V7, V8)", kpLabel, Bold]
            , GridLines → Automatic
          ];
        GraphicsRow[{pzGVDataA, pzGVDataB}, ImageSize → {1000, 600} ]
```

Out[∘]=



System Performance functions, testing:

```
In[∘]:= vecTimeDataRaw = hptData1[[All, 1]];
        vecFlowDataRaw = hptData1[[All, iFlow]];
```

For comparison, retrieve the raw flow data na prepare it for plotting

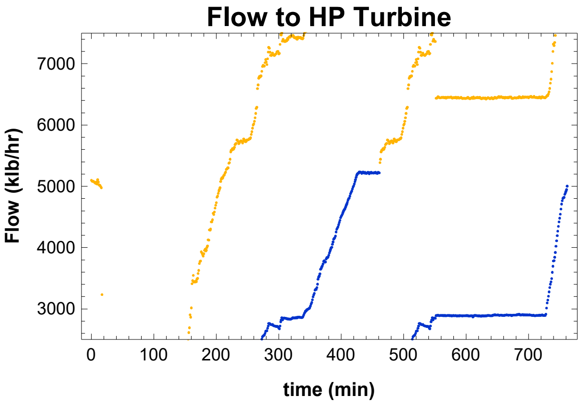
```
In[∘]:= vvFlowDataPlant = Transpose[{vecTimeDataRaw, vecFlowDataRaw}];
```

Use the new function and apply it to the operating data. Then prepare the data for pottling

In[ ]:=

```
gvFuncs00 = {aVCV1afunc, aVCV2afunc, aVCV3afunc, aVCV4afunc, aVCV1afunc,
   aVCV2afunc, aVCV3afunc, aVCV4afunc}; (*original gv funcs in model*)
vecFlowDataModel = calcBowenGovNozEngFlowTotal[#, 61.2,
     55.58, 7.72594, True, gvFuncs00] & /@ hptData1;
vvFlowDataModel = Transpose[{vecTimeDataRaw, vecFlowDataModel 3600 / 1000 }];
plotBowenPlantModelFlow[ vvFlowDataPlant, vvFlowDataModel]
```

Out[ ]=



Energy calculation for Bowen Unit 3

System Performance with Manipulate

Code

Pre-processing

```
In[ ]:= vecHth = i2stmHatPT[#[[1]], #[[2]] ] & /@
          Transpose@{Take[ hptData1[[All, iPth]], 20], Take[ hptData1[[All, iTth]], 20]}
        vecSth = i2stmSatPT[#[[1]], #[[2]] ] & /@
          Transpose@{Take[ hptData1[[All, iPth]], 20], Take[ hptData1[[All, iTth]], 20]}
        vecVth = i2stmVatPT[#[[1]], #[[2]] ] & /@
          Transpose@{Take[ hptData1[[All, iPth]], 20], Take[ hptData1[[All, iTth]], 20]}

Out[ ]= {1396.42, 1397.85, 1398.76, 1399.49, 1400.29, 1401.2, 1402.2, 1403.01, 1403.89, 1403.83,
         1403.57, 1405.37, 1406.64, 1407.72, 1408.59, 1409.71, 1410.71, 1411.27, 1411.17, 1410.77}

Out[ ]= {1.58245, 1.58377, 1.58446, 1.58491, 1.58566, 1.58652, 1.58726, 1.58802, 1.58819, 1.58879,
         1.58719, 1.58924, 1.59098, 1.59191, 1.59259, 1.59374, 1.59478, 1.594, 1.595, 1.593}

Out[ ]= {0.766905, 0.770438, 0.771513, 0.771622, 0.773703, 0.776032,
         0.777169, 0.779191, 0.776599, 0.781281, 0.770624, 0.777874, 0.785128,
         0.787282, 0.788472, 0.792033, 0.795366, 0.787157, 0.794981, 0.781999}
```

Manipulate

```
In[ ]:= manGVtrim02 = Manipulate[
           (*Downsample the data*)
           hptDataDown = Downsample[hptData1, {nDown, 1}];
           vecTimeDataDown = Downsample[vecTimeDataRaw, nDown];
           (*User given governor valve trim*)
           posSorted = Sort[pos];
           ptsExtended = Prepend[Append[posSorted, {1.5, 1}], {-2, 0}] ;
           gvfuncUser = Interpolation[ptsExtended, InterpolationOrder → 1];
           (*choose appropriate goveror valve trim*)
           gvFuncs = If[ igvf == 0, gvFuncs00, {gvfuncUser, gvfuncUser, gvfuncUser,
              gvfuncUser, gvfuncUser, gvfuncUser, gvfuncUser, gvfuncUser} ];
           (*Governor-Nozzle flow calculation*)
           rAll = calcBowenGovNozEng[# , N[kst], N[kgv], knz,
               If[iDynErr > 0, True, False], gvFuncs, iknzs] & /@ hptDataDown;
           (*First stage enthalpy, pressure, and temeprature*)
           rAllPT = hptDataDown[[All, {iTth, iPth}]];
           effDes = {120, 0.95, 0.2};
           (*To do: replace 120 with Design vol flow = Fdes / rDes * 1000/3600. *)
           (*rTestEnergy = Table[calcBowenGovNozEnergy[rAllPT[[i]], rAll[[i]], effDes],
              {i, 1, Length@hptDataDown} ];*)
           (*HP turbine J*)
```

```
vecfTotcalc = Total[ #[[All, 4]] ] & /@ rAll;
(*vecrhoFScalc = rTestEnergy[[All,1,5]];*)
vecdpHPdata = hptDataDown[[All, iPfs]] - hptDataDown[[All, iPcrh]];
(*vecjHPcalc = vecfTotcalc / Sqrt[vecrhoFScalc vecdpHPdata];*)
(*HP turbine expansion*)
(*HP turbine temperature*)
(*parse results for graphich*)
vecChokeGV1 = rAll[[All, 1, 1]];
vecChokeGV2 = rAll[[All, 2, 1]];
vecChokeGV3 = rAll[[All, 3, 1]];
vecChokeGV4 = rAll[[All, 4, 1]];
vecChokeGV5 = rAll[[All, 5, 1]];
vecChokeGV6 = rAll[[All, 6, 1]];
vecChokeGV7 = rAll[[All, 7, 1]];
vecChokeGV8 = rAll[[All, 8, 1]];

vecChokeJR1 = rAll[[All, 1, 2]];
vecChokeJR2 = rAll[[All, 2, 2]];
vecChokeJR3 = rAll[[All, 3, 2]];
vecChokeJR4 = rAll[[All, 4, 2]];
vecChokeJR5 = rAll[[All, 5, 2]];
vecChokeJR6 = rAll[[All, 6, 2]];
vecChokeJR7 = rAll[[All, 7, 2]];
vecChokeJR8 = rAll[[All, 8, 2]];

vecChokePR1 = rAll[[All, 1, 3]];
vecChokePR2 = rAll[[All, 2, 3]];
vecChokePR3 = rAll[[All, 3, 3]];
vecChokePR4 = rAll[[All, 4, 3]];
vecChokePR5 = rAll[[All, 5, 3]];
vecChokePR6 = rAll[[All, 6, 3]];
vecChokePR7 = rAll[[All, 7, 3]];
vecChokePR8 = rAll[[All, 8, 3]];

vecChokeFlow1 = rAll[[All, 1, 4]];
vecChokeFlow2 = rAll[[All, 2, 4]];
vecChokeFlow3 = rAll[[All, 3, 4]];
vecChokeFlow4 = rAll[[All, 4, 4]];
vecChokeFlow5 = rAll[[All, 5, 4]];
vecChokeFlow6 = rAll[[All, 6, 4]];
vecChokeFlow7 = rAll[[All, 7, 4]];
vecChokeFlow8 = rAll[[All, 8, 4]];

vecFlowtotal = vecChokeFlow1 + vecChokeFlow2 + vecChokeFlow3 + vecChokeFlow4
  + vecChokeFlow5 + vecChokeFlow6 + vecChokeFlow7 + vecChokeFlow8 ;
```

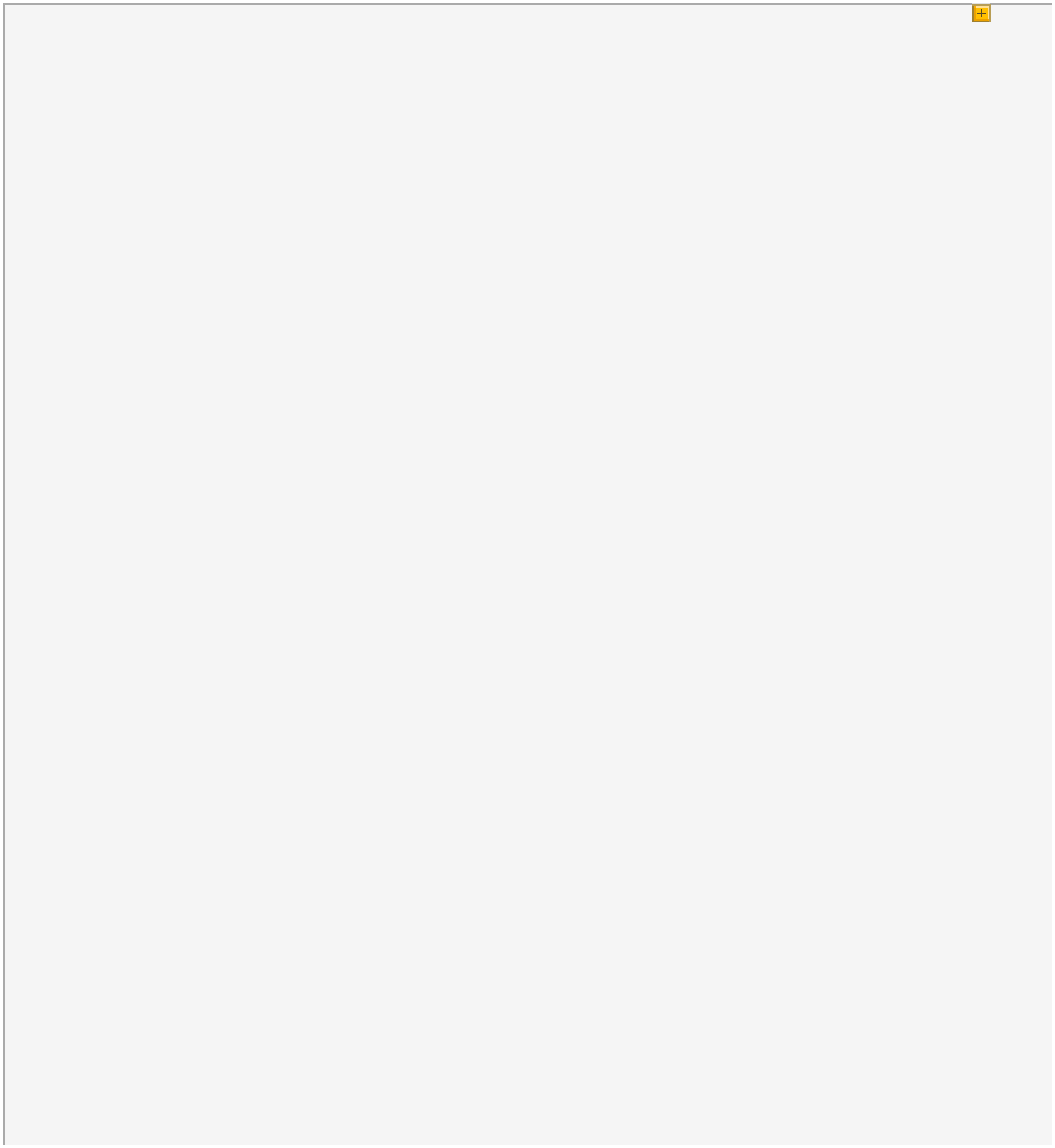
```
vecFlowDiff = 3600 / 1000. vecFlowtotal - hptDataDown[[All, 3]];

vecChokePBowl1 = rAll[[All, 1, 6]];
vecChokePBowl2 = rAll[[All, 2, 6]];
vecChokePBowl3 = rAll[[All, 3, 6]];
vecChokePBowl4 = rAll[[All, 4, 6]];
vecChokePBowl5 = rAll[[All, 5, 6]];
vecChokePBowl6 = rAll[[All, 6, 6]];
vecChokePBowl7 = rAll[[All, 7, 6]];
vecChokePBowl8 = rAll[[All, 8, 6]];

(*make vectors -- for system*)
vvFlowDataModel = Transpose[{vecTimeDataDown, vecFlowtotal 3600 / 1000}];
vvFlowDiff = Transpose[{vecTimeDataDown, vecFlowDiff }];
(*make vectors -- for Steam Chest A*)
vvFlowGViA = Transpose[{vecTimeDataDown, #}] & /@ {vecChokeFlow1 3600 / 1000,
    vecChokeFlow2 3600 / 1000, vecChokeFlow3 3600 / 1000, vecChokeFlow4 3600 / 1000};
vvJgvJnzGviA = Transpose[{vecTimeDataDown, #}] & /@
   {vecChokeJR1, vecChokeJR2, vecChokeJR3, vecChokeJR4};
vvChokeStatusA = Transpose[{vecTimeDataDown, #}] & /@
   {vecChokeGV1, vecChokeGV2, vecChokeGV3, vecChokeGV4};
vvPR1ModelA = Transpose[{vecTimeDataDown, vecChokePR1}];
(*make vectors -- for Steam Chest B*)
vvFlowGViB = Transpose[{vecTimeDataDown, #}] & /@ {vecChokeFlow5 3600 / 1000,
    vecChokeFlow6 3600 / 1000, vecChokeFlow7 3600 / 1000, vecChokeFlow8 3600 / 1000};
vvJgvJnzGviB = Transpose[{vecTimeDataDown, #}] & /@
   {vecChokeJR5, vecChokeJR6, vecChokeJR7, vecChokeJR8};
vvChokeStatusB = Transpose[{vecTimeDataDown, #}] & /@
   {vecChokeGV5, vecChokeGV6, vecChokeGV7, vecChokeGV8};
vvPR5ModelB = Transpose[{vecTimeDataDown, vecChokePR5}];
(*energy balance info*)
(*vvvecjHPcalc = Transpose[{vecTimeDataDown,vecjHPcalc}];*)
(*make plots*)
(*pGVtrim=plotGovValveTrim[gvFuncs00, gvfuncUser, ptsExtended, {600,400} ];*)
pGVtrim = plotGovValveTrim2Trims[{gvfuncUser}, ptsExtended, {600, 400}];
pFlowTotal = plotBowenPlantModelFlow[ vvFlowDataPlant, vvFlowDataModel];
pFlowDiff = plotBowenPlantModelFlowDiff[vvFlowDiff];
(*plots for Steam Chest A*)
pFlowGViA = plotBowenModelFlowGVi[vvFlowGViA];
pJgvJnzGviA = plotBowenModelFlowJRi[vvJgvJnzGviA];
pChokeStatusA = plotBowenModelChokeStatus[vvChokeStatusA];
pPRA = plotBowenModelPR[vvPR1ModelA];
(*plots for Steam Chest B*)
pFlowGViB = plotBowenModelFlowGVi[vvFlowGViB];
pJgvJnzGviB = plotBowenModelFlowJRi[vvJgvJnzGviB];
pChokeStatusB = plotBowenModelChokeStatus[vvChokeStatusB];
```

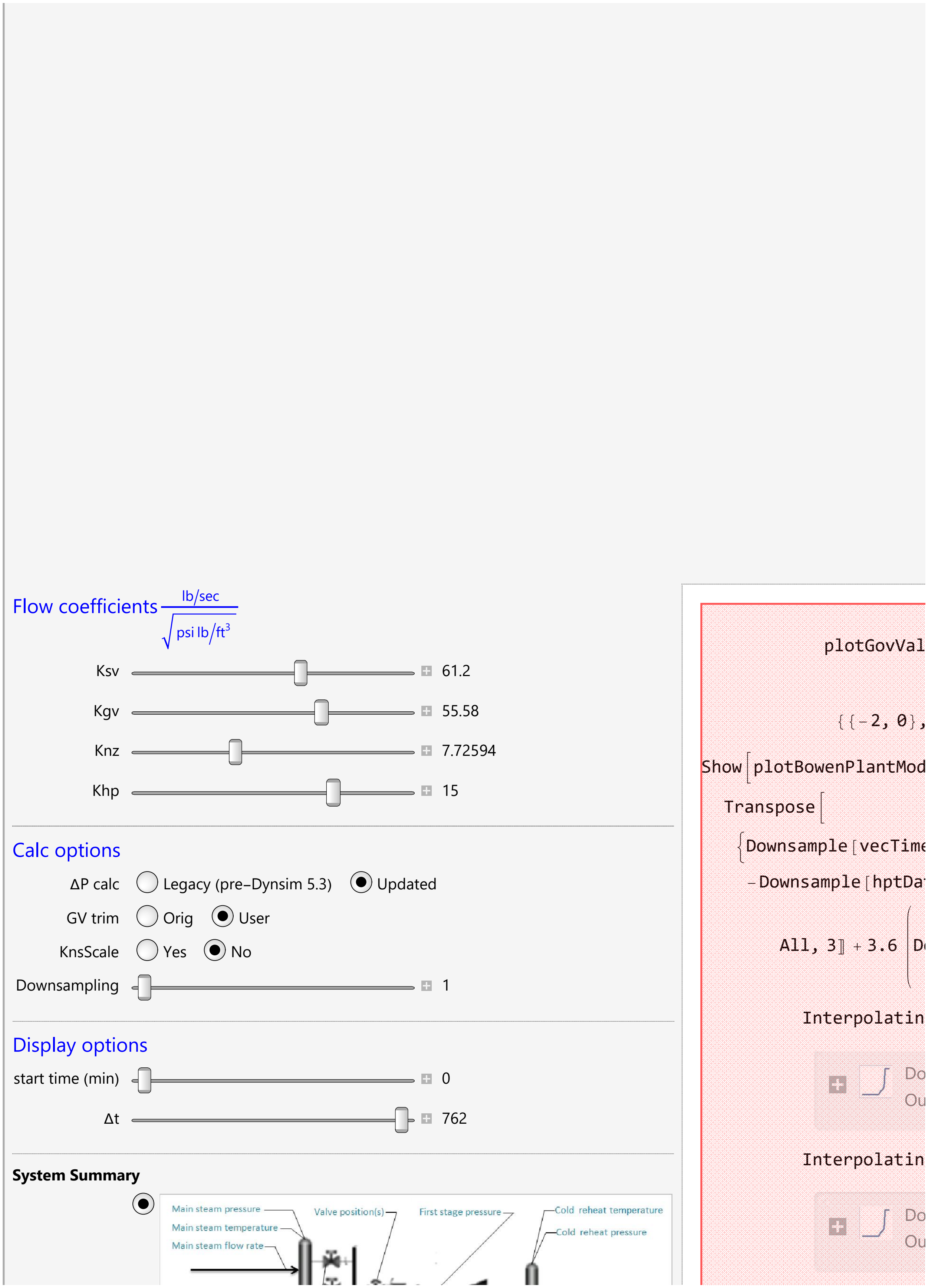
```
pPRB = plotBowenModelPR[vvPR5ModelB];
(*plots for HP turbine*)
pKhp = Plot[Sin[x], {x, 0, 6 π}];(*plotBowenModelK[vvvecjHPcalc]*);
(*this graph is a place holder, also display a few test values*)
pSin = Plot[Sin[x], {x, 0, 6 π}, ImageSize → {300, 300}
   , Epilog → {Text[iknzs, {4, 0.5}, {-1, -1}], Text[igvf, {4.5, 0.5}, {-1, -1}]}];
(*create a table with parameters to use in Dynsim: *)
rptRow01 = {Style["K stop valve", 12], NumberForm[kst, 4]};
rptRow02 = {Style["K governor", 12], NumberForm[kgv, 4]};
rptRow03 = {Style["K nozzle", 12], NumberForm[knz, 4] };
rptRow04 = {Style["Gov. valve trim", 12], Framed@TableForm[posSorted]};
rptRows = {rptRow01, rptRow02, rptRow03, rptRow04};
rptLabel = Style["HP Turbine Parameters", Bold, kpLabel];
rptTable = Labeled[ Grid[rptRows, Alignment → Left,
    Frame → True, Background → LightBlue], rptLabel, {Top}];
(*=======================*)
(*report result to HMI set of graphs in a grid*)
Grid[{{LocatorPane[Dynamic@pos,
     pGVtrim, LocatorAutoCreate → True, ContinuousAction → False ]
    , SpanFromLeft}
   , { Show[pFlowDiff, PlotRange → {{td0, tdDT}, {-1000, + 1000} }]
    , Show[pzGVDataA, PlotRange → {{td0, tdDT}, All}, ImageSize → {300, 300}]
    , Show[pzGVDataB, PlotRange → {{td0, tdDT}, All}, ImageSize → {300, 300}]
   }
   , { Show[plotGovNoz4Regions, ImageSize → {300, 300}]
    , Show[pFlowTotal,
     PlotRange → {{td0, tdDT}, {Floor[0.8 Min[vvFlowDataPlant[[All, 2]]], 500],
        Ceiling[Max[vvFlowDataPlant[[All, 2]]], 1000]}}]
    , Show[pFlowTotal, PlotRange → {{td0, tdDT}, {Floor[0.8 Min[vvFlowDataPlant[[
            All, 2]]], 500], Ceiling[Max[vvFlowDataPlant[[All, 2]]], 1000]}}]
   }
   , {rptTable
    , pJgvJnzGviA
    , pJgvJnzGviB
   }
   , {pKhp
    , pChokeStatusA
    , pChokeStatusB
   }
   , {pKhp
    , pFlowGViA
    , pFlowGViB
   }
   , {pKhp
    , pPRA
    , pPRB
```

```
      }
      , {pKNzfunc
        , pKNzfunc
        , pKNzfunc
      }
     }]
    (*======================*)
    ,
    (*list of controls*)
    {{pos, ipts}, ControlType → None, LocatorAutoCreate → True, Appearance → Automatic}
    , Delimiter
    , Style["Flow coefficients lb / sec / √(psi lb / ft^3)", Blue, 14]
    , {{kst, 61.2, "Ksv"}, 1, 100, 1, Appearance → "Labeled"}
    , {{kgv, 55.58, "Kgv"}, 1, 80, 1, Appearance → "Labeled"}
    , {{knz, 7.72594, "Knz"}, 1, 20, 0.2, Appearance → "Labeled"}
    , {{khp, 15, "Khp"}, 1, 20, 0.2, Appearance → "Labeled"}
    , Delimiter
    , Style["Calc options", Blue, 14]
    , {{iDynErr, 1, "ΔP calc"},
     {0 → "Legacy (pre-Dynsim 5.3)", 1 → "Updated"}, ControlType → RadioButtonBar}
    , {{igvf, 1, "GV trim"}, {0 → "Orig", 1 → "User"}, ControlType → RadioButtonBar}
    , {{iknzs, 0, "KnsScale"}, {1 → "Yes", 0 → "No"}, ControlType → RadioButtonBar}
    ,  {{nDown, 1, "Downsampling"}, 1, 10, 1, Appearance → "Labeled"}
    , Delimiter
    , Style["Display options", Blue, 14]
    ,
    {{td0, 0, "start time (min)"}, 0, Max@vecTimeDataRaw - 50, 50, Appearance → "Labeled"}
    , {{tdDT, Max@vecTimeDataRaw, "Δt"}, 0, Max@vecTimeDataRaw,
     50, Appearance → "Labeled"}
    , Delimiter
    , Style["System Summary", Bold]
  , {{bc, 1, " "}, {1 → g}, ControlType → RadioButton}
  , Delimiter
    , ControlPlacement :→ {Left, Left, Left, Left, Left, Left}
    , TrackedSymbols :→ {kst, kgv, knz, khp, igvf, iknzs, pos, iDynErr, td0, tdDT, nDown}
    , Initialization :→ (
      ipts = tabVCV3a[[{2, 5, 9, 15, 18, 19, 20}]];
      posSorted = {}; (*do this so that posSorted is local to this Manipulate*)
      gvFuncs00 = {aVCV1afunc, aVCV2afunc, aVCV3afunc, aVCV4afunc, aVCV1afunc,
         aVCV2afunc, aVCV3afunc, aVCV4afunc}; (*original governor valve trims*)
      gvfuncUser = {};
      gvFuncs = {};    (*do this so that posSorted is local to this Manipulate*)
      g = Show[fig01, ImageSize → Medium]; (* sytem sketch*)
      hptDataDown = hptData1;
    vecTimeDataDown = vecTimeDataRaw;
   )
  , SynchronousUpdating → False
 ];
```

Manipulate

In[ ]:=
```
manGVtrim02
```





Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 55.58

Knz 7.72594

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 0

Δt 762

System Summary

●

```
            plotGovVal

             {{-2, 0},

Show[plotBowenPlantMod
  Transpose[
   {Downsample[vecTime
    -Downsample[hptDat

      All, 3]] + 3.6 (D

         Interpolatin

             Do
             Ou

         Interpolatin

             Do
             Ou
```

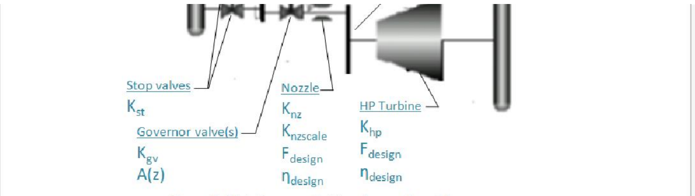


Figure 1: High Pressure Turbine data and model parameters

```
calcBowenGo
 hptData1,
 55.58, 7.7
 {Interpola

  Interpola

  Interpola

  Interpola

  Interpola

  Interpola

  Interpola

  Interpola

calcBowenGo
 {1, 1}, 61
```

```
{1, 1}, 01.
7.72594, Tı
{Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

All, 1, 4⟧ + D
calcBowenGo
hptData1, (
55.58, 7.7
{Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

 Interpola

  ⊞

calcBowenGo
{1, 1}, 61.
7.72594, T
{Interpola
```

```
                    Interpola

                    Interpola

                    Interpola

                    Interpola

                    Interpola

                    Interpola

                    Interpola

              All, 2, 4⟧ + D
                calcBowenGo
                  hptData1,
                  55.58, 7.7
                  {Interpola
```

```
Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

calcBowenGo
 {1, 1}, 61.
 7.72594, Tr
 {Interpola
```

```
Interpola

⊞

Interpola

⊞

Interpola

⊞

Interpola

⊞

Interpola

⊞

Interpola

⊞

Interpola

⊞

All, 3, 4⟧ + D

calcBowenGo

hptData1, (
55.58, 7.7
{Interpola

⊞
```

```
Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

calcBowenGo
 {1, 1}, 61.
 7.72594, T
 {Interpola

Interpola
```

```
                                   Interpola

                                   Interpola

                                   Interpola

                                   Interpola

                                   Interpola

                                   Interpola

                                   Interpola

                                   Interpola

                              All, 4, 4]] + D

                                calcBowenGo

                                 hptData1, (
                                 55.58, 7.7
                                 {Interpola

                                   Interpola
```

```
Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

calcBowenGo
{1, 1}, 61.
7.72594, T
{Interpola

Interpola
```

```
                 Interpola

                 Interpola

                 Interpola

                 Interpola

                 Interpola

                 Interpola

            All, 5, 4⟧ + D
              calcBowenGo
                hptData1,
                55.58, 7.7
               {Interpola

                 Interpola
```

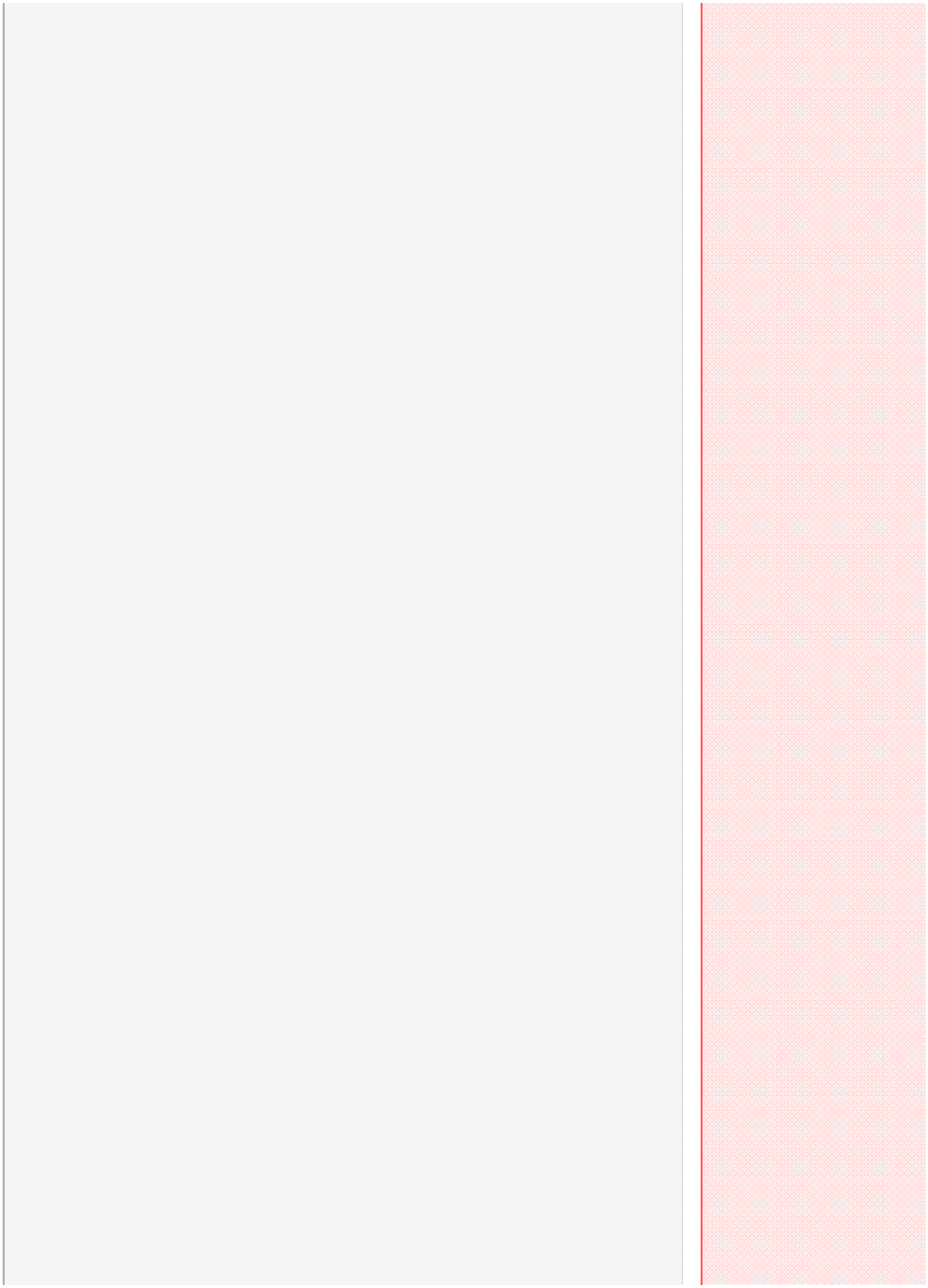
```
Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

calcBowenGo
{1, 1}, 61.
7.72594, Tr
{Interpola

Interpola
```

```
              Interpola

                ⊞

              Interpola

                ⊞

              Interpola

                ⊞

              Interpola

                ⊞

              Interpola

                ⊞

              Interpola

                ⊞

          All, 7, 4⟧ + D

           calcBowenGo

            hptData1, (
            55.58, 7.7
           {Interpola

                ⊞

              Interpola

                ⊞

              Interpola
```

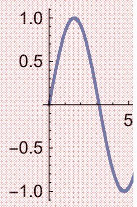
```
Interpola

Interpola

Interpola

Interpola

Interpola

Interpola

calcBowenGo
 {1, 1}, 61.
 7.72594, T
 {Interpola

Interpola

Interpola
```

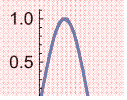
```
            ⊞
        Interpola
            ⊞
        Interpola
            ⊞
        Interpola
            ⊞
        Interpola
            ⊞
        Interpola
            ⊞

      All, 8, 4⟧ }}]
PlotRange → {{0,
   762},
  {-1000,
   1000}}]
 Show[plotGovNoz4Regi
```

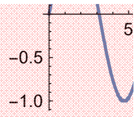
Out[ ]=

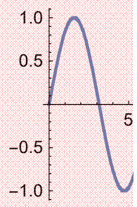


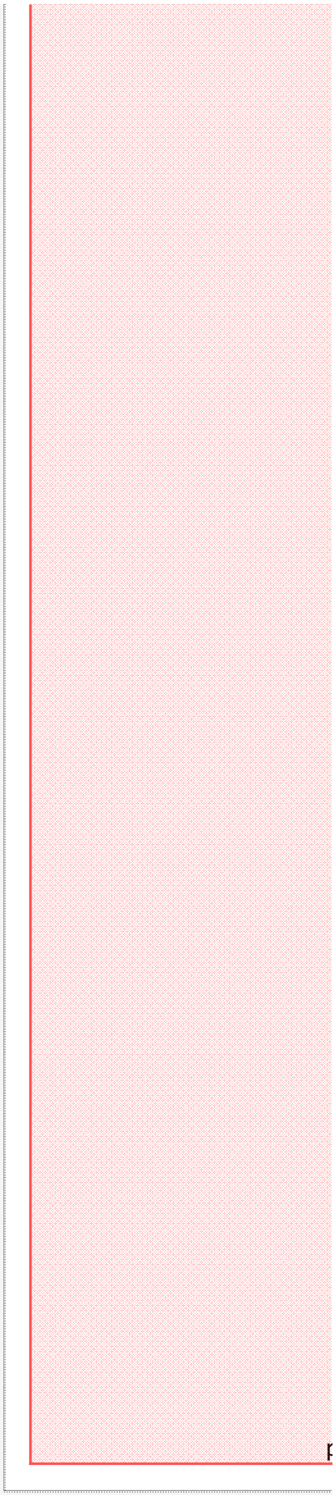
```
Style[HP Turbine F
      K stop valve
      K governor
      K nozzle




      Gov. valve tr
```


1.0
0.5
5
−0.5
−1.0


1.0
0.5


5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0


p

Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 7.72594

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

Δt 762

System Summary

●

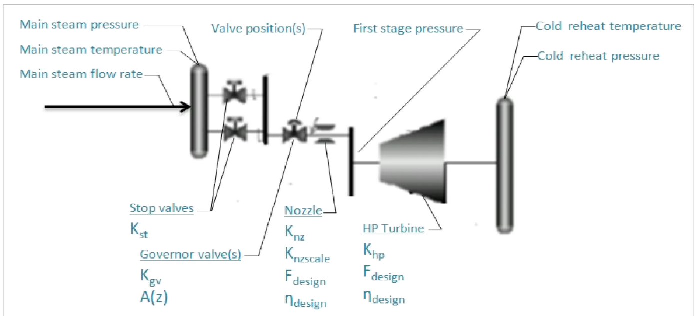
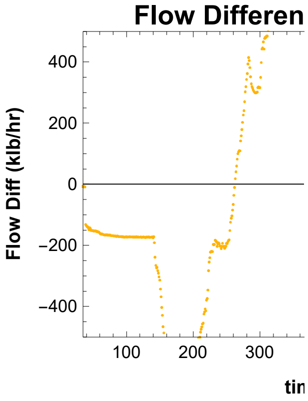
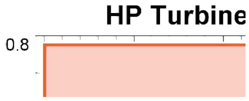
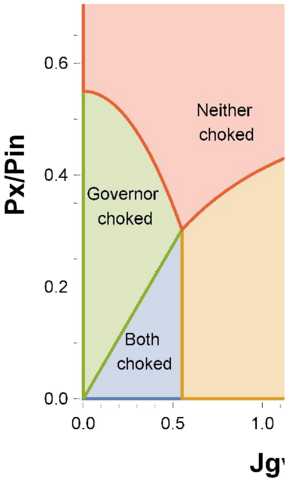


Figure 1: High Pressure Turbine data and model parameters



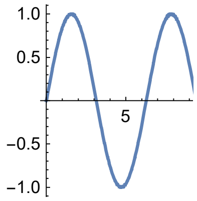


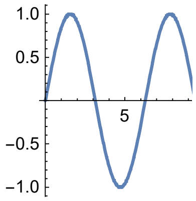


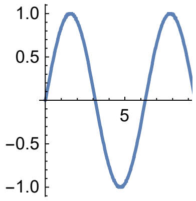
HP Turbine

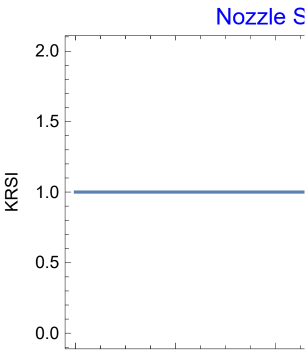
K stop valve	61.
K governor	40
K nozzle	7.7
Gov. valve trim	0. 0. 0. 0. 0. 0. 1.

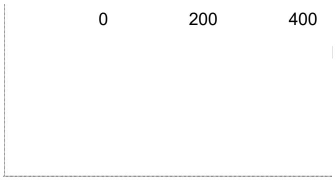
In[ ]:=

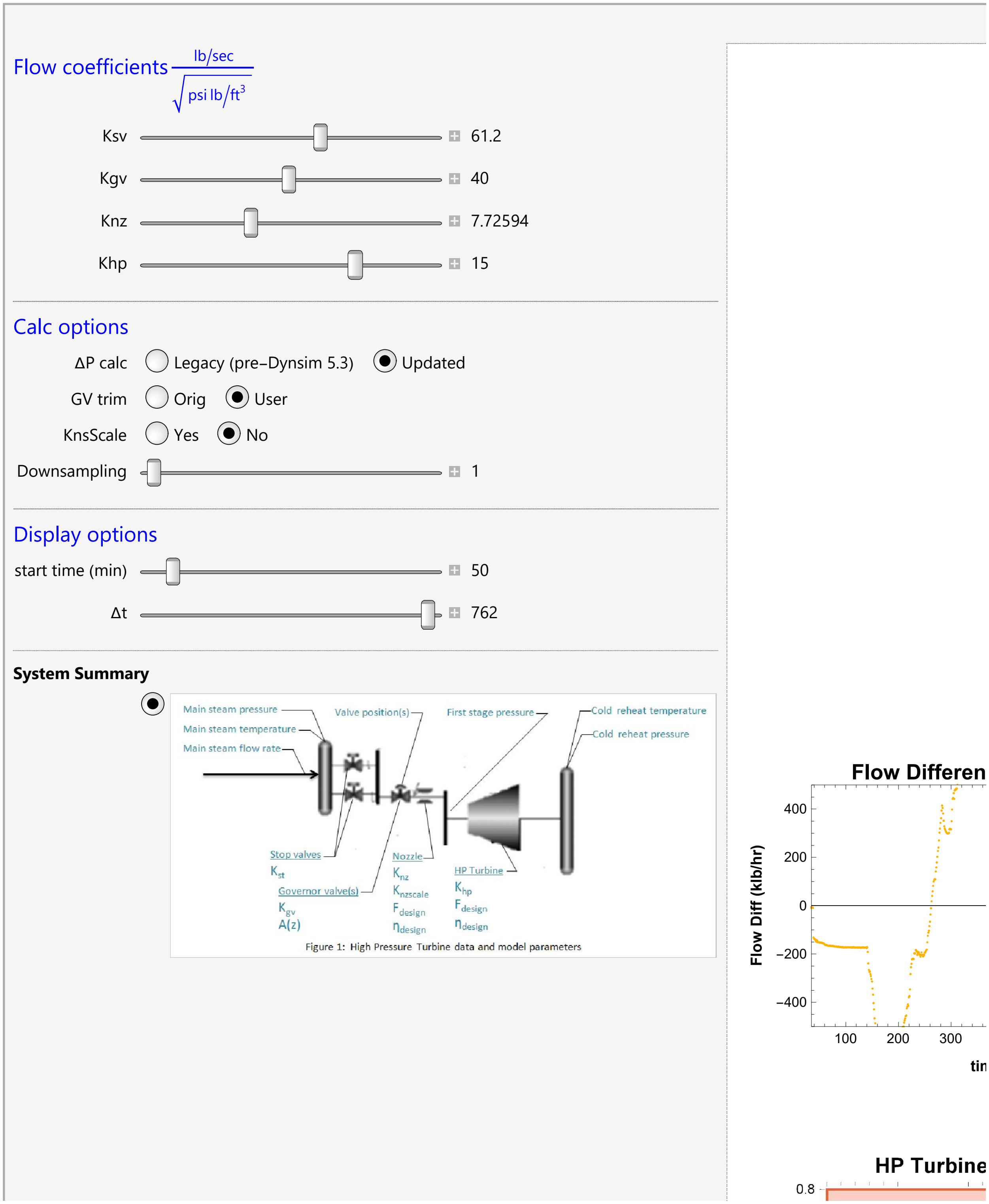



1.0
0.5
5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0


Nozzle S
2.0
1.5
KRSI
1.0
0.5
0.0





Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 7.72594

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

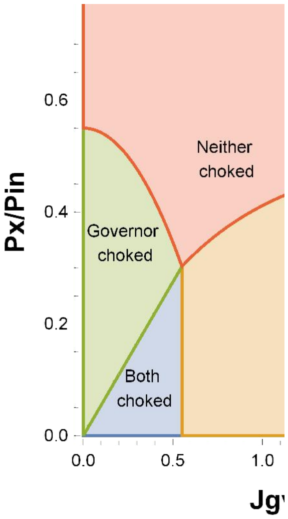
Δt 762

System Summary

●

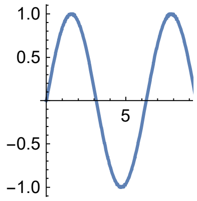
Figure 1: High Pressure Turbine data and model parameters

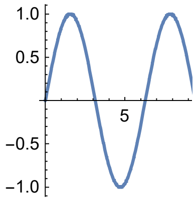
Out[ ]=

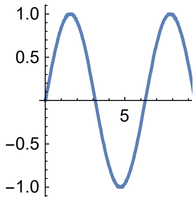


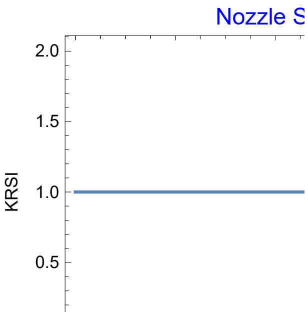
HP Turbine F

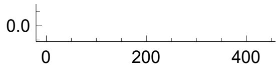
K stop valve	61.
K governor	40
K nozzle	7.7
Gov. valve trim	0. 0. 0. 0. 0. 0. 1.




1.0
0.5
5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0


Nozzle S
2.0
1.5
KRSI
1.0
0.5



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv	61.2
Kgv	40
Knz	6.6
Khp	15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

Δt 700

System Summary

●

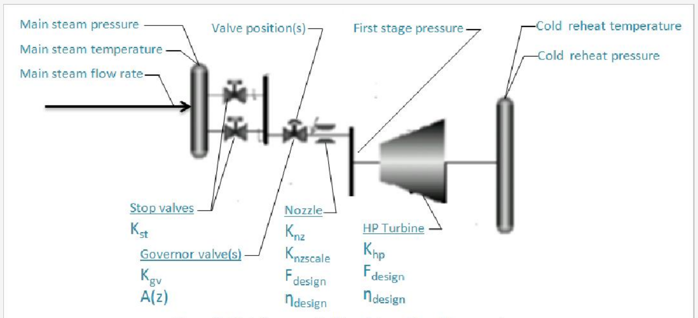
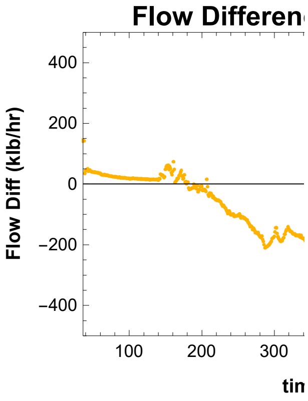
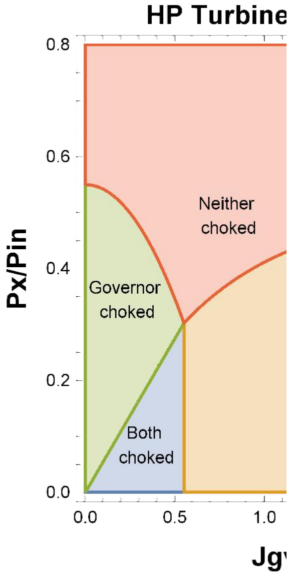


Figure 1: High Pressure Turbine data and model parameters

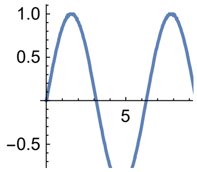


In[ ]:=

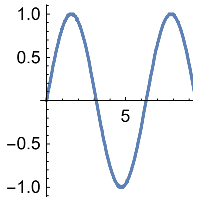


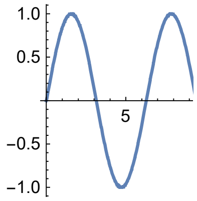
HP Turbine P

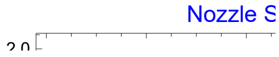
K stop valve	61.
K governor	40
K nozzle	6.6
Gov. valve trim	0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.

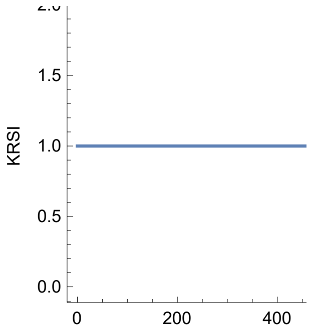


−1.0


1.0
0.5
5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0


Nozzle S
2.0



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 6.6

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

Δt 700

System Summary

●

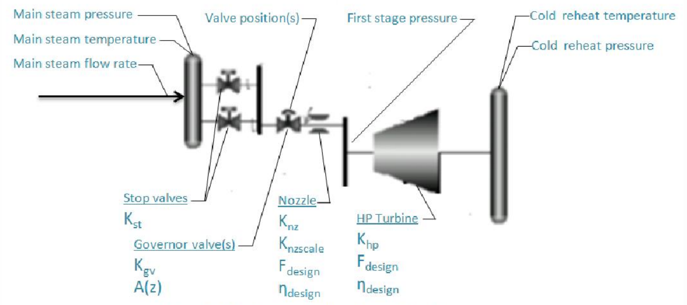
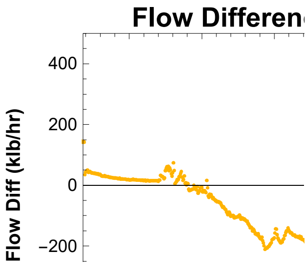
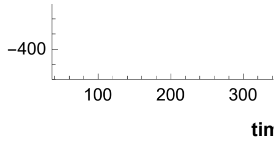
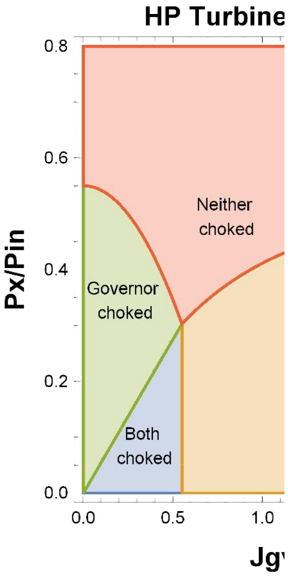


Figure 1: High Pressure Turbine data and model parameters



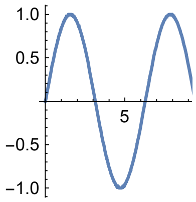


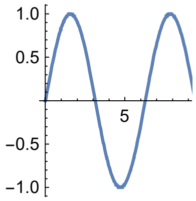


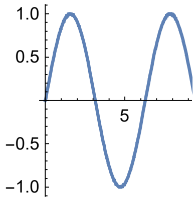
Out[ ]=

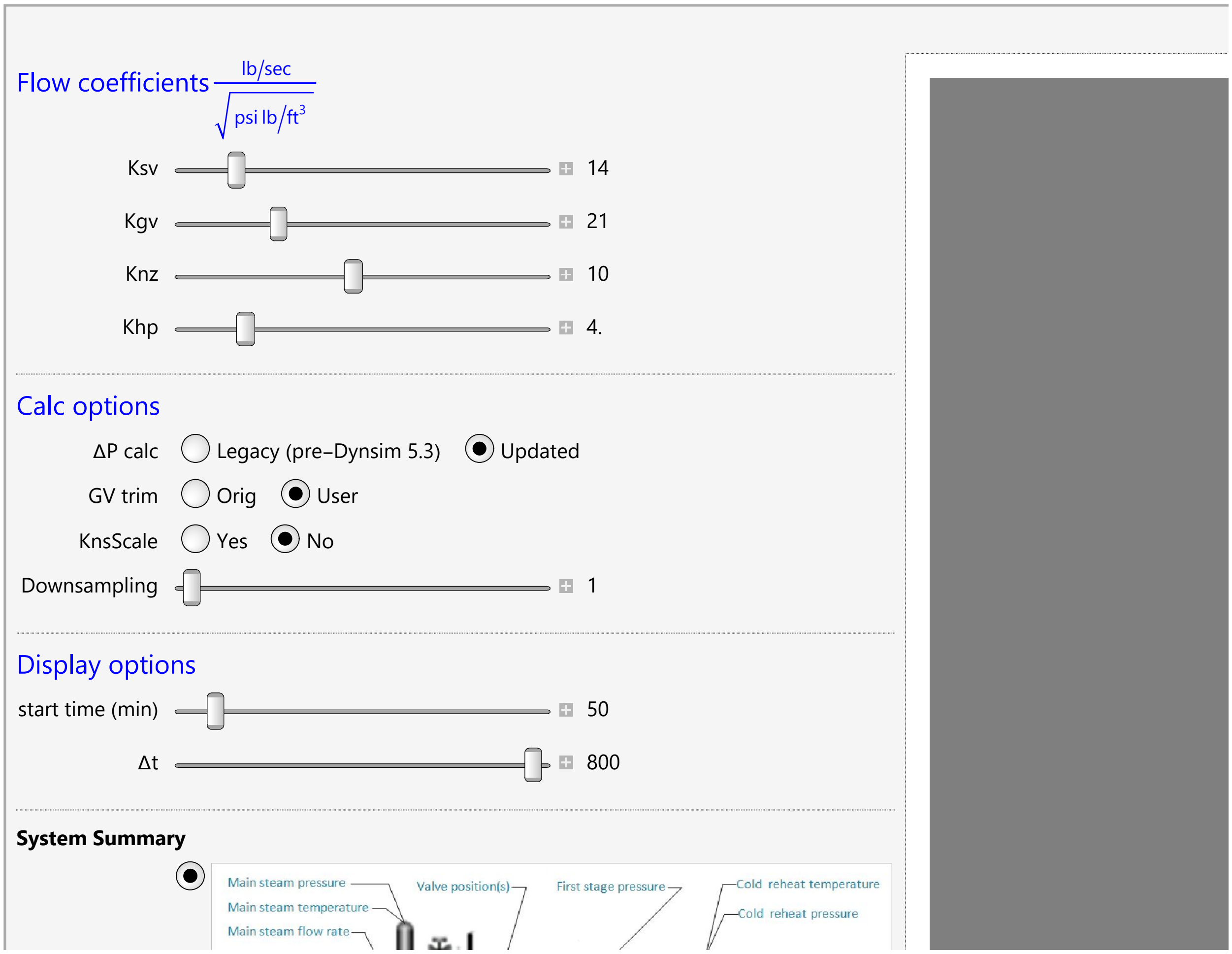
HP Turbine F

K stop valve	61.
K governor	40
K nozzle	6.6
Gov. valve trim	0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.


1.0
0.5
5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0


1.0
0.5
5
−0.5
−1.0



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 14

Kgv 21

Knz 10

Khp 4.

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

Δt 800

System Summary

●

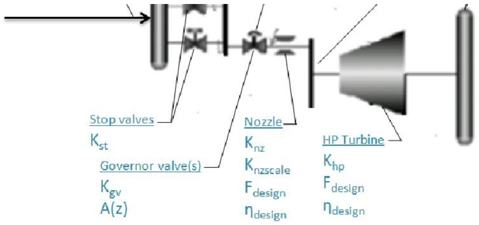
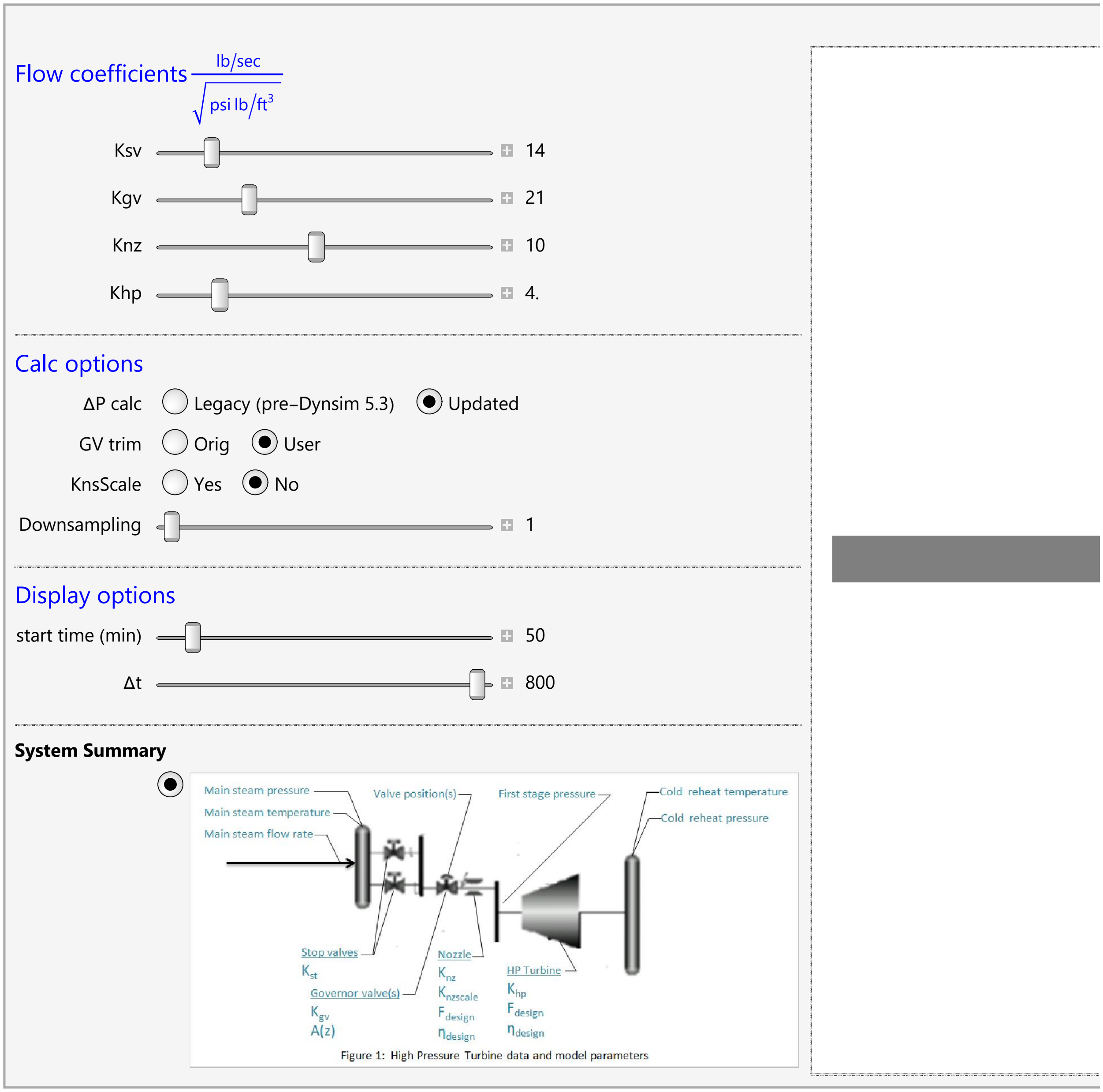


Figure 1: High Pressure Turbine data and model parameters

In[ ]:=

Out[ ]=



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 14

Kgv 21

Knz 10

Khp 4.

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 50

Δt 800

System Summary

●

Figure 1: High Pressure Turbine data and model parameters

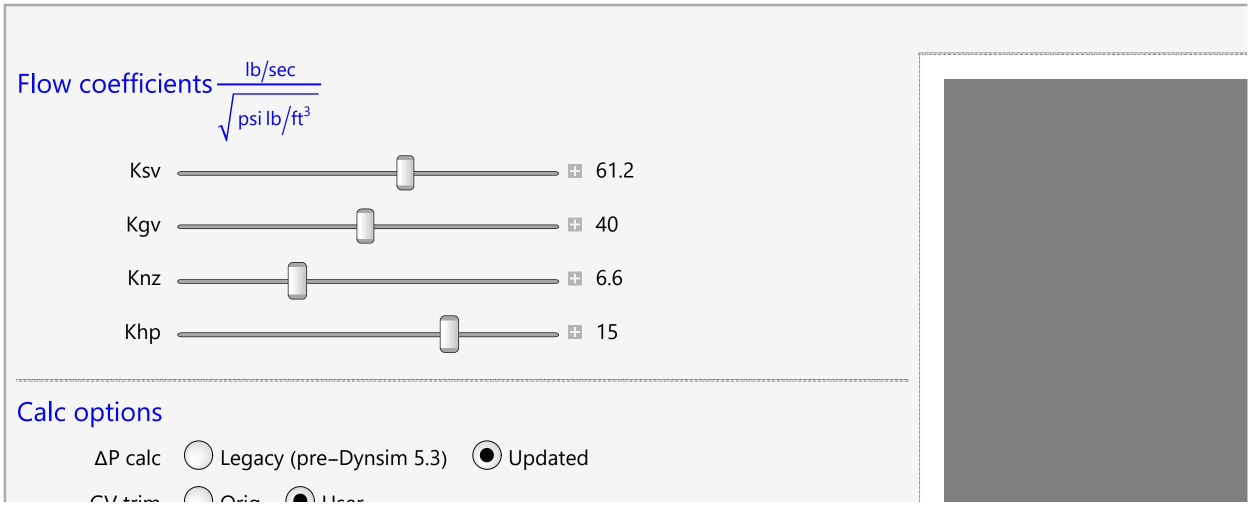
Part: Part specification

calcBowenGovNozEng[{0., 0.289352, 1., 805.178, 910.588, 14.7, 4.06133, 447.721, 97.7575, 99.5107, 98.9854, 99.6126, 99.3255, 99.444, 99.3594, 99.6774, 1.30518}, 14., 21., 10, True, {≪1≫}, 0]⟦All, 4⟧ is longer than depth of object.

Part: Part specification

calcBowenGovNozEng[{1., 0.217014, 8.33765, 807.433, 908.646, 18.6535, 5.96445, 452.682, 97.7311, 99.5094, 98.9811, 99.6107, 99.3278, 99.4609, 99.3577, 99.7002, 1.29923}, 14., 21., 10, True, {≪1≫}, 0]⟦All, 4⟧ is longer than depth of object.

```
Part: Part specification
  calcBowenGovNozEng[{2., 0.253183, 6.36044, 808.997, 908.858, 17.9676, 5.96445, 455.159, 97.7484, 99.5094, 98.974, 99.6117
        , 99.3135, 99.4447, 99.3688, 99.6901, 1.29755}, 14., 21., 10, True, {≪1≫, ≪7≫}, 0]⟦All, 4⟧ is longer
  than depth of object.

General: Further output of Part::partd will be suppressed during this calculation.

Part: Part 8 of
  calcBowenGovNozEng[{0., 0.289352, 1., 805.178, 910.588, 14.7, 4.06133, 447.721, 97.7575, 99.5107, 98.9854, 99.6126,
     99.3255, 99.444, 99.3594, 99.6774, 1.30518}, 14., 21., 10, True, {≪1≫}, 0] does not exist.

Part: Part 8 of
  calcBowenGovNozEng[{0., 0.289352, 1., 805.178, 910.588, 14.7, 4.06133, 447.721, 97.7575, 99.5107, 98.9854, 99.6126,
     99.3255, 99.444, 99.3594, 99.6774, 1.30518}, 14., 21., 10, True, {≪1≫}, 0] does not exist.

Part: Part 8 of
  calcBowenGovNozEng[{0., 0.289352, 1., 805.178, 910.588, 14.7, 4.06133, 447.721, 97.7575, 99.5107, 98.9854, 99.6126,
     99.3255, 99.444, 99.3594, 99.6774, 1.30518}, 14., 21., 10, True, {≪1≫}, 0] does not exist.

General: Further output of Part::partw will be suppressed during this calculation.

Transpose: The first two levels of
  {{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,

                                                          18
     38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, ≪713≫}, ── {≪1≫}⟦All, 2, 4⟧} cannot be transposed.
                                                          5

Transpose: The first two levels of
  {{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,

                                                          18
     38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, ≪713≫}, ── {≪1≫}⟦All, 3, 4⟧} cannot be transposed.
                                                          5

Transpose: The first two levels of
  {{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,

                                                          18
     38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, ≪713≫}, ── {≪1≫}⟦All, 4, 4⟧} cannot be transposed.
                                                          5

General: Further output of Transpose::nmtx will be suppressed during this calculation.
```



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 6.6

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

CV trim ○ Orig ● User

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 1200

Δt 1927

System Summary

●

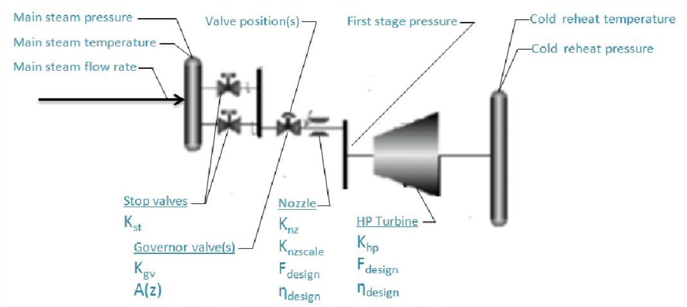
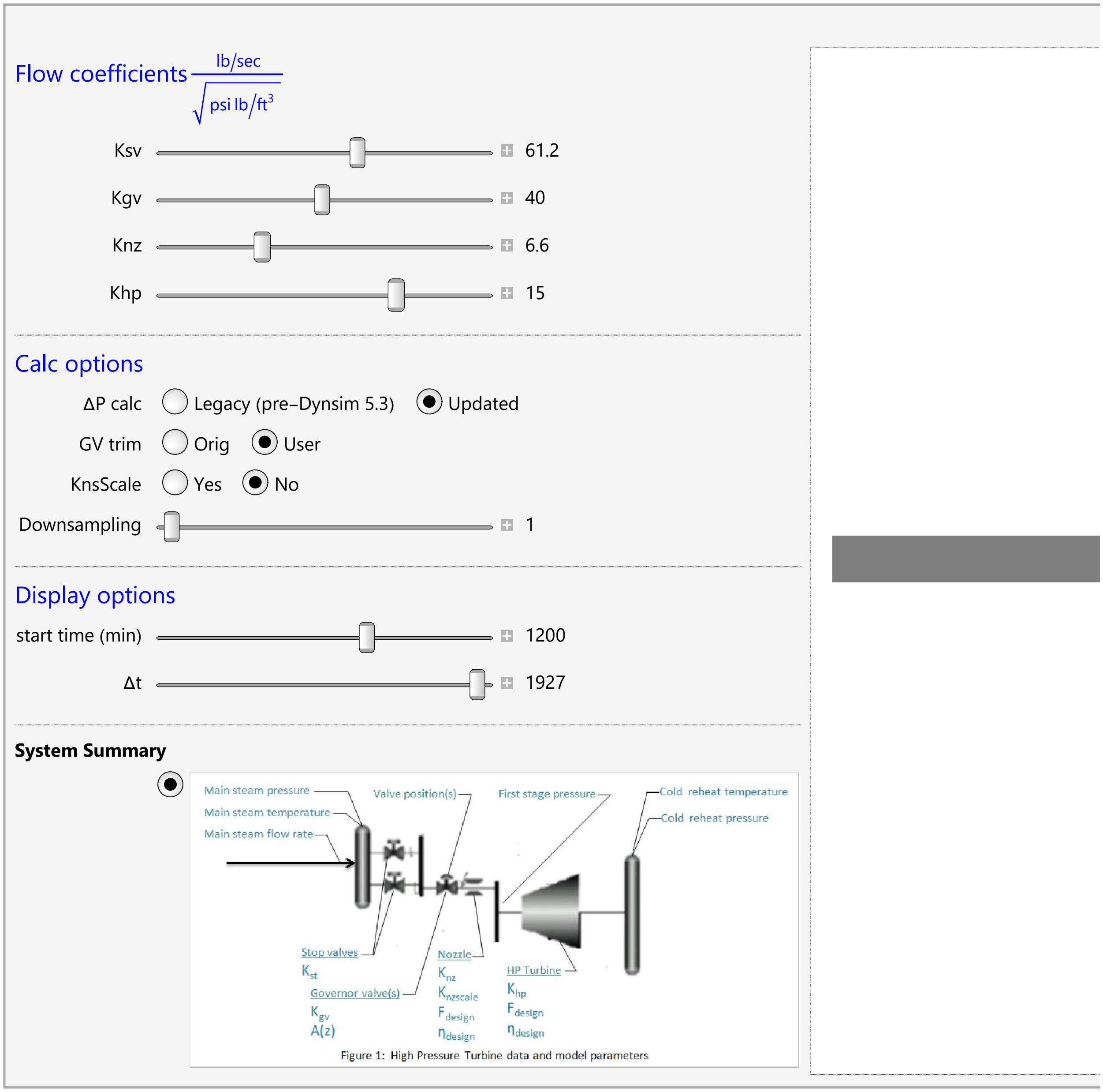


Figure 1: High Pressure Turbine data and model parameters

In[ ]:=

Out[◦]=



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 6.6

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

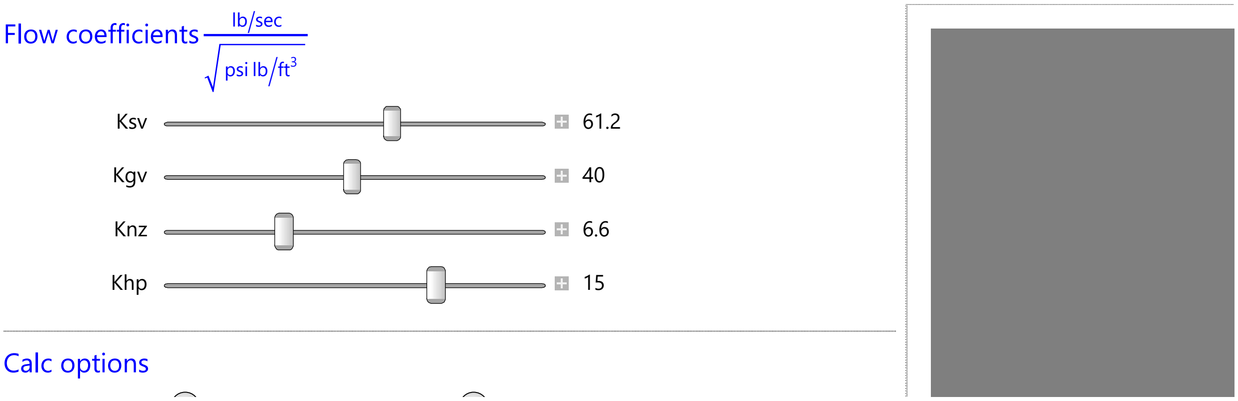
start time (min) 1200

Δt 1927

System Summary

●

Figure 1: High Pressure Turbine data and model parameters



Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 6.6

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Display options

start time (min) 0

Δt 1927

System Summary

●

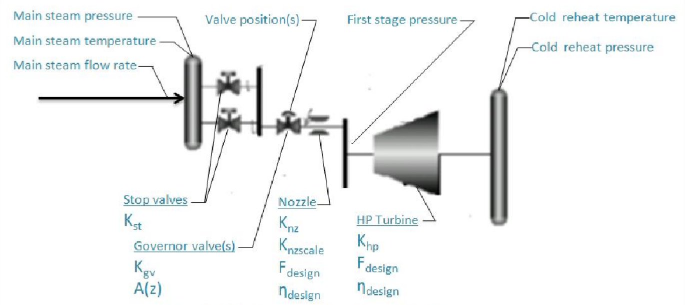


Figure 1: High Pressure Turbine data and model parameters

In[ ]:=

Flow coefficients $\frac{\text{lb/sec}}{\sqrt{\text{psi lb/ft}^3}}$

Ksv 61.2

Kgv 40

Knz 6.6

Khp 15

Calc options

ΔP calc ○ Legacy (pre–Dynsim 5.3) ● Updated

GV trim ○ Orig ● User

KnsScale ○ Yes ● No

Downsampling 1

Out[ ]=

Display options

start time (min) 0

Δt 1927

System Summary

●

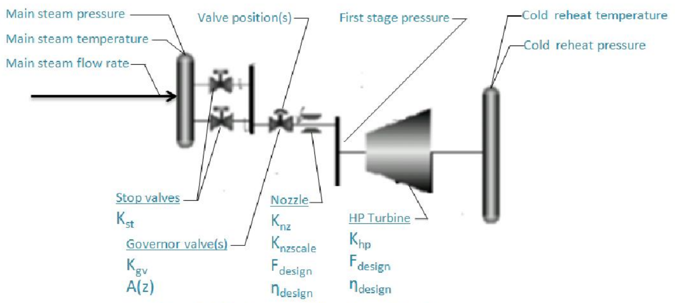


Figure 1: High Pressure Turbine data and model parameters