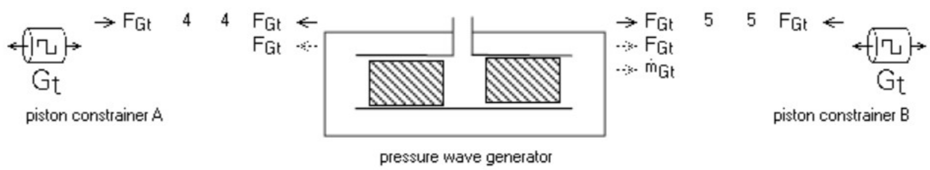
Sage Model Notes

OpposedPistonCompressor.scfn

D. Gedeon
13 December 2024

A gas pressure-wave generator that provides an AC gas flow produced by two opposed reciprocating pistons. Not to be confused with the conventional understanding of a compressor, of the type that employs check valves to rectify the flow into pulsatile DC flow.

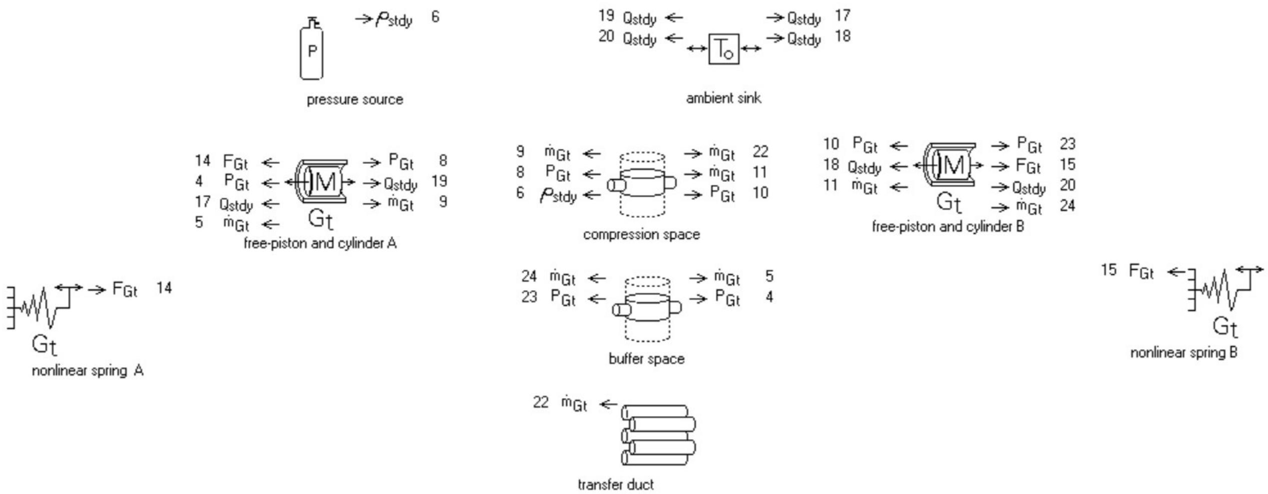
The Sage model consists of a *submodel* component renamed *pressure wave generator*, with the default submodel bitmap revised to look something like an opposed-piston compressor. Two *constrained piston* components renamed *piston constrainer A* and *piston constrainer B* force piston inside the submodel to move 180 degrees out of phase with motion determined by root model inputs `XpAmpDesign` and `XpPhsDesign`. Breaking the driver force connections allows the compressor to run in free-piston mode.



To make this model fully functional it must be copied into another model (presumable for a stirling-cycle cooler or engine) containing a mating $\dot{m}_{Gt}$ connector to attach to the one emerging from the right of the *pressure wave generator.* Typically, this connector would represent an inlet to the gas domain of a heat-rejecting heat exchanger.

The unconnected F_{Gt} connectors emerging from either side of the *pressure wave generator* allow optional linear actuator components (motors or alternators) to drive the pistons in parallel to the piston constrainers. Best practice is to leave the piston constrainers connected while designing the actuators, adjusting input voltage amplitude and phase (or current amplitude and phase), masses and springs so as to zero the forces provided by the constrainers. More on that later.

Inside of the *pressure wave generator* are quite a few components, mostly self-explanatory, appropriately renamed, with *A* and *B* suffixes when necessary to denote the two sides of the compressor:



The *pressure source* establishes the *piston compression space* time-average pressure. If pasted into another model keep in mind that only one pressure source is allowed for each contiguous gas domain.

Root level user-defined inputs establish the piston-driver motions.

```
XpAmpDesign      design piston amplitude (m)              5.000E-03
XpPhsDesign      design Xp phase (deg)                    9.000E+01
```

These will not come along for the ride if you copy and paste the compressor components into another model, so you will have to re-create them there or copy-and-paste them between models using two instances of the *Explore User-Customized Variables* dialog (Tools | Explore User Variables), using the Window's clipboard.

Piston constrainers recast inputs as follows:

A side
```
  FX = 0.000E+00...
    (XpAmpDesign) Amp
    (XpPhsDesign) Arg
```

B side
```
  PhsShift         opposed piston phase shift (deg)
1.800E+02
  FX = 0.000E+00...
    (XpAmpDesign) Amp
    (XpPhsDesign + PhsShift) Arg
```

The pressure wave generator submodel itself defines several inputs and outputs useful for recasting inputs of the components inside:

Inputs
```
  LenPisSeal       piston clearance seal length (m)         2.500E-02
  SwetBuff         buffer space wetted surface (m2)         1.000E-01
```

```
VolBuff          buffer space volume (m3)                  5.000E-04
Nsprings         number of springs each side (NonDim)      4.000E+00
Dpis             diam each piston (m)                      5.000E-02
Mpis             mass each piston (kg)                     1.000E+00
GapSeal          clearance seal gap (m)                    2.500E-05
EccenSeal        eccentricity seal 0 to 1 (NonDim)         5.000E-01
```

Outputs

```
pvPWG            net PV power                              0.000E+00
  pvCmp+pvBuff
Apis             piston frontal area                       1.963E-03
  0.25*Pi*Sqr(Dpis)
EccenFac         effective eccentric gap ratio             1.112E+00
  Power(1 + 1.5*Sqr(EccenSeal), 1/3)
```

The combination of `GapSeal`, `EccenSeal` and `EccenFac` combine to define the effective gap of the piston clearance seals based on a formulation for laminar flow between parallel plates. The ***clearance seals*** within the *free-piston and cylinder* components recast the `Gap` input to

```
Gap = EccenFac * GapSeal
```

The *free-piston and cylinder* components recast their inputs according to the above piston diameter and seal length inputs:

```
Dshell = Dpis
Length = LenPisSeal
```

Regarding the connectors emerging from the *free-pistons and cylinders*:

- F_{Gt} connectors drive the reciprocating masses (reciprocators) inside.
- P_{Gt} connectors provide volume displacements to the *compression space* and *buffer space* gas domains.
- Q_{stdy} connectors anchor the shuttle-heat transfer endpoint temperatures to the *ambient sink*.
- $\dot{m}_{Gt}$ connectors connect the *clearance seal* inlets to the *compression space* and *buffer space*.

The P_{Gt} connectors originate from *neg-facing* and *pos-facing areas* within the *reciprocator* components, which all recast their frontal-area inputs to

```
A = Apis
```

The *compression space* recasts its inputs as

```
Volume = 1.745*XpAmpDesign * 2 * Apis
Swet = 2*Apis
Length = 0.5*Dpis
```

The *compression space gas* defines a PV power output used by the *pressure wave generator*

```
pvCmp                                                      0.000E+00
  PV.Mean
  Export level: pressure wave generator
```

It implements a thick surface wall boundary condition inside so its solved temperature comes from the temperature of the gas entering from the *transfer duct* and to a lesser extent from the gas leaking through the *clearance seals*.

The *buffer space* recasts its inputs as

```
Swet = SwetBuff
Volume = VolBuff
```

These inputs are not very critical because the pressure ratio is generally small within the buffer spaces and the PV power losses low. The *buffer space gas* defines a PV power output used by the *pressure wave generator*

```
pvBuff                                                    0.000E+00
  PV.Mean
  Export level: pressure wave generator
```

It implements a thick surface wall boundary condition inside so its solved temperature comes from the temperature of the gas entering from the *clearance seals*.

The nonlinear spring components implement calculations to automatically account for the effective reciprocating mass of the spring itself by adjusting the static spring stiffness. This means there is no need to worry about effective spring mass when assigning the `Mpis` input of the *pressure wave generator*

Inputs

```
Kstatic        static spring stifffness (N/m)           8.112E+03
Fnat           1st mode resonant frequency (Hz)         1.909E+02
```

Recast

```
K0 = Nsprings * Kstatic * (1 - Sqr(Freq/Fnat  )  )
```

The presumption is that you know the 1st mode resonant frequency of the spring from finite-element modal analysis or actual experimentation and that the deflected shape of that first mode is close to the deflected shape of the spring in actual operation. Note that the effective spring stiffness `K0` approaches zero as the operating frequency approaches the spring natural frequency.

Optimization Framework

This model implements two constraints within the *piston constrainer A* component:

```
F.Cos.1 = 0
  Violated by 1.276E+03 (norm = 1.000E+02
F.Sin.1 = 0
  Violated by 2.502E+03 (norm = 1.000E+02
```

These anticipate a tuning optimization you might implement in a larger stirling-cycle model into which you paste this model. The meaning of the constraints is that the first-harmonic component of the constraining force shall be zero. Meaning that the compressor can run in free-piston mode. The reason the constraints are only implemented in one of the two piston constrainers is that the model is symmetric. If satisfied in one constrainer they will automatically be satisfied in the other.

For the optimizer to satisfy these constraints requires that you optimize some variables that give it slack to do so. For example you might optimize the `Mpis` or `Nsprings` inputs of the *pressure wave generator*. But these will not suffice because the inertial and spring forces are always in phase with the piston motion. You also need to optimize a variable that produces a perpendicular force component (in the sense of complex phasor notation) in phase with velocity, such as the voltage or current phase of a driving actuator.