# Modelica per la simulazione di sistemi e componenti oleodinamici

Paolo Ferraresi

Modelica è diventato uno standard nel campo della modellazione fisica a blocchi, e questo grazie alle sue enormi potenzialità, alla sua versatilità oltre al numero sempre crescente di utilizzatori e appassionati. Questo testo si preoccupa di favorire e impostare la creazione di un package per la modellazione di sistemi e componenti oleodinamici.



©2018 by Paolo Ferraresi

Copia disponibile

previa registrazione

al sito

*http://www.paoloferraresi.it*



Questa pagina è lasciata intenzionalmente vuota.

# 1 Indici e sommario

## 1.1 Sommario

1 Indici e sommario ..... 2
1.1 Sommario ..... 2
1.2 Indice delle figure ..... 6
1.3 Indice delle formule ..... 7
2 Modelica per la simulazione di sistemi e componenti ..... 8
2.1 Modellazione di sistemi con approccio basato sulle equazioni ..... 9
2.1.1 Sistemi di primo grado: crescita di una popolazione batterica / decadimento radioattivo ..... 9
2.1.2 Sistemi di secondo grado: oscillatore armonico ..... 10
2.2 Modellazione con blocchi acausali ..... 13
2.2.1 I modelli causali ..... 13
2.2.2 I modelli acausali ..... 14
2.2.3 L'importanza della modellazione a blocchi acausali ..... 16
3 Costruire una libreria ..... 18
3.1 Creare package e sotto-package ..... 18
3.2 Costanti ..... 20
3.3 Tipi ..... 22
3.4 Creare funzioni ..... 23
3.5 Creare porte di connessione ..... 25
3.5.1 Regola fondamentale per l'interconnessione di blocchi attraverso porte ..... 25
3.5.2 Definizione di una porta ..... 26
3.6 Costruire componenti semplici ..... 26
3.7 Buone pratiche per la scrittura di librerie ..... 27
4 Mechanical Library ..... 28
4.1 Modelli parziali d'interfaccia ..... 28
4.1.1 Classi connector ..... 28
4.1.2 Il modello Partial2Ports ..... 28
4.1.3 Il modello PartialRelativeMovements ..... 29
4.1.4 Il modello PartialViscoElastic2Ports ..... 29
4.2 La massa ..... 30
4.2.1 La classe parziale PartialMassWithIdealStopAndFriction ..... 30
4.2.2 Il modello MassWithIdealStop ..... 32
4.2.3 Esempio: caduta di un grave ..... 33

4.3 Molle e smorzatori elasto-viscosi ... 37
4.3.1 La molla ... 37
4.3.2 Smorzatore viscoso ... 38
4.3.3 Smorzatore elasto-viscoso ... 38
4.3.4 Esempio di un sistema massa-molla-smorzatore viscoso ... 39
4.4 Blocco ZeroSpeed ... 42
4.5 Blocco Force Convertion ... 42
4.5.1 Esempio: oscillatore armonico smorzato forzato ... 43
5 Hydraulic Library ... 46
5.1 Modelli parziali d'interfaccia ... 46
5.1.1 Classi connector ... 46
5.1.2 Il modello Partial2FluidPorts ... 46
5.1.3 Il modello Partial2FluidPortsWithNullNetFlow ... 46
5.1.4 Il modello Partial2FluidPortsWithVolume ... 46
5.2 Generatore di portata ideale ... 48
5.3 Sorgente di pressione ... 48
5.4 Serbatoio ... 48
5.5 Orificio ... 50
5.5.1 Esempio: Verifica di un orificio ... 51
5.6 Valvola limitatrice di pressione (lineare) ... 53
5.6.1 Esempio: schema idraulico con valvola limitatrice di pressione ... 54
5.7 Valvola di non ritorno (lineare) ... 57
5.7.1 Esempio: schemi con valvole di non ritorno ... 57
5.8 Valvola riduttrice di pressione (lineare) ... 60
5.8.1 Descrizione del modello lineare ... 60
5.8.2 Esempio: schema con valvola riduttrice di pressione (lineare) ... 61
5.9 Valvola selettrice di pressione ... 62
5.9.1 Esempio: schema con valvola selettrice di pressione ... 63
5.10 Valvola idraulica generica a 2 vie (normalmente chiusa) ... 65
5.10.1 Descrizione del modello ... 65
5.10.2 Personalizzazione delle leggi d'area ... 65
5.10.3 Codice ... 66
5.10.4 Esempio: realizzare una valvola regolatrice di flusso a 3 vie ... 69
5.10.5 Esempio: realizzare una valvola di sequenza ... 72
5.10.6 Esempio: realizzare una valvola overcenter ... 74

5.11 Valvola idraulica generica a 3 vie (normalmente aperta) ... 77
5.11.1 Descrizione del modello ... 77
5.11.2 Codice ... 77
5.11.3 Esempio: valvola riduttrice di pressione ad azione diretta ... 81
5.11.4 Esempio: valvola riduttrice di pressione ad azione pilotata ... 82
5.11.5 Esempio: valvola regolatrice di portata a 2 vie ... 84
6 Hydraulic Component Design ... 86
6.1 Modelli parziali d'interfaccia ... 86
6.1.1 La classe parziale PartialHCD2PortsWithVolume ... 86
6.1.2 La classe parziale PartialHCDEdge ... 87
6.2 Leakage ... 89
6.2.1 Parametri utilizzati nel modello ... 89
6.2.2 Modello delle perdite per trafilamento ... 89
6.2.3 Codice ... 92
6.2.4 Verifica (confronto con AMESim) ... 93
6.3 Pistone ... 97
6.3.1 Principali parametri e breve descrizione del modello ... 97
6.3.2 Codice ... 97
6.3.3 Esempio: pistone utilizzato per muovere una massa ... 98
6.4 Otturatore conico (Poppet) ... 100
6.4.1 Principali parametri ... 100
6.4.2 Codice ... 100
6.4.3 Esempio: schema minimale con otturatore conico ... 101
6.4.4 Esempio: valvola antiurto ad azione diretta ... 103
6.5 Ball Valve ... 108
6.5.1 Principali parametri ... 108
6.5.2 Codice ... 108
6.5.3 Esempio: ball valve (minimal) ... 109
6.6 AnularOrificeSpool ... 111
6.6.1 Principali parametri ... 111
6.6.2 Definire il ricoprimento ... 111
6.6.3 Codice ... 111
6.6.4 Esempio: verifica di una semplice valvola con spola e pistone di pilotaggio ... 113
6.7 GenericSpool ... 118
6.7.1 Principali parametri ... 118






6.7.2 Strategie per la parametrizzazione dell'area ..... 120

6.7.3 Codice ..... 121

6.7.4 Esempio 5.6.4 ma utilizzando GenericSpool e Underlap/Overlap gestito con $x_u$ ..... 122

6.7.5 Esempio 5.6.4 ma utilizzando GenericSpool e No Underlap/Overlap, quindi xu = 0 ..... 123

7 Listato completo ..... 124

8 Conclusione ..... 124

## 1.2 Indice delle figure

Figura 1-1: un oscillatore armonico della meccanica ..... 10
Figura 1-2: Schema a blocchi per il problema del paragrafo 1.1.2.1. ..... 13
Figura 1-3: schema a blocchi causali generico per il problema del paragrafo 1.1.2.1. ..... 13
Figura 1-4: uno schema a blocchi acausali della meccanica ..... 14
Figura 1-5: semplice circuito dell'elettrotecnica ..... 15
Figura 1-6: tipico esempio d'evoluzione di un progetto inizialmente semplice ..... 17
Figura 2-1: creazione di un package ..... 18
Figura 3-1: modello della massa con stop ideale e attrito ..... 32
Figura 3-2: legge di Hooke ..... 37
Figura 3-3: oscillatore armonico smorzato forzato ..... 43
Figura 4-1: schema Modelica per la verifica di un orificio ..... 51
Figura 4-2: schema Modelica per testare la valvola limitatrice di pressione (lineare) ..... 54
Figura 4-3: schemi con valvole unidirezionali ..... 58
Figura 4-4: schema Modelica per l'esempio su valvola riduttrice di pressione ..... 61
Figura 4-5: schema Modelica per l'esempio con valvola selettrice di pressione ..... 63
Figura 4-6: schema Modelica di una valvola idraulica generica a 2 vie normalmente chiusa ..... 65
Figura 4-7: valvola regolatrice di flusso a 3 vie / valvola limitatrice di pressione differenziale ..... 69
Figura 4-8: schema Modelica di una valvola idraulica generica a 3 vie normalmente aperta ..... 77
Figura 5-1: accoppiamento camicia-pistone ..... 89
Figura 5-2: problema del flusso nelle lastre parallele infinitamente estese ..... 90
Figura 5-3: problema del flusso nelle lastre parallele infinitamente estese di cui una in moto relativo rispetto all'altra ..... 91
Figura 5-4:Pistone accoppiato sul diametro D su camicia per un tratto L con gioco radiale $C_r$. ..... 104
Figura 5-5: schema per l'inserimento del ricoprimento nelle spole ..... 111
Figura 5-6: valvola d'esempio con spola e pistone di pilotaggio ..... 113
Figura 5-7: valvola d'esempio con spola generica (GenericSpool) e pistone di pilotaggio ..... 122

## 1.3 Indice delle formule

1-1: modello di crescita malthusiana nel caso continuo ..... 9
1-2: seconda legge di Newton applicata alla Figura 1-1 ..... 10
1-3: equazione [1-2] riscritta come sistema di equazioni di 1° grado ..... 10
3-1: velocità istantanea calcolata con x in millimetri ..... 29
3-2: seconda legge di Newton ..... 30
3-3: seconda legge di Newton per un grave in presenza d'attrito viscoso ..... 35
3-4: soluzione particolare alle C.I. nulle per la caduta di un grave in presenza d'attrito viscoso ..... 35
3-5: seconda legge di Newton per oscillatore armonico smorzato-forzato ..... 40
3-6: Oscillatore armonico smorzato ..... 40
3-7: oscillatore armonico smorzato forzato ..... 44
4-1: equazioni per un bipolo idraulico ..... 46
4-2: legge dello strozzatore in regime turbolento ..... 50
4-3: velocità massima del fluido in un orificio ..... 50
4-4: numero di Reynolds in un orificio o per una sezione qualunque ..... 50
4-5: coefficiente d'efflusso unificato laminare-turbolento ..... 50
4-6: velocità media del fluido in un orificio ..... 50
4-7: modello di una valvola limitatrice di pressione lineare ..... 53
4-8: caratteristica lineare (grado di apertura) di una valvola riduttrice di pressione ..... 60
5-1: metodo per il calcolo del ricoprimento ..... 87
5-2: forza d'attrito dovuta alla viscosità ..... 90
5-3: moto entro piastre parallele, soluzionge generale ..... 90
5-4: velocità del fluido nel moto tra due lastre parallele ..... 91
5-5: portata che scorre tra lastre parallele senza moto relativo ..... 91
5-6: gradiente di pressione in lastre parallele ad altezza costante ..... 91
5-7: andamento della pressione tra lastre parallele al altezza costante ..... 91
5-8: forza d'attrito viscoso nel moto tra due piastre parallele senza moto relativo ..... 91
5-9: velocità del fluido nel moto tra due lastre parallele, di cui una in moto relativo ..... 92
5-10: portata che scorre tra lastre parallele di cui una in moto relativo ..... 92
5-11: forza d'attrito viscoso nel moto tra due piastre parallele di cui una in moto relativo ..... 92
5-12: legge della pressione in un volume ..... 97
5-13: forza in apertura di un otturatore conico (poppet) ..... 100
5-14: forze di flusso su un otturatore conico (poppet) ..... 100
5-15: determinazione corretta del coefficiente $k_{jet}$ quando angolo del cono e angolo d'efflusso sono disgiunti ..... 103
5-16: attrito viscoso di un pistone immerso in un fluido ..... 104
5-17: coefficiente di smorzamento viscoso dato dal travaso di fluido ..... 104
5-18: forza in apertura di un otturatore a sfera (ball valve) ..... 108
5-19: forza in apertura nel risalto di una spola ..... 111
5-20: diametro idraulico equivalente per un orificio ..... 118
5-21: forza in apertura nel risalto di una spola ..... 119

# 2 Modelica per la simulazione di sistemi e componenti

Modelica è un **linguaggio di modellazione di sistemi a parametri concentrati**, **dichiarativo**, **basato su equazioni**, orientato agli oggetti.
E' un linguaggio formale utilizzabile per descrivere (**modellare**) sistemi di diversa natura, in vari domini della fisica (*meccanica, elettromagnetismo, termodinamica, fluidodinamica*, ecc.) anche simultaneamente (*multi domain analysis*) ad esempio un sistema elettroidraulico.
Modelica garantisce **efficienza** ed **elevate prestazioni** per le simulazioni poiché il codice è in seguito compilato, normalmente in linguaggio C.
*Nato nel 1997, inventato da Hilding Elmqvist, ha alla base lo sviluppo di un linguaggio object-oriented per la modellazione di sistemi tecnico-scientifici col fine di favorire lo sviluppo e il riuso di modelli per descrivere sistemi complessi. Potenzialmente è in grado di fare tutto quello che fanno software commerciali per la simulazione di sistemi fisici, come ad esempio LMS Imagine.Lab AMESim*. La differenza tra Modelica e le soluzioni commerciali spesso è data dal corredo (cioè dalla presenza dopo l'installazione) di librerie particolari, pronte all'uso, per la modellazione in campi specifici.
*Allo stato attuale, infatti, AMESim dispone di vaste librerie per la modellazione di componenti e sistemi oleodinamici, mentre Modelica è distribuito senza alternative valide. Esistono tuttavia librerie di terze parti più o meno valide*.
**In quest'articolo s'intende presentare le tecniche che portano alla realizzazione di una libreria per l'oleodinamica semplice ma aderente agli scopi fondativi di Modelica, vale a dire fornire soluzioni per lo sviluppo e il riuso di componenti**.



In pratica **l'obiettivo di questo testo è far vedere come impostare la creazione di una libreria oleodinamica per Modelica**.
**Questo non è un tutorial che insegna a usare Modelica, la cui conoscenza è data per acquisita.**

---

**Gli elementi del linguaggio sono tuttavia così semplici che per iniziare basterà guardare qualche esempio**.
***Naturalmente per usare al meglio Modelica è necessario leggere attentamente la documentazione e studiare su qualche libro di riferimento***, a tal proposito consiglio:

- *Introduction to Physical Modeling with Modelica* – Michael Tiller.
- *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica* – Peter Fritzson.

---

Poiché Modelica è un linguaggio, per programmare (modellare/simulare) è indispensabile utilizzare un ambiente di sviluppo. Ne esistono tanti: **Dymola** (***Dassault Systems***), **SimulationX** (***Maplesoft***), **JModelica**, **OpenModelica**, ecc. Questi ambienti offrono anche possibilità "grafiche" per interconnettere i vari blocchi e per crearne di nuovi.
In particolare lavoreremo con OpenModelica perché è strumento Open Source (*Free*, anche per scopi commerciali) e quindi è una soluzione il cui uso è certamente da incentivare.

## 2.1 Modellazione di sistemi con approccio basato sulle equazioni

Poiché **Modelica** è un linguaggio basato sulle equazioni**, il metodo più immediato e naturale per modellare, è sicuramente quello di scrivere direttamente le equazioni costitutive del modello**: Modelica è in grado di risolvere numericamente sistemi di equazioni algebriche e differenziali (DAE). Si tratta quindi di elencare le equazioni, definire eventuali parametri e condizioni iniziali e questo è possibile senza dover comprendere linguaggi di programmazioni complicati.
A scopo didattico, iniziamo con due problemi molto semplici, volutamente risolvibili analiticamente, così da prendere confidenza con il linguaggio e poter verificare la correttezza dei risultati.

### 2.1.1 Sistemi di primo grado: crescita di una popolazione batterica / decadimento radioattivo

Uno dei modelli di crescita delle popolazioni si basa sull'ipotesi (*Malthus, 1798*) che quest'ultime crescano a un **tasso proporzionale al numero d'individui**. Tale ipotesi è ragionevole per popolazioni di batteri o di animali in condizioni ideali (*ambiente illuminato, temperatura ottimale, nutrimento adeguato, assenza di predatori, immunità dalle malattie*).
Se *t* è il tempo (variabile indipendente), N(t) è il numero di batteri all'istante *t*, abbiamo:

$$\frac{d}{dt}N(t) = k \cdot N(t)$$

2-1: modello di crescita malthusiana nel caso continuo

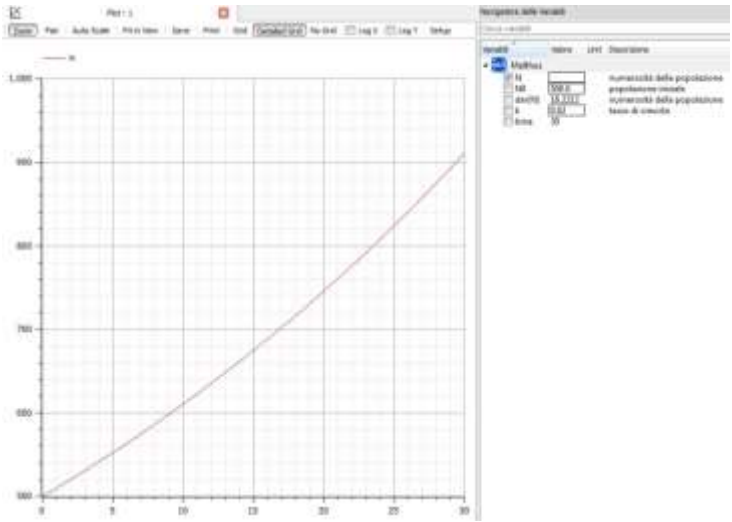
Dove *k* è il tasso di crescita. Questo modello risponde a problemi del tipo seguente: *una coltura di batteri cresce con una rapidità proporzionale al numero di batteri presenti k = 0.02. Se la coltura contiene inizialmente 500 batteri qual è l'evoluzione nel tempo del numero d'individui?*
La struttura del programma scritto con **Modelica** che risolve questo problema è molto semplice.

```
model Malthus
  parameter Real k = 0.02 "tasso di crescita";
  parameter Real N0(unit="unit",min=0) = 500 "popolazione iniziale";
  Real N(unit="unit",start=N0) "numerosità della popolazione";
equation
  der(N) = k*N; // definisce l'equazione: dN/dt = k*N
end Malthus;
```

In pratica, il listato, **dichiara i parametri *k* e $N_0$, la variabile dipendente *N*** e il suo valore iniziale (*start* = $N_0$).
Quindi definisce le equazioni (in questo caso una soltanto).
Il parametro di dichiara con la keyword `parameter`.
Le equazioni vanno inserite dopo la keyword `equation`.

La derivata nel tempo si indica con **der**(***nomeVariabile***), quindi der(N) è dN/dt.

Se impostiamo le opzioni (ad esempio integriamo per 30s) e lanciamo la simulazione, parte la doppia compilazione (da Modelica a C e da C ad eseguibile) e, automaticamente partono i calcoli. Alla fine (*il tempo dipende dalla complessità del problema, dalla durata della simulazione, ad esempio non qua abbiamo messo 30 secondi, dalla precisione e dall'algoritmo numerico*) ci troviamo nella finestra dei risultati, se selezioniamo N, vediamo il grafico.

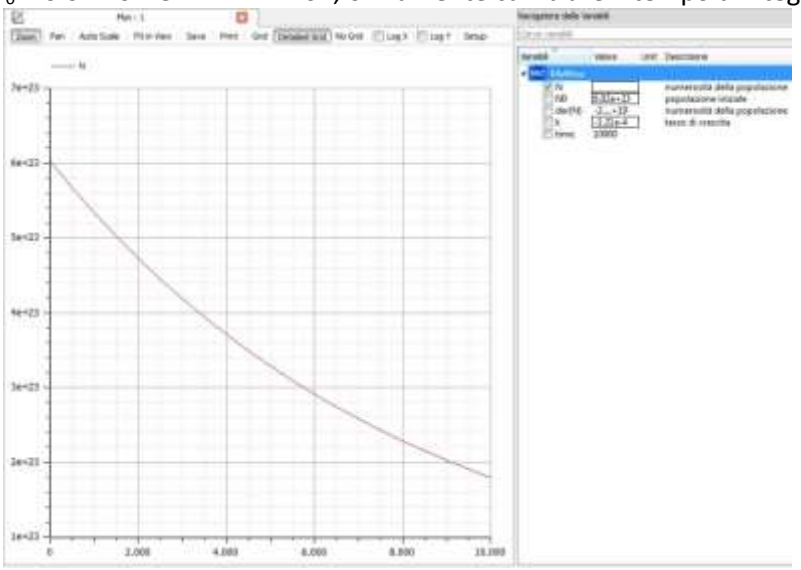


Possiamo verificare facilmente il risultato, ricordando che la soluzione "esatta" dell'equazione differenziale di primo grado del nostro modello è:

$$N(t) = N_0 \cdot e^{kt}$$

Avendo definito $N_0$ e k come parametri, posso modificarne il valore e rieseguire la simulazione senza dover modificare il programma o dover ricompilare.

Ad esempio, poiché **lo stesso modello risponde alla legge del decadimento radioattivo**, potrei simulare anche un problema del tipo: *al tempo t = 0 ho un certo numero di atomi di Carbonio 14, ad esempio $N_0$ = $6.02 \cdot 10^{23}$ atomi, quanti atomi avrò nel tempo (sapendo che k = $-1.21 \cdot 10^{-4}$)?*
Basterà modificare $N_0 = 6.02 \cdot 10^{23}$ e $k = -1.21 \cdot 10^{-4}$, ovviamente cambiare il tempo d'integrazione e ottenere:



Possiamo notare che dopo il tempo di dimezzamento $t_{1/2} = 5730$ anni, la quantità di atomi è la metà.

### 2.1.2 Sistemi di secondo grado: oscillatore armonico.

In Figura 2-1, è riportato un sistema fisico, costituito da una massa, da una molla e da uno smorzatore.

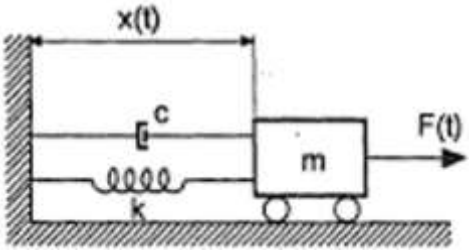



Figura 2-1: un oscillatore armonico della meccanica

Il modello di calcolo per questo sistema è quello dell'oscillatore armonico smorzato forzato.

Immaginiamo di dover risolvere il seguente problema: studiare il sistema di Figura 2-1 con massa m = 2 kg, costante elastica k = 800 N/m, coefficiente d'attrito viscoso c = 1000 N·s/m. All'istante t=0 il sistema è in quiete. F(t) ha un andamento sinusoidale di ampiezza 100 N e frequenza f = 1 Hz.

Se tralasciamo l'attrito delle rotelle e dell'aria e consideriamo la massa come puntiforme, l'equazione differenziale di secondo grado che descrive il modello si determina applicando la *seconda legge di Newton*:

$$m\frac{d^2x}{dt^2} = f(t) - c\cdot\frac{dx}{dt} - k\cdot x$$

2-2: seconda legge di Newton applicata alla Figura 2-1

O equivalentemente, il sistema fisico è descritto dal sistema di equazioni differenziali di primo grado:

$$\begin{cases} \dfrac{dx}{dt} = v \\ m\dfrac{dv}{dt} = f(t) - c\cdot v - k\cdot x \end{cases}$$

2-3: equazione [2-2] riscritta come sistema di equazioni di 1° grado

### 2.1.2.1 Soluzione tramite funzione di trasferimento (per sistemi lineari)

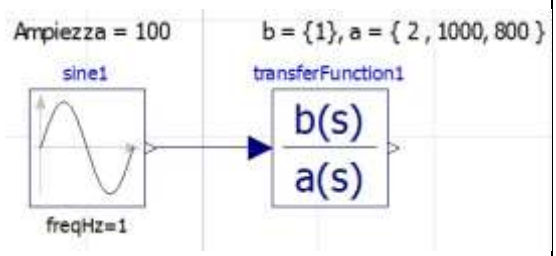
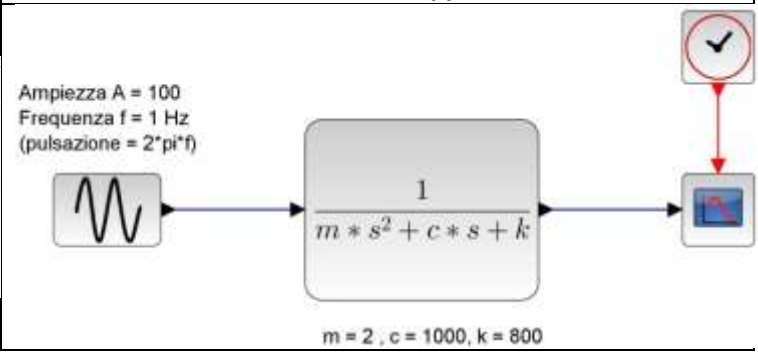
Prima di risolvere il problema con Modelica, cerchiamo una soluzione di cui siamo sicuri per poi verificare il risultato ottenuto.
La soluzione analitica è possibile ma non banale; al momento la omettiamo (*faremo i calcoli per verificare un esercizio al paragrafo 4.5.1*) poiché ci porta fuori dal discorso principale.
Vogliamo tuttavia verificare i risultati e lo possiamo fare abbastanza agevolmente poiché il modello è destritto da **equazioni differenziali ordinarie lineari a coefficienti costanti**, e quindi possiamo utilizzare la **trasformata di Laplace** per determinare la **funzione di trasferimento** (*f.d.t.*).
L-trasformiamo l'equazione [2-2] alle C.I. nulle:

$$ms^2X(s)+csX(s)+kX(s)=F(s)$$

Da cui ricaviamo la funzione di trasferimento:

$$G(s)=\frac{X(s)}{F(s)}=\frac{1}{ms^2+cs+k}=\frac{1}{2\cdot s^2+1000\cdot s+800}$$

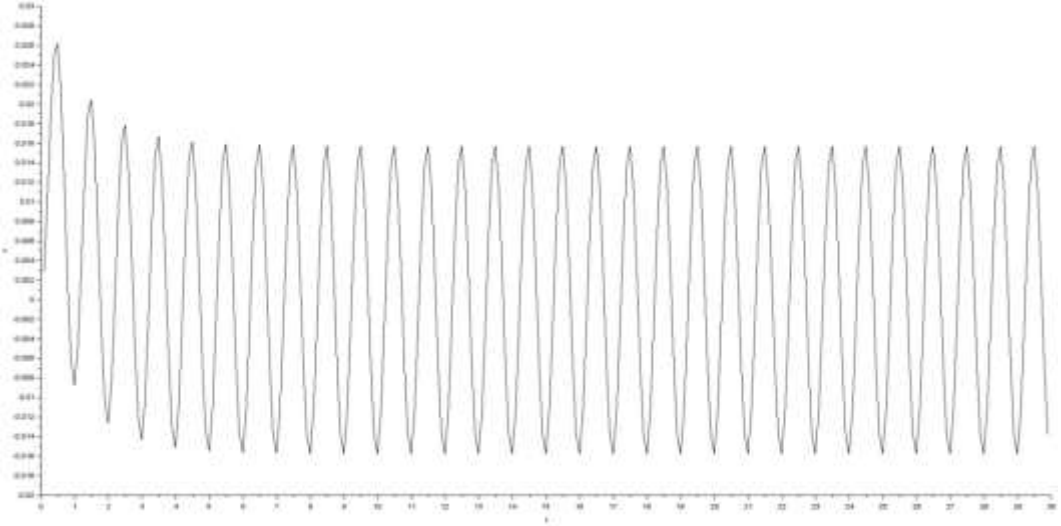
Ora possiamo tracciare un semplice schema a blocchi (con ***Simulink*** o ***Scilab*** o con lo stesso ***OpenModelica***) con un generatore (che rappresenta la Forza) e con un blocco *f.d.t*.

| Schema a blocchi fatto con OpenModelica | Schema a blocchi come appare con Scilab |
|---|---|
|  |  |





Se lanciamo la simulazione con il programma che preferiamo (*io per gli schemi a blocchi funzionali preferisco Scilab*) otteniamo:



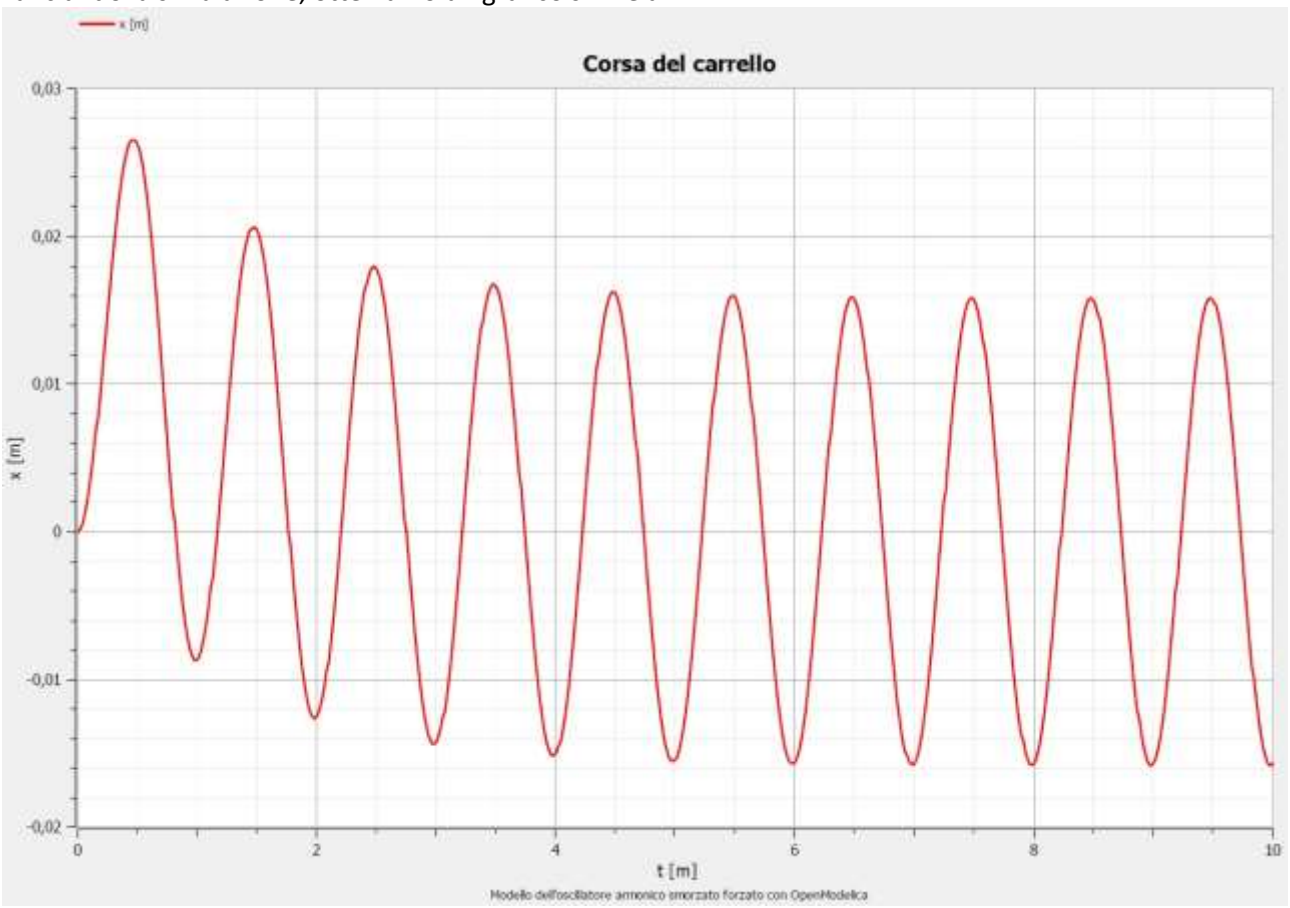
Questa è la soluzione numerica del nostro problema.

### 2.1.2.2 Soluzione tramite scrittura diretta delle equazioni

Scriviamo il sistema di equazioni [2-3] direttamente in codice Modelica:

```
model Oscillatore
  parameter Modelica.SIunits.Mass m = 2.0 "Mass";
  parameter Modelica.SIunits.TranslationalSpringConstant k(displayUnit="N/mm")=800.0 "Spring rate";
  parameter Modelica.SIunits.TranslationalDampingConstant c=1000.0 "Damping constant";
  parameter Real spread(unit="-", min = 0)  = 100 "Spread of F(t)";
  parameter Modelica.SIunits.Frequency f = 1 "Frequency";
  Modelica.SIunits.Length x(start=0, displayUnit="mm") "Stroke";
  Modelica.SIunits.Velocity v(start=0, displayUnit="km/h") "Velocity";
equation
  der(x)=v;
  m*der(v)=spread*sin(2*Modelica.Constants.pi*f*time)-c*v-k*x;
end Oscillatore;
```

Abbiamo introdotto le grandezze del SI (**Modelica.SIunits**) e la costante π (**Modelica.Constants.pi**).
Per risolvere questo esempio, le impostazioni per la simulazione potrebbero essere: **istante iniziale** = 0, **istante finale** = 10, **intervalli** = 1e-3, **metodo** = “rungekutta”, **tolleranza** = 1e-6.
Lanciando la simulazione, otteniamo un grafico simile a:




Ho settato anche il titolo del grafico, degli assi e ho aumentato un po’ lo spessore della linea.

## 2.2 Modellazione con blocchi acausali

### 2.2.1 I modelli causali

I normali schemi a blocchi funzionali, pensiamo a Simulink o Scilab, utilizzano un approccio causale, ovvero gli ingressi e le uscite sono sempre ben distinte.
Se ricordate, nel paragrafo 2.1.2.1, per calcolare la corsa avevamo trovato la *f.d.t.* e utilizzato uno schema a blocchi, tra cui una sorgente di segnale e un blocco *f.d.t.* di Scilab.

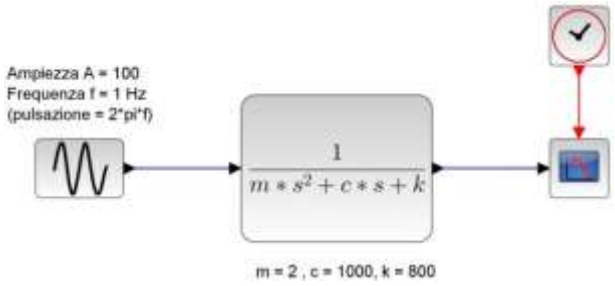



Figura 2-2: Schema a blocchi per il problema del paragrafo 1.1.2.1.

Il generatore di segnale invia il suo output alla *f.d.t.*, lo elabora, e lo invia lo blocco successivo.
Questo schema a blocchi è detto **causale** perché **il flusso del calcolo va sempre lungo le frecce**, passando da ciascun blocco lungo la direzione che va dalle porte di input alle porte di output. Ogni blocco che effettua calcoli, riceve in ingresso l'uscita del blocco che lo precede e invia la sua uscita all'ingresso del blocco che gli succede.
Se avessimo voluto modellare il problema con uno schema a blocchi senza utilizzare direttamente la *f.d.t.* avremmo potuto anche utilizzare lo schema seguente:

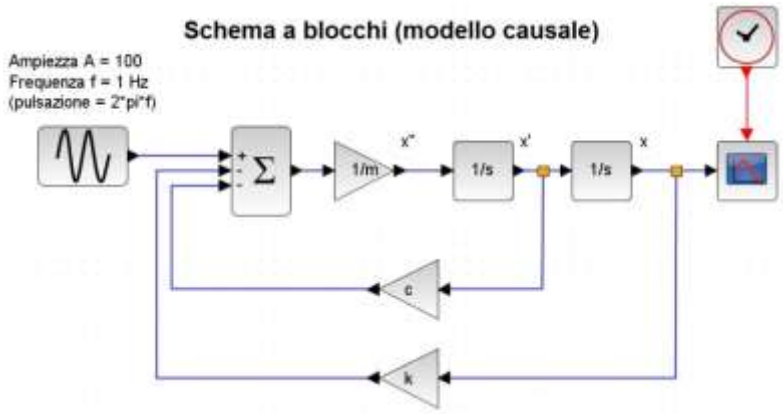



Figura 2-3: schema a blocchi causali generico per il problema del paragrafo 1.1.2.1.

Lo schema in Figura 2-3 in parole povere funziona così: dal nodo sommatore escono +F-c·x'-k·x, se pensiamo all'equazione [2-2], questo è uguale a m·x'', quindi il blocco dopo, dividendo per m, restituisce x'', il blocco dopo integra e quindi otteniamo x', quello dopo ancora lo integra e otteniamo finalmente x.
Questo è ancora uno schema causale, perché è evidente che ogni blocco aspetta una informazione da quello che lo precede, la elabora e la invia al blocco successivo. Gli ingressi e le uscite sono ben distinte (ad esempio nel blocco sommatore entrano F, c·x' e k·x ed esce m·x''). Il flusso del calcolo segue le interconnessioni dei blocchi (*m·x'' diventa x attraverso una divisione per m e una doppia integrazione*).
***Entrambi gli schemi hanno pro e contro: il primo è molto semplice (se conosco la f.d.t.) nel secondo devo interconnettere più blocchi e devo comunque conoscere il flusso dei calcoli.***

I limiti degli schemi a blocchi funzionali sono proprio questi:

1. L'uso della *f.d.t.* è pratico ed elegante ma confinanto in pratica per sistemi lineari e non molto complessi.
2. Anche senza l'uso della *f.d.t.* è necessaria un'analisi e una modellazione del flusso delle informazioni.
3. Difficoltà di simulare sistemi dove si scambia potenza.

Il punto 3, che è forse il limite più consistente, merita un esempio: pensiamo a una pompa oleodinamica che fornisce una certa portata. Mentre a vuoto la pressione è idealmente nulla, una volta che raggiunge una qualunque resistenza idraulica, l'informazione della pressione ritorna alla pompa. Infatti, non è la pompa a generare la pressione, quest'ultima si determina altrove, tuttavia "ritorna" alla mandata della pompa e proviene dalle resistenze che la prima incontra. La gestione di questa ***bidirezionalità tra i fattori della potenza*** è difficile da gestire in sistemi complessi nei blocchi causali. Siccome in oleodinamica si ha a che fare con scambi di potenza continuamente, va da sé che i sistemi oleodinamici con un buon numero di componenti non sono facili da modellare con schemi a blocchi funzionali causali.

### 2.2.2 I modelli acausali

La modellazione **acausale** è uno stile di modellazione dichiarativa, sulla base di equazioni piuttosto che d'istruzioni con un flusso ben definito. Le equazioni possono essere contenute nei blocchi oppure semplicemente elencate tramite un qualche linguaggio (ad esempio Modelica).
Le equazioni non devono specificare quali variabili siano ingressi o uscite, a differenza dei blocchi causali.
**Le equazioni**, dunque, **sono per loro natura acausali**.
**I blocchi acausali sono semplicemente gruppi di equazioni che descrivono una porzione del modello fisico d'interesse** e che si possono interconnettere mediante semplici regole (che vedremo) per cui, dal punto di vista del risultato finale, la scrittura diretta delle equazioni oppure la composizione di blocchi acausali produce lo stesso identico risultato. **Parleremo quindi di modelli acausali indistintamente nei due casi**.
Nei modelli acausali il flusso dei calcoli è delegato al codice (solver) che risolvere numericamente il sistema DAE.
***Nella modellazione acausale ci limitiamo a descrivere un sistema fisico senza interessarci alle modalità con cui esso verrà risolto, senza decidere cosa sia input e cosa sia output, ma ci possiamo concentrare sulle equazioni o sotto modelli costituenti.***

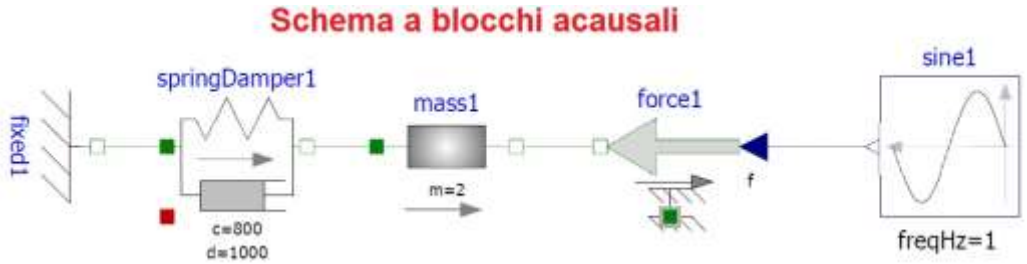
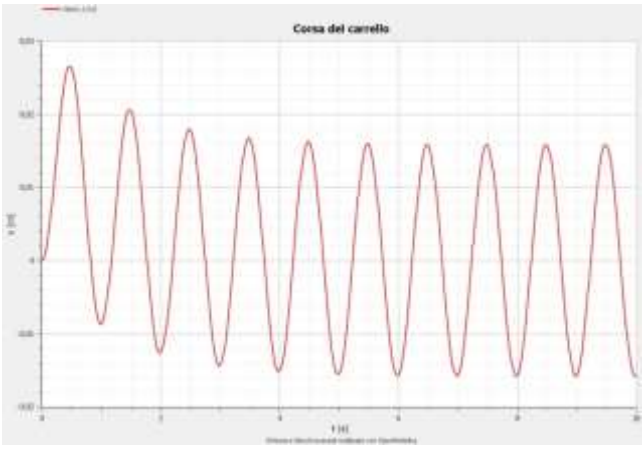



Figura 2-4: uno schema a blocchi acausali della meccanica

In Figura 2-4 vediamo il sistema descritto al paragrafo 2.1.2.
Notiamo subito che c'è una grande corrispondenza con la figura Figura 2-1, e questo è già un vantaggio poiché la modellazione diventa più naturale se utilizziamo sotto modelli elementari noti per creare qualcosa di più complesso. Al momento della compilazione, OpenModelica, crea un sistema di equazioni simile a quello del paragrafo 2.1.2.2. Lanciando la simulazione, se abbiamo inserito gli stessi parametri, otteniamo lo stesso risultato.



Ecco i principali vantaggi di quest'approccio:

1. Non è necessario conoscere la f.d.t.
2. Non è necessario definire un flusso di calcolo.
3. **Se ho utilizzato blocchi acausali, diventa molto facile modificare lo schema**.

Per capire meglio il punto 3, immaginiamo d'inserire una molla in serie al blocco springDamper1. E' molto facile: stacchiamo il collegamento, inseriamo un blocchettino con la molla, lo riconnettiamo con il blocco di sinistra e la massa. Serve un minuto. Se dovessimo invece modificare le equazioni, sarebbe più complicato. Per essere sicuri di non fare errori dovremmo pensarci sicuramente più di un minuto.
Per capire meglio come opera la modellazione con i blocchi acausali, proviamo ad imitare OpenModelica per un semplicissimo schema dell'elettrotecnica. Pensiamo ad esempio a un generatore di tensione costante V, resistenza R e messa a terra G.

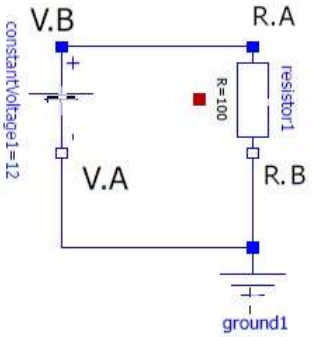



**Figura 2-5: semplice circuito dell'elettrotecnica**

**Modelica** esprime questo modello con 17 equazioni e altrettante variabili.
Noi risolveremo lo schema di Figura 2-5 con 5 equazioni e 5 incognite.
*Il fatto che i blocchi di Modelica siano così ricchi di variabili ed equazioni non ci deve stupire negativamente, anzi ci dovrebbe rallegrare: sono più ricchi d'informazioni e permettono di tenere conto ad esempio della temperatura e della sua influenza sulla resistività, dell'energia trasformata in calore per effetto Joule. Se non siamo interessati a tutte queste variabili, ci basta ignorarle quando creiamo i grafici con i risultati della simulazione*.
Naturalmente il sistema è elementare e la corrente si calcola immediatamente come i = V/R, tuttavia è istruttivo capire come si comporta Modelica.
Per noi le incognite sono la tensione (.v) nei punti V.B, V.A, R.B e R.A e la corrente che attraversa la maglia.
Abbiamo quindi un sistema di 5 equazioni e altrettante incognite, determinato che ammette una soluzione:

$$\begin{cases} V.B.v - V.A.v = 12 \\ R.A.v - R.B.v = R.R \cdot i \\ V.B.v = R.A.v \\ V.A.v = R.B.v \\ V.A.v = 0 \end{cases} \rightarrow \begin{cases} V.B.v = 12 \\ V.A.v = 0 \\ R.B.v = 0 \\ R.A.v = 12 \\ R.A.v = R.R \cdot i \end{cases} \rightarrow \begin{cases} V.B.v = 12 \\ V.A.v = 0 \\ R.B.v = 0 \\ R.A.v = 12 \\ i = \dfrac{R.A.v}{R.R} = \dfrac{12}{100} = 0.12A \end{cases}$$

Modelica è in grado automaticamente di passare dallo schema di Figura 2-5 a un sistema di equazioni DAE (*in questo caso non abbiamo equazioni differenziali*) ed ha la capacità di risolverlo numericamente.
Una volta che abbiamo un software che a partire dallo schema di Figura 2-5 è in grado di costruire un sistema di equazioni e risolverlo, la modellazione a blocchi causali appare meno conveniente.

### 2.2.3 L'importanza della modellazione a blocchi acausali

Abbiamo realizzato il calcolo di un oscillatore armonico smorzato forzato (Figura 2-1) in vari modi:

1) Attraverso la f.d.t (che è uno schema a blocchi causali, vedi 0).
2) Con uno schema a blocchi causali generico (Figura 2-3)
3) Con un programmino scritto con **Modelica** (**Equation Based Approach**, 2.1.2.2).
4) Con un modello a blocchi acausali con **Modelica** (**Component Model Approach**, Figura 2-4).

Per nessuno di questi modi è stato complicato arrivare a una soluzione.
In particolare, i metodi descritti nei punti 3) e 4) sembrano molto semplici; il programmino scritto al paragrafo 2.1.2.2, a parte la dichiarazione dei parametri e delle variabili è costituito da un elenco di due equazioni:

```
der(x) = v;
m*der(v) = 100*sin(2*Modelica.Constants.pi*f*time)-c*v-k*x;
```

Questo sembra ancora più semplice che scegliere i blocchi dalla vastissima libreria di OpenModelica, per costruire lo schema di Figura 2-4, connetterli e impostare i parametri di ciascun blocco.
Ci sono ragioni che giustificano la modellazione scrivendo direttamente le equazioni (**Equation Based Approach**):

1) **Soltanto attraverso le equazioni comprendiamo la vera natura fisica di un sistema**, quindi lo studio tramite la scrittura delle equazioni, da questo punto di vista, andrebbe incoraggiato.
2) Particolarmente adatto per sistemi costituiti da poche equazioni. Utile anche in sistemi più grandi se una volta scritti non si devono più modificare (a parte i parametri). La modifica, infatti, coinvolge la revisione di tutte o gran parte delle equazioni e risulta inefficiente e poco sicura.

Ci sono, viceversa, ragioni che giustificano la modellazione secondo il **Component Model Approach**:

A. L'interconnessione di più blocchi "semplici" permette di realizzare schemi molto complessi, facilmente. Il tempo necessario a ultimare uno schema di medie o grandi dimensioni risulta estremamente ridotto rispetto al tempo necessario per descrivere lo stesso modello mediante la scrittura diretta delle equazioni.
B. I modelli ottenuti sono facilmente modificabili. E' semplice aggiungere, rimuovere o modificare componenti. Il sistema di equazioni è aggiornato automaticamente da Modelica, senza correre il rischio di cancellare o corrompere equazioni per sbaglio.
C. L'uso e il riuso di blocchi collaudati favorisce simulazioni affidabili e limita le possibilità d'errore.

*Per contro, i blocchi delle librerie, come abbiamo già discusso al paragrafo 2.2.2, tendono a portarsi dentro un maggior numero di variabili (ed equazioni) per cui i sistemi generati dagli schemi risultano, in genere, più "pesanti" (più ricchi di dati). Con i moderni microprocessori questo raramente si traduce in un calo delle prestazioni di calcolo, poiché Modelica non ha problemi a gestire anche centinaia di migliaia di equazioni se la quantità di memoria RAM è adeguata. Il maggiore fastidio potrebbe essere dato dal fatto di dover cercare le variabili da graficare in un mare di grandezze calcolate.*

Considerando i pro e i contro, è sicuramente più produttivo e "sicuro" modellare con il **Component Model Approach**, tuttavia quando ci sono le condizioni, non escluderei la possibilità di scriversi le equazioni (**Equation Based Approach**), per via del punto 1).

Nella pratica spesso è necessario modificare e analizzare schemi più volte: nulla è più produttivo del **Component Model Approach**.

Facciamo un semplice esempio di cosa normalmente accade a un progetto.
In Figura 2-6 abbiamo uno schema di partenza molto semplice, che potrebbe essere facilmente modellato sia con uno schema a blocchi Modelica sia direttamente scrivendo le uniche due equazioni algebriche.

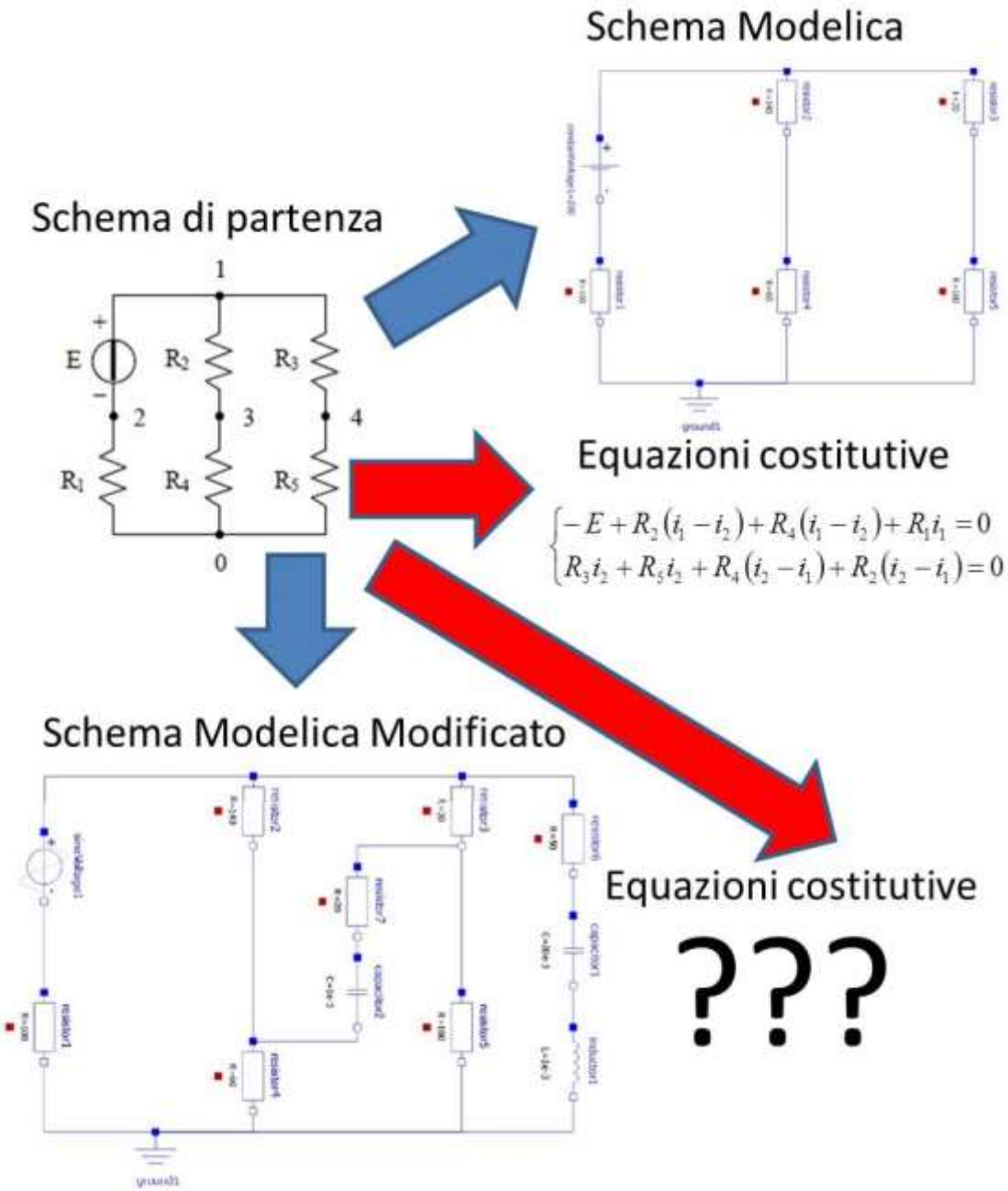



**Figura 2-6: tipico esempio d'evoluzione di un progetto inizialmente semplice**

Facilmente accade che la simulazione fornisce un risultato diverso da quello che volevamo e lo schema iniziale viene modificato e risulta naturalmente è più complesso.
A questo punto, benché sia ancora possibile scrivere direttamente le equazioni, ci rendiamo conto che non è conveniente descrivere i sistemi con le equazioni se poi c'è il rischio di doverli modificare più volte.

# 3 Costruire una libreria

## 3.1 Creare package e sotto-package

Il primo passo è creare un package (*un particolare tipo di classe Modelica*), che in generale rappresenta una collezione di classi Modelica tra cui altri sotto-package, costanti, funzioni, blocchi, modelli e altre classi costituite nel linguaggio Modelica.

Per creare un Package con OpenModelica è necessario fare click sul pulsante "**Nuova classe Modelica**" (***il simbolo del foglio vuoto con il segno +***).

Quindi si crea un package (specializzazione: **Package**) che nel nostro esempio, chiameremo *FPLib*.

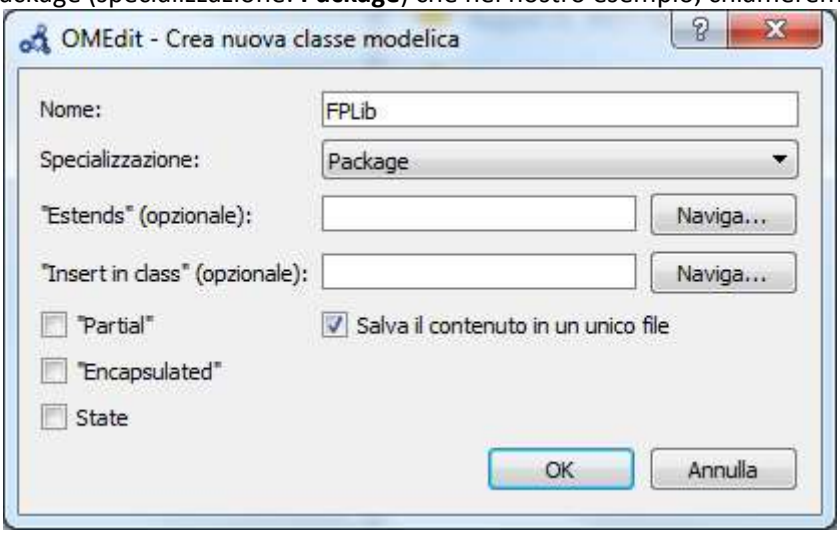



Figura 3-1: creazione di un package

Ora abbiamo creato un package vuoto:

```
package FPLib
end FPLib;
```

Il package è aggiunto nella struttura di sinistra, detta **Navigatore di librerie**, proprio sotto alla libreria standard di Modelica. L'icona assegnata inizialmente al package è l'icona di default con la lettere P.
Se vado con il mouse sul package appena creato e premo il tasto destro, selezionando "**Nuova classe Modelica**" posso creare classi all'interno del mio package, tra cui altra package detti sotto-package.
E' bene strutturare una libreria in package e sotto-package in modo da tenere i vari moduli distinti.
***Io consiglio di creare, al momento, i seguenti sotto-package:***

1. Constants
2. Types
3. Functions
4. Signals
5. Mechanics
6. Hydraulic
7. HCD
8. Icons

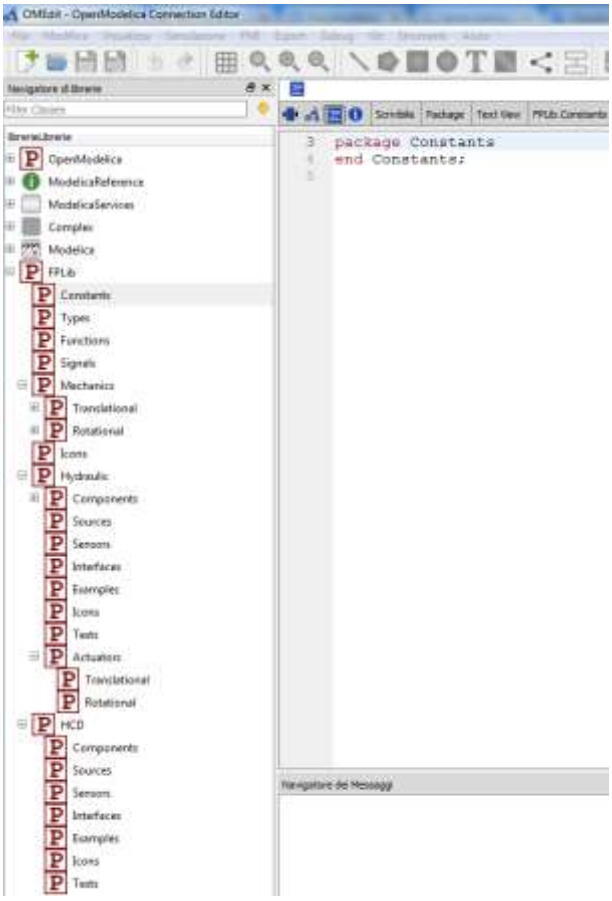
Il package Signals a sua volta lo dividere in almeno due package chiamati Continous e Sources.
Mechanics lo dividere in Translational e Rotational. Dentro ciascuno di questi creerei altri sotto-packages chiamati Components, Sources, Sensors, Interfaces, Examples, Icons e Tests.
Dentro Hydraulic andrei a creare Actuators, Components, Sources, Sensors, Interfaces, Examples, Icons e Tests. Dentro a Components dividerei ulteriormente in Resistances, Valves e DCV.
HCD è una libreria a un livello più basso rispetto a Hydraulic, per la realizzazione di componenti/valvole di qualsiasi tipo, simile alla libreria omonima presente dentro AMESim. Dentro HCD metterei ancora: Components, Sources, Sensors, Interfaces, Examples, Icons e Tests.

La struttura dei sotto-package dovrebbe diventare simile alla seguente:

Constants
Types
Functions
Signals
|_Continous
|_Sources
Mechanics
|_Translational
|__Components
|__Sources
|__Sensors
|__Interfaces
|__Expamples
|__Icons
|__Tests
|_Rotational
|__Components
|__Sources
|__Sensors
|__Interfaces
|__Expamples
|__Icons
|__Tests
Hydraulic
|_Actuators
|__Translational
|__Rotational
|_Components
|__Valves
|__DCV
|__Resistances
|_Sources
|_Sensors
|_Interfaces
|_Expamples
|_Icons
|_Tests
HCD
|_Components
|_Sources
|_Sensors
|_Interfaces
|_Expamples
|_Icons
|_Tests
Icons

Ovviamente questa è una scelta arbitraria, nessuno vieta di suddividere le varie classi con altri criteri.
Ora la finestra di OpenModelica si presenta pressappoco come rappresentato nell'immagine sotto a destra, con l'albero dei package nel navigatore di librerie.
Una volta fatto doppio click su un package, nella parte destra dello schermo compare una sua vista.
La vista è una modalità selezionabile da ciascuno dei quattro pulsanti nell'angolo in alto a sinistra della finestra di destra.






Immediatamente dopo la creazione di un package o di qualsiasi altro tipo di classe ciascuna vista è vuota, a parte la modalità testo (che vediamo già nell'immagine a destra), dove c'è soltanto la definizione della classe in lingaggio Modelica.
Nei quattro pulsanti delle viste, da sinistra, il primo è la "Vista icona". Nella vista icona possiamo disegnare l'icona utilizzata per riconoscere visivamente l'oggetto che andiamo a creare, ad esempio se devo modellare una valvola limitatrice di pressione, sarebbe bene disegnarne in vista icona il simbolo.
Il secondo è la "Vista diagramma": qua vedo l'eventuali interconnessioni tra più blocchi "semplici" che costituiscono la classe che sto andando a creare. Per i package questa non ha senso ma, se devo realizzare ad esempio una valvola antiurto e riempimento a partire da una valvola limitatrice di pressione e una valvola unidirezionale, supponendo che siano già preesistenti, allora posso semplicemente creare il mio modello di valvola antiurto, andare in vista diagramma e inserire i due modelli menzionati, collegarli opportunamente, ottenendo di fatto un nuovo modello senza dover scrivere alcuna riga di codice.
Il terzo è la vista "Text View" e in pratica sono le righe in linguaggio Modelica che costituiscono la classe sulla quale sto lavorando. Ad esempio, nell'immagini precedente, avevo fatto doppio click sul package Constants, e selezionado "Text View" vedo la definizione in linguaggio modelica di un package vuoto, chiamato appunto Constants:

```
package Constants
end Constants;
```

Il quarto mostra la documentazione per quella classe, se inserita correttamente, secondo le modalità previste dal linguaggio Modelica.

Nei prossimi paragrafi vedremo come “popolare” ciascun package in modo da sviluppare una libreria, in questo caso orientata alla simulazione di componenti e sistemi oleodinamici.

## 3.2 Costanti

In ogni calcolo sono presenti delle costanti. Dal punto di vista del linguaggio Modelica, una costante è dichiarata in modo simile ad una variabile, con la sola differenza che una volta dichiarata, non si potrà più modificarne il valore. Il compilatore segnala un errore se riconosce il tentativo di modificare il valore di una costante.

Ecco ad esempio come potremmo editare il package Constants:

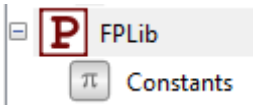
```
package Constants
  extends Modelica.Icons.Package;
  final constant Real pi = Modelica.Constants.pi;
  final constant Real FlowSI2Lmin = 60000.0;
  final constant Real FlowLmin2SI = 1.0 / FlowSI2Lmin;
  final constant Real SI2bar = 1e-5;
  final constant Real Bar2SI = 1e+5;
  final constant SI.Density rho = 850.0 "densità";
  final constant SI.DynamicViscosity mu = 0.051 "viscosita' dinamica";
  final constant SI.KinematicViscosity ni = mu / rho "viscosita' cinematica";
  final constant SI.BulkModulus bulk = 1.7e+9 "bulk modulus";
  final constant Real SteelYoungModulus(unit = "N/mm2") = 210000 "modulo di Young";
  final constant Real epsilon = 1e-9;
  final constant Types.Acceleration g = Modelica.Constants.g_n;
  final constant Real D2R = pi / 180 "Degree to Radian";
  final constant Real R2D = 180 / pi "Radian to Degree";
  annotation(
      Documentation(info = "<html>
<head>
</head>
<body>
<p>This package provides constants of mathematics, physics, fluid
properties, some material properties and other constants used in the
calculations.</p>
</body>
</html>"),
      Icon(coordinateSystem(extent = {{-100.0, -100.0}, {100.0, 100.0}}), graphics = {Polygon(origin
= {-9.2597, 25.6673}, fillColor = {102, 102, 102}, pattern = LinePattern.None, fillPattern =
FillPattern.Solid, points = {{48.017, 11.336}, {48.017, 11.336}, {10.766, 11.336}, {-25.684, 10.95},
{-34.944, -15.111}, {-34.944, -15.111}, {-32.298, -15.244}, {-32.298, -15.244}, {-22.112, 0.168},
{11.292, 0.234}, {48.267, -0.097}, {48.267, -0.097}}, smooth = Smooth.Bezier), Polygon(origin = {-
19.9923, -8.3993}, fillColor = {102, 102, 102}, pattern = LinePattern.None, fillPattern =
FillPattern.Solid, points = {{3.239, 37.343}, {3.305, 37.343}, {-0.399, 2.683}, {-16.936, -20.071},
{-7.808, -28.604}, {6.811, -22.519}, {9.986, 37.145}, {9.986, 37.145}}, smooth = Smooth.Bezier),
Polygon(origin = {23.753, -11.5422}, fillColor = {102, 102, 102}, pattern = LinePattern.None,
fillPattern = FillPattern.Solid, points = {{-10.873, 41.478}, {-10.873, 41.478}, {-14.048, -4.162},
{-9.352, -24.8}, {7.912, -24.469}, {16.247, 0.27}, {16.247, 0.27}, {13.336, 0.071}, {13.336, 0.071},
{7.515, -9.983}, {-3.134, -7.271}, {-2.671, 41.214}, {-2.671, 41.214}}, smooth = Smooth.Bezier)}));
end Constants;
```

A parte i riferimenti al linguaggio (prerequisito) bisogna aggiungere che **SI** deve essere definito con

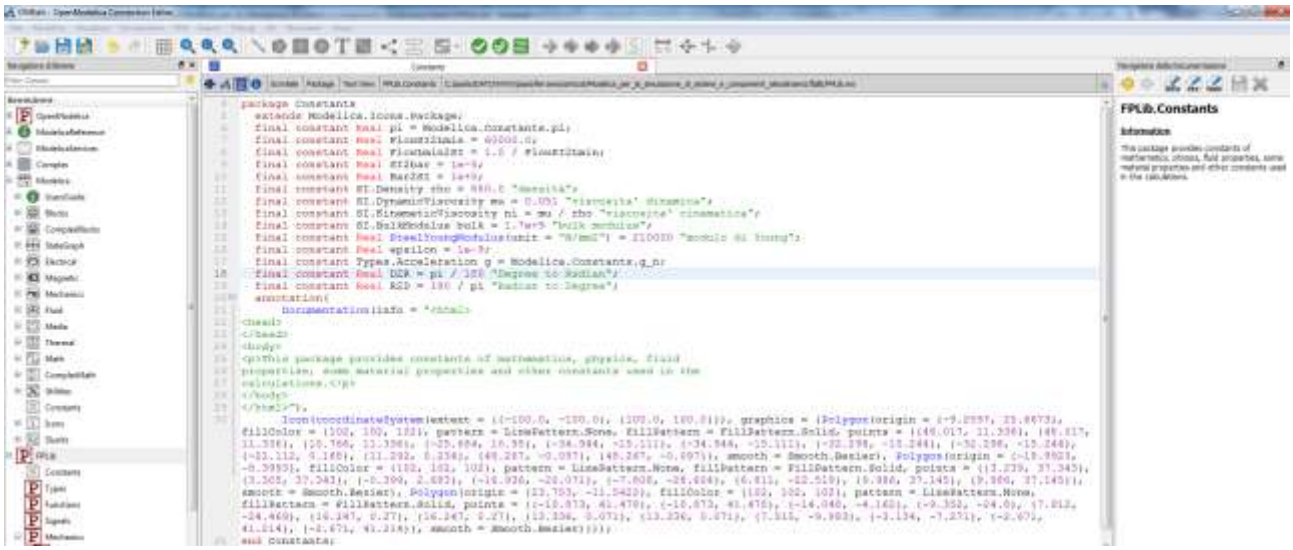
```
import SI = Modelica.SIunits;
```

direttamente sotto al package principale (FPLib). In Modelica.SIunits vengono definiti i tipi relativi alle unità di misura del Sistema Internazionale, per non scrivere tutte le volte Modelca.SIunits, si sostituisce con SI.

Il comando **extends Modelica.Icons.Package** e il testo dentro la keyword **annotation** definiscono l’icona associata al package Constants




e inseriscono anche le informazioni visibili per la guida (si vedono a destra dell’immagine sotto riportata)


FPLib.Constants
Information

## 3.3 Tipi

Sebbene siano già definite tulle le grandezze fisiche del Sistema Internazionale, risulta particolarmente utile lavorare con unità di misura del sistema tecnico. Possiamo andare direttamente nel package Types, in Text View ed editare i tipi che vogliamo aggiungere.

```
package Types
  extends Modelica.Icons.TypesPackage;
  type VolumeFlowRate = Real(quantity = "Volume Flow Rate", unit = "l/min");
  type Pressure = Real(quantity = "Pressure", unit = "bar");
  type Length = Real(quantity = "Length", unit = "mm", min = 0);
  type Position = Real(quantity = "Length", unit = "mm");
  type Diameter = Real(quantity = "Length", unit = "mm", min = 0);
  type Area = Real(quantity = "Area", unit = "mm2", min = 0);
  type Volume = Real(quantity = "Volume", unit = "cm3", min = 0);
  type Angle = Real(quantity = "Angle", unit = "°");
  type rpm = Real(quantity = "Rev per minute", unit = "rpm");
  type Dimensionless = Real(quantity = "Dimensionless", unit = "1");
  type SpringConstant = Real(quantity = "Spring Stiffness", unit = "N/mm", min = 0);
  type DampingConstant = Real(quantity = "Damping Constant", unit = "N.s/m");
  type YoungModulus = Real(unit = "N/mm2", min = 0);
  type Force = Real(quantity = "Force", unit = "F");
  type Velocity = Real(quantity = "Velocity", unit = "m/s");
  type Acceleration = Real(quantity = "Acceleration", unit = "m/s2");
  type Mass = Real(unit = "kg", min = 0);
  type ReynoldsNumber = Real(final quantity = "ReynoldsNumber", final unit = "1", min = 0);
  annotation(
    Documentation(info = "<html>
<head>
</head>
<body>
<p>This package provides for the physical quantities that the framework deals with. For reasons of
clarity of reading, a technical system has been adopted. This complicates the formulas a bit but
improves the editing of the various blocks and the reading of the graphs because the common units of
measurement in the hydraulics will be used directly.</p>
</body>
</html>"));
end Types;
```

## 3.4 Creare funzioni

Per creare funzioni andiamo nel package Functions e con il pulsante destro scegliamo **Nuova classe Modelica** e aggiungiamo una funzione dopo l'altra. Io ne propongo alcune.

```
package Functions
  extends Modelica.Icons.Package;

  function linterp "Linear interpolation"
    input Real x "Independent variable";
    input Real x_array[:] "x data array";
    input Real y_array[:] "y data array";
    output Real y "Interpolated result";
  protected
    Integer n;
  algorithm
    n := size(x_array, 1);
    assert(n == size(y_array, 1), "Size mismatch");
    if noEvent(x < x_array[1]) then
      y := smooth(2, y_array[1]);
    elseif noEvent(x > x_array[n]) then
      y := smooth(2, y_array[n]);
    else
      for i in 1:n - 1 loop
        if noEvent(x >= x_array[i] and x <= x_array[i + 1]) then
          y := smooth(2, y_array[i] + (y_array[i + 1] - y_array[i]) * ((x - x_array[i]) / (x_array[i
+ 1] - x_array[i])));
        end if;
      end for;
    end if;
  end linterp;

  function linearStager
    input Real v[:] "Array di (val1, val2, durata tra val1 e val2)";
    input Real T "Tempo in cui valutare la funzione";
    output Real y "Valore della funzione";
  protected
    Integer L "numero di elementi nell'array";
    Real oldTime = 0.0 "Tempo iniziale";
    Real v1, v2, t, slope, r;
    Integer i;
  algorithm
    r := 0;
    L := size(v, 1);
    assert(mod(L, 3) == 0, "numero di elementi nel vettore errato");
    for i in 1:3:L loop
      t := v[i + 2];
      if t > 0 then
        if T > oldTime and T <= oldTime + t then
          v1 := v[i];
          v2 := v[i + 1];
          slope := (v2 - v1) / t;
          r := v1 + slope * (T - oldTime);
          i := L + 1;
          break;
        end if;
      end if;
      oldTime := oldTime + t;
    end for;
    y := noEvent(if abs(r) > Constants.epsilon then r else 0);
  end linearStager;

  function orif3f
    input Types.Pressure p1 "Pressure upstream the orifice";
    input Types.Pressure p2 "Pressure downstream the orifice";
    input Types.Area A "Area of the orifice";
    input Types.Diameter hd "Hydraulic Diameter";
    input Types.Dimensionless Cqmax "Max. Discharge coefficient";
    input Types.ReynoldsNumber Rec = 1000 "Critical Reynolds number";
    output Types.VolumeFlowRate Q "Flow Rate";
    output Types.Velocity Wmean "Mean velocity";
    output Types.ReynoldsNumber Re "Reynolds number";
    output Types.Dimensionless Cq "Discharge coefficient";
  protected
    Types.Velocity Wmax "Max. velocity";
    Types.Pressure DeltaP "=p1-p2";
    final constant Real kw = sqrt(200000);
  algorithm
```






```
    DeltaP := p1 - p2;
    Wmax := smooth(2, kw * sqrt(abs(DeltaP) / Constants.rho));
    Re := smooth(2, 1e-3 * Wmax * hd * Constants.rho / Constants.mu);
    Cq := smooth(2, Cqmax * tanh(2 * Re / Rec));
    Wmean := smooth(2, Cq * Wmax * sign(DeltaP));
    Q := smooth(2, Cq * A * sqrt(720 * abs(DeltaP) / Constants.rho) * sign(DeltaP));
  end orif3f;

  function hCapacitance
    input Types.VolumeFlowRate Qin "Incoming flow rate [l/min]";
    input Types.VolumeFlowRate Qout "Outcoming flow rate [l/min]";
    input Types.Volume V "Volume [cm3]";
    input Real Vdot(unit = "cm3/s") "Variation of volume over time [cm3/s]";
    input Types.Pressure p "Actual pressure in the volume";
    input SI.BulkModulus bulk = Constants.bulk "bulk modulus [Pa]";
    output Real pdot(unit = "bar/s") "Pressure variation over time [bar/s]";
  protected
    Types.VolumeFlowRate Qnet = Qin - Qout "Net flow rate";
  algorithm
    assert(p > (-1), "The relative pressure is less than -1 bar, there is likely to be cavitation
somewhere.", AssertionLevel.warning);
    pdot := smooth(2, if p > (-1) or Qnet > 0.0 then bulk / (6000 * V) * (Qnet - 3 / 50 * Vdot) else
0);
  end hCapacitance;

  function bilimit
    input Real val;
    input Real lowLimit;
    input Real highLimit;
    output Real limitedVal;
  algorithm
    if val < lowLimit then
      limitedVal := lowLimit;
    elseif val > highLimit then
      limitedVal := highLimit;
    else
      limitedVal := val;
    end if;
  end bilimit;
  annotation(
    Icon(coordinateSystem(extent = {{-100.0, -100.0}, {100.0, 100.0}}), graphics = {Polygon(origin = {1.3835, -4.1418}, rotation =
45.0, fillColor = {64, 64, 64}, pattern = LinePattern.None, fillPattern = FillPattern.Solid, points = {{-15.0, 93.333}, {-15.0,
68.333}, {0.0, 58.333}, {15.0, 68.333}, {15.0, 93.333}, {20.0, 93.333}, {25.0, 83.333}, {25.0, 58.333}, {10.0, 43.333}, {10.0, -
41.667}, {25.0, -56.667}, {25.0, -76.667}, {10.0, -91.667}, {0.0, -91.667}, {0.0, -81.667}, {5.0, -81.667}, {15.0, -71.667}, {15.0, -
61.667}, {5.0, -51.667}, {-5.0, -51.667}, {-15.0, -61.667}, {-15.0, -71.667}, {-5.0, -81.667}, {0.0, -81.667}, {0.0, -91.667}, {-
10.0, -91.667}, {-25.0, -76.667}, {-25.0, -56.667}, {-10.0, -41.667}, {-10.0, 43.333}, {-25.0, 58.333}, {-25.0, 83.333}, {-20.0,
93.333}}), Polygon(origin = {10.1018, 5.218}, rotation = -45.0, fillColor = {255, 255, 255}, fillPattern = FillPattern.Solid, points
= {{-15.0, 87.273}, {15.0, 87.273}, {20.0, 82.273}, {20.0, 27.273}, {10.0, 17.273}, {10.0, 7.273}, {20.0, 2.273}, {20.0, -2.727},
{5.0, -2.727}, {5.0, -77.727}, {10.0, -87.727}, {5.0, -112.727}, {-5.0, -112.727}, {-10.0, -87.727}, {-5.0, -77.727}, {-5.0, -2.727},
{-20.0, -2.727}, {-20.0, 2.273}, {-10.0, 7.273}, {-10.0, 17.273}, {-20.0, 27.273}, {-20.0, 82.273}})})),
    Documentation(info = "<html>
<p>
This package contains Modelica <b>functions</b>
</p>
</html>"));
end Functions;
```

## 3.5 Creare porte di connessione

Una porta è una classe, di tipo **Connector**, che serve a connettere tra loro componenti che scambiano grandezze dello stesso dominio fisico.
E' costituita da almeno due grandezze di cui una è detta grandezza potenziale e l'altra grandezza di flusso (*flow*). Esse hanno una caratteristica ben precisa: la potenza scambiata attraverso la porta è sempre il prodotto della grandezza potenziale (*o di una sua derivata nel tempo*) e della grandezza di flusso. La grandezza di trasporto (Carrier) è la grandezza che, in un sistema chiuso, si conserva e si lega alla grandezza flow per il fatto che quest'ultima, è la derivata nel tempo della grandezza Carrier.
Nella tabella sottostante vediamo le grandezze potenziali, di flusso, di trasporto per alcuni domini della fisica e le relazioni che legano quest'ultime con la potenza scambiata nella porta di connessione.

| Domain | Potential quantity | Flow quantity | Carrier | Basic relationships |
|---|---|---|---|---|
| Electrical | V (Voltage) | i (Current) | q (Charge) | N = V·i, i = dq/dt |
| Translational | x (Position) | F (Force) | p (Linear momentum) | N = F·ẋ, F = dp/dt |
| Rotational | φ (Angle) | M (Torque) | L (Angular momentum) | N = M·φ', M = dL/dt |
| Hydraulic | P (pressure) | Q (Volume flow rate) | V (Volume) | N = P·Q, Q = dV/dt |

In Tabella 1 vediamo le principali grandezze potenziali e di flusso per i principali domini della fisica.

| Domain | Potential | Flow | Carrier |
|---|---|---|---|
| **Electrical** | Voltage | Current | Charge |
| **Translational** | Position | Force | Linear Momentum |
| **Rotational** | Angle | Torque | Angular Momentum |
| **Magnetic** | Magnetic potential | Magnetic flux rate | Magnetic flux |
| **Hydraulic** | Pressure | Volume flow | Volume |
| **Pneumatic** | Pressure | Mass flow | Air |
| **Heat** | Temperature | Heat flow | Heat |
| **Chemical** | Chemical potential | Particle flow | Particles |

**Tabella 1: grandezze potenziali, di flusso e carrier tipiche della fisica**

Noi abbiamo bisogno di porte meccaniche e di porte idrauliche per poter modellare componenti oleodinamici.

### 3.5.1 Regola fondamentale per l'interconnessione di blocchi attraverso porte

Il fatto che una grandezza sia potenziale oppure di flusso determina il modo in cui le equazioni dei blocchi verranno legate.
Innanzitutto è possibile collegare soltanto porte dello stesso tipo.
La connessione tra due o più porte comporta sempre che vengano aggiunte le seguenti equazioni (oltre a quelle presenti nei vari blocchi):

1. Tutte le grandezze potenziali confluenti in un nodo vengono eguagliate ($p_1 = p_2 = \ldots = p_n$).
2. La sommatoria di tutte le grandezze di flusso viene egugliata a zero ($f_1 + f_2 + \ldots + fn = 0$).

Pensando a un circuito dell'elettrotecnica e alla Tabella 1, un nodo è equipotenziale e la sommatoria algebrica delle correnti in un nodo è nulla.
Vedremo la creazione di una porta meccanica al paragrafo 4.1.1 e di una porta idraulica al paragrafo 5.1.1.

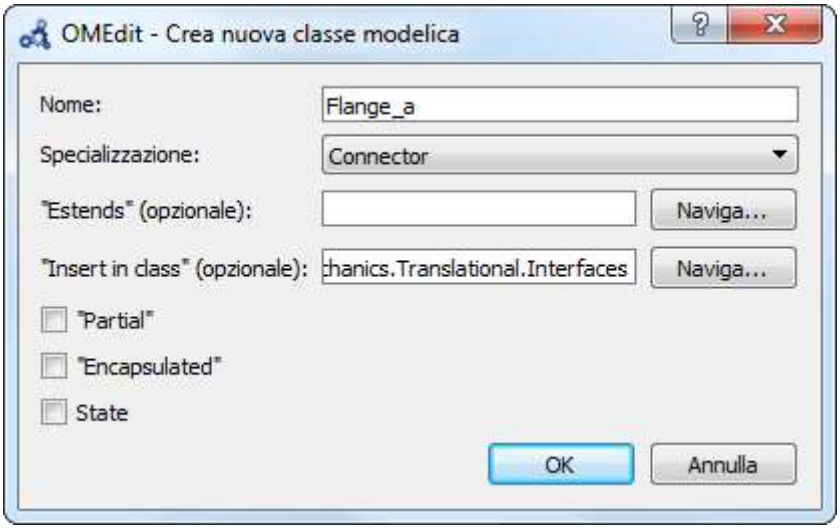


Per Modelica ogni porta è bidirezionale, salvo specifica indicazione. **Quello che differenzia una grandezza entrante da una uscente è una convenzione che usata dalla libreria standard di Modelica per cui si ritiene una grandezza positiva se entrante e negativa se uscente**.

Ad esempio se un polo di una resistena ha una corrente posiviva significa che la corrente è entrante, viceversa se è negativa è uscente. Quando scriviamo le equazioni dobbiamo tenere conto di questo fatto, ad esempio nel generatore di portata con una porta, la portata sarà –q.

### 3.5.2 Definizione di una porta

Per creare una porta andiamo nel package d'interesse (sotto-package Interfaces di una qualunque libreria) e creiamo una Nuova classe Modelica, di tipo Connector




Nell'esempio è il nome che abbiamo dato a una porta meccanica. Vedremo il codice per le porte meccaniche al paragrafo 4.1.1. e il codice per le porte idrauliche al paragrafo 5.1.1.

## 3.6 Costruire componenti semplici

Per costruire un componente semplice è necessario selezionare con il tasto destro del mouse il package di destinazione (dove si vuole aggiungere il componente), quindi "**Nuova classe Modelica**", e digitare il nome del componente. Verificare che la specializzazione sia **Model** e premere OK.
Ovviamente il componente potrebbe essere l'estensione di una classe preesistente (Extends), in pratica una classe derivata da una classe base (o parziale).
Dovremo **aggiungere porte** (**connectors**) che consentiranno al nostro componente di scambiare grandezze fisiche: **il modo migliore per farlo è andare in "vista icona" e trascinare le porte che vogliamo nei punti desiderati (ci dovremo poi ricordare di rinominarle nel codice se lo desideriamo)**. Se invece dobbiamo aggiungere una classe parziale, non la possiamo trascinare ma dovrà essere essere inclusa per estensione (si veda l'esempio 5.1.4).
**Si dovrà quindi completare lo schema aggiungendo grafica e testo**: il testo **%name%** consente di visualizzare il nome del componente nello schema, il testo **%nome_parametro** serve a visualizzare il valore del parametro stesso, ad esempio "q = %q" mostra nell'icona del componente il valore del parametro q.
Si dovrà infine **aggiungere il resto del codice** (***parametri, equazioni, eventualmente algoritmi***).
Ricapitolando si deve:

1. Creare un componente con "**Nuova classe Modelica**" su un package di destinazione, di **specializzazione** "**Model**".
2. In "**vista icona**" **si trascinano le porte di cui necessita il componente** (o aggiungere una classe derivata con Extends che già le contiene); se le abbiamo importate trascinandole, eventualmente si può andare in Text View a rinominare i nomi delle porte.
3. Sempre in "**Vista icona**" **si completa la grafica del componente con il testo** %name% con il nome del componente ed eventualmente con i parametri più importanti (es. %q).
4. **Aggiungere il resto del codice (parametri, equazioni, algoritmi)**.

E' altresì possibile creare un modello vuoto e copiare tutto il codice, comprese le annotazioni.

## 3.7 Buone pratiche per la scrittura di librerie

Molti modelli hanno interfacce in comune: pensiamo ai bipoli dell'elettrotecnica ad esempio.
Ricreare questi bipoli manualmente ogni volta, oltre che una gran perdita di tempo, è anche un modo per introdurre errori.
Un esempio di buona programmazione è creare classi parziali che abbiano un certo numero di elementi in comune (porte, variabili, equazioni, etc.). Ad esempio gli elementi con due porte meccaniche potrebbero essere tutti derivati da "Il modello Partial2Ports", elementi con due porte idrauliche da "Il modello Partial2FluidPorts".
Una classe parziale per una categoria di componenti idraulici a due porte dove non c'è accumulo di portata, come ad esempio una resistenza idraulica, potrebbe essere definita come "Il modello Partial2FluidPortsWithNullNetFlow": in questo tutti i componenti conoscono già il ΔP tra le porte, il fatto che la portata che entra e quella che esce sono identiche (in valore assoluto) per cui non resta che definire il legame tra portata e differenza di pressione.
L'uso di classi parziali consente di:

1. Scrivere meno codice. Aggiungo potenza alla scrittura del codice (con meno codice faccio più cose).
2. Avere codice più robusto perché richiamando sempre gli stessi sotto-modelli rischio meno errori rispetto a riscrivere parti di codice che fanno le stesse cose.
3. Avere più coerenza tra i vari modelli nel nome e nella descrizione di parametri e variabili.

Nei prossimi capitoli vedremo alcuni blocchi dei diversi gruppi (meccanica, oleodinamica, hydraulic component design).

# 4 Mechanical Library

Trattiamo qui i principali blocchi della meccanica. Ce ne sarebbero tantissimi da inserire ma il nostro scopo è far vedere come si realizzano e quindi ci limitiamo ai principali. Gli altri, a parte le equazioni costitutive, si realizzano nello stesso modo. Una libreria meccanica semplice e flessibile sta alla base di una buona libreria oleodinamica poiché ogni valvola reale è costituita da almeno una parte mobile (una massa in movimento) e da una molla.

## 4.1 Modelli parziali d'interfaccia

Per la meccanica abbiamo bisogno di alcuni modelli d'interfaccia al fine di semplificare e standardizzare la scrittura del resto del codice.

### 4.1.1 Classi connector

Creiamo due porte meccaniche "Flange_a" e "Flange_b" come visto al paragrafo 3.5.2.
Sono le porte attraverso le quali viene scambiata energia meccanica. In basa alla Tabella 1, quando parliamo di energia meccanica traslazionale, parliamo dunque di posizione e forza.

```
connector Flange_a "(left) 1D translational flange (flange axis directed INTO cut plane,
e. g. from left to right)"
  Types.Position s "Absolute position of flange";
  flow Types.Force f "Cut force directed into flange";
  annotation(
   defaultComponentName = "flange_a",
   Icon(coordinateSystem(preserveAspectRatio = true, extent = {{-100, -100}, {100, 100}}), graphics = {Rectangle(extent = {{-100, -
100}, {100, 100}}, lineColor = {0, 127, 0}, fillColor = {0, 127, 0}, fillPattern = FillPattern.Solid)}),
   Diagram(coordinateSystem(initialScale = 0.1), graphics = {Rectangle(lineColor = {0, 127, 0}, fillColor = {0, 127, 0}, fillPattern
= FillPattern.Solid, extent = {{-40, -40}, {40, 40}}), Text(origin = {80, -10}, lineColor = {0, 127, 0}, extent = {{-160, 110}, {0,
70}}, textString = "%name")}));
end Flange_a;
connector Flange_b "(right) 1D translational flange (flange axis directed OUT OF cut
plane)"
  Types.Position s "Absolute position of flange";
  flow Types.Force f "Cut force directed into flange";
  annotation(
   defaultComponentName = "flange_b",
   Icon(coordinateSystem(preserveAspectRatio = true, extent = {{-100, -100}, {100, 100}}), graphics = {Rectangle(extent = {{-100, -
100}, {100, 100}}, lineColor = {0, 127, 0}, fillColor = {255, 255, 255}, fillPattern = FillPattern.Solid)}),
   Diagram(coordinateSystem(initialScale = 0.1), graphics = {Rectangle(lineColor = {0, 127, 0}, fillColor = {255, 255, 255},
fillPattern = FillPattern.Solid, extent = {{-40, -40}, {40, 40}}), Text(origin = {-40, -10}, lineColor = {0, 127, 0}, extent = {{-40,
110}, {120, 70}}, textString = "%name")}));
end Flange_b;
```

In pratica Flange_a e Flange_b sono identiche, quello che cambia è che la prima è un quadratino verde e l'altra è un quadratino bianco. *Anche se dal punto di vista della codifica avremmo potuto usare soltanto una delle due porte, averle differenziate aiuta la creazione dei modelli*.

### 4.1.2 Il modello Partial2Ports

Dichiara un blocco con una porta Flange_a e una porta Flange_b.

```
partial model Partial2Ports
  FPLib.Mechanics.Translational.Interfaces.Flange_a flange_a annotation(
   Placement(visible = true, transformation(origin = {-104, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Mechanics.Translational.Interfaces.Flange_b flange_b annotation(
   Placement(visible = true, transformation(origin = {104, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
end Partial2Ports;
```

In questo modo, ogni volta che ho bisogno di un blocco con due porte meccaniche, mi basterà creare una classe che estende questo modello per avere le stesse due porte (istanziate con ***flange_a*** e ***flange_b***) sempre nella stessa posizione.

### 4.1.3 Il modello PartialRelativeMovements

E' una classe d'utilità che calcola le seguenti variabili:

- Position ***xa_rel***: è la posizione relativa della flangia a, rispetto allo stato iniziale, ovvero se flange_a.s inizialmente è 2 e passa 4, xa_rel = 2.
- Position ***xb_rel***: è la posizione relativa della flangia b, rispetto allo stato iniziale, ovvero se flange_b.s inizialmente è 8 e passa 6, xb_rel = -2.
- Velocity ***va***: è la velocità della flangia_a.
- Velocity ***vb***: è la velocità della flangia_b.
- Length ***Delta_x***: è la differenza tra xa_rel e xb_rel.
- Velocity ***Delta_v***: è la differenza tra la velocità della flangia a e la velocità della flangia b.

Ecco com'è definita.

```
model PartialRelativeMovements
  Types.Length Delta_x "= flange_a.s - flange_b.s";
  Types.Velocity Delta_v "= der(Delta_x)";
  Types.Position xa_rel(start = 0) "relative flange_a position";
  Types.Position xb_rel(start = 0) "relative flange_b position";
  Types.Velocity va(start = 0) "flange_a velocity";
  Types.Velocity vb(start = 0) "flange_b velocity";
equation
  Delta_x = xa_rel - xb_rel;
  Delta_v = 0.001 * der(Delta_x);
  va = 0.001 * der(flange_a.s);
  vb = 0.001 * der(flange_b.s);
  0.001 * der(xa_rel) = va;
  0.001 * der(xb_rel) = vb;
  annotation(
   Documentation(info = "<html><head><title></title></head>
<body>Calculates:<br>xa_rel = relative movement (from zero) flange_a<br>xb_rel = relative movement (from zero) flange_b<br>va =
velocity flange_a<br>vb = velocity flange_b<br>Delta_x = flange_a.s - flange_b.s<br>Delta_v = der(Delta_x)<br></body></html>"));
end PartialRelativeMovements;
```

Nota: la velocità è calcolata come

$$v = 10^{-3}\frac{dx}{dt}$$

4-1: velocità istantanea calcolata con x in millimetri

Perché la posizione x è in millimetri. Se avessimo semplicemente utilizzato der(flange_a.s) avremmo avuto una velocità in mm/s.

### 4.1.4 Il modello PartialViscoElastic2Ports

La classe estende Partial2Ports e PartialRelativeMovements:

```
partial model PartialViscoElastic2Ports
  extends Partial2Ports;
  extends PartialRelativeMovements;
  Types.Force F "= flange A - flange B force";
equation
  flange_a.f = F;
  flange_b.f = -F;
end PartialViscoElastic2Ports;
```

Inoltre stabilisce che la forza che "entra" nella porta flange_a è quella che "esce" dalla porta flange_b.
Se ho una molla "in piedi" su un tavolo, e con un dito applico una certa forza per accorciarla, detta forze è entrante dalla parte in cui spingo con il dito, ma la stessa forza è uscente dalla molla dalla parte in cui spinge contro il tavolo.

## 4.2 La massa

L'equazione fondamentale per la massa è la seconda legge di Newton:

$$m \cdot a = \sum F$$

4-2: seconda legge di Newton

Normalmente le forze comprendono sia quelle esterne applicate sia le forze d'attrito e la gravitazionale.
Normalmente nelle nostre applicazioni dovremo tenere conto di attriti e fine corsa.
Esistono vari modelli per il calcolo dei fine corsa (ad esempio Kelvin-Voigt) ma s'è scelto di trattarli come ideali perché nella maggior parte dei casi funziona bene ed è il metodo che in genere assicura le migliori prestazioni (esecuzione delle simulazioni più veloce).
Infatti, le prestazioni non risentono tanto dal numero di equazioni o variabili quanto dalla presenza e dal numero di elementi di discontinuità, in particolare le collisioni dei fine corsa, perché forzano il solutore a interrompere l'integrazione, a reinizializzare le variabili e a riprendere il processo.
Rispetto ad altri modelli, la classe ***PartialMassWithIdealStopAndFriction***, deriva dalla libreria standard di Modelica e riduce al minimo (non potendole evitare) le situazioni di stop e restart del solutore.

### 4.2.1 La classe parziale PartialMassWithIdealStopAndFriction

E' una classe parziale che implementa la [4-2] per una massa puntiforme con attrito e stop ideali:

1. **Attrito statico** (parametro **F_Stribeck = forza d'attrito statico in N**).
2. **Attrito dinamico** (parametro **F_Coulomb = forza d'attrito dinamico in N**).
3. **Attrito viscoso** (parametro **F_prop = coefficiente d'attrito viscoso in N.s/m**).

*Non staremo a spiegare come viene calcolato l'attrito perché è fuori dai nostri scopi, anche perché questa classe in pratica ricalca quella della libreria standard MassWithStopAndFriction con l'unica differenza che la posizione è in millimetri*.
Altri parametri importanti sono:

1. **Massa** (parametro **m**, in kg).
2. **Limite di posizione inferiore** (parametro $\mathbf{x_1}$, in millimetri).
3. **Limite di posizione superiore** (parametro $\mathbf{x_2}$, in millimetri).

Le variabili di stato x (posizione in millimetri) e v (velocità in m/s) vengono calcolate istante per istante, così come l'accelerazione istantanea (in $m/s^2$).
La forza netta che agisce sulla massa è F-f dove F è la forza esterna e f è la risultante delle forze d'attrito.
Ecco il codice Modelca:

```
model PartialMassWithIdealStopAndFriction
  parameter Types.Mass m = 1.0 "Mass";
  parameter Types.Position x1 = -10.0 "Left stop for (left end of) sliding mass";
  parameter Types.Position x2 = 10.0 "Right stop for (right end of) sliding mass";
  parameter Types.Force F_Coulomb = 0.0 "Dynamic friction (Coulomb force)";
  parameter Types.Force F_Stribeck = 0.0 "Stribeck effect (max. static Coulomb force)";
  parameter Types.DampingConstant F_prop = 0.0 "Velocity dependent friction";
  parameter Types.Velocity v_small = 1e-3 "Relative velocity near to zero (see model info text)";
  parameter Real fexp(unit = "s/m", min = 0.0) = 2.0 "Exponential decay" annotation(
    Dialog(tab = "Advanced"));
  Types.Position x(start = x1, stateSelect = StateSelect.always) "Absolute position of component";
  Types.Velocity v(start = 0.0) "Absolute velocity of flange_a and flange_b";
  Types.Acceleration a(start = 0.0) "Absolute acceleration of flange_a and flange_b";
  Types.Force F(start = 0.0) "External force applied to the mass";
protected
  Types.Velocity v_relfric "Relative velocity between frictional surfaces" annotation(HideResult =
true);
  Types.Acceleration a_relfric "Relative acceleration between frictional surfaces"
annotation(HideResult = true);
  Types.Force f "Friction force (positive, if directed in opposite direction of v_rel)"
annotation(HideResult = true);
  Types.Force f0 "Friction force for v=0 and forward sliding" annotation(HideResult = true);
  Types.Force f0_max "Maximum friction force for v=0 and locked" annotation(HideResult = true);
  Boolean free "true, if frictional element is not active" annotation(HideResult = true);
  Real sa(unit = "1") "Path parameter of friction characteristic f = f(a_relfric)"
annotation(HideResult = true);
  Boolean startForward(start = false, fixed = true) "= true, if v_rel=0 and start of forward sliding
or v_rel > v_small" annotation(HideResult = true);
  Boolean startBackward(start = false, fixed = true) "= true, if v_rel=0 and start of backward
sliding or v_rel < -v_small" annotation(HideResult = true);
```

```
  Boolean locked(start = false) "true, if v_rel=0 and not sliding" annotation(HideResult = true);
  Integer mode(min = -1, max = 3, start = 3, fixed = true) annotation(HideResult = true);
  Integer stopped annotation(HideResult = true);
equation
  f0 = F_Coulomb + F_Stribeck;
  f0_max = 1.001 * f0;
  free = f0 <= 0.0 and F_prop <= 0.0 and x > x1 and x < x2;
  v = 0.001 * der(x);
  a = der(v);
  v_relfric = v;
  a_relfric = a;
  m * a = F - f;
  f = if locked then sa else if free then 0.0 else if startForward then F_prop * v + F_Coulomb +
F_Stribeck else if startBackward then F_prop * v + (-F_Coulomb) - F_Stribeck else if pre(mode) == 1
then F_prop * v + F_Coulomb + F_Stribeck * exp((-fexp) * abs(v)) else F_prop * v + (-F_Coulomb) -
F_Stribeck * exp((-fexp) * abs(v));
  stopped = if x <= x1 then -1 else if x >= x2 then 1 else 0;
  when stopped <> 0 then
    reinit(x, if stopped < 0 then x1 else x2);
    reinit(v, 0.0);
  end when;
  startForward = pre(mode) == 0 and (sa > f0_max and x < x2 or pre(startForward) and sa > f0 and x <
x2) or pre(mode) == (-1) and v_relfric > v_small or initial() and v_relfric > 0.0;
  startBackward = pre(mode) == 0 and (sa < (-f0_max) and x > x1 or pre(startBackward) and sa < (-f0)
and x > x1) or pre(mode) == 1 and v_relfric < (-v_small) or initial() and v_relfric < 0.0;
  locked = not free and not (pre(mode) == 1 or startForward or pre(mode) == (-1) or startBackward);
  a_relfric = if locked then 0.0 else if free then sa else if startForward then sa - f0_max else if
startBackward then sa + f0_max else if pre(mode) == 1 then sa - f0_max else sa + f0_max;
  mode = if free then 2 else if (pre(mode) == 1 or pre(mode) == 2 or startForward) and v_relfric >
0.0 and x < x2 then 1 else if (pre(mode) == (-1) or pre(mode) == 2 or startBackward) and v_relfric <
0.0 and x > x1 then -1 else 0;
end PartialMassWithIdealStopAndFriction;
```

Un'altra differenza è che nella libreria standard la massa è dotata di una lunghezza mentre qua non se ne tiene conto. In pratica la massa è trattata come puntiforme.

### 4.2.2 Il modello MassWithIdealStop

Il modello estende PartialMassWithIdealStopAndFriction per ottenere una classe istanziabile di una massa con stop ideali, attrito di Coulomb, attrito viscoso da usare con gli schemi a blocchi acausali.

```
model MassWithIdealStop
  parameter Types.Angle inclination = 0 "Inclination (+90 flange_b lowest, -90 flange_b
higher)";
  extends Interfaces.PartialMechanicalBase;
  extends Interfaces.Partial2Ports;
  extends Interfaces.PartialOrientedReferenceSystemDisplay;
  extends Interfaces.PartialMassWithIdealStopAndFriction;
protected
  parameter SI.Angle angle = Constants.D2R * inclination;
equation
  flange_a.s = x;
  flange_b.s = x;
  F = flange_a.f + flange_b.f + m * Constants.g * sin(angle);
  annotation(
    defaultComponentName = "mass",
    Icon(graphics = {Polygon(origin = {0, -60}, fillColor = {255, 255, 255}, fillPattern = FillPattern.Backward, points = {{-100,
20}, {-100, -20}, {100, -20}, {100, 20}, {80, 20}, {80, 0}, {-80, 0}, {-80, 20}, {-100, 20}}), Rectangle(origin = {0, 10}, fillColor
= {0, 170, 0}, fillPattern = FillPattern.HorizontalCylinder, extent = {{-60, 50}, {60, -70}}), Rectangle(origin = {-80, 0}, fillColor
= {0, 170, 0}, fillPattern = FillPattern.HorizontalCylinder, extent = {{-20, 20}, {20, -20}}), Rectangle(origin = {80, 0}, fillColor
= {0, 170, 0}, fillPattern = FillPattern.HorizontalCylinder, extent = {{-20, 20}, {20, -20}}), Text(origin = {0, 2}, extent = {{-30,
30}, {30, -30}}, textString = "M"), Text(origin = {-2, -29}, extent = {{-78, 9}, {82, -11}}, textString = "mass = %m"), Text(origin =
{-64, 82}, extent = {{-26, 18}, {24, -2}}, textString = "[ %x1 mm"), Text(origin = {66, 82}, extent = {{-26, 18}, {22, -2}},
textString = "%x2 mm ]")}, coordinateSystem(initialScale = 0.1)));
end MassWithIdealStop;
```

Quello che otteniamo è un blocchetto con due porte meccaniche

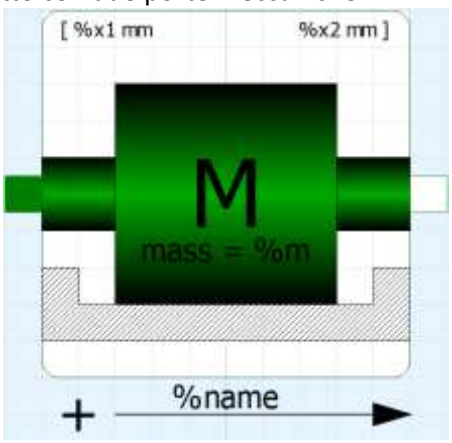



**Figura 4-1: modello della massa con stop ideale e attrito**

La freccia indica che la x aumenta (tra $x_1$ e $x_2$) se la massa si sposta nel verso della freccia.
La massa si sposta nella direzione della freccia se la forza netta applicata alle due porte è positiva.
Nel modello è anche presente un angolo (*inclination*) che determina l'inclinazione nei confronti del piano orizzontale, in particolare:

- Se inclination = 0 (default), la massa è parallela al terreno.
- Se inclination = 90, la massa è orientata con la flangia B verso il basso (poiché la flangia B è dalla parte della freccia, se inclination = 90 la massa ha la freccia rivolta verso il basso).
- Se inclination = -90, la massa è orientata con la flangia B verso l'alto, quindi se inclination = -90 la massa ha la freccia rivolta verso l'alto).

Se inclination è diverso da zero, la forza di gravità verrà calcolata in modo da essere coerente con il senso della freccia.

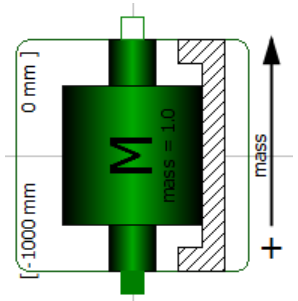
### 4.2.3 Esempio: caduta di un grave

L'esempio più semplice che possiamo fare, soltanto con il blocchetto di una massa, è la caduta di un grave. Lo faremo per due casi:

1) senza considerare l'attrito
2) con coefficiente d'attrito viscoso (β = 5 N·s/m).

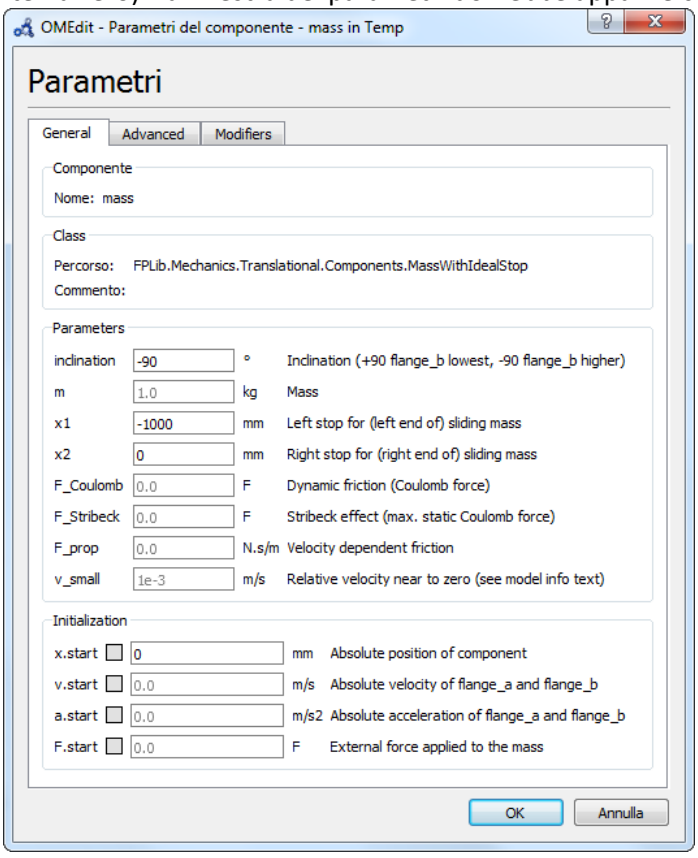
Il corpo è inizialmente posto all'altezza x = 0 mm (altezza del suolo) e dopo l'istante t = 0 inizierà a cadere in un dislivello di 1 metro.

Creiamo un nuovo modello, posizionando il blocchetto appena creato, lo ruotiamo in verticale (con la freccia verso l'alto)

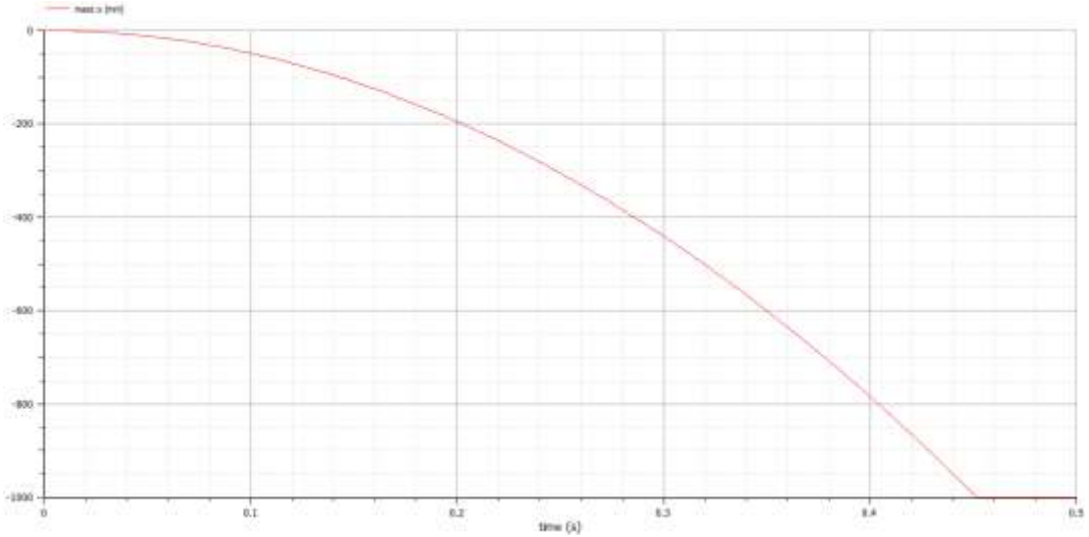



Con un doppio click sull'icona modificare i parametri in:

m = 1, x1 = -1000 mm, x2 = 0 mm, inclination = -90 (perché ho la freccia verso l'alto) e x.start = 0 (il corpo inizialmente è posto all'altezza zero). La finestra dei parametri dovrebbe apparire così:




Se lancio la simulazione ottengo:



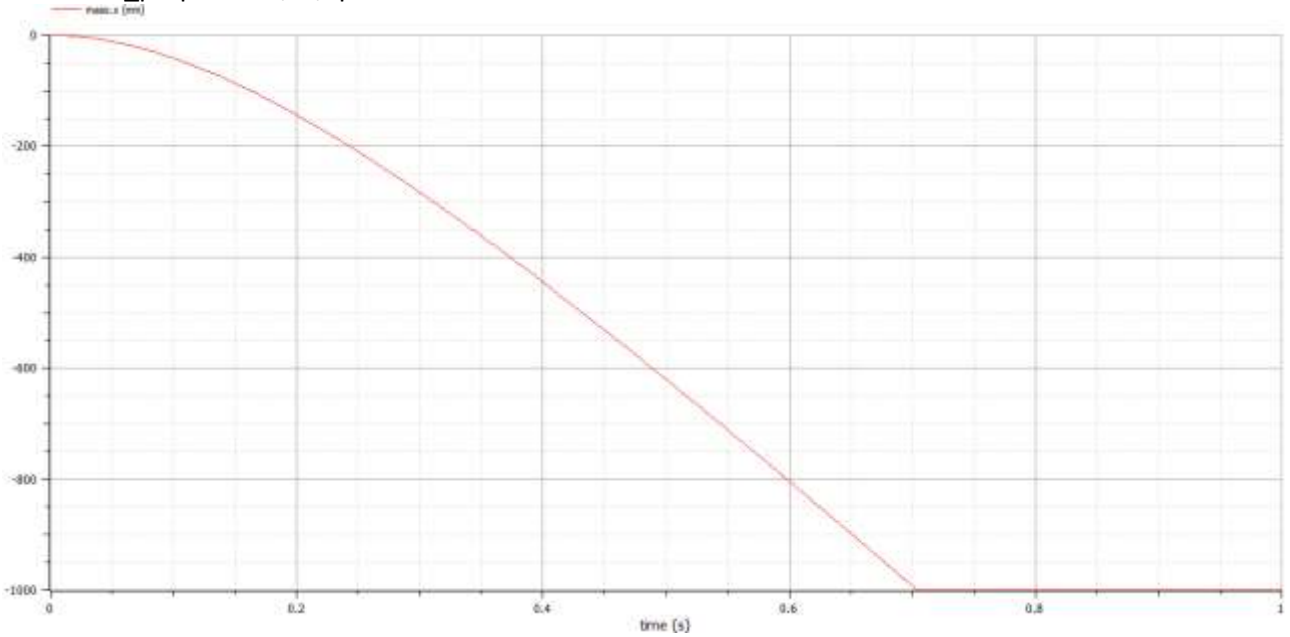

Cioè tocca il suolo (x = -1000 mm) dopo un tempo di 0.452 s.
Il risultato è facilmente verificabile, infatti:

$$\ddot{x} = -g \rightarrow x = -\frac{gt^2}{2} \rightarrow t = \sqrt{-\frac{2x}{g}}\Bigg|_{x=-1} = \sqrt{\frac{2}{g}} \approx 0.452s$$

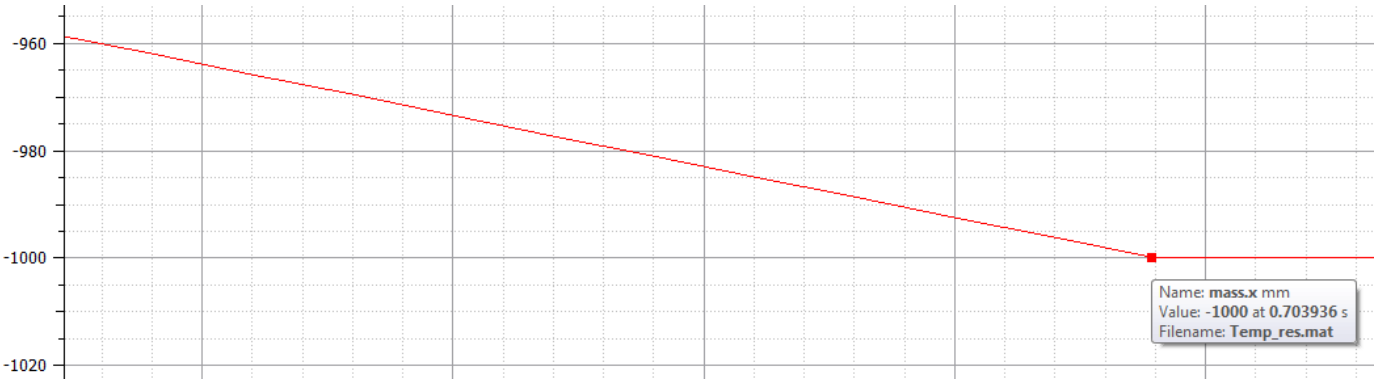
Caso 2). Immaginiamo ora che sia presente una forza d'attrito viscosa

$$F_v = \beta \cdot v$$

Se andiamo nel nostro modello Modelica, facciamo doppio click sull'icona della massa e settiamo la variabile F_prop a 5 N·s/m, quindi lanciamo la simulazione otteniamo:




Se ci muoviamo con il puntatore del mouse sulla curva, dove interseca x = -1000 mm (cioè quando il corpo è caduto di un metro), una piccola etichetta ci restituisce il valore della x e del tempo. Possiamo anche ingrandire la curva vicino a quel punto, come vediamo sotto:




**In pratica Modelica ci dice che il grave tocca il suolo all'istante t = 0.703936 s**.

Per la verifica del caso 2), con β = 5 N·s/m, sappiamo, applicando la seconda legge di Newton che:

$$m \cdot \ddot{x} = -m \cdot g - \beta \cdot \dot{x}$$

Quindi abbiamo:

$$\ddot{x} + \frac{\beta}{m}\dot{x} = -g$$

4-3: seconda legge di Newton per un grave in presenza d'attrito viscoso

Notiamo subito una soluzione "particolare":

$$x_p = -\frac{gmt}{\beta}$$

Mentre per l'equazione omogenea associata calcoliamo l'equazione caratteristica:

$$\lambda^2 + a\lambda + b = 0$$

Con a = β/m e b = 0, abbiamo:

$$\lambda_1 = -\frac{\beta}{m} \text{ e } \lambda_2 = 0$$

La soluzione generale è dunque:

$$x_g = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$$

Da cui, la soluzione generale della non omogenea è:

$$x = c_1 e^{-\frac{\beta}{m}t} + c_2 - \frac{gmt}{\beta}$$

Quindi alle C.I. (t = 0 → x = 0 e t = 0 → x' = 0) abbiamo:

$$\begin{cases} c_1 + c_2 = 0 \\ -\frac{\beta}{m}c_1 - \frac{gm}{\beta} = 0 \end{cases} \rightarrow \begin{cases} c_1 = -g\frac{m^2}{\beta^2} \\ c_2 = g\frac{m^2}{\beta^2} \end{cases}$$

Quindi la soluzione analitica della [4-3] alle C.I. nulle è:

$$x(t) = -g\frac{m^2}{\beta^2}e^{-\frac{\beta}{m}t} + g\frac{m^2}{\beta^2} - \frac{gmt}{\beta}$$

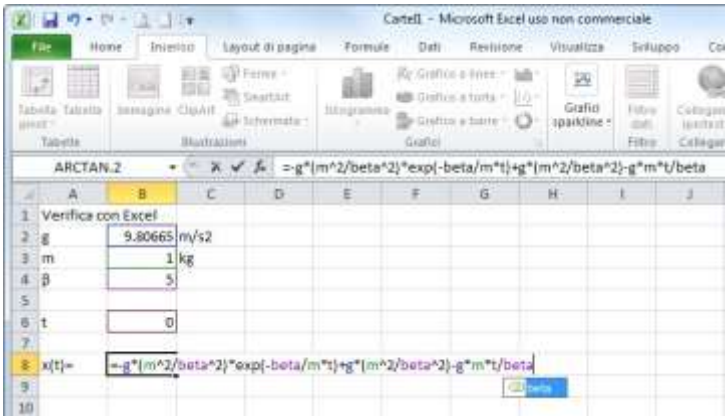
4-4: soluzione particolare alle C.I. nulle per la caduta di un grave in presenza d'attrito viscoso

Per la verifica, sostituiamo alla [4-4], g = 9.80665, m = 1, β= 5, così otteniamo:

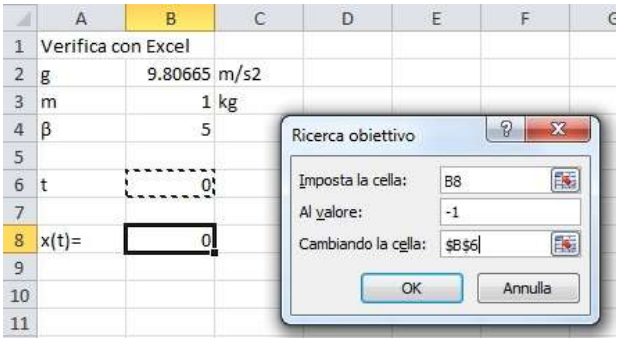
$$x(t) = -0.392266e^{-5t} - 1.96133t + 0.392266$$

Questa è certamente una equazione trascendente, quindi ci dovremo accontentare di una soluzione numerica approssimata[1].

Possiamo ad esempio utilizzare lo strumento **ricerca obiettivo** di Microsoft Excel: creiamo un foglio di calcolo con l'equazione da risolvere.




E invochiamo il comando ricerca obiettivo:




E quasi immediatamente compare una soluzione approssimata:

| | A | B | C |
|---|---|---|---|
| 1 | Verifica con Excel | | |
| 2 | g | 9.80665 | m/s2 |
| 3 | m | 1 | kg |
| 4 | β | 5 | |
| 5 | | | |
| 6 | t | 0.703518 | |
| 7 | | | |
| 8 | x(t)= | -0.9992 | |

Se vogliamo un'approssimazione ancora migliore possiamo utilizzare ad esempio l'algoritmo di Newton. Una comoda implementazione, pronta da usare, la possiamo trovare dentro a Maxima.

```
load(newton1);
newton(-0.392266*%e^(-5*t)-1.96133*t+0.392266+1, t, 0.7, 1e-15);
```

che fornisce come risultato: 0.70393633469845.

Abbiamo dunque effettuato alcune verifiche sul blocco massa che hanno confermato che le equazioni sono corrette e vengono risolte correttamente da OpenModelica.

[1] Una soluzione analitica di questa equazione è stata proposta attraverso lo sviluppo in serie della funzione W di Lambert $W_n(z)$, tramite la quale, è possibile eliminare la trascendenza, per cui la soluzione diviene:

$$b \neq 0, \quad g \neq 0, \quad m \neq 0, \quad t = \frac{g\,m^2\,W_n\left(-e^{-\frac{b^2}{g\,m^2}-1}\right) + b^2 + g\,m^2}{b\,g\,m}, \quad n \in \mathbb{Z}$$

## 4.3 Molle e smorzatori elasto-viscosi

### 4.3.1 La molla

Ovviamente la molla è fondamentale per una libreria meccanica ed è un componente indispensabile nell'oleodinamica, infatti è il primo strumento a disposizione del progettista per regolare la pressione.
La molla è descritta dalla **legge di Hooke**, rappresentata in Figura 4-2.

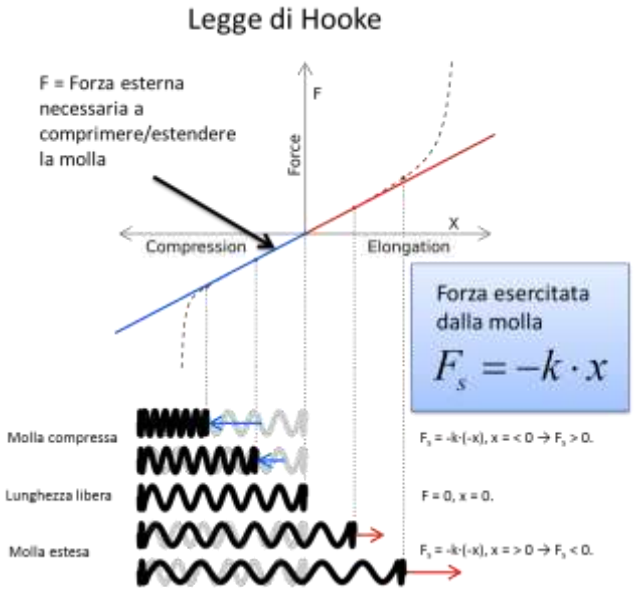



Figura 4-2: legge di Hooke

Nella legge di Hooke così espressa, la x ha il significato della posizione relativa rispetto alla lunghezza libera (alla quale facciamo corrisponde x = 0) che ne indica il grado di compressione o di estensione. Nella molla compressa x < 0 e nella molla estesa x>0.

1. Comprimere la molla (esercitando una forza negativa, verso sinistra) significa che la molla spinge verso destra con una forza $F_s = -k\cdot x$, essendo x negativa allora la forza esercitata dalla molla $F_s$ è positiva e quindi spinge verso destra.
2. Estendere la molla (esercitando una forza positiva, verso destra) significa che la molla tira verso sinistra con una forza $F_s = -k\cdot x$, essendo x positiva allora la forza esercitata dalla molla $F_s$ è negativa e quindi tira verso sinistra.

Nel modello Spring proposto è possibile impostare un eventuale forza iniziale di precarico $F_0$ (Spring preload) che è positiva se la molla è precompressa.
La forza elastica tiene conto del precarico iniziale e della compressione (o estensione) della molla.

```
model Spring "Compression Spring Model"
  extends Interfaces.PartialMechanicalBase;
  extends Interfaces.PartialViscoElastic2Ports;
  parameter Types.SpringConstant k = 100 "Spring stiffness value";
  parameter Types.Force F0 = 0 "Spring preload";
  Types.Length offset "Length due to the preload force";
equation
  offset = F0 / k;
  F = k * (offset + Delta_x);
  annotation(
   Icon(graphics = {Line(origin = {-18, -1}, points = {{-76, 1}, {-54, 1}, {-42, 61}, {-22, -59}, {8, 61}, {38, -59}, {58, 61}, {78,
-59}, {88, 1}, {116, 1}}, thickness = 1), Text(origin = {-8, -71}, extent = {{-92, 11}, {108, -9}}, textString = "k = %k N/mm"),
Text(origin = {-50, -92}, extent = {{-50, 12}, {150, -8}}, textString = "F0 = %F0 N")}, coordinateSystem(initialScale = 0.1)));
end Spring;
```

La classe parziale PartialViscoElastic2Ports si occupa di "mettere" F al giusto posto e col giusto segno nelle due flange.

### 4.3.2 Smorzatore viscoso

Lo smorzatore viscoso o damper ha un modello ancora più semplice rispetto alla molla: restituisce una forza, contraria al moto, proporzionale alla velocità. La costante c (in N·s/m) è detta coefficiente di smorzamento viscoso (damping constant).

```
model Damper "Damper Model"
  extends Interfaces.PartialMechanicalBase;
  extends Interfaces.PartialViscoElastic2Ports;
  parameter Types.DampingConstant c = 1000 "Damping constant";
equation
  F = c * Delta_v;
  annotation(
    Icon(graphics = {Text(origin = {-8, -71}, extent = {{-92, 11}, {108, -9}}, textString = "c = %c N.s/m"), Line(points = {{-20,
40}, {20, 40}, {20, -40}, {-20, -40}}, thickness = 1), Line(origin = {56, 0}, points = {{36, 0}, {-36, 0}}, thickness = 1),
Line(origin = {-48, 0}, points = {{-48, 0}, {48, 0}}, thickness = 1), Line(points = {{0, 36}, {0, -36}}, thickness = 1)},
coordinateSystem(initialScale = 0.1)));
end Damper;
```

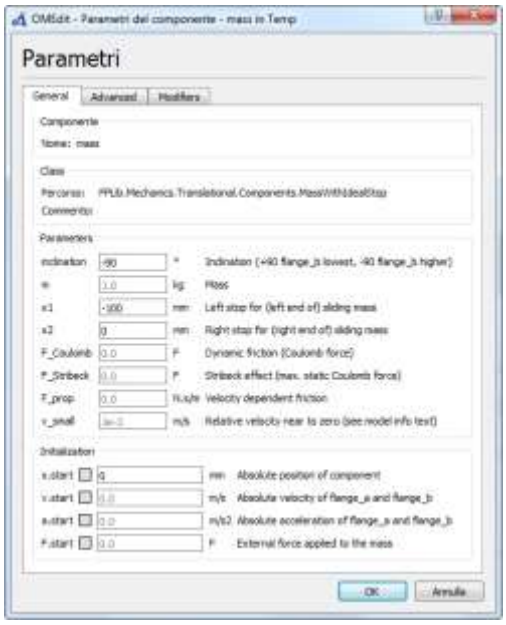
### 4.3.3 Smorzatore elasto-viscoso

E' costituito da una molla e da uno smorzatore viscoso.

```
model SpringDamper "Compression Spring Model"
  extends Interfaces.PartialMechanicalBase;
  extends Interfaces.PartialViscoElastic2Ports;
  parameter Types.SpringConstant k = 100 "Spring stiffness value";
  parameter Types.Force F0 = 0 "Spring preload";
  parameter Types.DampingConstant c = 1000 "Damping constant";
  Types.Length offset "Length due to the preload force";
equation
  offset = F0 / k;
  F = k * (offset + Delta_x) + c * Delta_v;
  annotation(
    Icon(graphics = {Line(origin = {-18, 39}, points = {{-62, 1}, {-54, 1}, {-42, 41}, {-22, -39}, {8, 41}, {38, -39}, {58, 41}, {78,
-39}, {88, 1}, {98, 1}}, thickness = 1), Text(origin = {-8, -71}, extent = {{-84, 11}, {98, -9}}, textString = "k = %k N/mm, F0 = %F0
N"), Text(origin = {-50, -92}, extent = {{-42, 12}, {136, -6}}, textString = "c = %c N.s/m"), Line(origin = {-46, -40}, points = {{-
34, 0}, {46, 0}}, thickness = 1), Line(origin = {0, -40}, points = {{-20, -20}, {20, -20}, {20, 20}, {-20, 20}}, thickness = 1),
Line(origin = {56, -40}, points = {{24, 0}, {-36, 0}}, thickness = 1), Line(origin = {-92, 1}, points = {{12, 39}, {12, -41}},
thickness = 1), Line(origin = {92, 0}, points = {{-12, 40}, {-12, -40}}, thickness = 1), Line(origin = {-94, 0}, points = {{14, 0},
{-2, 0}}, thickness = 1), Line(origin = {82, 0}, points = {{14, 0}, {-2, 0}}, thickness = 1), Line(origin = {0, -40}, points = {{0,
16}, {0, -16}}, thickness = 1)}, coordinateSystem(initialScale = 0.1)));
end SpringDamper;
```

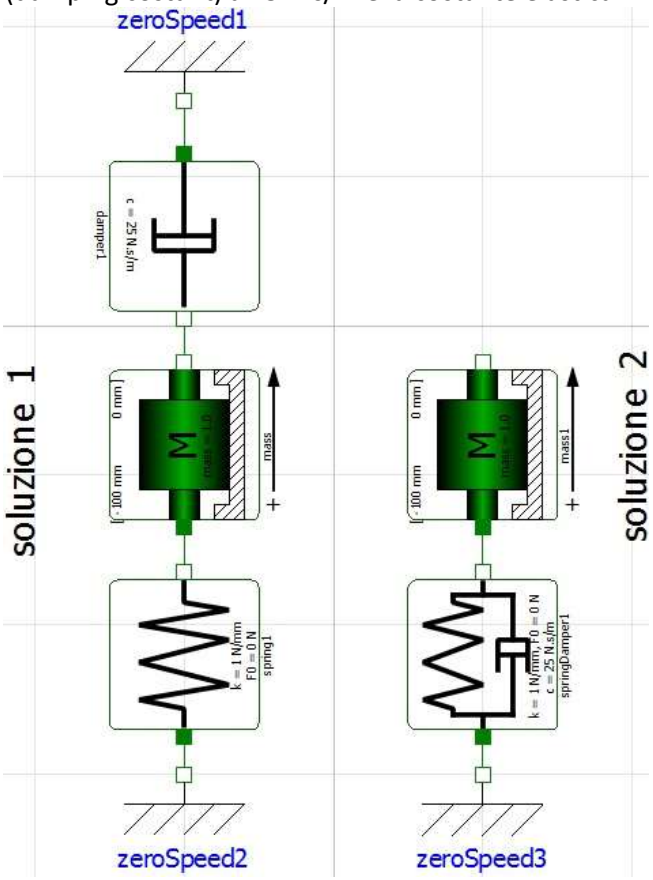
### 4.3.4 Esempio di un sistema massa-molla-smorzatore viscoso

Problema: Una massa m = 1 kg, viene appoggiata su una molla di lunghezza libera L0 = 0.1 m, con costante elastica k = 1000 N/m. Immaginiamo che sia presente anche un coefficiente d'attrito viscoso c = 25 N·s/m.

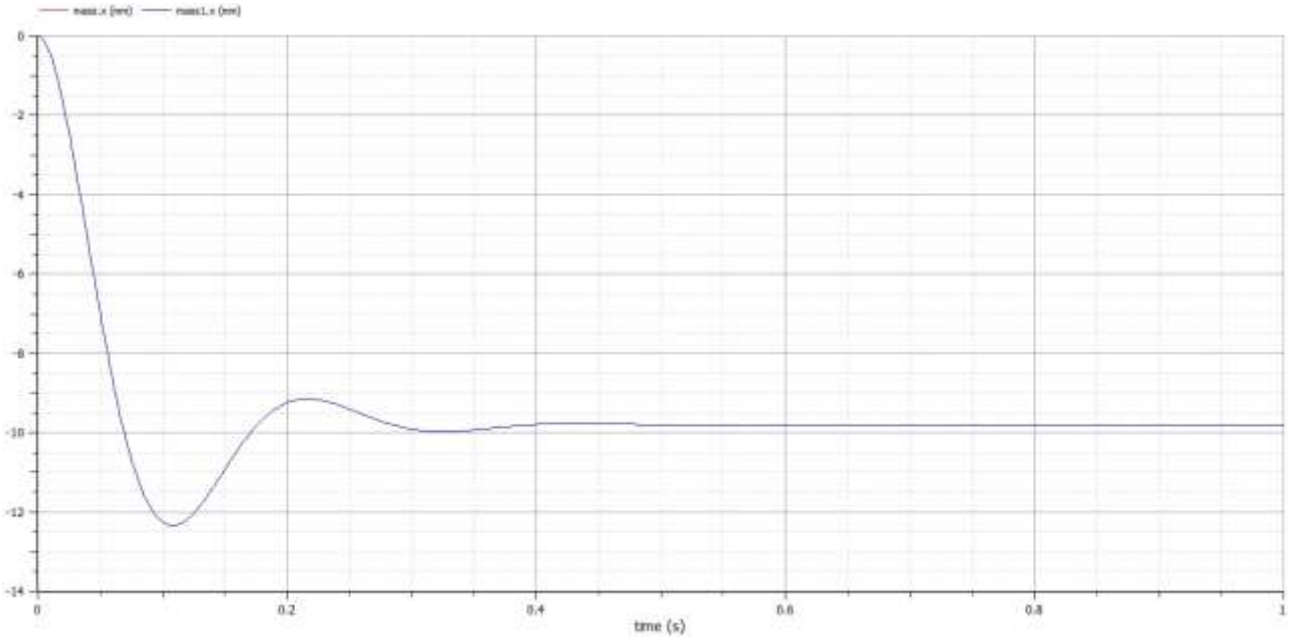



Soluzione: posizioniamo un blocchetto MassWithIdealStop rivolto con la flangia b verso l'alto. Cambiamo i parametri con: inclination = -90 ° (così la forza peso è contraria alla freccia), x1 = -100 mm, x2 = 0 e x.start = 0.
La scheda parametri dovrà risultare dunque come nella figura a lato.

Andiamo a connettere uno smorzatore viscoso e una molla (soluzione 1) oppure uno smorzatore elasto-viscoso (soluzione 2). Nel primo caso, allo smorzatore viscoso va aggiunto un Blocco ZeroSpeed perché un terminale dello smorzatore ha velocità nulla mentre nel secondo caso non ci deve essere perché la massa si deve poter muovere.
Sotto alla molla nel primo caso e, nel secondo caso sotto allo smorzatore elasto-viscoso, si deve usare il Blocco ZeroSpeed perché la molla e lo smorzatore elasto-viscoso sono vincolati nella parte inferiore.
Impostiamo il parametro c (damping costant) a 25 N·s/m e la costante elastica k = 1 N/mm.




Se lanciamo la simulazione otteniamo.




Cioè la molla s'accorcia di circa 9.81 mm, infatti la x passa da zero a -9.80665 mm.
La soluzione "all'equilibrio" è corretta, come si può facilmente dimostrare eguagliando la forza peso con la forza elastica:

$$m \cdot g = -kx \rightarrow x = -\frac{m \cdot g}{k} = -\frac{1 \cdot 9.80665}{1000} = -9.80665 \cdot 10^{-3} m$$

Per la soluzione transiente invece:

$$m \cdot \ddot{x} + c \cdot \dot{x} + k \cdot x = -m \cdot g$$

**4-5: seconda legge di Newton per oscillatore armonico smorzato-forzato**

Nella [4-5] il termine forzante non è periodico. Matematicamente è una equazione differenziale ordinaria lineare del secondo ordine, a coefficienti costanti, non omogenea, la cui omogenea è priva del termine forzante. La soluzione particolare della [4-5] è ovviamente $x_p = -m \cdot g/k$.
L'equazione omogenea è:

$$\ddot{x} + \frac{c}{m} \cdot \dot{x} + \frac{k}{m} \cdot x = 0$$

Che possiamo scrivere come

$$\ddot{x} + 2\gamma \cdot \dot{x} + \omega_0^{\,2} \cdot x = 0$$

**4-6: Oscillatore armonico smorzato**

Con $\gamma = c/(2m)$ = coefficiente di smorzamento e $\omega_0^{\,2} = k/m$ ($\omega_0$ = pulsazione propria).
Possiamo avere tre casi:

1. $\gamma^2 > \omega_0^{\,2}$ (smorzamento forte).
2. $\gamma^2 = \omega_0^{\,2}$ (smorzamento critico).
3. $\gamma^2 < \omega_0^{\,2}$ (smorzamento debole).

Noi siamo in quest'ultimo caso, con i valori di m, c e k correnti.
Le soluzioni dell'equazione caratteristica sono complesse coniugate.

$$\alpha_1 = -\gamma + i\sqrt{\omega_0^{\,2} - \gamma^2} = -\gamma + i\omega \quad \text{e} \quad \alpha_2 = -\gamma - i\sqrt{\omega_0^{\,2} - \gamma^2} = -\gamma - i\omega$$

E la soluzione generale della [4-6] è:

$$x_g(t) = Ae^{\alpha_1 t} + Be^{\alpha_2 t} = e^{-\gamma t}\left(Ae^{i\omega t} + Be^{-i\omega t}\right)$$

Utilizzando la formula di Eulero

$$e^{\pm i\omega t} = \cos(\omega t) \pm i \cdot sin(\omega t)$$

Si ottiene

$$x_g(t) = e^{-\gamma t}\left[(A+B)\cos(\omega t) + (A-B)i\sin(\omega t)\right]$$

Il risultato deve essere reale e inoltre A e B devono essere diversi, ciò comporta che anche A e B siano complessi coniugati:

$$A = a + ib \quad , \quad B = a - ib \rightarrow A + B = 2a \quad , \quad A - B = 2ib$$

Effettuando queste sostituzioni l'espressione di x(t) diventa:

$$x_g(t) = e^{-\gamma t}\left[2a\cos\omega t - 2b\sin\omega t\right]$$

Che si può scrivere come:

$$x_g(t) = A_0 e^{-\gamma t}\sin(\omega t + \varphi)$$

Quindi la soluzione generale della [4-5] risulta (principio di sovrapposizione degli effetti):

$$x(t) = A_0 e^{-\gamma t}\sin(\omega t + \varphi) - \frac{mg}{k}$$

Dalle C.I. si ha:

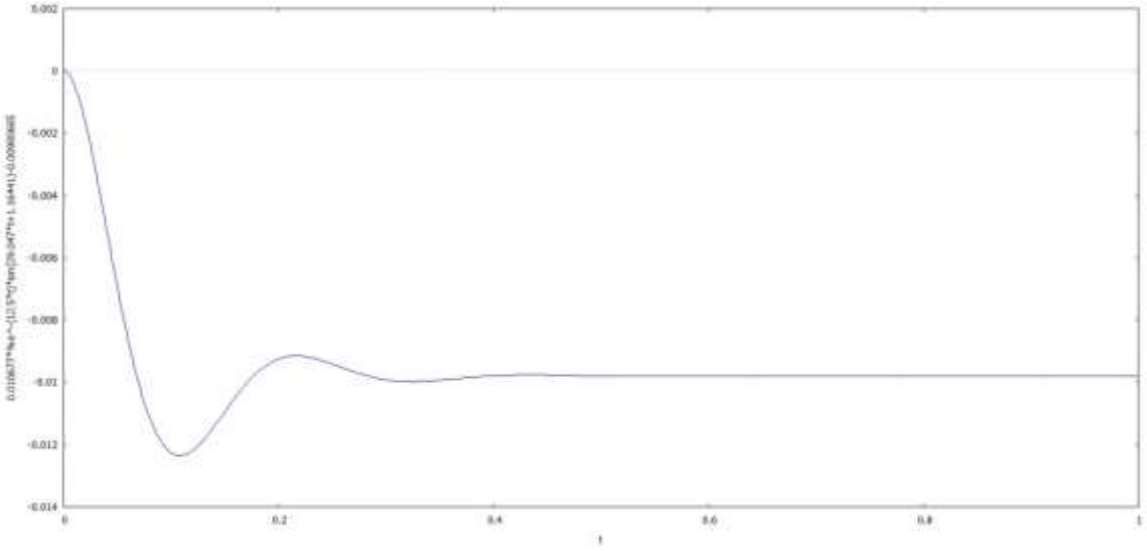
$$\begin{cases} A_0 \sin\varphi - \dfrac{gm}{k} = 0 \\ A_0\left(\omega\cos\varphi - \gamma\sin\varphi\right) = 0 \end{cases} \rightarrow \begin{cases} A_0 = \dfrac{gm}{k\sin\varphi} \\ \omega\cos\varphi - \gamma\sin\varphi = 0 \end{cases} \rightarrow \begin{cases} A_0 = \dfrac{gm}{k\sin\varphi} \\ \tan\varphi = \dfrac{\omega}{\gamma} \end{cases}$$ g

Nel nostro caso abbiamo $\gamma = c/(2m)$ = 12.5, $\omega_0^2 = k/m$ = 1000, $\omega = \sqrt{\omega_0^2 - \gamma^2}$ ≈ 29.047, quindi ricaviamo: $\varphi$ ≈ 1.16441 rad e $A_0$ ≈ 0.010677.

Se sostituiamo alla soluzione generale le C.I. e abbiamo la soluzione particolare:

$$x(t) = 0.010677e^{-12.5t}\sin(29.047t + 1.16441) - 9.80665\cdot 10^{-3}$$

Che per t = [0,1] fornisce il seguente grafico:



Ovviamente questo risultato è in metri, ma corrisponde esattamente alla soluzione fornita da OpenModelica.

Disponiamo quindi di alcuni blocchi fondamentali corrispondenti ad altrettanti elementi della meccanica: la massa, la molla e lo smorzatore.

## 4.4 Blocco ZeroSpeed

Anche se non ne abbiamo parlato, si vede dagli schemi a blocchi Modelica che abbiamo dovuto introdurre dei blocchi ZeroSpeed.
Semplicemente, la classe zeroSpeed, ha una porta in cui la posizione x è mantenuta costante.
La costanza è determinata dall'equazione:

$$\frac{dx}{dt} = 0$$

Come si può vedere dal pezzo di codice seguente:

```
model ZeroSpeed "Fixed flange"
  Interfaces.Flange_b flange annotation( ...);
  Types.Position x(start = 0) "Position";
equation
  flange.s = x;
  der(x) = 0;
  annotation( ...);
end ZeroSpeed;
```

Poiché il codice non vincola la forza ad una equazione, la forza sarà ricavata dalla soluzione delle equazioni contenute negli altri blocchi.

## 4.5 Blocco Force Convertion

E' un blocco che converte un segnale (generato da un blocco standard della libreria Modelica, Modelica→Blocks→Sources) in una forza.
Il vantaggio dell'approccio suggerito da Modelica e utilizzato in questa libreria è quello di poter utilizzare gran parte dei blocchi della libreria standard. Il fatto di poter lavorare ad esempio con la libreria dei segnali (Modelica→Blocks) significa che come input sarà possibile utilizzare i segnali "sorgenti" (Modelica→Blocks→Sources) ma anche di compiere tutte le operazioni previste dalla libreria standard.
Ecco un frammento di codice per il semplice blocco:

```
model ForC
  extends Interfaces.PartialOrientedReferenceSystemConfirmation;
  // F = -u
  Modelica.Blocks.Interfaces.RealInput u annotation( ...);
  Interfaces.Flange_a flange_a annotation( ...);
equation
  flange_a.f = -u * v_sgn;
  annotation( ...);
end ForC;
```

In pratica la cosa più importante da comprendere, per ora, è che la forza sull'attacco flange_a, è il segnale di input cambiato di segno (-u). Il motivo del cambiamento di segno dovrebbe essere noto: Modelica considera uscente una grandezza con segno negativo.
La moltiplicazione per v_sgn è necessaria poiché la forza è orientata e quindi se ruotiamo il blocco (e impostiamo il segno come negativo) allora il segno della forza viene cambiato. Il booleano v_sgn è collegato ad un parametro che si chiama FromLeft2Right (il nome è improprio ma quando avevo iniziato a scrivere la libreria il codice era meno sofisticato). Tuttavia, a parte il nome, FromLeft2Right deve essere settato a true, quando la freccia è concorde alla freccia presente nella massa. *Impostare correttamente questo flag è fondamentale per indicare che una forza positiva tende a far crescere la x. In pratica se ho una massa posta in x = 0, orientata con freccia verso destra, se la voglio spostare verso destra (x>0), inserisco un blocco ForC con freccia concorde e segnale positivo*.
Vedremo meglio come si usa, nel prossimo esempio.

### 4.5.1 Esempio: oscillatore armonico smorzato forzato

Il sistema massa-molla-smorzatore (Figura 4-3),con massa m = 1 kg, molla di costante elastica k = 1 N/mm non precaricata ($F_0$ = 0), costante di smorzamento viscoso c = 25 Ns/m è sollecitato da una forza a gradino unitario.

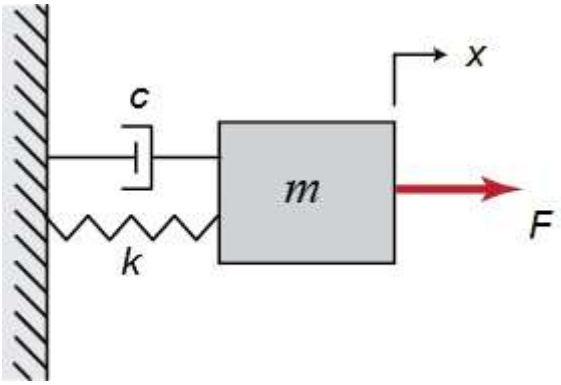
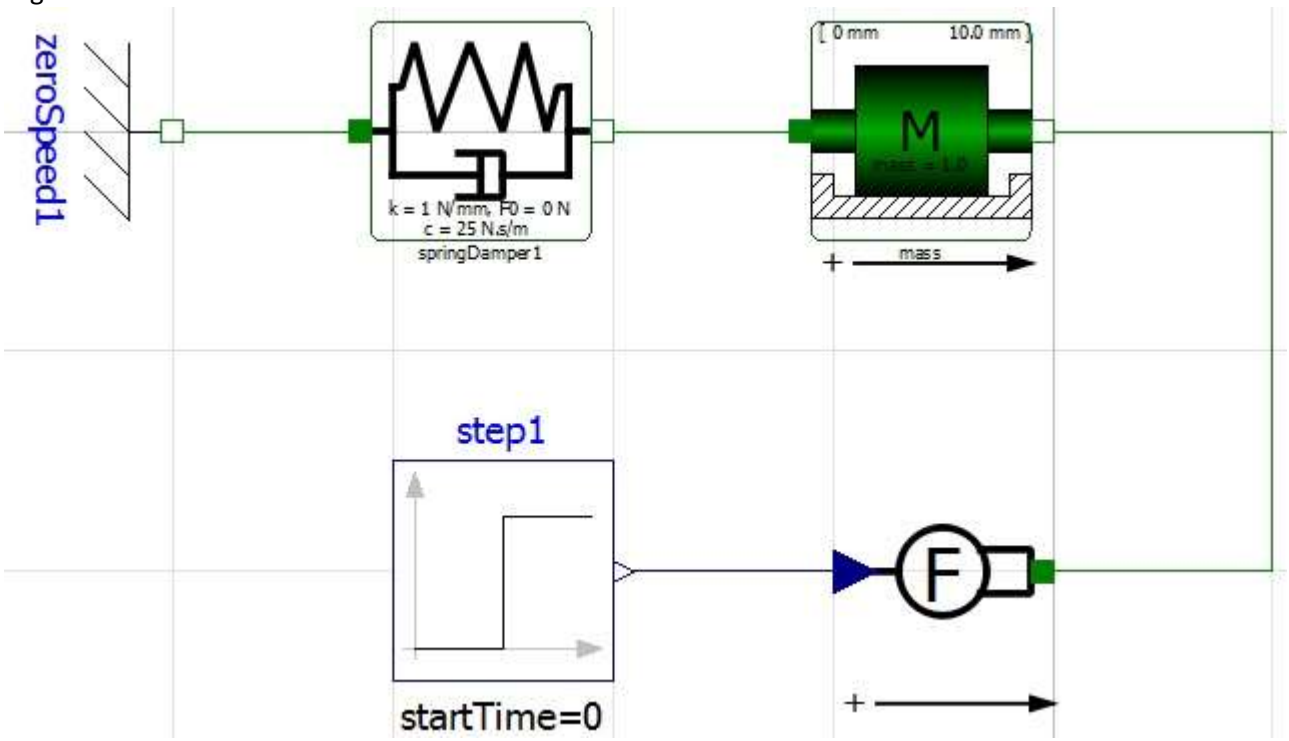
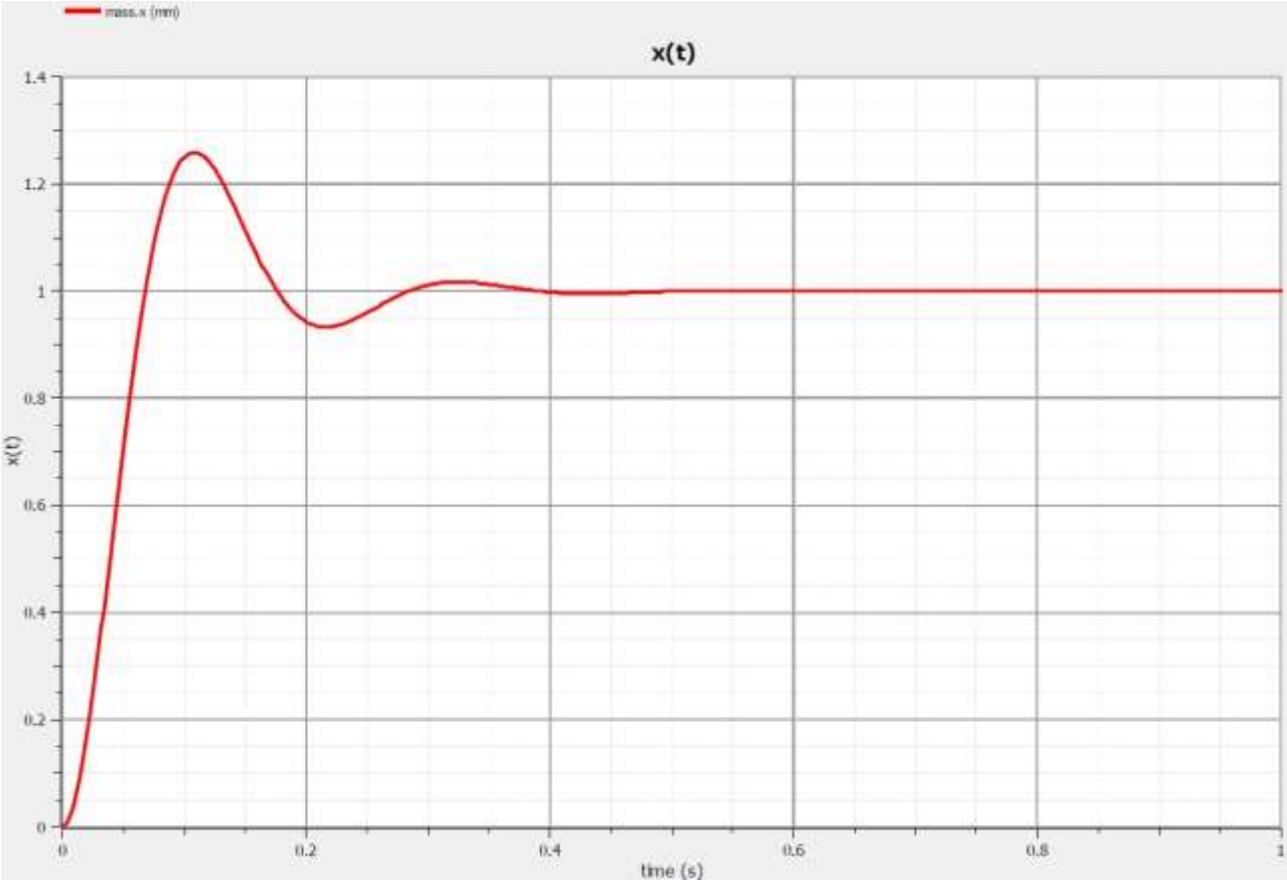



Figura 4-3: oscillatore armonico smorzato forzato

Con OpenModelica e con i comandi della libreria che abbiamo descritto fino ad ora possiamo realizzare il seguente schema:




La massa la facciamo scorrere da 0 mm fino a 10 mm.
Notare come la freccia della forza sia concorde con la freccia della massa. La freccia verso destra della massa intende dire che la x (posizione) cresce verso destra. La freccia concorde del blocco ForC con la massa indica che un segnale positivo tira (o spinge) la massa in modo da fare aumentare la x (quindi verso destra).
Se lanciamo la simulazione otteniamo:




Per verificare la correttezza del risultato, applichiamo la seconda legge di Newton:

$$m\frac{d^2x}{dt^2} = F - c\frac{dx}{dt} - kx$$

Definito $\gamma = \frac{c}{2m}$ e $\omega_0{}^2 = \frac{k}{m}$ riscriviamo la seconda di Newton come:

$$\ddot{x} + 2\gamma\dot{x} + \omega_0{}^2 x = F$$

**4-7: oscillatore armonico smorzato forzato**

Anche in questo caso abbiamo smorzamento debole ($\gamma^2 < \omega_0{}^2$), quindi il metodo di soluzione analitica sarà simile a quello adottato nel paragrafo 4.3.4. Quindi la soluzione particolare della [4-7] è:

$$x_p = F\frac{m}{k}$$

La soluzione generale della [4-7] è dunque (principio di sovrapposizione):

$$x(t) = A_0 e^{-\gamma t}\sin(\omega t + \varphi) + F\cdot\frac{m}{k}$$

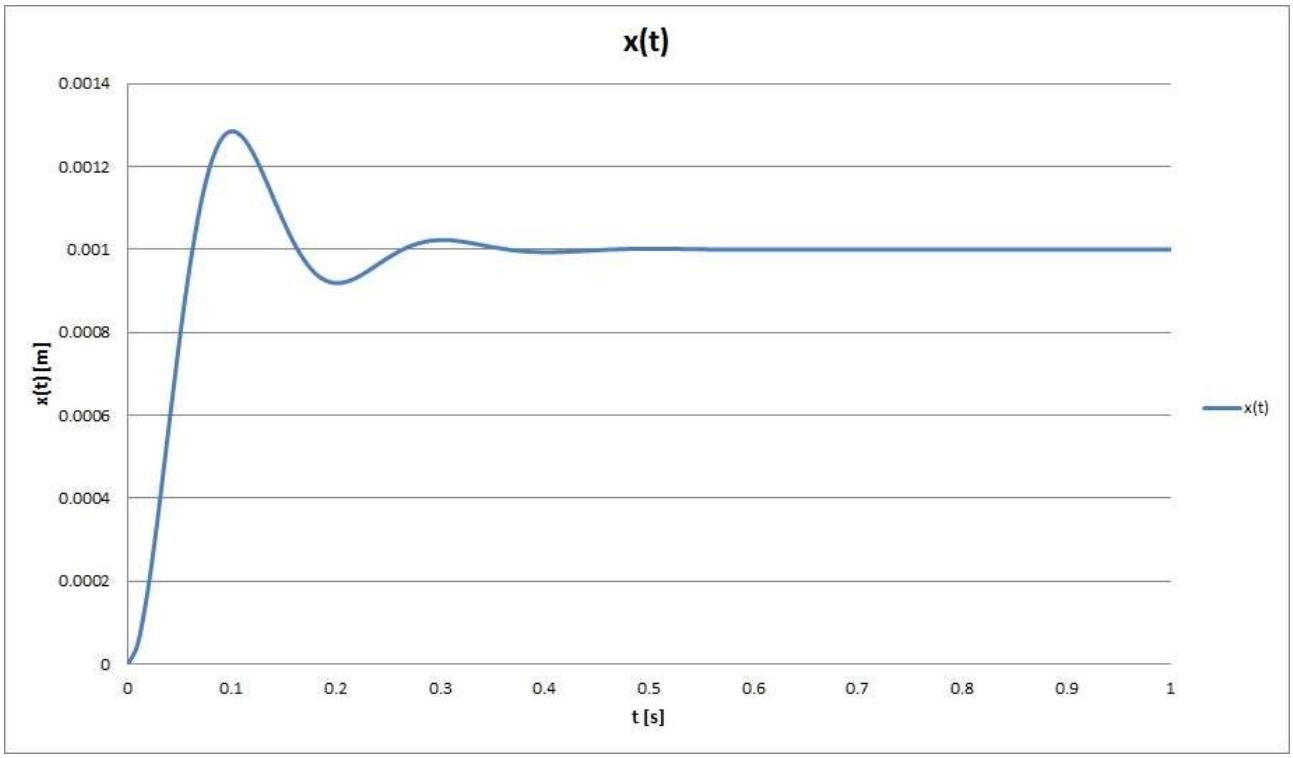
Dalle C.I. x(0)=0, ẋ(0) = 0 abbiamo:

$$\begin{cases} A_0\sin\varphi - F\frac{m}{k} = 0 \\ A_0\omega\cos\varphi - A_0\gamma\sin\varphi = 0 \end{cases} \rightarrow \begin{cases} A_0 = -F\frac{m}{k}\cdot\frac{\gamma}{\omega}\cdot\sqrt{\frac{\omega^2}{\gamma^2}+1} \\ \varphi = \arctan\frac{\omega}{\gamma} \end{cases}$$

Poiché $\omega = \sqrt{\omega_0{}^2 - \gamma^2}$, con m = 1 kg, k = 1000 N/m, c = 25 Ns/m abbiamo $A_0 = -1.08\cdot10^{-3}$ e $\varphi = 1.19$, risulta:

$$x(t) = \frac{1}{1000} - 1.08\cdot10^{-3}e^{-\frac{25}{2}t}\cdot\sin\left(5\cdot\sqrt{39}\,t + 1.19\right)$$

Se grafichiamo questa funzione otteniamo la posizione, in metri:




Che verifica il risultato ottenuto utilizzando OpenModelica e la nostra libreria.
Sebbene esistono altri importanti componenti meccanici, consideriamo esaurita questa parte.
L'obiettivo di questo testo non è infatti quello di fornire una libreria completa, come quella di AMESim, ma di far vedere come sia possibile implementarne una. La porta meccanica e i componenti che la utilizzano delle libreria meccanica sono infatti fondamentali per procedere verso la realizzazione di una libreria in stile ***Hydraulic Component Design*** di AMESim.

# 5 Hydraulic Library

## 5.1 Modelli parziali d'interfaccia

### 5.1.1 Classi connector

Una porta idraulica ha come grandezza potenziale la pressione e come grandezza di flusso la portata (volumica). Creiamo quindi due porte, FluidPort_a e FluidPort_b, così definite:

```
connector FluidPort_a "(left) 1D translational flange (flange axis directed INTO cut plane, e. g.
from left to right)"
  Types.Pressure p "Absolute position of flange";
  flow Types.VolumeFlowRate q "Cut force directed into flange";
  annotation(
    defaultComponentName = "A",
    Icon(coordinateSystem(initialScale = 0.1), graphics = {Ellipse(fillColor = {255, 0, 0}, fillPattern = FillPattern.Sphere, extent
= {{-100, 100}, {100, -100}}, endAngle = 360)}),
    Diagram(coordinateSystem(initialScale = 0.1), graphics = {Text(origin = {76, 24}, lineColor = {0, 127, 0}, extent = {{-160, 110},
{0, 70}}, textString = "%name"), Ellipse(fillColor = {255, 0, 0}, fillPattern = FillPattern.Sphere, extent = {{-100, 100}, {100, -
100}}, endAngle = 360)}));
end FluidPort_a;
```

```
connector FluidPort_b "(left) 1D translational flange (flange axis directed INTO cut plane, e. g.
from left to right)"
  Types.Pressure p "Absolute position of flange";
  flow Types.VolumeFlowRate q "Cut force directed into flange";
  annotation(
    defaultComponentName = "B",
    Icon(coordinateSystem(initialScale = 0.1), graphics = {Ellipse(fillColor = {0, 0, 255}, fillPattern = FillPattern.Sphere, extent
= {{-100, 100}, {100, -100}}, endAngle = 360)}),
    Diagram(coordinateSystem(initialScale = 0.1), graphics = {Text(origin = {76, 24}, lineColor = {0, 127, 0}, extent = {{-160, 110},
{0, 70}}, textString = "%name"), Ellipse(fillColor = {0, 0, 255}, fillPattern = FillPattern.Sphere, extent = {{-100, 100}, {100, -
100}}, endAngle = 360)}));
end FluidPort_b;
```

In pratica sono identiche, ma le differenziamo per chiarezza nei blocchi. La stessa scelta è stata fatta per le porte meccaniche Flange_a e Flange_b.

### 5.1.2 Il modello Partial2FluidPorts

E' una classe parziale che incorporta le due porte appena create:

```
partial model Partial2FluidPorts
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation( ...);
  FPLib.Hydraulic.Interfaces.FluidPort_b B annotation( ...);
end Partial2FluidPorts;
```

### 5.1.3 Il modello Partial2FluidPortsWithNullNetFlow

E' una classe parziale che incorpora le seguenti equazioni:

$$\begin{cases} \Delta p = A.p - B.p \\ A.q + B.q = 0 \\ q = A.q \end{cases}$$

5-1: equazioni per un bipolo idraulico

Ecco il codice:

```
partial model Partial2FluidPortsWithNullNetFlow
  extends Partial2FluidPorts;
  Types.Pressure dp;
  Types.VolumeFlowRate q;
equation
  dp = A.p - B.p;
  A.q + B.q = 0;
  q = A.q;
end Partial2FluidPortsWithNullNetFlow;
```

### 5.1.4 Il modello Partial2FluidPortsWithVolume

E' una classe parziale che incorpora due volumi per le due porte idrauliche, estendendo Partial2FluidPorts.

```
partial model Partial2FluidPortsWithVolume
```

```
  extends Interfaces.PartialHydraulicBase;
  extends Interfaces.Partial2FluidPorts;
  // parametri e variabili generali
  parameter Types.Diameter dA = 100 "Inlet diameter at fluid port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Diameter dB = 100 "Inlet diameter at fluid port B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdAmax = 0.7 "Max. Cd at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdBmax = 0.7 "Max. Cd at port B" annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecA = 1000 "Critical Reynolds number at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecB = 1000 "Critical Reynolds number at port B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolA0 = 10 "Initial volume at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolB0 = 10 "Initial volume at port B" annotation(
    Dialog(tab = "Advanced"));
  Types.Pressure pA(start = 0) "Pressure on volume A";
  Types.Pressure pB(start = 0) "Pressure on volume B";
  Types.Pressure dp(start = 0) "DeltaP across the edge (dp = pA - pB)";
  Types.VolumeFlowRate q(start = 0) "Flow across the edge (from A to B)";
  Types.VolumeFlowRate qA(start = 0) "Flow at inlet port A";
  Types.VolumeFlowRate qB(start = 0) "Flow at inlet port B";
  SI.ReynoldsNumber ReA "Reynold number at port A";
  SI.ReynoldsNumber ReB "Reynold number at port B";
  SI.Velocity wA "fluid mean velocity at port A";
  SI.Velocity wB "fluid mean velocity at port B";
  Types.Dimensionless CdA "Discharge coefficient at port A";
  Types.Dimensionless CdB "Discharge coefficient at port B";
protected
  Types.Area AA = Constants.pi * dA ^ 2 / 4 "Inlet section at port A";
  Types.Area AB = Constants.pi * dB ^ 2 / 4 "Inlet section at port B";
equation
// hydraulic ports equations
  (qA, wA, ReA, CdA) = Functions.orif3f(A.p, pA, AA, dA, CdAmax, RecA);
  (qB, wB, ReB, CdB) = Functions.orif3f(pB, B.p, AB, dB, CdBmax, RecB);
  der(pA) = Functions.hCapacitance(qA, q, VolA0, 0, pA, Constants.bulk);
  der(pB) = Functions.hCapacitance(q, qB, VolB0, 0, pB, Constants.bulk);
  A.q = qA;
  B.q = -qB;
  dp = pA - pB;
end Partial2FluidPortsWithVolume;
```

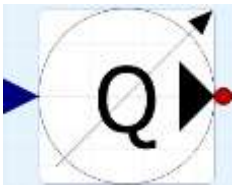
Una caratteristica importante di questa classe, che ritroviamo anche in molti altri blocchi idraulici, è che definisce gli attacchi delle porte:

1. I diametri dA e dB delle due porte idrauliche A e B.
2. I coefficienti d'efflusso massimi $C_{d,max}$ per le due porte idrauliche A e B.
3. I numeri di Reynolds critici per le due porte idrauliche A e B.
4. I volumi (fissi) per due porte idrauliche A e B.

La scelta di considerare già i volumi ha il vantaggio che non è necessario costellare i modelli con blocchi che costituiscono i volumi, i contro sono che ciascuna porta, naturalmente, avrà ora un ΔP di attraversamento. A tal fine, per limitare questo effetto negativo, è stato scelto di default un attacco di diametro 100 mm: in questo modo la caduta di pressione sarà pressoché nulla, anche a portate elevate.
Di contro, sistemi con portate basse, possono creare qualche problema al risolutore che emetterà dei warning, ma sarà generalmente in grado di risolvere il sistema. Se dovessero comparire troppi messaggi del tipo "**Warning: maximal number of iteration reached but no root found**" si può provare a diminuire i diametri dA e dB. Il problema, infatti, risiede nel fatto che con un diametro così grande, il gradiente di pressione calcolato dalla funzione hCapacitance diventa molto grande e quindi il solutore numerico incontra difficoltà a determinare il valore corretto anche con passi d'integrazione molto piccoli.
Un altro aspetto da valutare è che questa classe, costituisce l'impalcatura di tutte le valvole di regolazione a due vie e avendo al suo interno due volumi, istantaneamente è possibile che anche una valvola in cui il flusso sia interdetto, la portata sulle sue bocche sia temporaneamente diversa da zero.
Tuttavia compilando debitamente i parametri succede quello che avviene nella realtà, poiché all'ingresso e all'uscita di ogni valvola ci sono piccoli volumi tra le parti mobili e le tubazioni.

## 5.2 Generatore di portata ideale

Il generatore di portata ideale è un blocco che accetta un segnale e genera una portata uguale al segnale in ingresso.

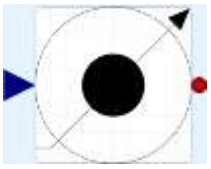



Ecco il codice:

```
model VariableFlowSource
  extends Interfaces.PartialHydraulicBase;
  Modelica.Blocks.Interfaces.RealInput u annotation(
    Placement(visible = true, transformation(origin = {-120, -100}, extent = {{-20, -20}, {20, 20}}, rotation = 0),
iconTransformation(origin = {-120, 0}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
    Placement(visible = true, transformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Types.VolumeFlowRate q "Volume Flow Rate";
  Types.Pressure p "Pressure";
equation
  q = u;
  A.q = -q;
  p = A.p;
  annotation(
    Icon(graphics = {Ellipse(lineThickness = 0.5, extent = {{-100, 100}, {100, -100}}, endAngle = 360), Text(extent = {{-60, 60},
{60, -60}}, textString = "Q"), Line(origin = {-15.8, 6.2}, points = {{-84.1996, -86.1996}, {-64.1996, -86.1996}, {105.8, 83.8004}},
thickness = 0.5), Polygon(origin = {88, 88}, fillPattern = FillPattern.Solid, points = {{10, 10}, {-18, -2}, {-2, -18}, {10, 10}}),
Polygon(origin = {80, 0}, fillPattern = FillPattern.Solid, points = {{20, 0}, {-20, 40}, {-20, -40}, {20, 0}})},
coordinateSystem(initialScale = 0.1)));
end VariableFlowSource;
```

## 5.3 Sorgente di pressione

La sorgente di pressione accetta un segnale in ingresso e imposta all'uscita la pressione definita dal segnale stesso.

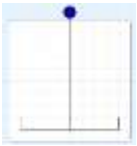


Ecco il codice:

```
model VariablePressureSource
  extends Interfaces.PartialHydraulicBase;
  Modelica.Blocks.Interfaces.RealInput u annotation(
    Placement(visible = true, transformation(origin = {-120, -100}, extent = {{-20, -20}, {20, 20}}, rotation = 0),
iconTransformation(origin = {-120, 0}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
    Placement(visible = true, transformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Types.Pressure p "Pressure";
  Types.VolumeFlowRate q "Volume Flow Rate";
equation
  p = u;
  A.p = p;
  q = A.q;
  annotation(
    Icon(graphics = {Ellipse(lineThickness = 0.5, extent = {{-100, 100}, {100, -100}}, endAngle = 360), Line(origin = {-15.8, 6.2},
points = {{-84.1996, -86.1996}, {-64.1996, -86.1996}, {105.8, 83.8004}}, thickness = 0.5), Polygon(origin = {88, 88}, fillPattern =
FillPattern.Solid, points = {{10, 10}, {-18, -2}, {-2, -18}, {10, 10}}), Ellipse(fillPattern = FillPattern.Solid, extent = {{-40,
40}, {40, -40}}, endAngle = 360)}, coordinateSystem(initialScale = 0.1)));
end VariablePressureSource;
```

## 5.4 Serbatoio

Imposta la pressione a zero.



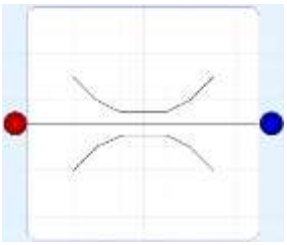
Ecco il codice:

```
model Tank
  extends Interfaces.PartialHydraulicBase;
  FPLib.Hydraulic.Interfaces.FluidPort_b B annotation(
    Placement(visible = true, transformation(origin = {0, 108}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {0, 112}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
equation
  B.p = 0;
  annotation(
    Icon(graphics = {Line(origin = {0, -70}, points = {{-80, 10}, {-80, -10}, {80, -10}, {80, 10}}, thickness = 0.5), Line(origin =
{0, 12}, points = {{0, -92}, {0, 88}}, thickness = 0.5)}, coordinateSystem(initialScale = 0.1)));
end Tank;
```

Come esercizio, si potrebbe aggiungere un parametro, così da simulare anche serbatoi pressurizzati.

## 5.5 Orificio

Andiamo ora a definire un orificio:



Un orificio ha un diametro d, che rappresenta un brusco restringimento di sezione.
In questo caso il diametro idraulico equivalente è uguale a d, quindi $d_h$ = d.
A un orificio s'associa un coefficiente d'efflusso $C_d$, un numero di Reynolds critico, che dipendono dalla geometria.
La formula per il calcolo della portata, per un orificio, è definita come:

$$Q = C_d \cdot A_0 \cdot \sqrt{\frac{2}{\rho} \cdot |\Delta P|} \cdot sign(\Delta P)$$

**5-2: legge dello strozzatore in regime turbolento**

La funzione ***orif3f*** è in grado di adattare la legge [5-2] anche al caso d'efflusso laminare.
Se $p_1$ è la pressione a monte dello strozzatore e $p_2$ è la pressione a valle, allora, la velocità massima è:

$$w_{\max} = \sqrt{\frac{2}{\rho} \cdot \Delta P}$$

**5-3: velocità massima del fluido in un orificio**

Dove chiaramente $\Delta P = p_1 - p_2$.

Il numero di Reynolds è dato da:

$$\mathrm{Re} = w_{\max} \cdot d_h \cdot \frac{\rho}{\mu}$$

**5-4: numero di Reynolds in un orificio o per una sezione qualunque**

La [5-4] vale per un orificio o per qualunque sezione circolare in cui dh = d oppure per qualunque sezione calcolando il diametro idraulico equivalente ( dh = 4·A/P, dove A è la sezione di passaggio e P il perimetro "bagnato dal fluido").

La formula [5-2] è valida sia nel caso d'efflusso turbolento, che nel caso d'efflusso laminare, sostituendo a $C_d$, un coefficiente d'efflusso $C_q$ calcolato in funzione del numero di Reynolds e del numero di Reynolds critico:

$$C_q = C_d \cdot \tanh\left(\frac{2 \cdot \mathrm{Re}}{\mathrm{Re}_{\mathrm{critic}}}\right)$$

**5-5: coefficiente d'efflusso unificato laminare-turbolento**

La velocità media, infine è calcolata con:

$$w_{mean} = C_q \cdot w_{\max}$$

**5-6: velocità media del fluido in un orificio**

Ed ecco il codice:

```
model Orifice
  extends Interfaces.PartialHydraulicBase;
  extends Interfaces.Partial2FluidPortsWithNullNetFlow;
  parameter Types.Diameter d(min = 0) = 5 "orifice diameter";
  parameter Types.Dimensionless Cd = 0.7 "discharge coefficient";
  parameter SI.ReynoldsNumber Rec = 1000 "critical Reynolds number";
  SI.ReynoldsNumber Re(start = 0) "Reynolds number";
  SI.Velocity w(start = 0) "mean fluid velocity";
  Types.Dimensionless Cq(start = 0);
  Types.Area A0 "Area of the restrictor";
equation
  A0 = Constants.pi * d ^ 2 / 4;
  (q, w, Re, Cq) = Functions.orif3f(A.p, B.p, A0, d, Cd, Rec);
  annotation(
    Icon(graphics = {Line(points = {{-98, 0}, {98, 0}}, thickness = 0.5), Line(origin = {0, 25.21}, points = {{-60, 14.7929}, {-40, -
5.20711}, {-20, -15.2071}, {20, -15.2071}, {40, -5.20711}, {60, 14.7929}}, thickness = 0.5), Line(origin = {0, -25.21}, points = {{-
60, -14.7929}, {-40, 5.20711}, {-18, 15.2071}, {20, 15.2071}, {40, 5.20711}, {60, -14.7929}}, thickness = 0.5)},
coordinateSystem(initialScale = 0.1)));
end Orifice;
```

### 5.5.1 Esempio: Verifica di un orificio

Consideriamo uno strozzatore di diametro d = 5 mm, con Cd = 0.7, numero di Reynolds critico Rec = 1000 e un fluido avente densità ρ = 850 kg/m$^3$ e viscosità μ = 51 cP = 0.051 Pa·s.
Effettuiamo una simulazione per determinare la velocità media del fluido, il numero di Reynolds e la portata nei due casi: a) ΔP = 0.25 bar e b) ΔP= 100 bar.
Ecco il nostro schema effettuato con Modelica e la nostra libreria.

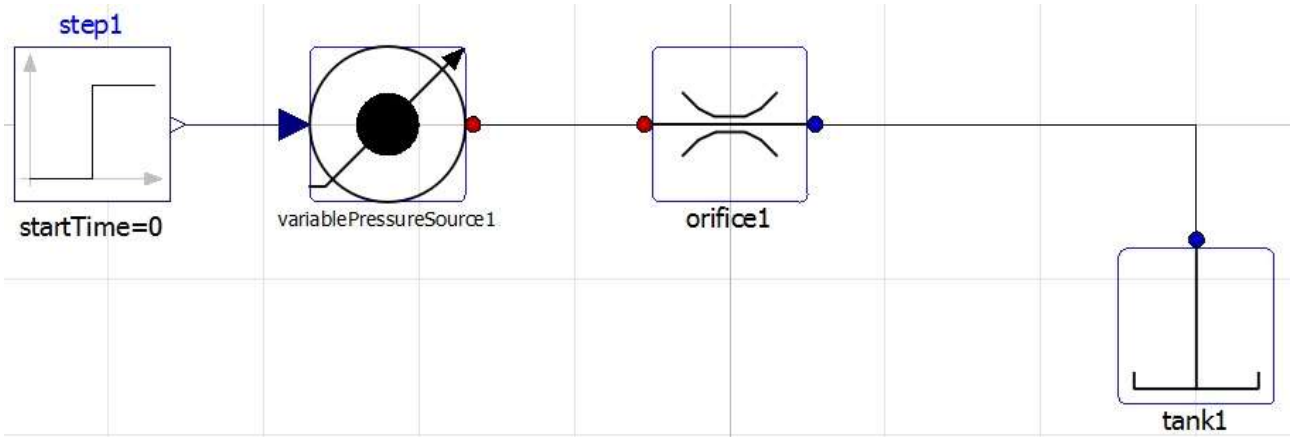
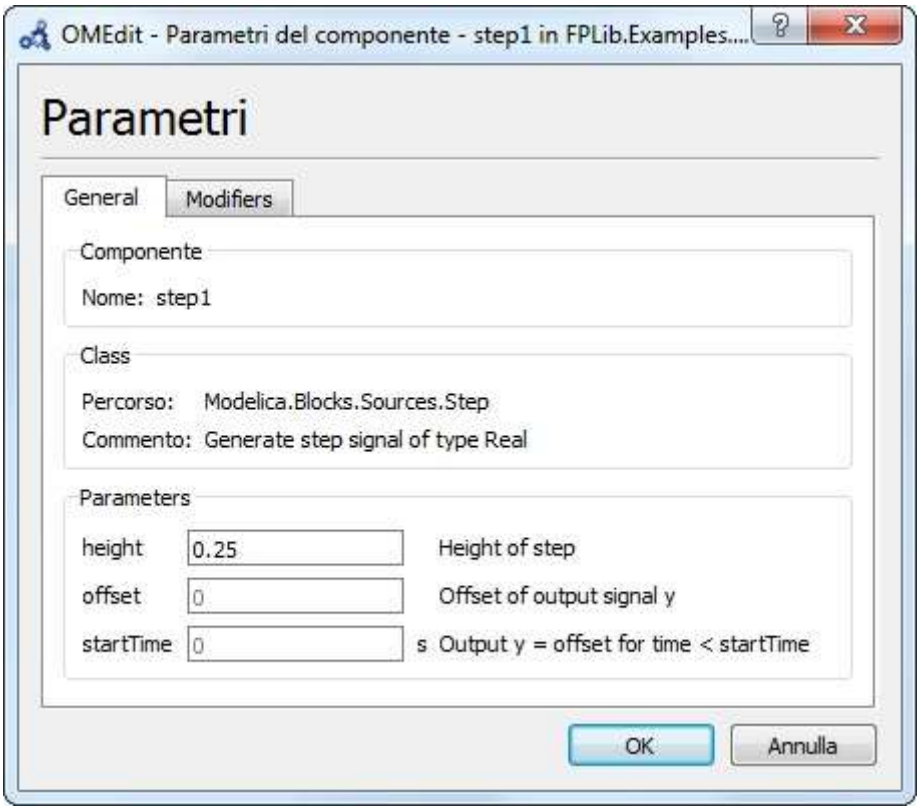



Figura 5-1: schema Modelica per la verifica di un orificio

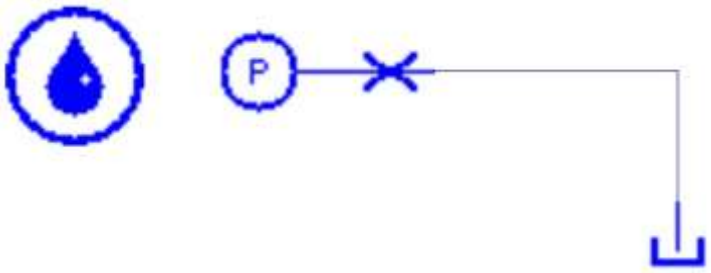
Abbiamo lasciato i valori di default per lo strozzatore, poiché coincidono con i dati del problema.
Le costanti ρ e μ sono anch'esse quelle indicate dal problema, quindi non le modifichiamo.
L'unico parametro da settare è il segnale (step1) che nel primo caso, settiamo a 0.25.




Se lanciamo la simulazione otteniamo, per lo strozzatore orifice1: dp = 0.25 bar, $C_q$ = 0.599, Re = 639.137, w (media) = 4.596 m/s e q = 5.414 l/min.
Se lanciamo la simulazione otteniamo, per lo strozzatore orifice1: dp = 100 bar (inserire 100 nel blocco step1), $C_q$ = 0.7, Re = 12783, w (media) = 107.375 m/s e q = 126.498 l/min.
Se applichiamo le formule [5-2, 5-3, 5-4, 5-5 e 5-6] nei due casi otteniamo:

| | |
|---|---|
| $w_{\max,1} = \sqrt{\frac{2}{\rho}\cdot\Delta P} = \sqrt{\frac{2}{850}\cdot 0.25\cdot 10^5} = 7.669$ m/s | $w_{\max,2} = \sqrt{\frac{2}{\rho}\cdot\Delta P} = \sqrt{\frac{2}{850}\cdot 100\cdot 10^5} = 153.393$ m/s |
| $\mathrm{Re}_1 = w_{\max,1}\cdot d_h\cdot\frac{\rho}{\mu} = 7.669\cdot 5\cdot 10^{-3}\,\frac{850}{0.051} = 639$ | $\mathrm{Re}_2 = w_{\max,2}\cdot d_h\cdot\frac{\rho}{\mu} = 153.393\cdot 5\cdot 10^{-3}\,\frac{850}{0.051} = 12783$ |
| $C_{q,1} = 0.7\cdot\tanh\left(\frac{2\cdot 639}{1000}\right) = 0.599$ | $C_{q,2} = 0.7\cdot\tanh\left(\frac{2\cdot 12783}{1000}\right) = 0.7$ |
| $w_{mean,1} = C_{q,1}\cdot w_{\max,1} =$ 4.59 m/s | $w_{mean,2} = C_{q,2}\cdot w_{\max,2} =$ 107.375 m/s |
| $Q_1 = C_{q,1}\cdot A_0\cdot\sqrt{\frac{2}{\rho}\cdot 0.25\cdot 10^5} =$<br>$= 0.599\cdot 1.963\cdot 10^{-5}\cdot\sqrt{\frac{2}{850}\cdot 0.25\cdot 10^5} =$<br>$= 9.018\cdot 10^{-5}\,\frac{m^3}{s} = 5.41\,\frac{l}{\min}$ | $Q_2 = C_{q,2}\cdot A_0\cdot\sqrt{\frac{2}{\rho}\cdot 100\cdot 10^5} =$<br>$= 0.7\cdot 1.963\cdot 10^{-5}\cdot\sqrt{\frac{2}{850}\cdot 100\cdot 10^5} =$<br>$= 2.11\cdot 10^{-3}\,\frac{m^3}{s} \approx 126.5\,\frac{l}{\min}$ |

Possiamo anche realizzare lo stesso modello con AMESim.



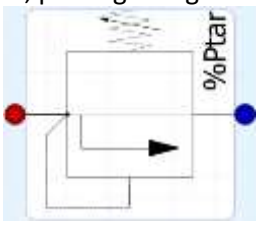

Impostando la pressione costante prima a 0.25 bar, poi a 100 bar otteniamo i seguenti risultati:

| a) ΔP = 0.25 bar | b) ΔP= 100 bar |
|---|---|
| flow rate at port 1 = 5.4188 L/min | flow rate at port 1 = 126.825 L/min |
| pressure at port 1 = 0 bar | pressure at port 1 = 0 bar |
| pressure at port 2 = 0.25 bar | pressure at port 2 = 100 bar |
| flow coefficient ($C_q$) = 0.599344 - | flow coefficient ($C_q$) = 0.7 - |
| flow number(λ) = 639.477 - | flow number(λ) = 12802 - |
| mean fluid velocity = 4.59917 m/s | mean fluid velocity = 107.221 m/s |

Come si vede i risultati sono abbastanza allineati. Ci sono piccole differenze non significative, dovute al fatto che AMESim tratta la densità e la viscosità in funzione della pressione, il che è corretto ma personalmente ho scelto di semplificare i calcoli. In ogni caso le differenze tra i valori calcolati da AMESim e quelli calcolati dalla libreria sono veramente minime, quindi è una buona approssimazione.

## 5.6 Valvola limitatrice di pressione (lineare)

Una valvola limitatrice di pressione ha due porte non intercambiabili (come ad esempio lo strozzatore). Una porta ($p_1$) è connessa ad un ramo nel quale si vuole limitare la pressione, mentre l'altra ($p_2$) è connessa al serbatoio. Di conseguenza abbiamo $\Delta P = p_1 - p_2$.
Idealmente, la connessione tra la porta 1 e 2 è interrotta se $\Delta P \leq P_{tar}$, dove quest'ultima è la pressione di taratura, ovvero quel valore limite di pressione che non si vorrebbe superare, altrimenti, la connessione tra 1 e 2 è stabilita, di modo che l'olio eccedente, possa guadagnare lo scarico.




In pratica, se ΔP > Ptar, allora, la portata che passa tra la porta 1 e la porta 2 è:

$$Q_r = \frac{p_1 - p_2 - P_{tar}}{G}$$

5-7: modello di una valvola limitatrice di pressione lineare

Dove G è un parametro della valvola che rappresenta l'errore di regolazione: la pressione è quella aggiuntiva al valore di taratura, raggiunta alla portata di riferimento; ad esempio, se una valvola tarata a 350 bar, a 100 L/min regola una pressione di 390 bar, allora G = (390-350) bar / (100 L/min) = 0.4. Di default si assume G = 1·10-3 bar/(L/min) che significa assumere una valvola dal comportamento praticamente ideale. Ecco il codice:

```
model LinearReliefValve
  extends Interfaces.Partial2FluidPortsWithVolume;
  parameter Types.Pressure Ptar = 350 "cracking pressure";
  parameter Real G(unit = "bar/(L/min)", min = 1e-3) = 1e-3 "pressure/flow coefficient";
  parameter Types.Pressure pHysteresis = 0 "Hysteresis pressure";
  Types.Pressure cp;
  Boolean cracking;
algorithm
  cp := dp - Ptar + pHysteresis;
  when dp <= Ptar - pHysteresis then
    cracking := false;
```

```
  end when;
  when dp > Ptar then
    cracking := true;
  end when;
  if cracking == true then
    q := cp / G;
  else
    q := 0;
  end if;
  annotation(
    Icon(graphics = {Rectangle(origin = {10, -12}, lineThickness = 0.5, extent = {{-70, 72}, {50, -48}}), Line(origin = {-80, 0},
points = {{-20, 0}, {20, 0}}, thickness = 0.5), Line(origin = {80.687, 0.381679}, points = {{-20, 0}, {20, 0}}, thickness = 0.5),
Line(origin = {1.19, 63.73}, points = {{3.08625, 35.9473}, {-20.9138, 31.9473}, {19.0862, 25.9473}, {-20.9138, 17.9473}, {19.0862,
11.9473}, {-20.9138, 5.9473}, {19.0862, 1.9473}, {-4.91375, -4.0527}}), Line(origin = {196.097, -50.7871}, points = {{-167, 131}, {-
219, 141}}, thickness = 0.5), Polygon(origin = {-53, 76}, fillPattern = FillPattern.Solid, points = {{23, 16}, {33, 16}, {31, 12},
{23, 16}}), Polygon(origin = {37, 32}, fillPattern = FillPattern.Solid, points = {{13, 0}, {-17, 8}, {-17, -8}, {13, 0}}),
Ellipse(origin = {-61, 1}, fillPattern = FillPattern.Solid, extent = {{-1, 1}, {3, -3}}, endAngle = 360), Line(origin = {-40.359, -
139.739}, points = {{-20, 139.171}, {-40, 119.171}, {-40, 51.1708}, {40, 51.1708}, {40, 79.1708}}, pattern = LinePattern.Dash,
thickness = 0.5), Line(origin = {-5, 16}, points = {{-45, -16}, {-45, 16}, {45, 16}}, thickness = 0.5), Text(origin = {88, 64},
rotation = 90, extent = {{-44, 28}, {36, -12}}, textString = "%Ptar")}, coordinateSystem(initialScale = 0.1)));
end LinearReliefValve;
```

Al modello è applicato uno schema di isteresi.

Per evitare situazioni in cui la simulazione possa oscillare attorno al valore di taratura ed emettere così dei warning o possa addirittura rallentare fino a dare errore, aggiungiamo a $\Delta P$-$P_{tar}$, il valore dell'isteresi (parametro, valore di default = 0, che significa no isteresi).

1. Se $\Delta P \leq P_{tar} - P_{isteresi}$, allora la valvola è senza dubbio chiusa (cracking = false).
2. Se $\Delta P > P_{tar}$ allora la valvola è aperta (cracking = true).

Quindi rimane un $\Delta P$ compreso tra $P_{tar} - P_{isteresi}$ e $P_{tar}$ in cui la valvola mantiene il suo stato di apertura o chiusura.

Facciamo un esempio numerico: Ptar = 350 bar, $p_{Hysteresis}$ = 5 bar.

Se $\Delta P$ è minore di 345 bar, sicuramente la valvola è chiusa.

Se $\Delta P$ è maggiore di 350 bar, sicuramente la valvola è aperta.

Poiché non viene valutata la situazione di $\Delta P$ compresa tra $P_{tar} - P_{isteresi}$ e $P_{tar}$ significa che in questa situazione, lo stato della valvola (boolean cracking) rimane inalterato e quindi, se a valvola chiusa la pressione sale da 300 bar fino a 350, la valvola rimane chiusa, se la pressione sale a 351 bar, si apre, poi se scende fino a 346 bar, rimane ancora aperta e si richiude soltanto se il $\Delta P$ è minore o uguale a 345 bar.

Quindi otteniamo una valvola che si apre alla pressione di taratura, ma che si richiude un po' sotto, del valore $p_{Hysteresis}$.

### 5.6.1 Esempio: schema idraulico con valvola limitatrice di pressione

Riprendiamo lo schema in Figura 5-1 e addattiamolo mettendo una rampa al posto del segnale step, e inserendo una valvola limitatrice di pressione (lineare). Lo schema dovrà risultare come il seguente:

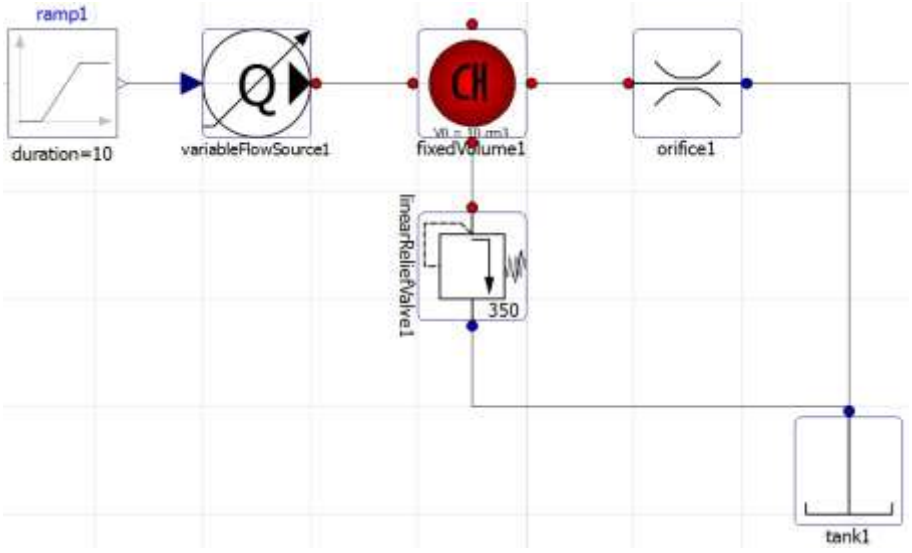
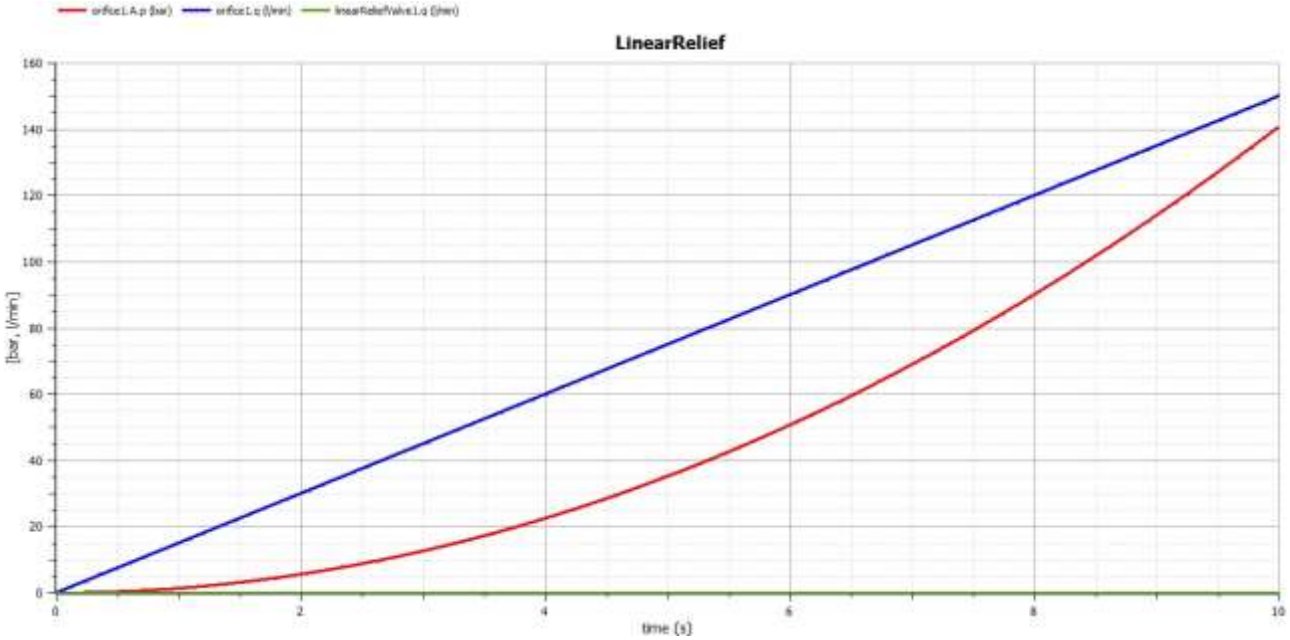


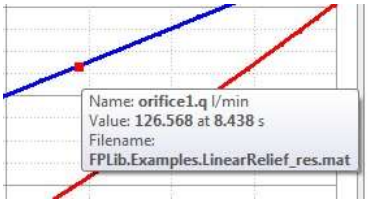

Figura 5-2: schema Modelica per testare la valvola limitatrice di pressione (lineare)

Impostiamo la rampa con un'altezza di 150 e una durata di 10 s, questo significa che avremo una portata variabile linearmente da 0 a 150 L/min, in 10 s.

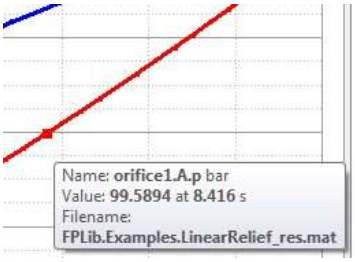
Con le formule [5-2, 5-3, 5-4, 5-5 e 5-6], che abbiamo già usato, è facile verificare che una portata Q = 150 L/min corrisponde ad un ΔP sullo strozzatore (stessi dati del problema 5.5.1) di circa 140 bar.
Dunque, con una valvola limitatrice di pressione tarata a 350 bar io mi devo aspettare che quest'ultima non entri mai in regolazione (portata attraverso essa, nulla), e che nell'istante finale, la pressione a monte dello strozzatore sia di circa 140 bar, che a valle sia zero, e che la portata attraverso lo strozzatore sia 150 L/min.
Se lanciamo la simulazione infatti otteniamo:



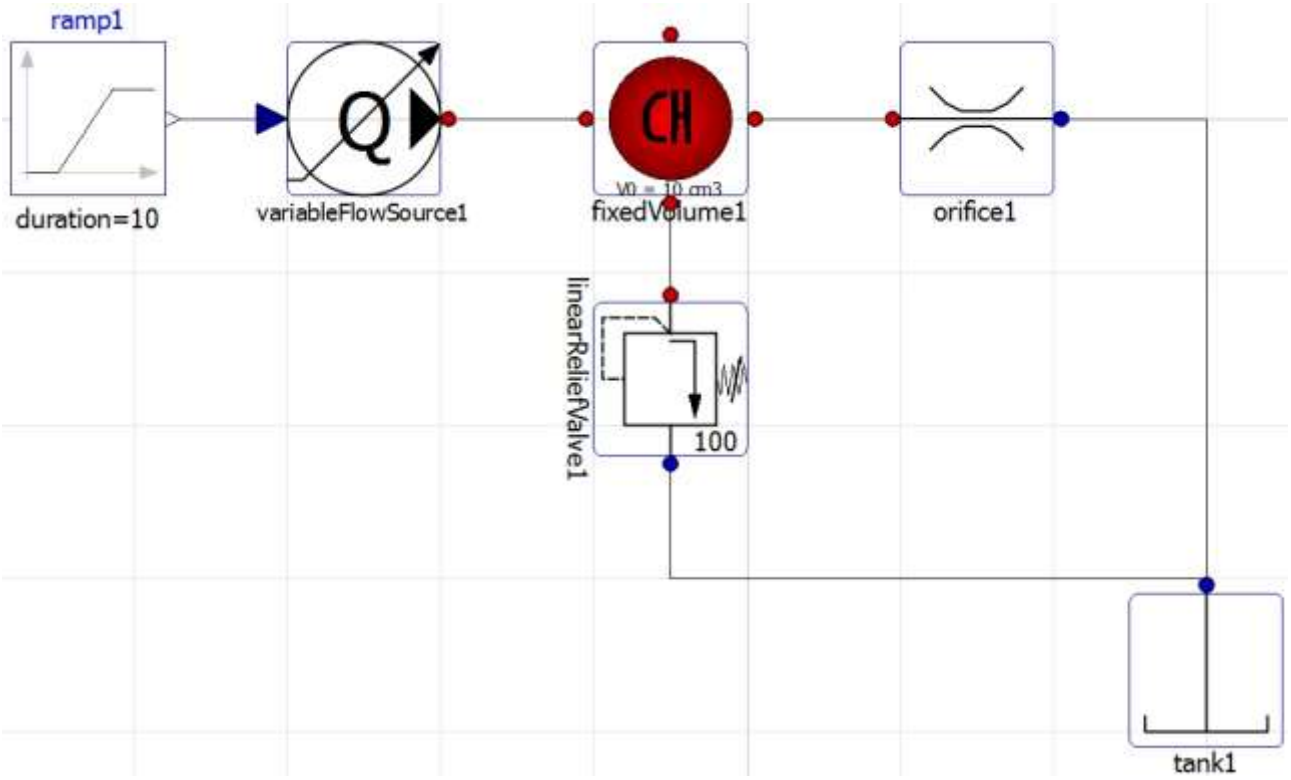

Infatti accade proprio quello che abbiamo previsto. La pressione arriva a 140.6 bar, la portata è naturalmente 150 L/min. La portata elaborata dalla valvola limitatrice di pressione è nulla (linea verde).
Inoltre, ricordiamo dall'esempio 5.5.1 che a 126.5 L/min il ΔP sullo strozzatore era di 100 bar. Se indugiamo con il puntatore del mouse sulla linea blu, che è la portata, intorno ai 126.5 vediamo che si apre una piccola finesta pop-up.



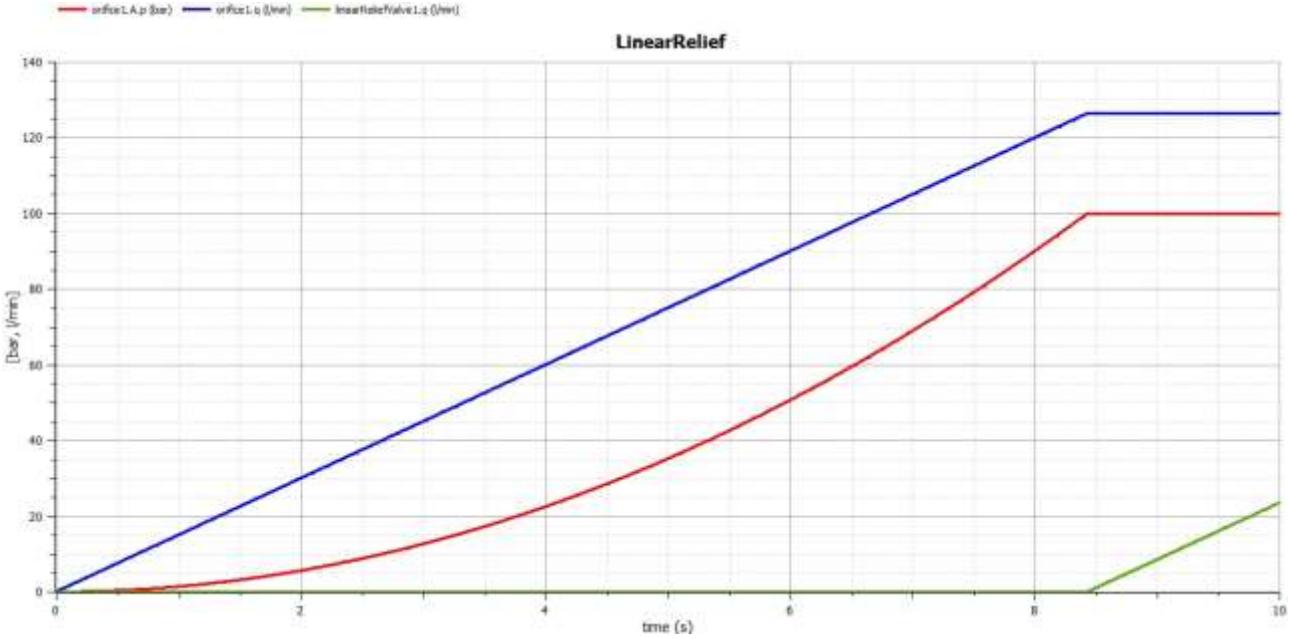

La finestra ci dice che il valore è circa 126.5 a circa 8.4 s.
Se ora vado nella linea rossa intorno agli 8.4 secondi dovrei verificare che la pressione a monte dello strozzatore è circa 100 bar.



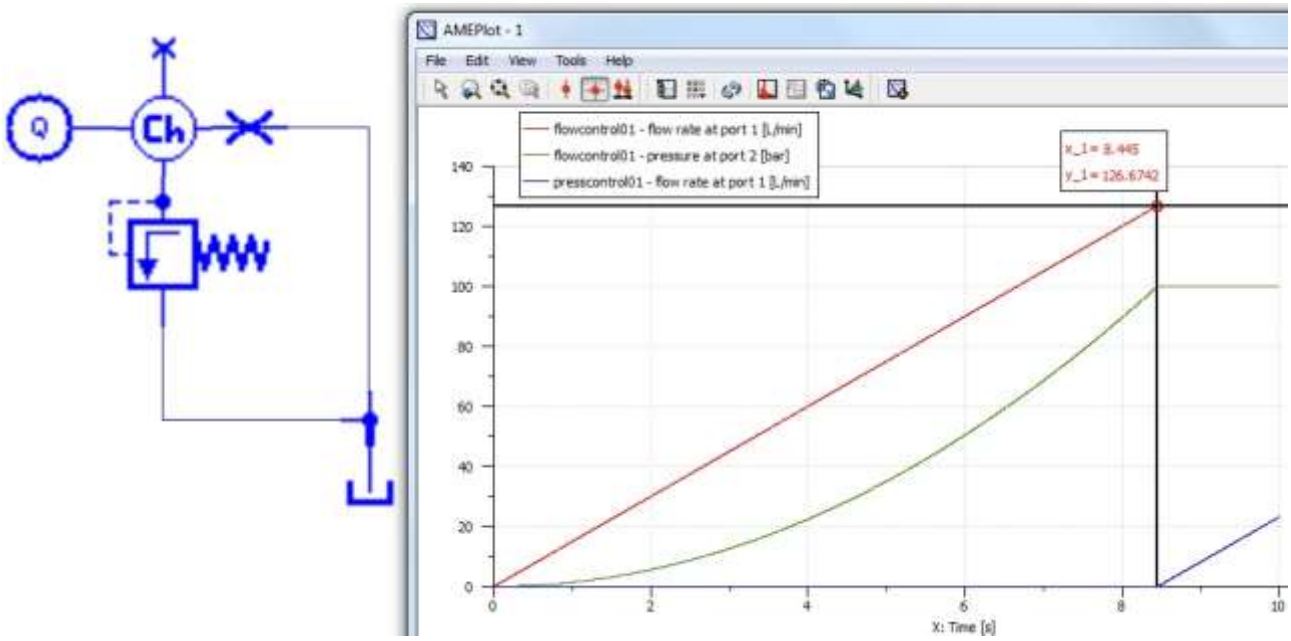

Immaginiamo ora di voler limitare la pressione a 100 bar. Andiamo nello schema di Figura 5-2 e modifichiamo la pressione di taratura a 100 bar.




Ora dovremmo vedere che la nuova taratura è stata impostata (100 bar).
A questo punto, lanciando la simulazione, mi devo attendere un grafico simile al precedente, fino a circa 8.4 secondi, quindi, dovrei vedere la pressione limitata a 100 bar sullo strozzatore, la portata dello strozzatore limitata a circa 126.5 L/min e circa 23.5 bar che escono dalla limitatrice.



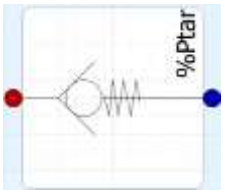

Infatti il risultato della simulazione rispecchia le aspettative.
*Fino a questo momento avevamo fatto sempre delle verifiche con formule analitiche. Da ora in poi la complessità dei problemi non ci consentirà più di fare così: faremo tuttavia un confronto diretto con schemi equivalenti di AMESim*.
Di seguito ecco lo schema e il risultato della simulazione con AMESim.




Il risultato è praticamente identico.

## 5.7 Valvola di non ritorno (lineare)

La valvola di non ritorno, lineare, è praticamente identica (come codice) alla valvola limitatrice descritta nel paragrafo 5.6.




### 5.7.1 Esempio: schemi con valvole di non ritorno

Ecco un esempio con la valvola montata in modo da consentire il flusso e nel modo in cui il flusso è interdetto.

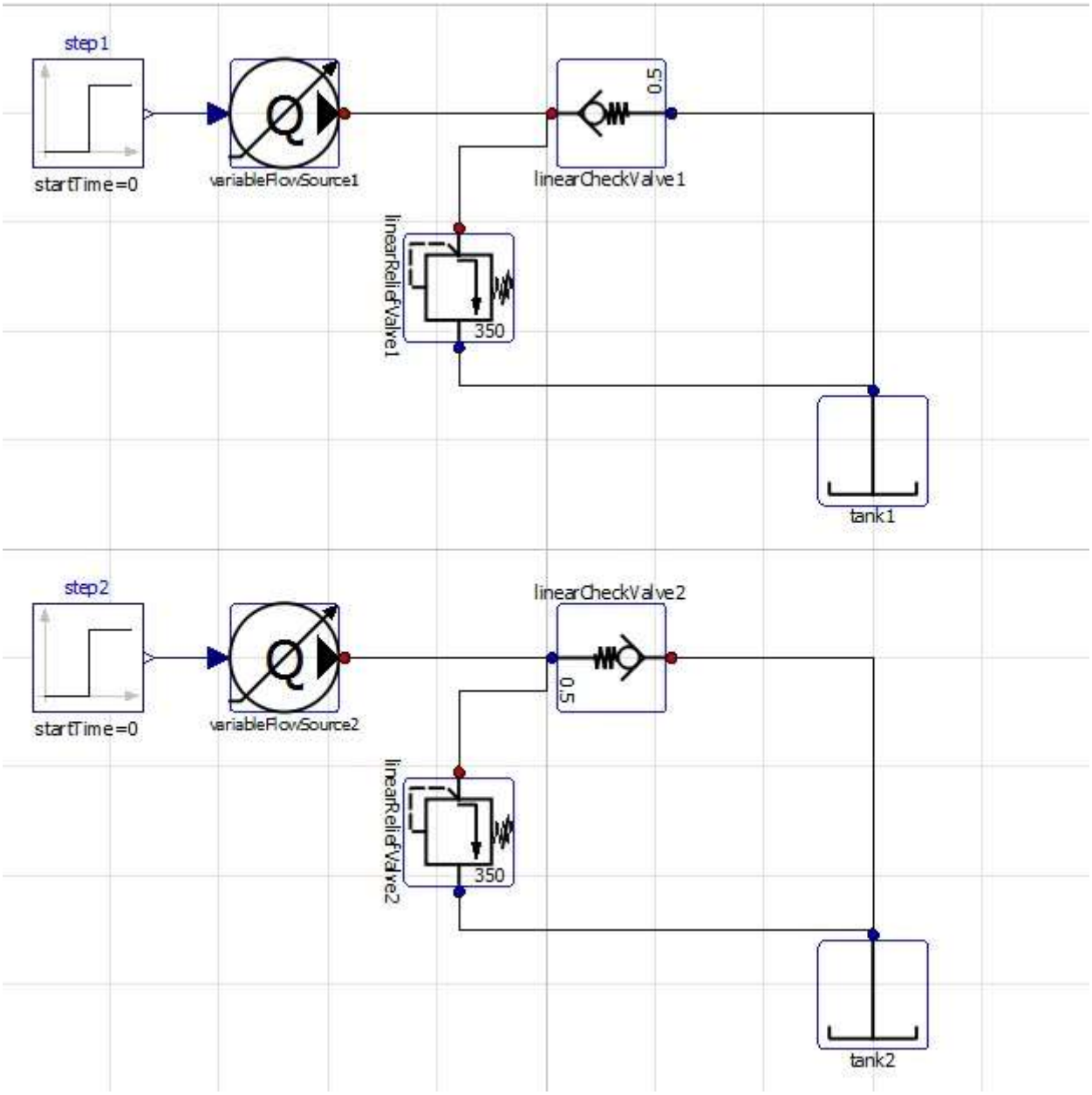
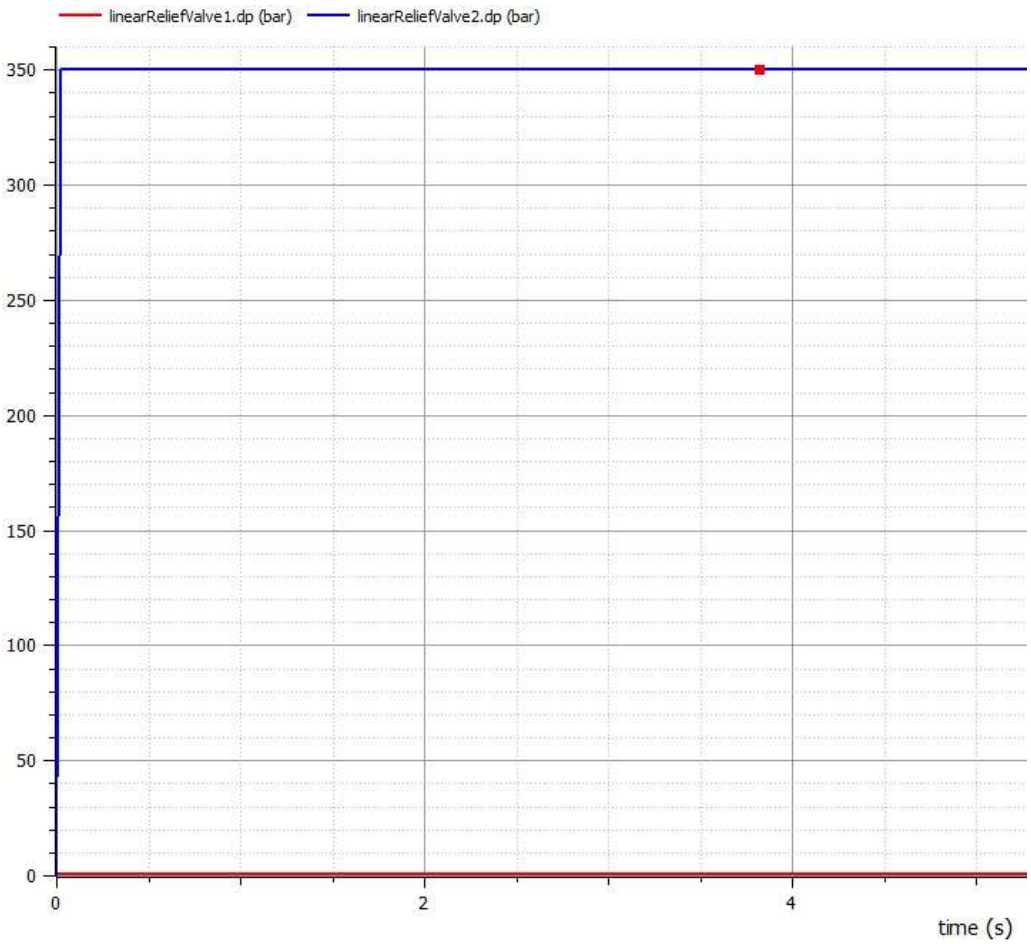



**Figura 5-3: schemi con valvole unidirezionali**

Naturalmente in questo caso non è necessario effettuare calcoli ma una discussione qualitativa andrà più che bene. La valvola unidirezionale linearCheckValve1 (caso A, in alto) sarà attraversata da una portata, la pressione a monte di essa sarà circa 0.5 bar (avendo lasciato il blocco step al valore di default, 1, la portata sarà così piccola da presentare un errore di regolazione minimo). L'altra valvola unidirezionale (caso b) invece è montata interdetta, e quindi è chiusa, la portata che attraversa la check valve è nulla mentre la pressione a valle viene limitata dalla limitatrice di pressione, per tanto, nei due casi troveremo:

A. Portata attraverso la check valve q = 1 l/min, pressione a monte della check valve p ≈ 0.5 bar.
B. Portata attraverso la check valve q = 0 l/min, pressione a monte della check valve p ≈ 350 bar.

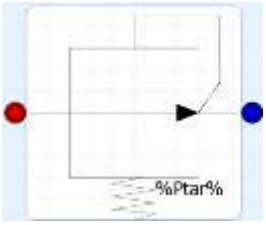
Se lanciamo la simulazione otteniamo:




In pratica la linea rossa è a 0.5 mm (per questo appare molto vicina allo zero), mentre la linea blu è in effetti a 350 bar.

## 5.8 Valvola riduttrice di pressione (lineare)

E' una valvola che regola la pressione a valle a un valore impostato.




*Questa valvola non ha la **funzione relief**, cioè limita la pressione chiudendo la via da A (attacco rosso) verso B, ma non consente di alimentare in B e scaricare in A. Cioè B non può salire oltre il valore $P_{max}$, se alimento da A, ma se la pressione in B aumenta per cause esterne (esempio una pompa attaccata a B), allora la pressione cresce illimitatamente*.

### 5.8.1 Descrizione del modello lineare

Sia la pressione all'attacco A, $p_A$ e la pressione all'attacco B, $p_B$.
La pressione netta che apre la valvola è $c_p = P_{tar} - p_B$.
L'apertura della valvola (valore frazionario, rispetto alla corsa massima) è data da

$$x_v = 1 - \frac{P_{tar} - p_B}{P_{tar} - P_{\max}}$$

5-8: caratteristica lineare (grado di apertura) di una valvola riduttrice di pressione

Dalla [5-8] ne deriva che:

1. Se $p_B$ è inferiore alla taratura della valvola $P_{tar}$, allora $x_v > 1$.
2. Se $p_B = P_{tar}$, allora $x_v = 1$.
3. Se $P_{tar} \le p_B \le P_{max}$ allora $x_v$ è compreso tra 1 e 0.
4. Se $p_B \ge P_{max}$ allora $x_v$ è minore di zero.

Questo valore è usato come segue:
Il grado d'apertura $x_v$ è considerato sempre tra 0 e 1, in pratica se la pressione in B è minore della pressione di taratura, il valore di $x_v$ è limitato al massimo a 1, se la pressione in B è maggiore della pressione massima, il valore di $x_v$ è limitato al minimo a 0. Il risultato di questa operazione di taglio lo chiamiamo ***d***.
La sezione d'apertura è il prodotto di d e dell'area massima $A_{max}$:

$$A(d) = d \cdot A_{\max}$$

Dove $A_{max}$ è la sezione massima calcolata sapendo che alla portata nominale $Q_n$, la valvola presenta un ΔP dato dal parametro $DeltaP_n$.
Una stima di questa area massima è calcolata con la [5-2], assumendo $A_0$ come area massima, $\Delta P = DeltaP_n$ e $Q = Q_n$.
Per una stima ancora più corretta, dovremmo inserire il coefficiente $C_q$ calcolato con la [5-5], ma nelle valvole, tra gli elementi mobili il moto è praticamente sempre turbolento e quindi otteniamo un'approssimazione valida per l'area massima.
Ecco il codice:

```
model PressureReducingValve
  extends Interfaces.Partial2FluidPortsWithVolume;
  parameter Types.Pressure Ptar = 10 "cracking pressure";
  parameter Types.Pressure Pmax = 11 "maximum pressure";
  parameter Types.VolumeFlowRate Qn = 10 "nominal flowrate";
  parameter Types.Pressure DeltaPn = 1 "pressure drop at Qn";
  parameter Types.Dimensionless Cd = 0.7 "discharge coefficient";
  parameter Types.Area Amax = Qn * sqrt(Constants.rho / (5 * DeltaPn)) / (12 * Cd);
protected
  Types.Pressure cp;
  Real d;
equation
  cp = Ptar - pB;
  if cp <= 0 then
    d = max(0, 1 - cp / (Ptar - Pmax));
  else
    d = min(1 - cp / (Ptar - Pmax), 1);
```

```
  end if;
  q = Cd * d * Amax * sqrt(720 * abs(pA - pB) / Constants.rho) * sign(pA - pB);
 annotation(
   Icon(graphics = {Rectangle(lineThickness = 0.5, extent = {{-60, 60}, {60, -60}}), Line(origin = {-0.0238878, -77.9589}, points =
{{10.0239, 17.9589}, {-19.9761, 11.9589}, {20.0239, 5.95892}, {-19.9761, -0.041078}, {20.0239, -6.04108}, {-19.9761, -12.0411},
{20.0239, -16.0411}, {0.0238878, -22.0411}, {0.0238878, -22.0411}}), Line(points = {{-96, 0}, {96, 0}, {96, 0}, {96, 0}}, thickness =
0.5), Polygon(origin = {49.69, 0}, fillPattern = FillPattern.Solid, points = {{10.3143, 0}, {-9.6857, 8}, {-9.6857, -8}, {10.3143,
0}}), Line(origin = {40, 44.8}, points = {{20, -44.8025}, {40, -16.8025}, {40, 45.1975}, {-40, 45.1975}, {-40, 15.1975}}, pattern =
LinePattern.Dash, thickness = 0.5), Text(origin = {52, -70}, extent = {{-34, 10}, {34, -10}}, textString = "%Ptar%")},
coordinateSystem(initialScale = 0.1)));
end PressureReducingValve;
```

### 5.8.2 Esempio: schema con valvola riduttrice di pressione (lineare)

Abbiamo una sorgente di pressione a 140 bar, ma vogliamo alimentare il nostro strozzatore ai soliti 100 bar (circa).

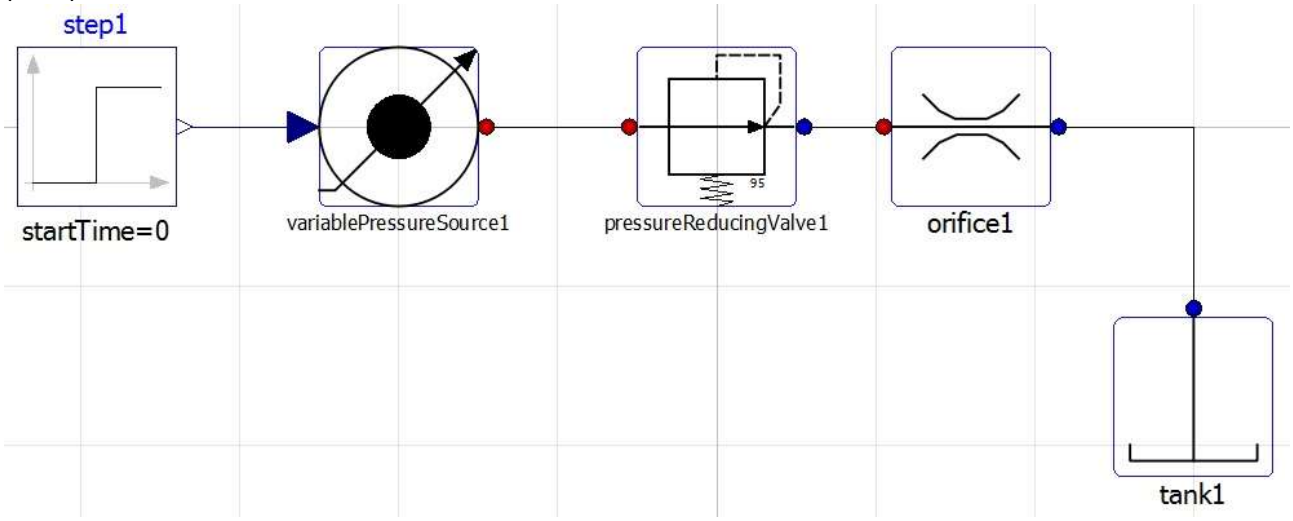



Figura 5-4: schema Modelica per l'esempio su valvola riduttrice di pressione

Abbiamo compilato i parametri della valvola riduttrice come segue:



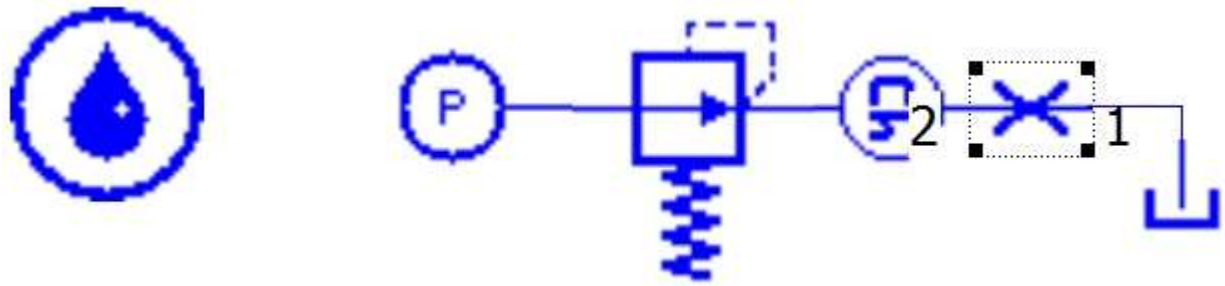

Se lanciamo la simulazione otteniamo: pressione sullo strozzatore = 98.5 bar e portata nello strozzatore = 125.6 l/min.

Com'è ragionevole attendersi, la pressione non è esattamente 100 bar, infatti se la pressione in cui inizia a chiudere è 95 bar e quella in cui è chiusa è 100 bar, è ovvio che in fase di regolazione qualunque punto tra 95 e 100 bar può essere raggiunto, inoltre, anche la grandezza nominale della valvola (definita

implicitamente dalla coppia di valori $Q_n$ e $DeltaP_n$) influenza la pressione regolata poiché più $DeltaP_n$ è grande, minore risulta $A_{max}$ (a parità di $Q_n$) e quindi, minore è la sezione, maggiore sarà il ΔP tra A e B, quindi se A è assegnato può essere soltanto che B è minore.

#### 5.8.2.1 *Verifica con AMESim*

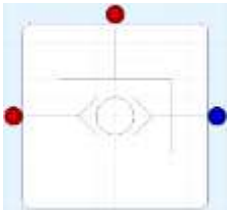
Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




Le impostazioni sono esattamente le stesse: P = 140 bar, riduttrice con cracking pressure = 95 bar, maximum pressure = 100 bar, nominal flow rate = 150 L/min, nominal pressure drop = 5 bar, e il solito strozzatore con i soliti parametri (orifice diameter = 5 mm, maximum flow coefficiente = 0.7, critical flow number from laminar to turbolent = 1000).
Lanciando la simulazione risulta:
flow rate at port 1 = 125.9 L/min;
pressure at port 2 = 98.55 bar.
Diciamo che siamo lì (125.6 contro 125.9 per la portata, mentre la pressione è circa la stessa). Si tratta di una differenza veramente contenuta.

## 5.9 Valvola selettrice di pressione

Se rispetto allo schema, la porta a sinistra è A, la porta a destra è B e la porta superiore è C, la valvola selettrice di pressione individua tra le porte A e B, quella posta alla pressione maggiore e la mette in connessione con la porta d'uscita C.



Applicando un modello d'isteresi si ha che la commutazione avviene alternativamente se:

- la pressione in A e minore di quella in B meno l'isteresi;
- la pressione in A è maggiore di quella in B più l'isteresi.

In questo modo c'è una fascia compresa tra $\pm p_{Hysteresis}$ a cavallo tra $p_A$ e $p_B$ per la quale non avviene commutazione.
Se c'è commutazione allora la portata è:

- $q = q_{AC} = \frac{p_A - p_C}{G}$, se $p_A > p_B$.
- $q = q_{BC} = \frac{p_B - p_C}{G}$, altrimenti.

Ecco il codice:

```
model ShuttleValve
  extends FPLib.Hydraulic.Interfaces.Partial3FluidPortsWithVolume;
  parameter Types.Pressure pHysteresis = 0.001 "Hysteresis pressure";
  parameter Real G(unit = "bar/(L/min)", min = 1e-3) = 1e-3 "pressure/flow coefficient";
  Types.Pressure cp;
  Types.VolumeFlowRate q(start = 0);
initial equation
```

```
  cp = pA - pB;
equation
  when {pA < pB - pHysteresis, pA > pB + pHysteresis} then
    cp = pA - pB;
  end when;
//cp = pA - pB;
  if noEvent(cp >= 0) then
    q = (pA - pC) / G;
    qAC = q;
    qAB = 0;
    qBC = 0;
  else
    q = (pB - pC) / G;
    qAC = 0;
    qAB = 0;
    qBC = q;
  end if;
  annotation(
    Icon(graphics = {Ellipse(extent = {{-20, 20}, {20, -20}}, endAngle = 360), Line(origin = {-29.6464, 0}, points = {{9.64645, 20},
{-10.3536, 0}, {9.64645, -20}}), Line(origin = {29.6464, 0}, points = {{-9.64645, 20}, {10.3536, 0}, {-9.64645, -20}}),
Rectangle(extent = {{-60, 40}, {60, -40}}), Line(origin = {-68, 0}, points = {{-28, 0}, {28, 0}}), Line(origin = {68, 0}, points =
{{28, 0}, {-28, 0}}), Line(origin = {0, 67}, points = {{0, -27}, {0, 27}})}, coordinateSystem(initialScale = 0.1)));
end ShuttleValve;
```

### 5.9.1 Esempio: schema con valvola selettrice di pressione

Abbiamo due sorgenti di pressione variabili nel tempo, la prima è una rampa che parte da 0 e arriva a 140 bar in 10 secondi, la seconda parte da 140 bar e arriva a 0 nello stesso tempo.
Tramite una valvola selettrice di pressione, selezioniamo la sorgente di pressione maggiore e la connettiamo a un'orificio.
Di seguito lo schema Modelica con la libreria proposta:

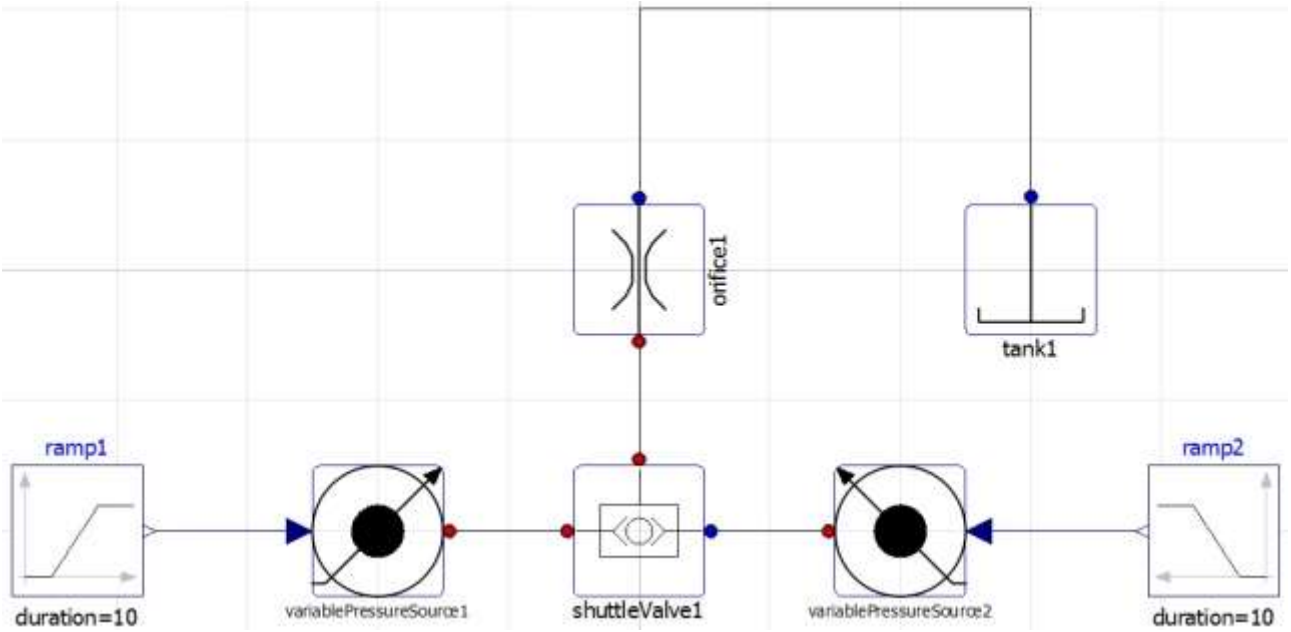
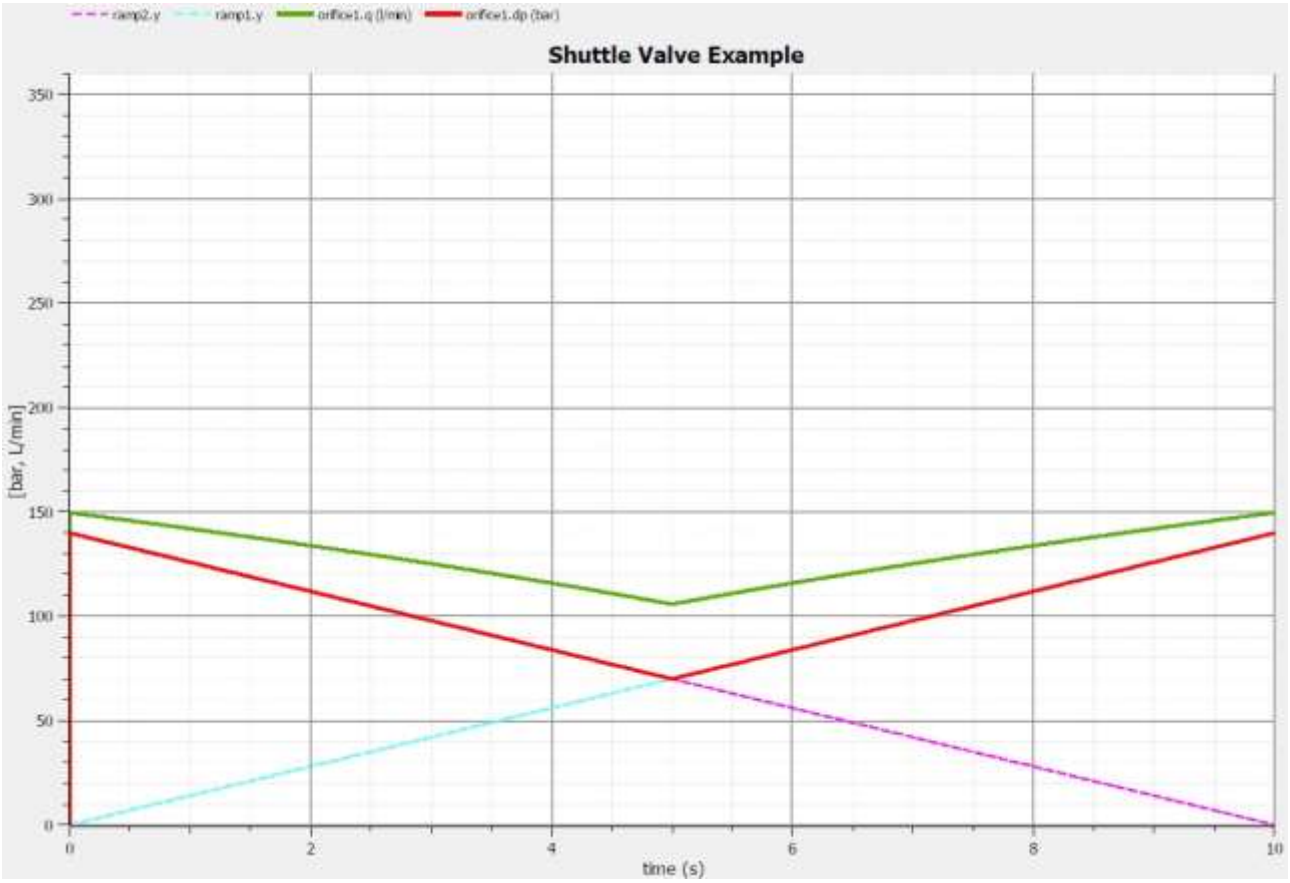



Figura 5-5: schema Modelica per l'esempio con valvola selettrice di pressione

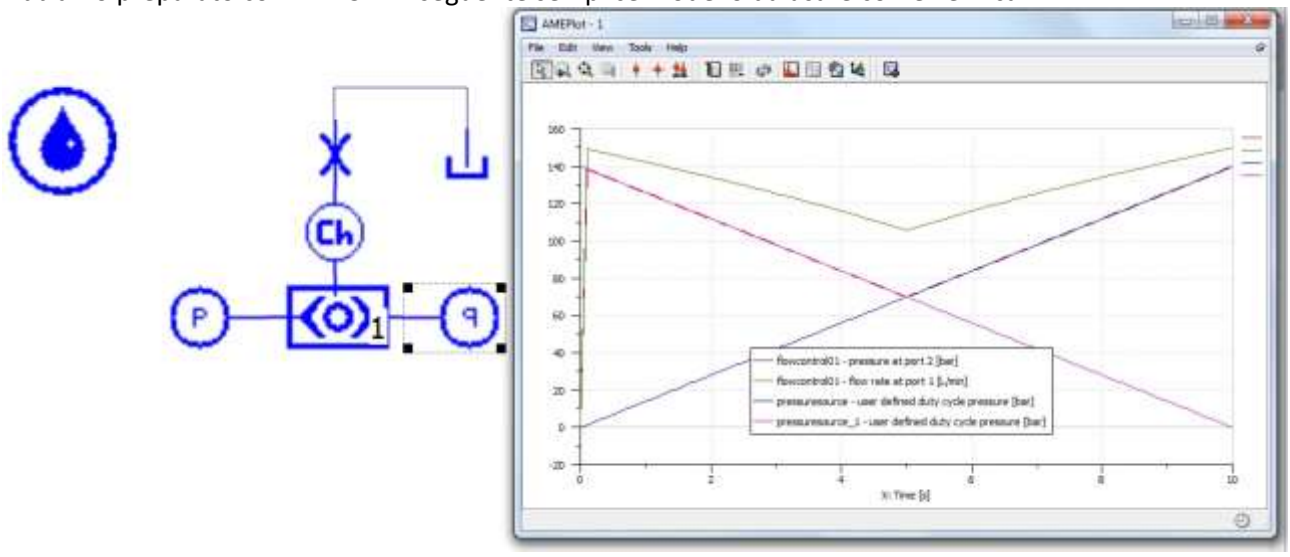
Nota: per la ramp2 bisogna settare l'altezza = -140 e un offset di 140.
Lanciando la simulazione, per 10 s, otteniamo:




Come si vede, la linea rossa (che in pratica è l'output della selettrice) seleziona il valore massimo tra ramp1.y e ramp2.y. Conseguentemente, la portata è rappresentata dalla linea verde (orifice1.q). Notiamo sempre che a circa 100 bar, corrisponde una portata di circa 126.5 L/min.

#### *5.9.1.1 Verifica con AMESim*

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




In pratica il risultato è identico.

## 5.10 Valvola idraulica generica a 2 vie (normalmente chiusa)

Il ruolo di questa valvola è quello d'essere utilizzata in combinazione con altri componenti per formare diverse valvole di controllo della pressione e della portata.
E' una valvola generica, con due porte, e tre segnali di pilotaggio.

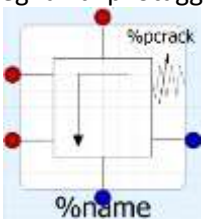



**Figura 5-6: schema Modelica di una valvola idraulica generica a 2 vie normalmente chiusa**

Se un attacco di pilotaggio non esiste/non serve, nello schema che si vuole realizzare, va messo a scarico.

### 5.10.1 Descrizione del modello

L'area massima è calcolata mediante i parametri $q_{nom}$ e $p_{nom}$ che sono la portata nominale (alla massima apertura) e il relativo ΔP d'attraversamento. Vengono applicate le formule [5-2, 5-3, 5-4, 5-5 e 5-6].
Una prima stima di questa area massima è calcolata con la [5-2], assumendo $A_0$ come area massima, ΔP = pnom e Q = $q_{nom}$.
Poi è inserita una equazione che lega il coefficiente $C_q$ calcolato con la [5-5] alla portata nominale e al ΔP di passaggio, combinata alle equazioni [5-2, 5-3, 5-4, 5-5 e 5-6]
La pressione netta in apertura è data da:

$$dp_{net} = a_E \cdot p_E + a_D \cdot p_D - a_C \cdot p_C - p_{crack}$$

Dove $a_E$, $a_D$ e $a_C$ sono fattori che definiscono i rapporti di pilotaggio.
Il primo ($a_E$) è quello in basso a sinistra, il secondo ($a_D$) è quello in alto a sinistra, il terzo ($a_C$) è quello dal lato molla.
*Se ad esempio voglio realizzare una valvola limitatrice di pressione: $a_D$ = 1, perché $p_{crack}$ è direttamente riferita alla pressione della porta D, gli altri, se sono a scarico, possiamo lasciare 1 che è di default.*
*Se voglio realizzare una overcenter, normalmente questa è tarata sulla relief, e quindi proseguo sempre con $a_D$ = 1, perché $p_{crack}$ è direttamente riferita alla pressione della porta D ma devo impostare $a_E$ con il rapporto di pilotaggio. Ad esempio per una overcenter con rapporto di pilotaggio 8:1, $a_E$ sarà uguale a 8.*
L'apertura della valvola (valore frazionario, rispetto alla corsa massima) è data da:

$$x_v = \frac{dp_{net}}{p_{\max}}$$

Dove $p_{max}$ è la pressione che porta l'elemento mobile dalla posizione di riposo (completamente chiusa) a quella completamente aperta.

- Se $x_v$ è minore di 0 → la valvola è chiusa.
- Se $x_v$ è minore di 1 → la valvola è parzialmente aperta, la sezione aperta $A_e$ è $x_v \cdot A_array(x_v) \cdot A_{max}$.
- Se $x_v$ è maggiore o uguale a 1 → la valvola è totalmente aperta, la sezione aperta $A_e$ è $A_{max}$.

### 5.10.2 Personalizzazione delle leggi d'area

I vettori xv_array[:] e A_array[:] valgono di default {0, 1}. Questo significa che la sezione aperta $A_e$ sarà lineare rispetto a $x_v$. E' possibile impostare una legge d'apertura diversa se necessario, ad esempio con:

xv_array[:] = {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1} e
A_array[:] = {0, 0.01, 0.04, 0.09, 0.16, 0.25, 0.36, 0.49, 0.64, 0.81, 1}

Possiamo impostare una legge quadratica. Quanto più sono numerosi i vettori quanto più la legge è precisa.

### 5.10.3 Codice

Ecco il codice per il modello della valvola:

```
model Generic2PortValve
  extends Interfaces.PartialHydraulicBase;
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
    Placement(visible = true, transformation(origin = {0, 108}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {0, 110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b B annotation(
    Placement(visible = true, transformation(origin = {0, -108}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {0, -110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b C annotation(
    Placement(visible = true, transformation(origin = {112, -30}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {110, -40}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_a D annotation(
    Placement(visible = true, transformation(origin = {-150, -10}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-110, 40}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_a E annotation(
    Placement(visible = true, transformation(origin = {-184, -68}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-110, -40}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  // Volume A
  parameter Types.Diameter dA = 100 "Inlet diameter at fluid port A" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdAmax = 0.7 "Max. Cd at port A" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecA = 1000 "Critical Reynolds number at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolA0 = 10 "Initial volume at port A" annotation(
    Dialog(tab = "Advanced"));
  Types.Pressure pA(start = 0) "Pressure on volume A";
  Types.VolumeFlowRate qA(start = 0) "Flow at inlet port A";
  SI.ReynoldsNumber ReA "Reynold number at port A";
  SI.Velocity wA "fluid mean velocity at port A";
  Types.Dimensionless CdA "Discharge coefficient at port A";
  // Volume B
  parameter Types.Diameter dB = 100 "Inlet diameter at fluid port B" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdBmax = 0.7 "Max. Cd at port B" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecB = 1000 "Critical Reynolds number at port B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolB0 = 10 "Initial volume at port B" annotation(
    Dialog(tab = "Advanced"));
  Types.Pressure pB(start = 0) "Pressure on volume B";
  Types.VolumeFlowRate qB(start = 0) "Flow at inlet port B";
  SI.ReynoldsNumber ReB "Reynold number at port B";
  SI.Velocity wB "fluid mean velocity at port B";
  Types.Dimensionless CdB "Discharge coefficient at port B";
  // Volume C
  parameter Types.Diameter dC = 100 "Inlet diameter at fluid port C" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdCmax = 0.7 "Max. Cd at port C" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecC = 1000 "Critical Reynolds number at port C" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolC0 = 10 "Initial volume at port C" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pC(start = 0) "Pressure on volume C";
  Types.VolumeFlowRate qC(start = 0) "Flow at inlet port C";
  SI.ReynoldsNumber ReC "Reynold number at port C";
  SI.Velocity wC "fluid mean velocity at port C";
  Types.Dimensionless CdC "Discharge coefficient at port C";
  // Volume D
  parameter Types.Diameter dD = 100 "Inlet diameter at fluid port D" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdDmax = 0.7 "Max. Cd at port D" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecD = 1000 "Critical Reynolds number at port D" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolD0 = 10 "Initial volume at port D" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pD(start = 0) "Pressure on volume D";
  Types.VolumeFlowRate qD(start = 0) "Flow at inlet port D";
  SI.ReynoldsNumber ReD "Reynold number at port D";
  SI.Velocity wD "fluid mean velocity at port D";
  Types.Dimensionless CdD "Discharge coefficient at port D";
  // Volume E
  parameter Types.Diameter dE = 100 "Inlet diameter at fluid port E" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdEmax = 0.7 "Max. Cd at port E" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecE = 1000 "Critical Reynolds number at port E" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolE0 = 10 "Initial volume at port E" annotation(
```

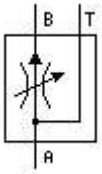
```
    Dialog(tab = "Advanced"));
  Types.Pressure pE(start = 0) "Pressure on volume E";
  Types.VolumeFlowRate qE(start = 0) "Flow at inlet port E";
  SI.ReynoldsNumber ReE "Reynold number at port E";
  SI.Velocity wE "fluid mean velocity at port D";
  Types.Dimensionless CdE "Discharge coefficient at port D";
  // Generic 2 port valve parameter and variable
  parameter Types.Pressure pcrack = 10 "Spring pre-tension";
  parameter Types.Pressure pmax = 10 "Pressure required to open the valve fully from pcrak (=
kspring*stroke/A)";
  parameter Types.Dimensionless aC = 1 "Dimensionless area for pilot pressure at port C (rel. to
spring chamber active area)";
  parameter Types.Dimensionless aD = 1 "Dimensionless area for pilot pressure at port D (rel. to
spring chamber active area)";
  parameter Types.Dimensionless aE = 1 "Dimensionless area for pilot pressure at port E (rel. to
spring chamber active area)";
  parameter Types.VolumeFlowRate qnom = 10 "Nominal flow rate at fully opened valve";
  parameter Types.Pressure pnom = 10 "Nominal pressure difference at fully opened valve";
  parameter Types.Dimensionless Cdmax = 0.7 "Maximum discharge coefficient" annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber Recritic = 1000 "Critical Reynold number" annotation(
    Dialog(tab = "Advanced"));
  parameter Real xv_array[:] = {0, 1} "Fractional opening area range for Area calculation"
annotation(
    Dialog(tab = "Advanced"));
  parameter Real A_array[:] = {0, 1} "Fractional Area corresponding at xv_array -> Area =
A_array(xv_array)*Amax" annotation(
    Dialog(tab = "Advanced"));
  Types.VolumeFlowRate q(start = 0) "Flow across the edge (from A to B)";
  Types.Pressure dp(start = 0) "DeltaP across the edge (dp = pA - pB)";
  SI.ReynoldsNumber Re(start = 0) "Reynold number";
  Types.Dimensionless Cd(start = Cdmax) "Discharge coefficient";
  SI.Velocity w "fluid mean velocity";
  Types.Area Amax(start = Amax0) "Valve maximum area";
  Types.Dimensionless xv(start = 0) "Fractional valve opening";
  Types.Area Ae(start = 0) "Valve opening";
  Types.Diameter Dh(start = 0) "Hydraulic diameter corresponding to the opened section";
  Real dpnet(start = 0) "if >0 valve is regulating";
protected
  parameter Types.Area Amax0 = qnom / (12 * sqrt(5) * Cdmax * sqrt(pnom / Constants.rho));
  Types.Area AA = Constants.pi * dA ^ 2 / 4 "Inlet section at port A";
  Types.Area AB = Constants.pi * dB ^ 2 / 4 "Inlet section at port B";
  Types.Area AC = Constants.pi * dC ^ 2 / 4 "Inlet section at port C";
  Types.Area AD = Constants.pi * dD ^ 2 / 4 "Inlet section at port D";
  Types.Area AE = Constants.pi * dE ^ 2 / 4 "Inlet section at port E";
equation
// inflows / outflows on ports and volumes
  (qA, wA, ReA, CdA) = Functions.orif3f(A.p, pA, AA, dA, CdAmax, RecA);
  (qB, wB, ReB, CdB) = Functions.orif3f(pB, B.p, AB, dB, CdBmax, RecB);
  (qC, wC, ReC, CdC) = Functions.orif3f(pC, C.p, AC, dC, CdCmax, RecC);
  (qD, wD, ReD, CdD) = Functions.orif3f(D.p, pD, AD, dD, CdDmax, RecD);
  (qE, wE, ReE, CdE) = Functions.orif3f(E.p, pE, AE, dE, CdEmax, RecE);
  A.q = qA;
  B.q = -qB;
  C.q = -qC;
  D.q = qD;
  E.q = qE;
  der(pA) = Functions.hCapacitance(qA, q, VolA0, 0, pA, Constants.bulk);
  der(pB) = Functions.hCapacitance(q, qB, VolB0, 0, pB, Constants.bulk);
  der(pC) = Functions.hCapacitance(0, qC, VolC0, 0, pC, Constants.bulk);
  der(pD) = Functions.hCapacitance(qD, 0, VolD0, 0, pD, Constants.bulk);
  der(pE) = Functions.hCapacitance(qE, 0, VolE0, 0, pE, Constants.bulk);
// generic 2 ports valve equations
  qnom = 12 * Amax * Cdmax * sqrt(5 * pnom / Constants.rho) * tanh(4 * sqrt(Amax * pnom / (5 *
Constants.pi * Constants.rho)) * Constants.rho / (Constants.mu * Recritic));
  dp = pA - pB;
  dpnet = aE * pE + aD * pD - aC * pC - pcrack;
  xv = dpnet / pmax;
  if noEvent(xv < 0) then
    Ae = 0;
    Dh = 0;
  elseif noEvent(xv < 1) then
    Ae = Functions.linterp(xv, xv_array, A_array) * Amax;
    Dh = 2 * sqrt(Ae / Constants.pi);
  else
    Ae = Amax;
    Dh = 2 * sqrt(Amax / Constants.pi);
  end if;
```






```
  (q, w, Re, Cd) = Functions.orif3f(pA, pB, Ae, Dh, Cdmax, Recritic);
  annotation(
    Icon(graphics = {Rectangle(origin = {10, -12}, lineThickness = 0.5, extent = {{-70, 72}, {50, -
48}}), Line(origin = {95.7702, 32.3102}, rotation = 90, points = {{3.08625, 35.9473}, {-20.9138,
31.9473}, {19.0862, 25.9473}, {-20.9138, 17.9473}, {19.0862, 11.9473}, {-20.9138, 5.9473}, {19.0862,
1.9473}, {-4.91375, -4.0527}}), Line(origin = {246.112, -16.1611}, points = {{-177, 21}, {-167,
81}}, thickness = 0.5), Polygon(origin = {-55, 108}, fillPattern = FillPattern.Solid, points =
{{135, -42}, {131, -52}, {135, -54}, {135, -42}}), Polygon(origin = {15, -30}, fillPattern =
FillPattern.Solid, points = {{-39, -4}, {-51, -4}, {-45, -18}, {-39, -4}}), Line(origin = {-4.67939,
16.6412}, points = {{-25, -56}, {-25, 24}, {35, 24}}, thickness = 0.5), Text(origin = {74, 80},
extent = {{-44, 28}, {36, -12}}, textString = "%pcrack"), Line(origin = {80.6412, -36.9771}, points
= {{-20, -2}, {18, -2}}, pattern = LinePattern.Dash, thickness = 0.5), Line(origin = {-80.8855,
38.1069}, points = {{-18, 2}, {20, 2}}, pattern = LinePattern.Dash, thickness = 0.5), Line(origin =
{-80.8244, -42.3054}, points = {{-18, 2}, {20, 2}}, pattern = LinePattern.Dash, thickness = 0.5),
Line(origin = {0, 80}, points = {{0, 18}, {0, -20}}, thickness = 0.5), Line(origin = {0.381679, -
80.2443}, points = {{0, 20}, {0, -18}}, thickness = 0.5)}, coordinateSystem(initialScale = 0.1)));
end Generic2PortValve;
```

### 5.10.4 Esempio: realizzare una valvola regolatrice di flusso a 3 vie




Realizziamo il seguente schema con Modelica, sfruttando la libreria FPLib.

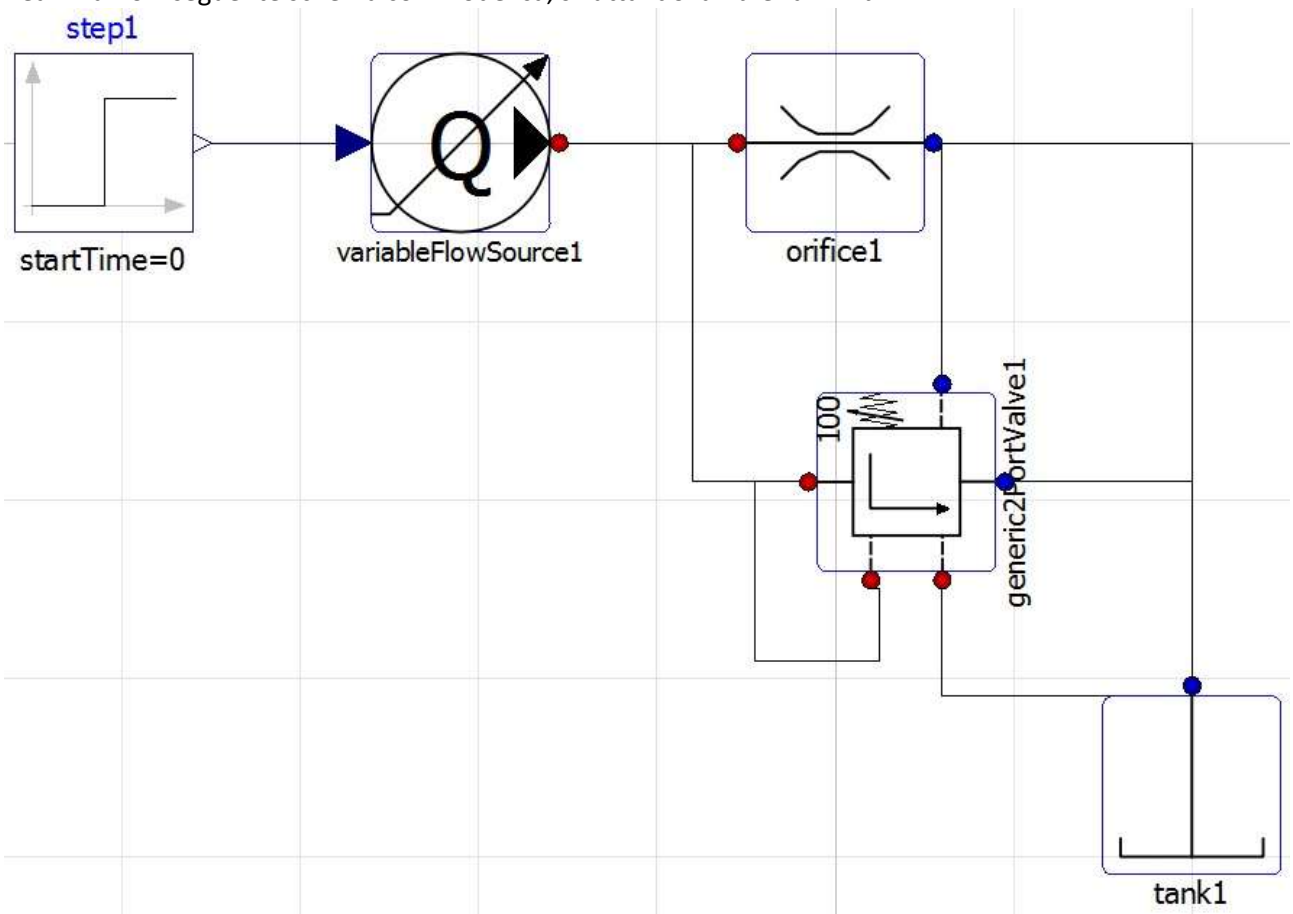
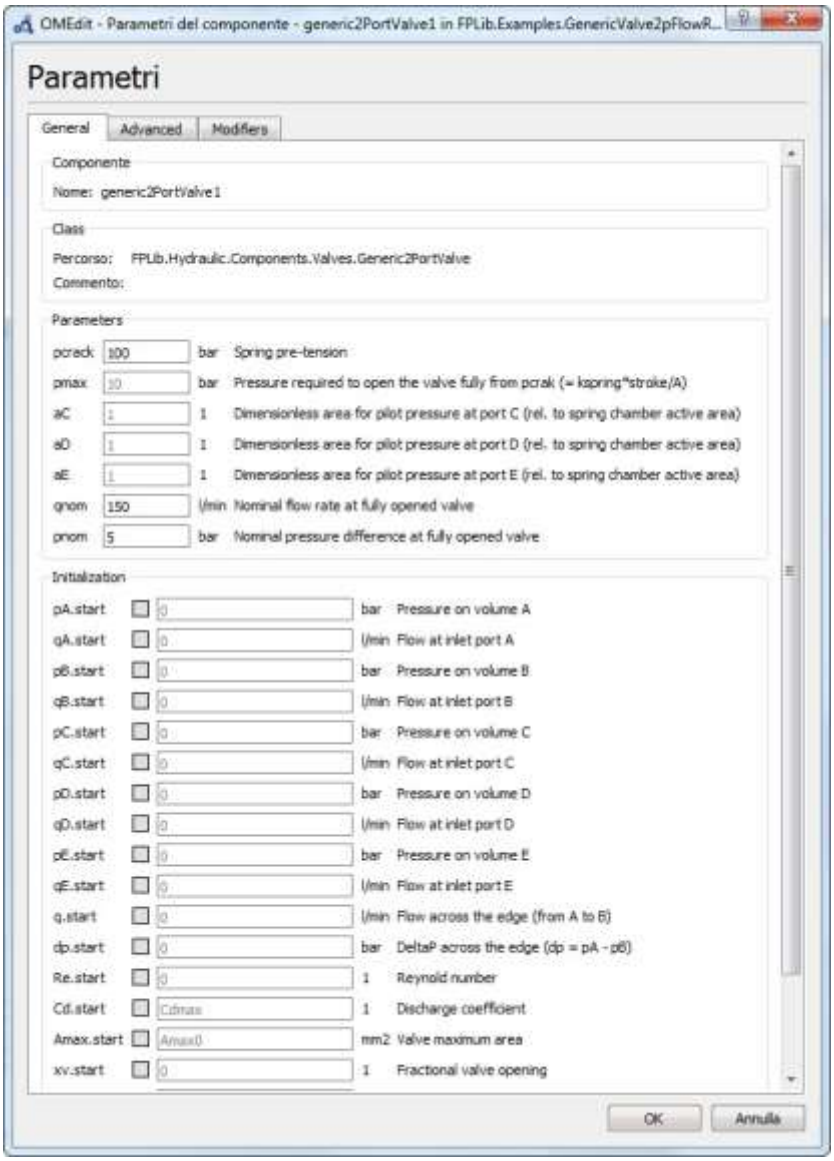



Figura 5-7: valvola regolatrice di flusso a 3 vie / valvola limitatrice di pressione differenziale

La valvola funzione come valvola limitatrice di pressione differenziale, infatti apre quando la pressione a monte dello strozzatore meno la pressione a valle, supera il valore di taratura della valvola.
Ne consegue che la pressione sullo strozzatore è limitata a $p_{crack}$, al quale potrà aggiungersi l'errore di regolazione dovuto alla pressione che dobbiamo fornire per aprire completamente la valvola $p_{max}$.
Ecco come abbiamo impostato i parametri per la valvola a 2 vie:

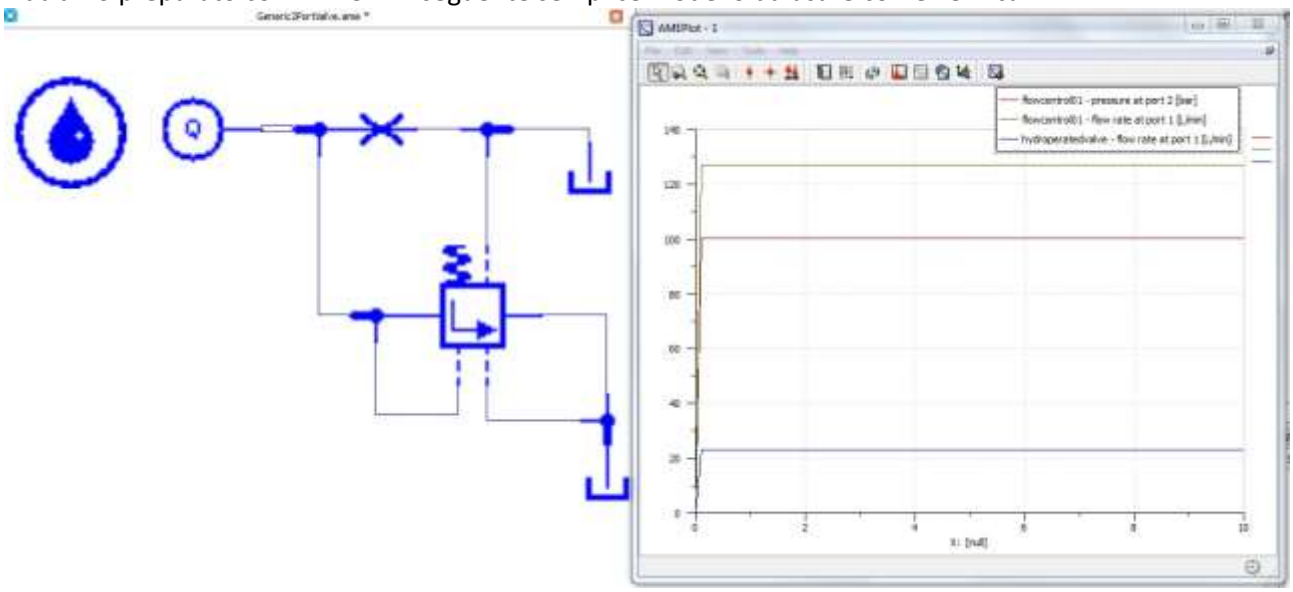



Il gradino invece lo abbiamo impostato a 150.

Se lanciamo la simulazione otteniamo i seguenti risultati, a regime, pressione ai capi dello strozzatore = 100.346 bar, portata = 126.7 L/min, portata che esce dalla valvola a 2 vie = 23.3 L/min.

#### *5.10.4.1 Verifica con AMESim*

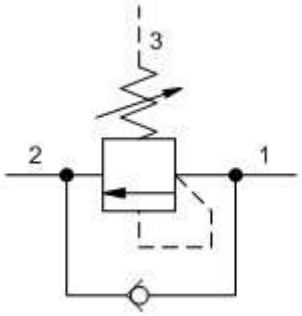
Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:



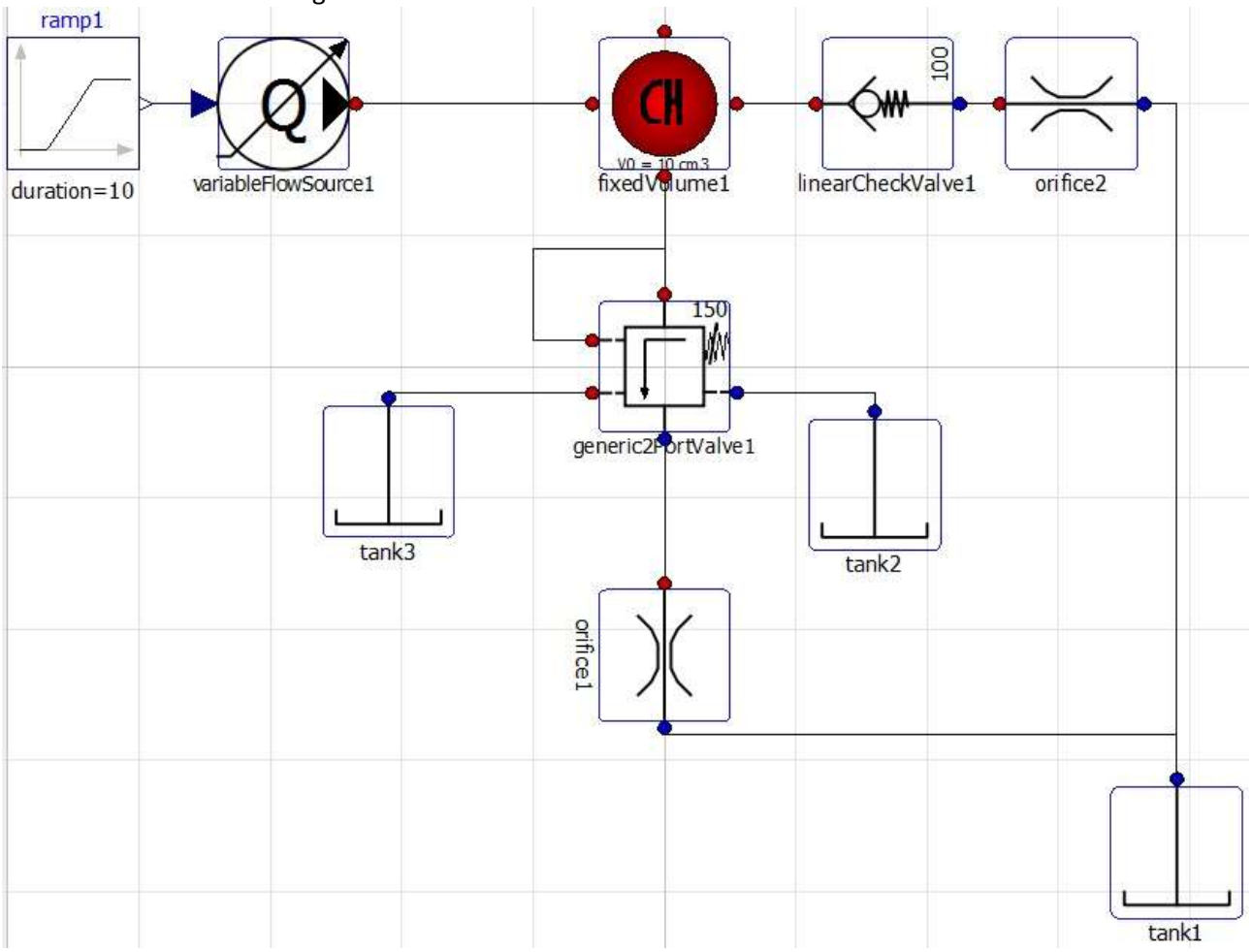
I risultati sono pressoché identici.

### 5.10.5 Esempio: realizzare una valvola di sequenza

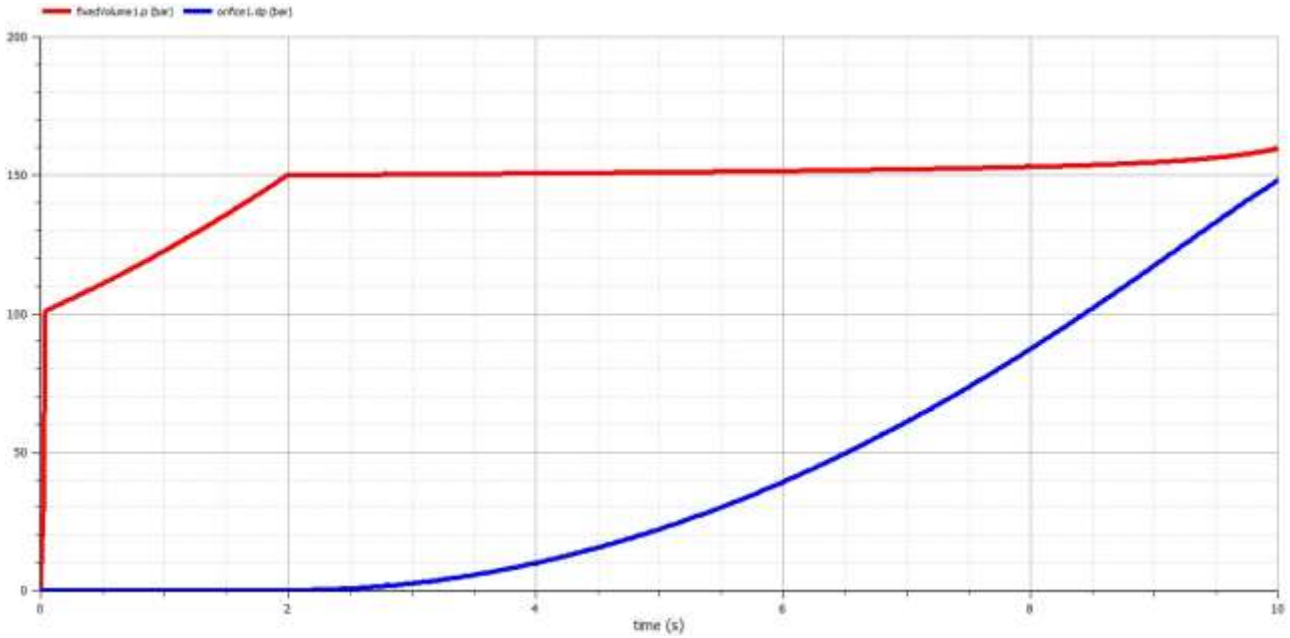
Con la valvola idraulica generica a 2 vie realizziamo una valvola di sequenza.




Creiamo con Modelica il seguente schema:



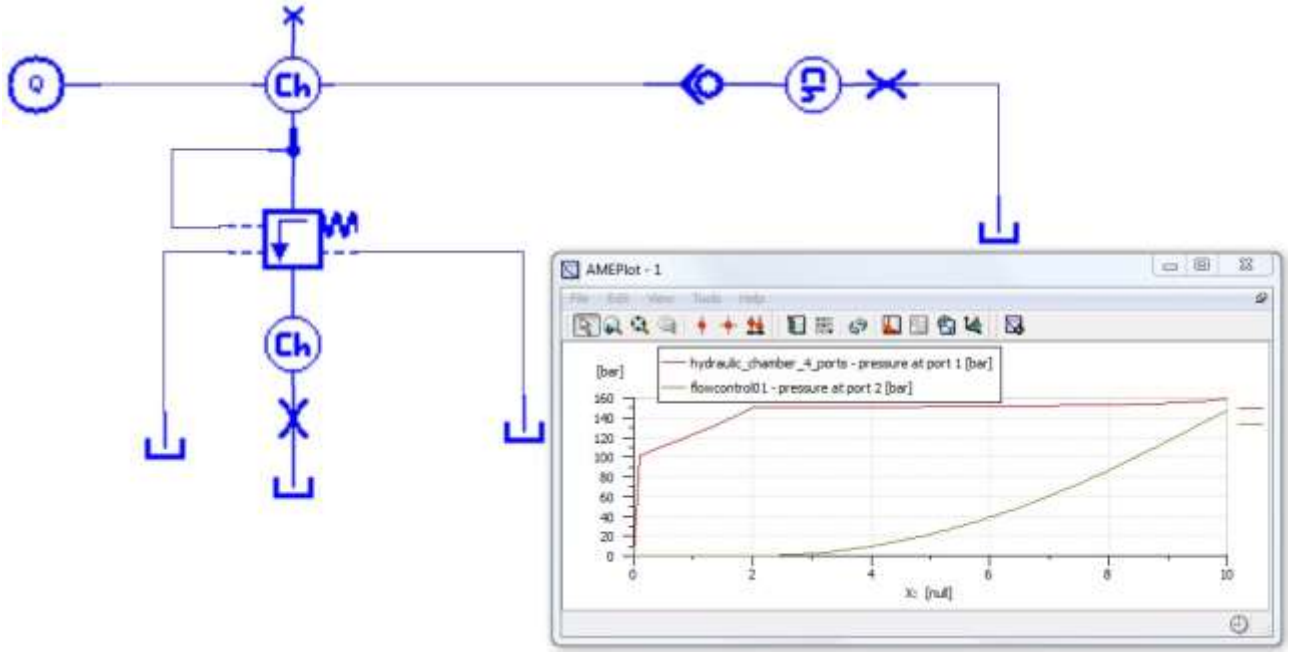

Un generatore ideale di portata immette olio nel volume. Poiché trova resistenza nella check-valve, la pressione nel volume cresce, poi a 100 bar la check valve si apre e l'olio inizia a fluire nello strozzatore orifice2. Si determina una ulteriore resistenza che fa salire ulteriormente la pressione. Tale resistenza è crescente poichè il generatore di portata è comandato da una rampa che fa da 0 a 200 L/min in 10 s. Quando ad un certo punto nel volume ci sono 150 bar, la valvola generic2PortValve1 si apre e opera come valvola di sequenza.
Se lanciamo la simulazione, otteniamo:




In pratica dopo 2 s la pressione arriva a 150 bar e quindi si apre la valvola, come testimonia il ΔP sullo strozzatore a valle della valvola di sequenza. Sebbene la valvola di sequenza sia tarata a 150 bar, la pressione a valle impiega ulteriore tempo per arrivare al valore prossimo a quello della taratura, perché nel sistema è necessaria la giusta portata per poter alimentare le due utenze (i due strozzatori).
La simulazione ha mostrato una criticità di questo circuito: ad una occhiata veloce dello schema si poteva supporre che la pressione a valle fosse sempre disponibile invece dipende fortemente dalla portata.

### *5.10.5.1 Verifica con AMESim*

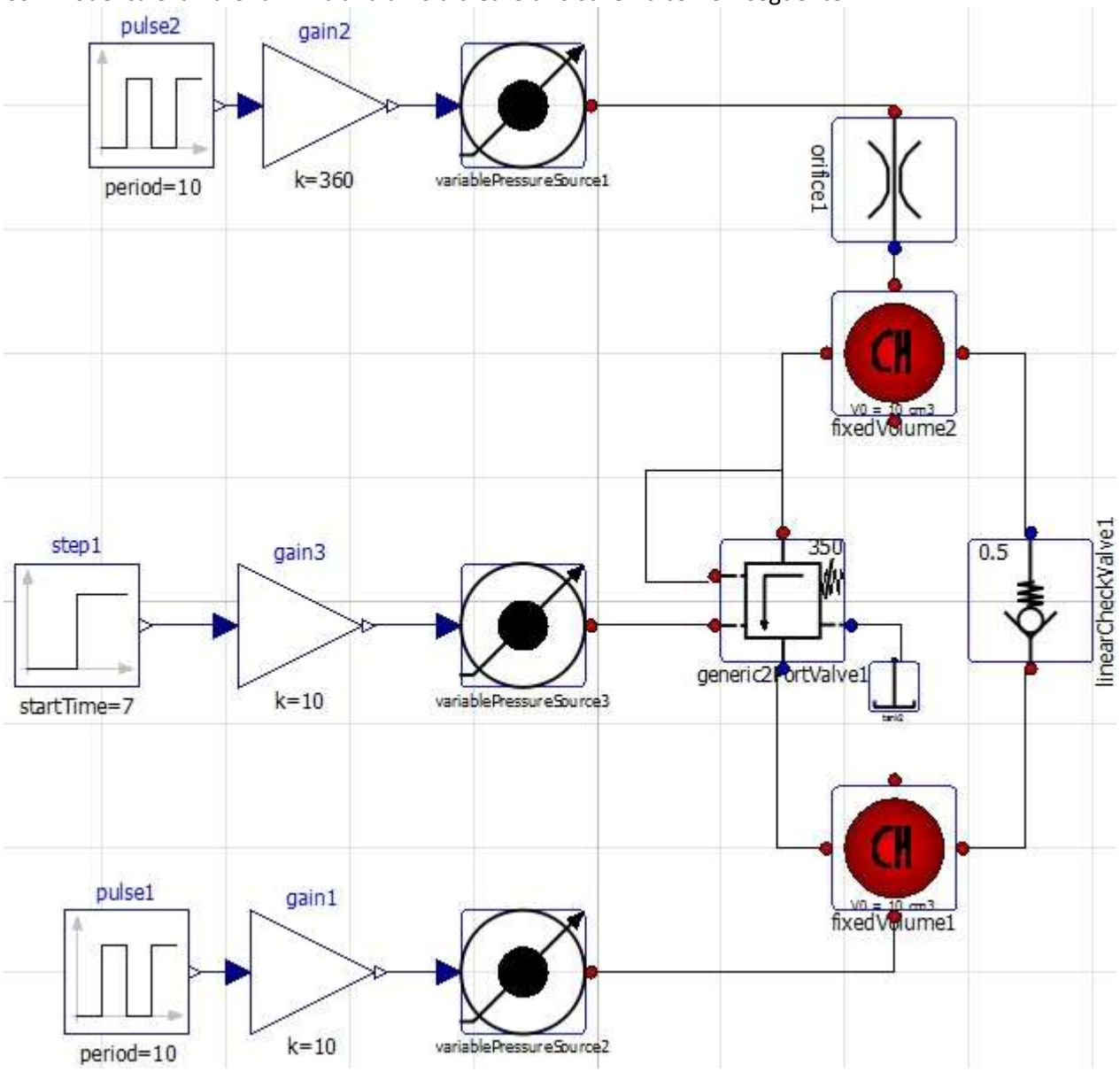
Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




In pratica i valori sono gli stessi.

### 5.10.6 Esempio: realizzare una valvola overcenter

Creare una valvola overcenter con rapporto di pilotaggio 10:1, tarata a 350 bar[2].
Con Modelica e la libreria FPLib andiamo a creare uno schema come il seguente:



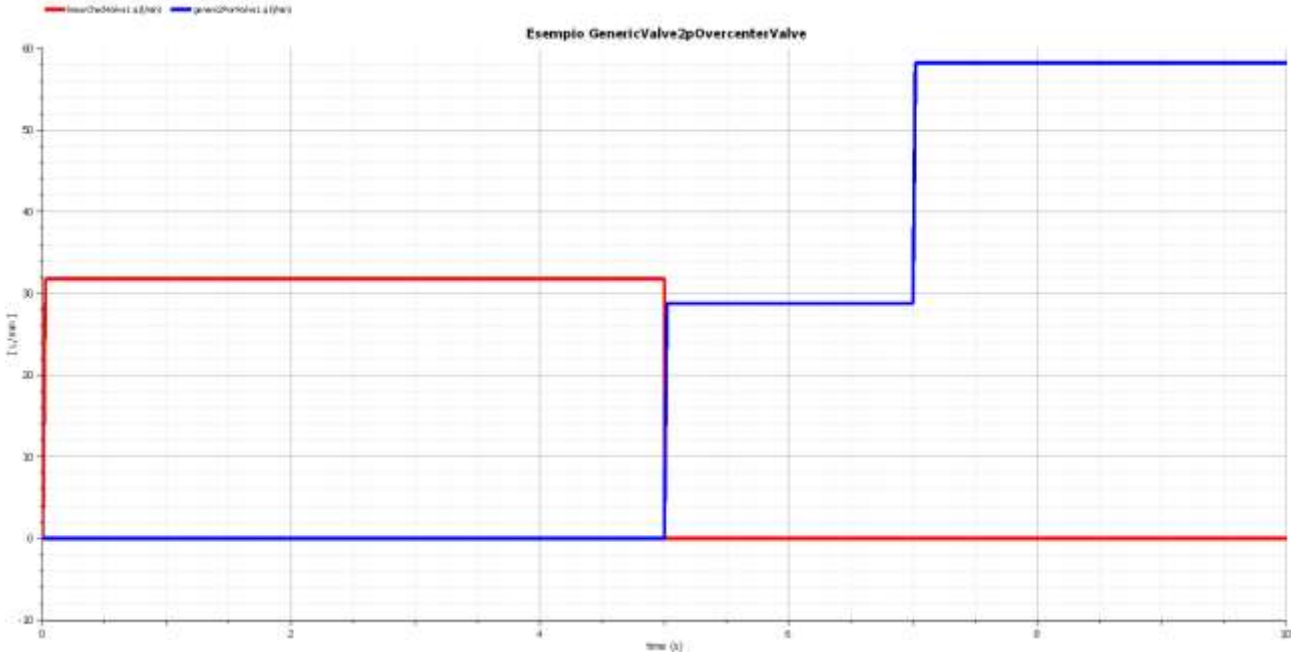

I parametri per i tre blocchi/segnali sono:

| Blocco pulse1 | Blocco pulse1 | Blocco step1 |
|---|---|---|
| amplitude = 1 (default)<br>width = 50 (default)<br>period = 10 (modificato)<br>nperiod = -1 (default)<br>offset = 0 (default)<br>startTime = 0 (default). | amplitude = -1 (modificato)<br>width = 50 (default)<br>period = 10 (modificato)<br>nperiod = -1 (default)<br>offset = 1 (modificato)<br>startTime = 0 (default). | height = 1 (default)<br>offset = 0 (default)<br>startTime = 7 (modificato) |

Tre blocchi **gain** vanno poi a settare la magnitudo del segnale (il guadagno è indicato sotto ciascun blocco gain). Il blocco gain fa parte della libreria standard di Modelica (**Modelica.Blocks.Math**).

[2] La difficoltà sta più che altro nel circuito di test, poiché non abbiamo ancora creato le valvole direzionali, è difficile realizzare un movimento di salita e discesa. Per fare questo creiamo due sorgenti di pressione variabili nel tempo, che lavorano alternativamente, anzi, tre se includiamo il pilotaggio della valvola overcenter.

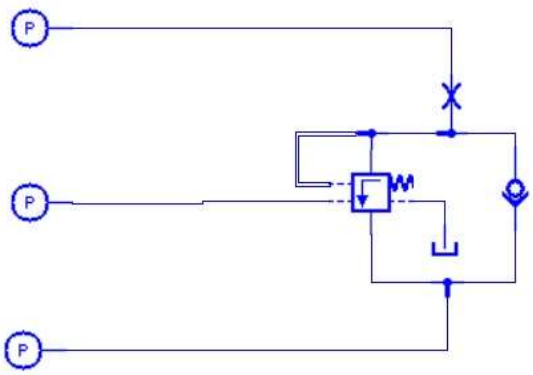
La linearCheckValve1 è settata con i seguenti parametri: Ptar = 0.5 (default), G = 0.1 (modificato) e pHysteresis = 0 (default).
La generic2PortValve1 ha i seguenti parametric: $p_{crack}$ = 350 (modificato), $p_{max}$ = 10 (default), $a_C$ = 1 (defualt), $a_D$ = 1 (default), $a_E$ = 10 (modificato), $q_{nom}$ = 10 (default) e $p_{nom}$ = 10 (default).
In particolare, **$a_E$ definisce il rapporto di pilotaggio rispetto ad $a_D$** (nell'esempio **10 : 1 ↔ $a_E$ : $a_D$**), il rapporto **R = $a_E$ / $a_D$ corrisponde al rapporto tra area di pilotaggio / area di tenuta** in una valvola reale corrispondente.
Se lanciamo la simulazione (per 10 s) otteniamo:

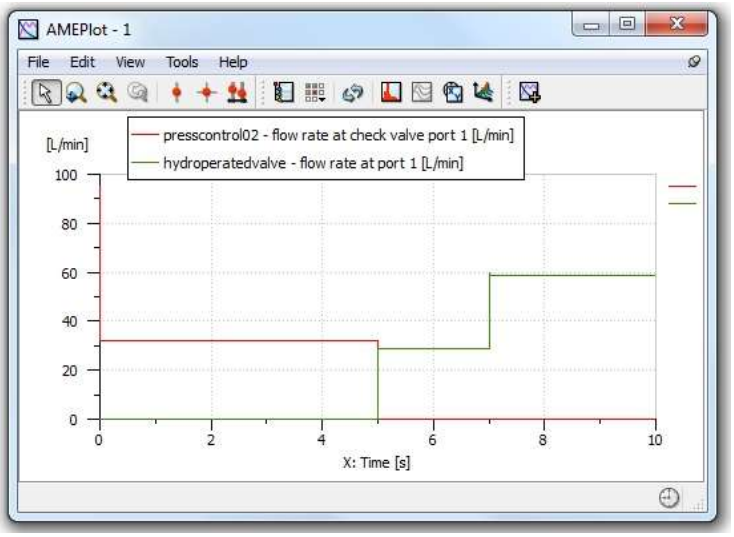



In pratica, la linea rossa è la portata attraverso la valvola unidirezionale, mentre la linea blu è la portata attraverso la valvola generic2PortValve1.
Nei primi 5 s, pulse2 è nullo e pulse1 è 1, quindi l'azione combinata delle sorgenti di pressione, comandate in questo modo, simula la salita di un carico, si vede infatti che la portata nella linearCheckValve1 è 31.8 L/min.
Nei successivi 5 s, pulse 1 e pulse 2 si scambiano, quindi intendiamo simulare una manovra di discesa.
La manovra s'avvia (notiamo che è una discesa perché la valvola unidirezionale rimane chiusa, mentre si apre la generic2PortValve1). La portata che passa tra $5 \le t \le 7$ è circa 28.8 L/min.
Ma poi a partire da t = 7 s, c'è il segnale di pilotaggio, che fa aprire ulteriormente la generic2PortValve1, infatti la portata aumenta e raggiunge il valore di 58.2 L/min.

### *5.10.6.1 Verifica con AMESim*

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




AMESim fornisce 31.8 L/min per la check valve, quindi 29 L/min per la valvola a 2 vie e 58.6 quando interviene il segnale di pilotaggio.

Anche in questo caso la libreria FPLib fornisce una buona approssimazione dei risultati.

## 5.11 Valvola idraulica generica a 3 vie (normalmente aperta)

Il ruolo di questa valvola è quello d'essere utilizzata in combinazione con altri componenti per formare diverse valvole di controllo della pressione e della portata.
E' una valvola generica, con tre porte, e due segnali di pilotaggio.

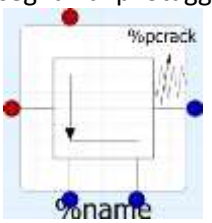



**Figura 5-8: schema Modelica di una valvola idraulica generica a 3 vie normalmente aperta**

Se un attacco di pilotaggio non esiste/non serve, nello schema che si vuole realizzare, va messo a scarico.

### 5.11.1 Descrizione del modello

L'area massima è calcolata mediante i parametri $q_{nomA2B}$, $p_{nomA2B}$, $q_{nomA2C}$ e $p_{nomA2C}$ che sono le portate nominali (alla massima apertura) ed i relativi ΔP d'attraversamento per le due vie.
Vengono applicate le formule [5-2, 5-3, 5-4, 5-5 e 5-6].
Una prima stima delle due aree massime è calcolata con la [5-2], assumendo $A_0$ come area massima, ΔP = pnom e Q = $q_{nom}$ nei due casi A2B e A2C.
Poi sono inserite due equazioni che legano i coefficienti $C_q$ calcolati con la [5-5] alle portate nominali e ai ΔP di passaggio, combinate alle equazioni [5-2, 5-3, 5-4, 5-5 e 5-6]
La pressione netta in apertura è data da:

$$dp_{net} = a_E \cdot p_E - a_D \cdot p_D - p_{crack}$$

Dove $a_E$ e $a_D$ sono fattori che definiscono i rapporti di pilotaggio.
Il primo ($a_E$) è quello a sinistra, il secondo ($a_D$) è quello a destra (dal lato molla).
L'apertura della valvola (valore frazionario, rispetto alla corsa massima) è data da:

$$x_v = \frac{dp_{net}}{p_{\max}}$$

Dove $p_{max}$ è la pressione che porta l'elemento mobile dalla posizione di riposo (completamente chiusa) a quella completamente aperta.

- Se $x_v$ è minore di -1: A → B completamente aperta (B è in basso a sinistra), da A a C è chiusa.
- Se $x_v$ è compreso tra -1 e 0: A → B, la sezione aperta, $A_{e,A2B}$ è $x_v \cdot A_arrayA2B(x_v) \cdot A_{max,A2B}$, $A_{e,A2C} = 0$.
- Se $x_v$ è compreso tra 0 e +1: A → C, la sezione aperta, $A_{e,A2C}$ è $x_v \cdot A_arrayA2C(x_v) \cdot A_{max,A2C}$, $A_{e,A2B} = 0$.
- Se $x_v$ è maggiore di 1: A → C completamente aperta (C è in basso a destra), da A a B è chiusa.

Le due coppie di vettori **xv_arrayA2B**, **A_arrayA2B** e **xv_arrayA2C**, **A_arrayA2C** si possono utilizzare allo stesso modo descritto nel paragrafo 5.10.2.

### 5.11.2 Codice

Ecco il codice:

```
model Generic3PortValve
  extends Interfaces.PartialHydraulicBase;
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
    Placement(visible = true, transformation(origin = {0, 108}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-40, 110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b B annotation(
    Placement(visible = true, transformation(origin = {0, -108}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-40, -110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b C annotation(
    Placement(visible = true, transformation(origin = {112, -30}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {40, -110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b D annotation(
    Placement(visible = true, transformation(origin = {126, -4}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_a E annotation(
    Placement(visible = true, transformation(origin = {-184, -68}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-110, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
```





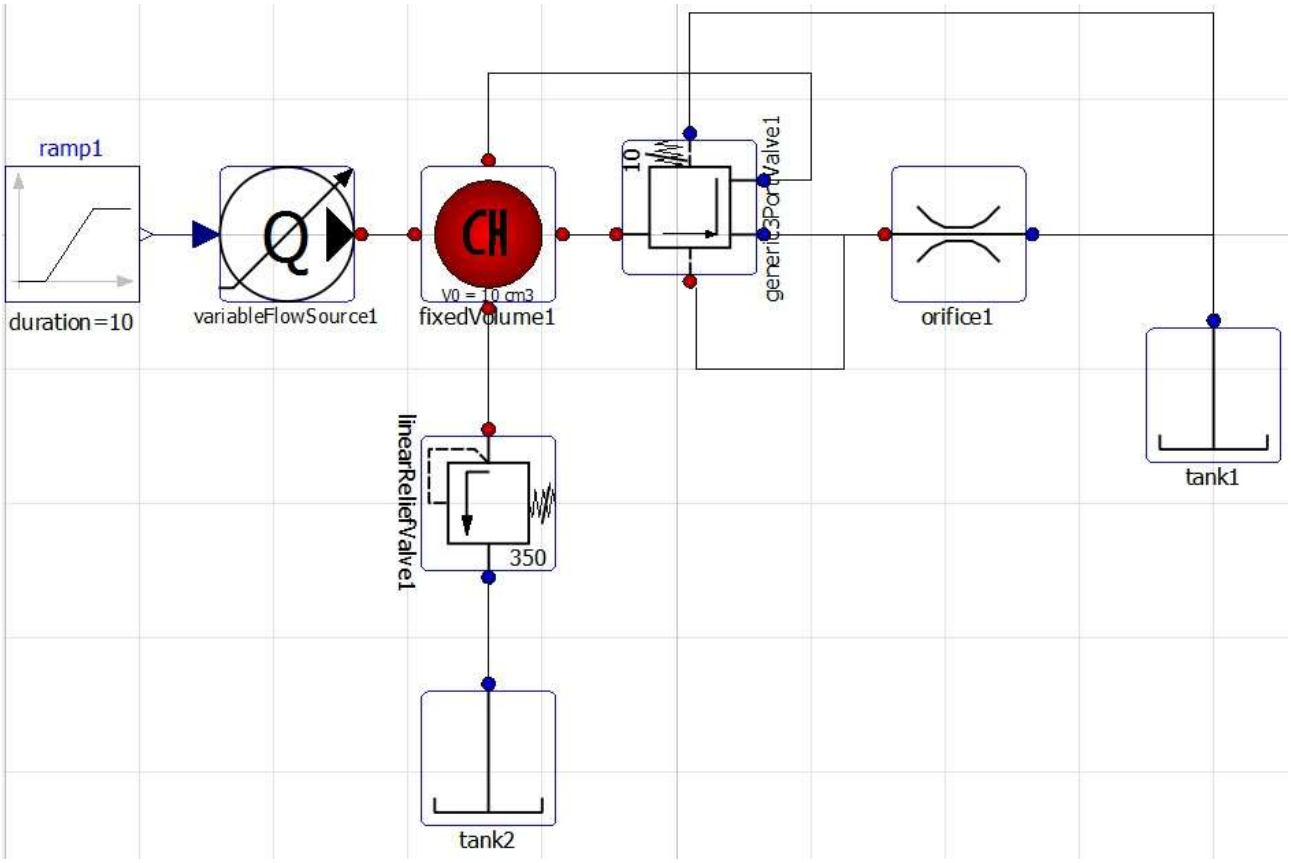

```
  // Volume A
  parameter Types.Diameter dA = 100 "Inlet diameter at fluid port A" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdAmax = 0.7 "Max. Cd at port A" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecA = 1000 "Critical Reynolds number at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolA0 = 10 "Initial volume at port A" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pA(start = 0) "Pressure on volume A";
  Types.VolumeFlowRate qA(start = 0) "Flow at inlet port A";
  SI.ReynoldsNumber ReA "Reynold number at port A";
  SI.Velocity wA "fluid mean velocity at port A";
  Types.Dimensionless CdA "Discharge coefficient at port A";
  // Volume B
  parameter Types.Diameter dB = 100 "Inlet diameter at fluid port B" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdBmax = 0.7 "Max. Cd at port B" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecB = 1000 "Critical Reynolds number at port B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolB0 = 10 "Initial volume at port B" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pB(start = 0) "Pressure on volume B";
  Types.VolumeFlowRate qB(start = 0) "Flow at inlet port B";
  SI.ReynoldsNumber ReB "Reynold number at port B";
  SI.Velocity wB "fluid mean velocity at port B";
  Types.Dimensionless CdB "Discharge coefficient at port B";
  // Volume C
  parameter Types.Diameter dC = 100 "Inlet diameter at fluid port C" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdCmax = 0.7 "Max. Cd at port C" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecC = 1000 "Critical Reynolds number at port C" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolC0 = 10 "Initial volume at port C" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pC(start = 0) "Pressure on volume C";
  Types.VolumeFlowRate qC(start = 0) "Flow at inlet port C";
  SI.ReynoldsNumber ReC "Reynold number at port C";
  SI.Velocity wC "fluid mean velocity at port C";
  Types.Dimensionless CdC "Discharge coefficient at port C";
  // Volume D
  parameter Types.Diameter dD = 100 "Inlet diameter at fluid port D" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdDmax = 0.7 "Max. Cd at port D" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecD = 1000 "Critical Reynolds number at port D" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolD0 = 10 "Initial volume at port D" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pD(start = 0) "Pressure on volume D";
  Types.VolumeFlowRate qD(start = 0) "Flow at inlet port D";
  SI.ReynoldsNumber ReD "Reynold number at port D";
  SI.Velocity wD "fluid mean velocity at port D";
  Types.Dimensionless CdD "Discharge coefficient at port D";
  // Volume E
  parameter Types.Diameter dE = 100 "Inlet diameter at fluid port E" annotation(Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdEmax = 0.7 "Max. Cd at port E" annotation(Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecE = 1000 "Critical Reynolds number at port E" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolE0 = 10 "Initial volume at port E" annotation(Dialog(tab = "Advanced"));
  Types.Pressure pE(start = 0) "Pressure on volume E";
  Types.VolumeFlowRate qE(start = 0) "Flow at inlet port E";
  SI.ReynoldsNumber ReE "Reynold number at port E";
  SI.Velocity wE "fluid mean velocity at port D";
  Types.Dimensionless CdE "Discharge coefficient at port D";
  // Generic 3 port valve parameter and variables
  parameter Types.Pressure pcrack = 10 "Spring pre-tension";
  parameter Types.Pressure pmax = 10 "pilot pressure required to open valve fully port 4-2";
  parameter Types.Dimensionless aD = 1 "Dimensionless area for pilot pressure at port D (rel. to
spring chamber active area)";
  parameter Types.Dimensionless aE = 1 "Dimensionless area for pilot pressure at port E (rel. to
spring chamber active area)";
  parameter Types.VolumeFlowRate qnomA2B = 10 "Nominal flow rate at fully opened valve froma A to
B";
  parameter Types.Pressure pnomA2B = 10 "Nominal pressure drop froma A to B (at nominal flow rate)";
  parameter Types.VolumeFlowRate qnomA2C = 10 "Nominal flow rate at fully opened valve froma A to
C";
  parameter Types.Pressure pnomA2C = 10 "Nominal pressure drop froma A to C (at nominal flow rate)";
  parameter Types.Dimensionless CdmaxA2B = 0.7 "Maximum discharge coefficient from A to B"
annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecriticA2B = 1000 "Critical Reynold number from A to B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdmaxA2C = 0.7 "Maximum discharge coefficient from A to C"
annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecriticA2C = 1000 "Critical Reynold number from A to C" annotation(
    Dialog(tab = "Advanced"));
```

```
  Types.VolumeFlowRate q1(start = 0) "Flow from A to B";
  Types.VolumeFlowRate q2(start = 0) "Flow from A to C";
  SI.ReynoldsNumber Re1(start = 0) "Reynold number from A to B";
  Types.Dimensionless Cd1(start = CdmaxA2B) "Discharge coefficient from A to B";
  SI.Velocity w1 "fluid mean velocity from A to B";
  Types.Area AmaxA2B(start = AmaxA2B0) "Valve maximum area from A to B";
  SI.ReynoldsNumber Re2(start = 0) "Reynold number from A to C";
  Types.Dimensionless Cd2(start = CdmaxA2C) "Discharge coefficient from A to C";
  SI.Velocity w2 "fluid mean velocity from A to C";
  Types.Area AmaxA2C(start = AmaxA2C0) "Valve maximum area from A to C";
  Types.Area Ae1(start = 0) "Valve opening from A to B";
  Types.Diameter Dh1(start = 0) "Hydraulic diameter corresponding to the opened section from A to
B";
  Types.Area Ae2(start = 0) "Valve opening from A to C";
  Types.Diameter Dh2(start = 0) "Hydraulic diameter corresponding to the opened section from A to
C";
  Types.Dimensionless xv(start = 0) "Fractional valve opening";
  parameter Real xv_arrayA2B[:] = {-1, 0} "Fractional opening area range for Area calculation"
annotation(
    Dialog(tab = "Advanced"));
  parameter Real A_arrayA2B[:] = {1, 0} "Fractional Area (A=1 at xv=-1 corresponds to AmaxA2B)"
annotation(
    Dialog(tab = "Advanced"));
  parameter Real xv_arrayA2C[:] = {0, 1} "Fractional opening area range for Area calculation"
annotation(
    Dialog(tab = "Advanced"));
  parameter Real A_arrayA2C[:] = {0, 1} "Fractional Area (A=1 at xv=1 corresponds to AmaxA2C)"
annotation(
    Dialog(tab = "Advanced"));
  Real dpnet(start = 0) "if >0 valve is regulating";
protected
  parameter Types.Area AmaxA2B0 = qnomA2B / (12 * sqrt(5) * CdmaxA2B * sqrt(pnomA2B /
Constants.rho));
  parameter Types.Area AmaxA2C0 = qnomA2C / (12 * sqrt(5) * CdmaxA2C * sqrt(pnomA2C /
Constants.rho));
  Types.Area AA = Constants.pi * dA ^ 2 / 4 "Inlet section at port A";
  Types.Area AB = Constants.pi * dB ^ 2 / 4 "Inlet section at port B";
  Types.Area AC = Constants.pi * dC ^ 2 / 4 "Inlet section at port C";
  Types.Area AD = Constants.pi * dD ^ 2 / 4 "Inlet section at port D";
  Types.Area AE = Constants.pi * dE ^ 2 / 4 "Inlet section at port E";
equation
// inflows / outflows on ports and volumes
  (qA, wA, ReA, CdA) = Functions.orif3f(A.p, pA, AA, dA, CdAmax, RecA);
  (qB, wB, ReB, CdB) = Functions.orif3f(pB, B.p, AB, dB, CdBmax, RecB);
  (qC, wC, ReC, CdC) = Functions.orif3f(pC, C.p, AC, dC, CdCmax, RecC);
  (qD, wD, ReD, CdD) = Functions.orif3f(pD, D.p, AD, dD, CdDmax, RecD);
  (qE, wE, ReE, CdE) = Functions.orif3f(E.p, pE, AE, dE, CdEmax, RecE);
  A.q = qA;
  B.q = -qB;
  C.q = -qC;
  D.q = -qD;
  E.q = qE;
  der(pA) = Functions.hCapacitance(qA, q1 + q2, VolA0, 0, pA, Constants.bulk);
  der(pB) = Functions.hCapacitance(q1, qB, VolB0, 0, pB, Constants.bulk);
  der(pC) = Functions.hCapacitance(q2, qC, VolC0, 0, pC, Constants.bulk);
  der(pD) = Functions.hCapacitance(0, qD, VolD0, 0, pD, Constants.bulk);
  der(pE) = Functions.hCapacitance(qE, 0, VolE0, 0, pE, Constants.bulk);
// generic 2 ports valve equations
  qnomA2B = 12 * AmaxA2B * CdmaxA2B * sqrt(5 * pnomA2B / Constants.rho) * tanh(4 * sqrt(AmaxA2B *
pnomA2B / (5 * Constants.pi * Constants.rho)) * Constants.rho / (Constants.mu * RecriticA2B));
  qnomA2C = 12 * AmaxA2C * CdmaxA2C * sqrt(5 * pnomA2C / Constants.rho) * tanh(4 * sqrt(AmaxA2C *
pnomA2C / (5 * Constants.pi * Constants.rho)) * Constants.rho / (Constants.mu * RecriticA2C));
  dpnet = aE * pE - aD * pD - pcrack;
  xv = dpnet / pmax;
  if noEvent(xv <= (-1)) then
    Ae1 = AmaxA2B;
    Dh1 = 2 * sqrt(AmaxA2B / Constants.pi);
    Ae2 = 0;
    Dh2 = 0;
  elseif noEvent(xv > (-1) and xv < 0) then
    Ae1 = Functions.linterp(xv, xv_arrayA2B, A_arrayA2B) * AmaxA2B;
    Dh1 = 2 * sqrt(Ae1 / Constants.pi);
    Ae2 = 0;
    Dh2 = 0;
  elseif noEvent(xv > 0 and xv < 1) then
    Ae1 = 0;
    Dh1 = 0;
    Ae2 = Functions.linterp(xv, xv_arrayA2C, A_arrayA2C) * AmaxA2C;
```

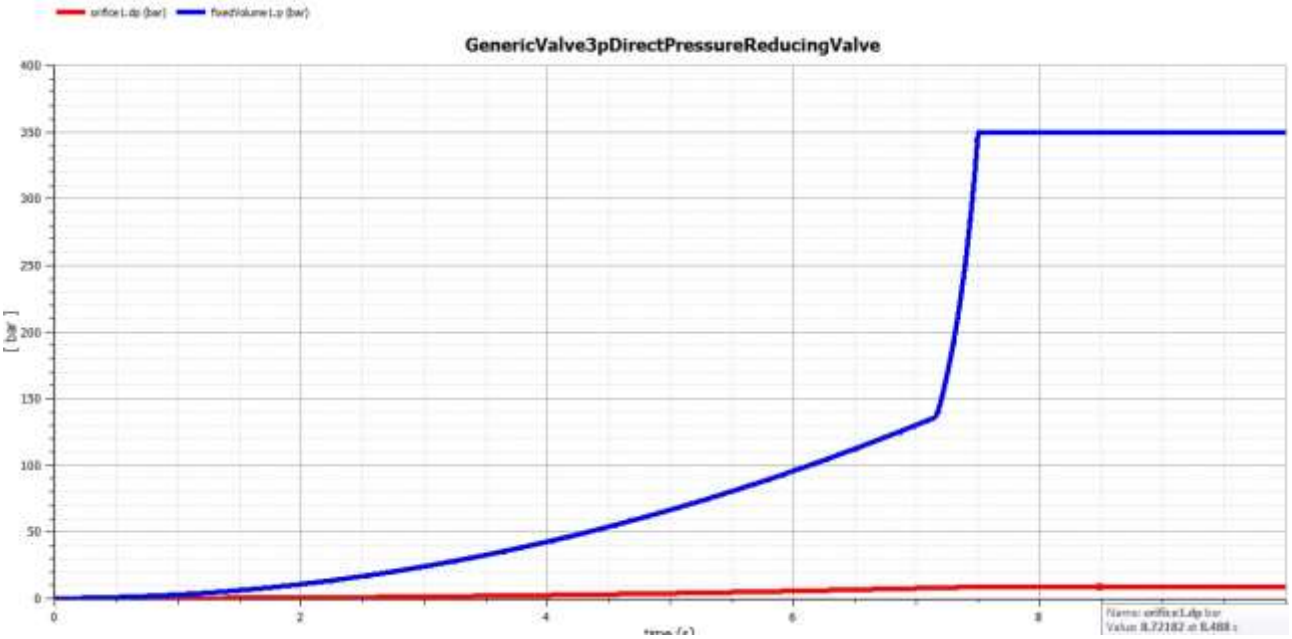
```
    Dh2 = 2 * sqrt(Ae2 / Constants.pi);
  elseif noEvent(xv >= 1) then
    Ae1 = 0;
    Dh1 = 0;
    Ae2 = AmaxA2C;
    Dh2 = 2 * sqrt(AmaxA2C / Constants.pi);
  else
    Ae1 = 0;
    Dh1 = 0;
    Ae2 = 0;
    Dh2 = 0;
  end if;
  (q1, w1, Re1, Cd1) = Functions.orif3f(pA, pB, Ae1, Dh1, CdmaxA2B, RecriticA2B);
  (q2, w2, Re2, Cd2) = Functions.orif3f(pA, pC, Ae2, Dh2, CdmaxA2C, RecriticA2C);
  annotation(
    Icon(graphics = {Rectangle(origin = {10, -12}, lineThickness = 0.5, extent = {{-70, 72}, {50, -
48}}), Line(origin = {95.7702, 32.3102}, rotation = 90, points = {{3.08625, 35.9473}, {-20.9138,
31.9473}, {19.0862, 25.9473}, {-20.9138, 17.9473}, {19.0862, 11.9473}, {-20.9138, 5.9473}, {19.0862,
1.9473}, {-4.91375, -4.0527}}), Line(origin = {246.112, -16.1611}, points = {{-177, 21}, {-167,
81}}, thickness = 0.5), Polygon(origin = {-55, 108}, fillPattern = FillPattern.Solid, points =
{{135, -42}, {131, -52}, {135, -54}, {135, -42}}), Polygon(origin = {5, -20}, fillPattern =
FillPattern.Solid, points = {{-39, -4}, {-51, -4}, {-45, -18}, {-39, -4}}), Text(origin = {74, 80},
extent = {{-44, 28}, {36, -12}}, textString = "%pcrack"), Line(origin = {80.4022, 2.16305}, points =
{{-20, -2}, {18, -2}}, pattern = LinePattern.Dash, thickness = 0.5), Line(origin = {-80.906, -
2.68709}, points = {{-18, 2}, {20, 2}}, pattern = LinePattern.Dash, thickness = 0.5), Line(origin =
{-40.2174, 80.4022}, points = {{0, 18}, {0, -20}}, thickness = 0.5), Line(origin = {-40.6401, -
80.6464}, points = {{0, 20}, {0, -18}}, thickness = 0.5), Line(origin = {40.1426, -80.7007}, points
= {{0, 20}, {0, -18}}, thickness = 0.5), Line(origin = {1.61, -39.95}, points = {{-42, 0}, {40, 0}},
thickness = 0.5), Line(origin = {-39.6087, 1}, points = {{0, 39}, {0, -39}})},
coordinateSystem(initialScale = 0.1)));
end Generic3PortValve;
```

### 5.11.3 Esempio: valvola riduttrice di pressione ad azione diretta

Adatta per piccole portate, nella simulazione adotteremo una rampa da 0 a 50 l/min.



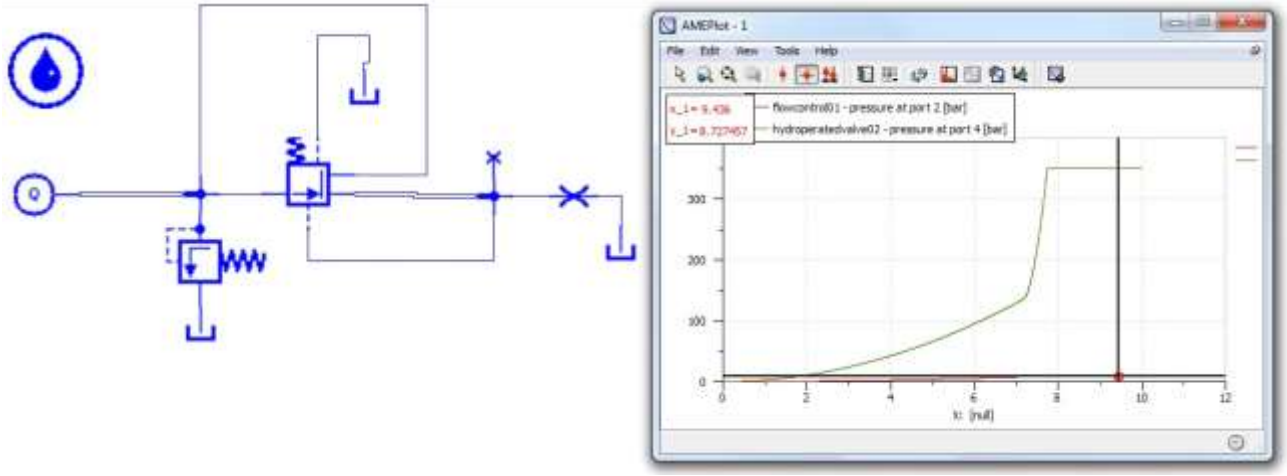

Adottiamo per la valvola $p_{crack}$ = 10 e $p_{max}$ = 2 e lasciamo tutto il resto di default.
Avviamo la simulazione…




La pressione sullo strozzatore è circa 8.7 bar.

### *5.11.3.1 Verifica con AMESim*

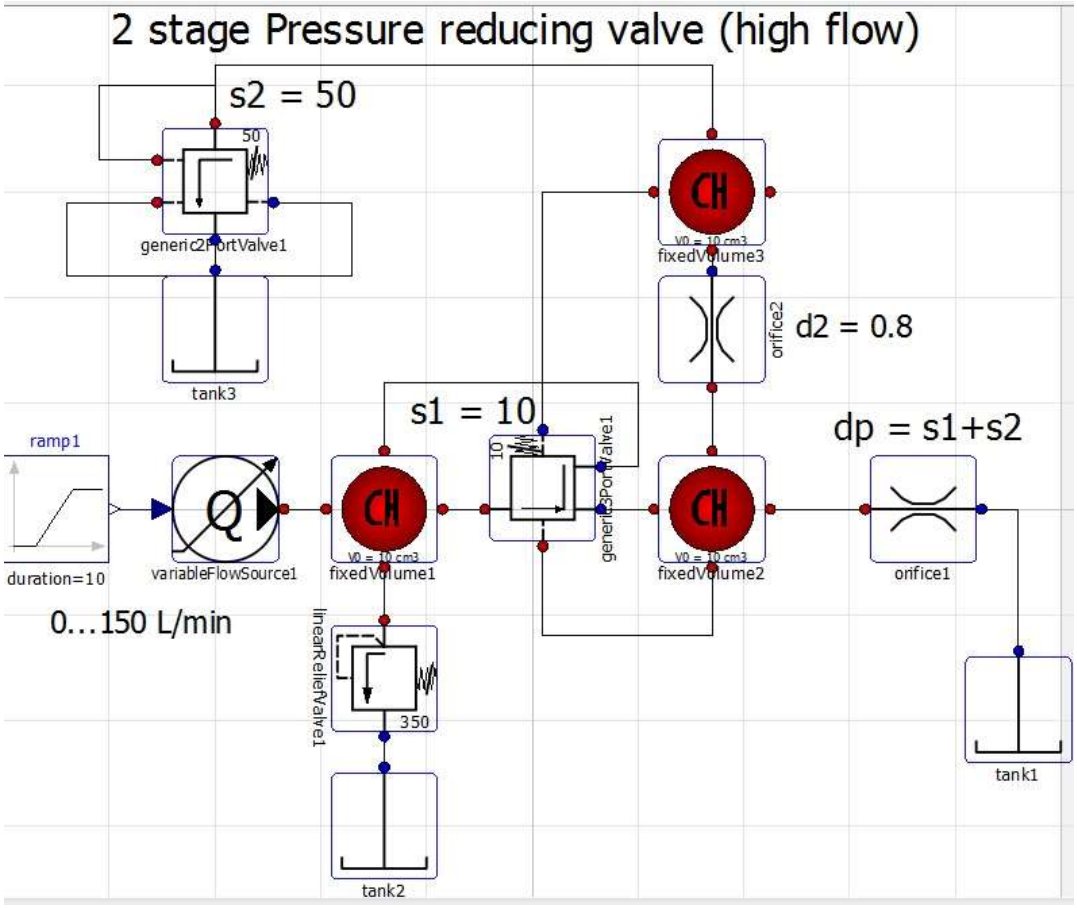
Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:



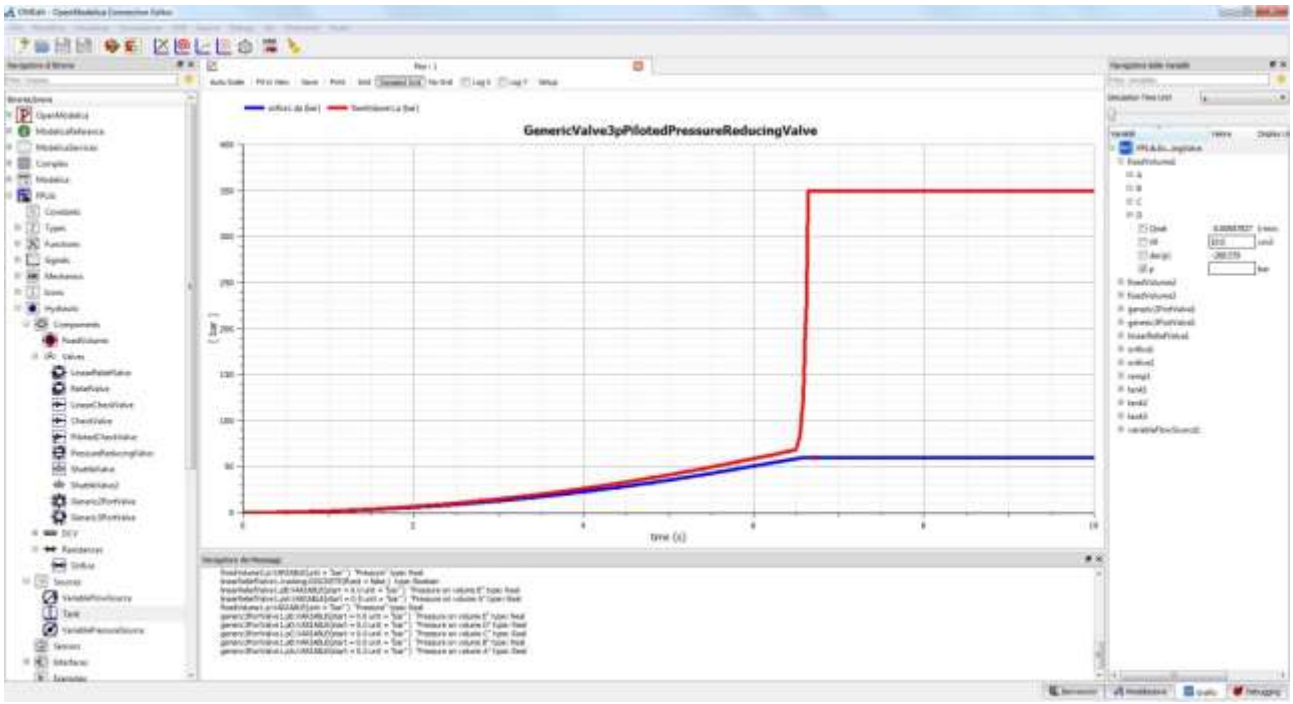
La pressione sullo strozzatore è circa 8.7 bar.

## 5.11.4 Esempio: valvola riduttrice di pressione ad azione pilotata

Intendiamo simulare il seguente schema, che impiega uno stadio principale formato da una valvola a 3 vie e uno pilota (lo stadio pilota è usato come una limitatrice di pressione) per realizzare una valvola riduttrice di pressione, a 2 vie, ad azione pilotata, adatta per grandi portate.



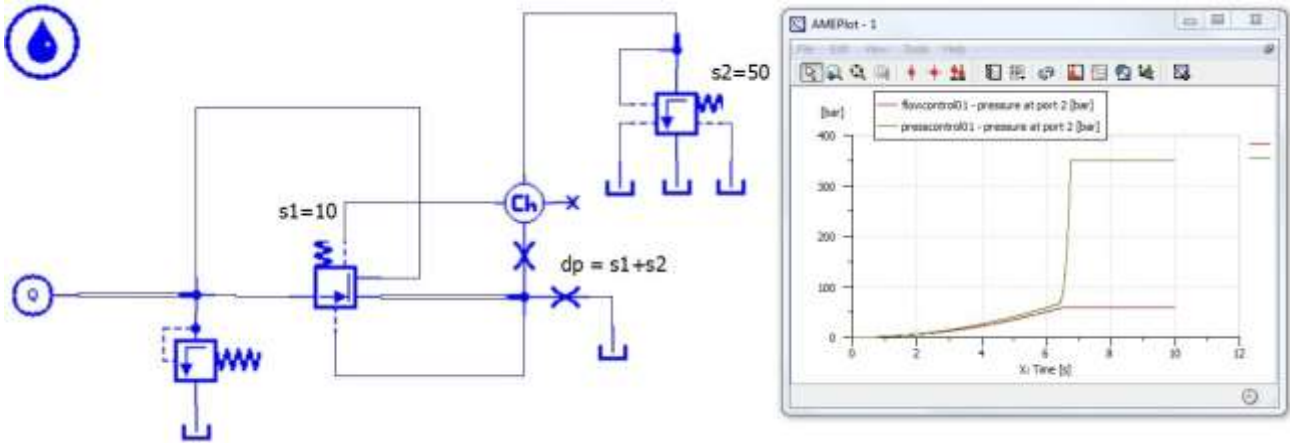
La rampa è ora da zero a 150, il primo stadio ha $p_{crack}$ = 10 e $p_{max}$ = 2, infatti ha una molla molto piccola. Inoltre ha $q_{nomA2B}$ = 100 e $p_{nomA2B}$ = 10, infatti è uno stadio di potenza e deve avere una grande apertura. Il secondo stadio ha $p_{crack}$ = 50, tutto il resto ha i valori di default.
Se lanciamo la simulazione otteniamo:




La pressione che questa valvola riduttrice regola, è la somma di $s_1$ + $s_2$, quindi in questo caso, 60 bar.

### *5.11.4.1 Verifica con AMESim*

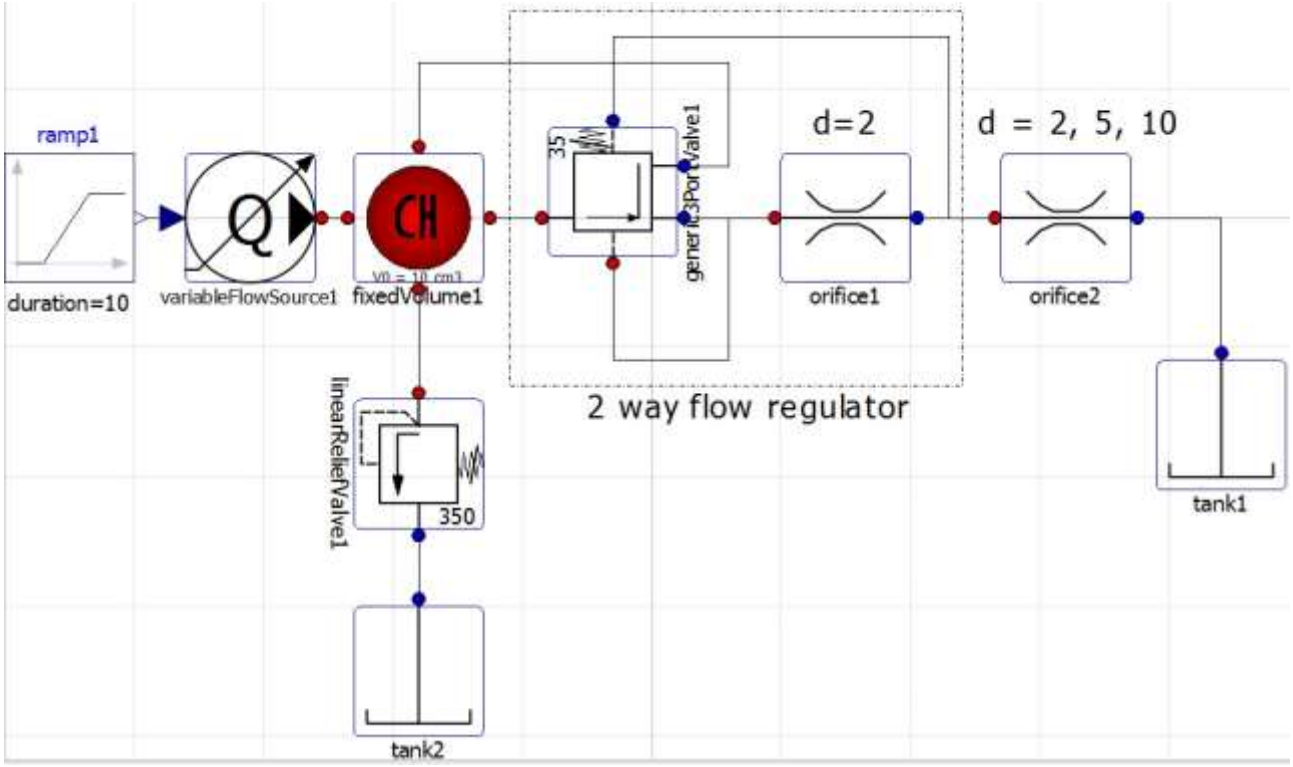
Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:



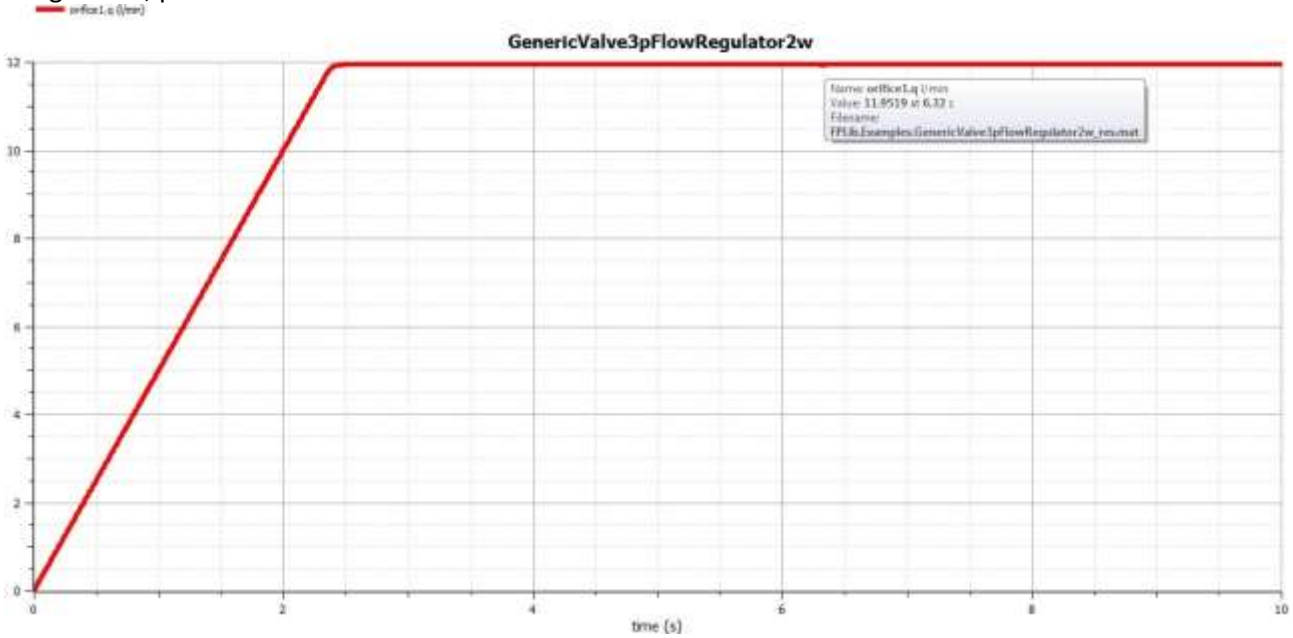

Anche AMESim ci dice che la pressione regolata è $s_1$ + $s_2$, e quindi circa 60 bar.

### 5.11.5 Esempio: valvola regolatrice di portata a 2 vie

Utilizziamo ora la valvola idraulica generica a 3 vie per realizzare un regolatore di flusso a 2 vie.




La rampa va da 0 a 50 per 10 s. La generic3PortValve1 ha $p_{crack}$ = 35 bar, $p_{max}$ = 6 bar e $q_{nomA2B}$ = 100 L/min perché in generale su queste valvole devono passare portate non trascurabili.
Poiché la valvola mantiene costante la pressione di 35 bar sullo strozzatore orifice1 (d = 2), evidentemente non la stiamo utilizzando per portate tanto grandi, quindi un altro scopo della simulazione è quello di meditare di utilizzare per quest'applicazione, una valvola di dimensioni nominali più piccole.
In ogni caso, procediamo con la simulazione…

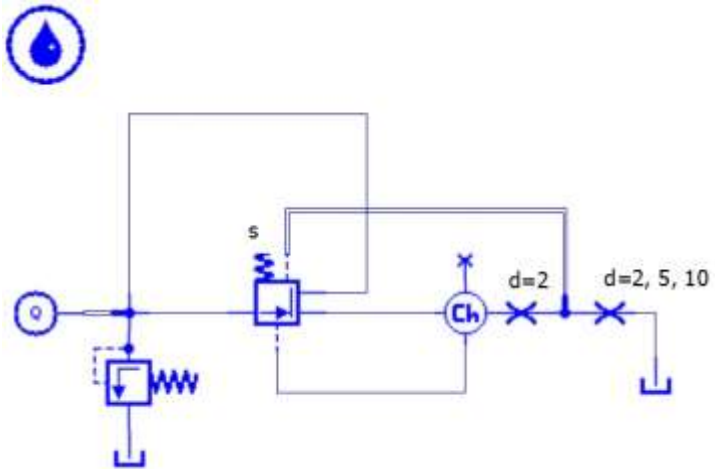


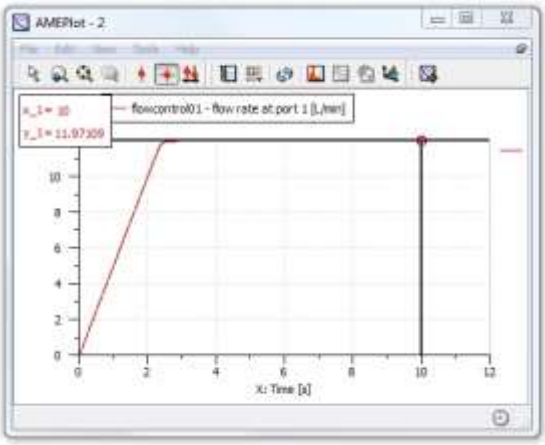

Per verificare che effettivamente il regolatore di portata sia compensato, ho provato a modificare il "carico", simulato dalla strozzatura orifice2, impostandolo a d = 2 mm, 5 mm e infine a 10 mm: la portata di 12 L/min si mantiene pressoché costante.

Ho anche provato a ripristinare $q_{nomA2B}$ al valore di defualt (= 10 L/min) per verificare che la portata si mantenga più o meno uguale in tutte le condizioni: effettivamente per regolare 12 L/min non serve una valvola con una luce di passaggio dimensionata per 100 L/min.

### *5.11.5.1 Verifica con AMESim*

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




La differenza di pressione sullo strozzatore di sinistra rimane costante, di conseguenza, la portata nello strozzatore di destra rimane costante, anche cambiando la dimensione dello strozzatore.
Con uno strozzatore d = 2 mm e s = 35 bar, la portata è costante a 12 L/min ed è indipendente dal carico (e quindi dal diametro dello strozzatore di destra).

# 6 Hydraulic Component Design

Questo titolo ha il nome della libreria più apprezzata di AMESim, per chi si occupa di oleodinamica.
Come per le altre sezioni presentate, non si pretende certamente di egualiare il celebre software di simulazione per completezza e semplicità d'uso, tuttavia, almeno fino ad ora credo che siamo riusciti a dimostrare che è possibile modellare blocchi abbastanza agevolmente per svolgere lavori analoghi a quello che si potrebbe fare con AMESim, seppure limitatamente a certi contesti.

## 6.1 Modelli parziali d'interfaccia

Sostanzialmente dobbiamo modellare blocchi con porte meccaniche e porte idrauliche.
Delle prime ce ne siamo occupati nel capitolo 4, delle altre al capitolo 5. Per poter cooperare con i blocchi già presentati è necessario che le porte siano le stesse.
Possiamo intanto creare una interfaccia parziale da usare per tutti i diversi blocchi costituiti da due porte meccaniche e due porte idrauliche con gli elementi di base in comue (la forza di Modelica è proprio il riuso del codice).

### 6.1.1 La classe parziale PartialHCD2PortsWithVolume



La classe dovrà contenere due FluidPort A e B (rossa e blu, 5.1.1) gestire due volumi, e deve estendere le due porte meccaniche ereditate da Partial2Ports (4.1.2) e PartialRelativeMovements (0).
Ecco il codice:

```
partial model PartialHCD2PortsWithVolume
  extends Interfaces.PartialHCDBase;
  extends Mechanics.Translational.Interfaces.Partial2Ports;
  extends Mechanics.Translational.Interfaces.PartialOrientedReferenceSystemConfirmation;
  extends Mechanics.Translational.Interfaces.PartialRelativeMovements;
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
   Placement(visible = true, transformation(origin = {-50, -106}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {-70, -110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  FPLib.Hydraulic.Interfaces.FluidPort_b B annotation(
   Placement(visible = true, transformation(origin = {38, -68}, extent = {{-10, -10}, {10, 10}}, rotation = 0),
iconTransformation(origin = {70, -110}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  // parametri e variabili generali
  parameter Types.Diameter dA = 100 "Inlet diameter at fluid port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Diameter dB = 100 "Inlet diameter at fluid port B" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdAmax = 0.7 "Max. Cd at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless CdBmax = 0.7 "Max. Cd at port B" annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecA = 1000 "critical Reynolds number" annotation(
    Dialog(tab = "Advanced"));
  parameter SI.ReynoldsNumber RecB = 1000 "critical Reynolds number" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolA0 = 10 "Initial volume at port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Volume VolB0 = 10 "Initial volume at port B" annotation(
    Dialog(tab = "Advanced"));
  Types.Pressure pA(start = 0) "Pressure on volume A";
  Types.Pressure pB(start = 0) "Pressure on volume B";
  Types.Pressure dp(start = 0) "DeltaP across the edge (dp = pA - pB)";
  Types.Volume VolA "Volume at port A";
  Types.Volume VolB "Volume at port B";
  Types.VolumeFlowRate q(start = 0) "Flow across the edge (from A to B)";
  Types.VolumeFlowRate qA(start = 0) "Flow at inlet port A";
  Types.VolumeFlowRate qB(start = 0) "Flow at inlet port B";
  SI.ReynoldsNumber ReA "Reynold number at port A";
  SI.ReynoldsNumber ReB "Reynold number at port B";
  SI.Velocity wA "fluid mean velocity at port A";
  SI.Velocity wB "fluid mean velocity at port B";
  Types.Dimensionless CdA "Discharge coefficient at port A";
  Types.Dimensionless CdB "Discharge coefficient at port B";
protected
```

```
  Types.Area AA = Constants.pi * dA ^ 2 / 4 "Inlet section at port A";
  Types.Area AB = Constants.pi * dB ^ 2 / 4 "Inlet section at port B";
  Types.Area VolSezA "Normal section for vol. A computation";
  Types.Area VolSezB "Normal section for vol. B computation";
equation
// hydraulic ports equations
  VolA = VolA0 + 0.001 * xa_rel * VolSezA * v_sgn;
  VolB = VolB0 - 0.001 * xb_rel * VolSezB * v_sgn;
  (qA, wA, ReA, CdA) = Functions.orif3f(A.p, pA, AA, dA, CdAmax, RecA);
  (qB, wB, ReB, CdB) = Functions.orif3f(pB, B.p, AB, dB, CdBmax, RecB);
  der(pA) = Functions.hCapacitance(qA, q, VolA, va * VolSezA * v_sgn, pA, Constants.bulk);
  der(pB) = Functions.hCapacitance(q, qB, VolB, -vb * VolSezB * v_sgn, pB, Constants.bulk);
  A.q = qA;
  B.q = -qB;
  dp = pA - pB;
  annotation(
    Icon(graphics = {Rectangle(origin = {0, 85}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-100, -5},
{100, 5}}), Rectangle(origin = {0, -83}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-100, -5}, {-80, 3}}),
Rectangle(origin = {180, -83}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-100, -5}, {-80, 3}}),
Rectangle(origin = {40, -83}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-100, -5}, {20, 3}}}));
end PartialHCD2PortsWithVolume;
```

### 6.1.2 La classe parziale PartialHCDEdge

Questa classe parziale aggiunge funzionalità ai blocchi della Hydraulic Component Design:

1. Coefficiente d'efflusso massimo per il tipo di geometria (***$C_q$*** effettivo è calcolato con la [5-5]).
2. Ricoprimento negativo (*underlap*) o positivo (*overlap*), vedi 6.6.2.
3. Fattore ***$k_{jet}$*** per il computo delle forze di flusso ($k_{jet}$ = 0 → calcoli condotti senza considerare le forze di flusso, $k_{jet}$ = 1 → calcoli condotti con le forze di flusso, 0 < $k_{jet}$ < 1 → i calcoli relativi a quel blocco considerano parzialmente le forze di flusso, l'entità della forza sarà proporzionale al fattore $k_{jet}$ rispetto al valore teorico delle forze di flusso).
4. Possibilità d'impostare una sezione massima (***MaxDownstreamArea***) per cui l'apertura della sezione d'efflusso dell'elemento mobile, durante la sua corsa, non potrà comunque superare.

In particolare, per il punto 2, consente di gestire il ricoprimento:

$$x_{lap} = x_{a,rel} \cdot v_{sgn} + x_u$$

6-1: metodo per il calcolo del ricoprimento

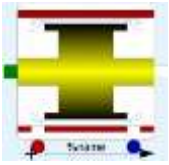
Dove ***$x_{a,rel}$*** è lo spostamento della parte mobile dall'inizio della simulazione, ***$x_u$*** è un offset relativo alla posizione zero e ***$x_{lap}$*** è il grado d'apertura.
Infatti, qualunque legge d'area prenderà come argomento $x_{lap}$ e non $x_{a,rel}$: questo chiaramente consente di traslare l'inizio dell'apertura della distanza del ricoprimento $x_u$ con facilità.
Ad esempio: spola con *underlap* di 2 mm ($x_u$ = 2), alla posizione di riposo ($x_{a,rel}$ = 0) abbiamo che $x_{lap}$ = 2, quindi l'area è calcolata con la parte mobile già aperta di $x_{lap}$; viceversa un *overlap* di 2 mm (xu = -2), determina in posizione di riposo $x_{lap}$ = -2 e quindi l'area è nulla e prima di aprire, si dovrà spostare di 2 mm (la direzione dipende dal flag **FromLeft2Right**, in particolare se è true, la x cresce in accordo con la freccia associata alla massa, viceversa la x cresce nell'altro senso. La variabile $v_{sgn}$ vale 1 se FromLeft2Right è true, altrimenti vale -1).
Ecco il codice:

```
partial model PartialHCDEdge
  extends Interfaces.PartialHCD2PortsWithVolume;
  // parametri e variabili generali
  parameter Types.Dimensionless Cdmax = 0.7 "Max. Cd at control edge";
  parameter SI.ReynoldsNumber Rec = 100 "critical Reynolds number at control edge";
  parameter Types.Position xu = 0 "Underlap at zero displacement, xu<0 = overlap, x=0=critical, x>0
= underlap";
  parameter Types.Dimensionless kjet = 1.0 "Flow force coefficient (0 = no flow force, 1 = max. flow
forces)";
  parameter Types.Area MaxDownstreamArea = 1e+30 "Max. downstream area" annotation(
    Dialog(tab = "Advanced"));
  Types.Position xlap "Edge opening / poppet lift once removed underlap/overlap";
  Types.Area Ae(start = 0) "Flow area (effective)";
  Types.Length Dh "Hydraulic diameter";
  SI.ReynoldsNumber Re "Reynold number across the edge";
  SI.Velocity w "fluid mean velocity across the edge";
  Types.Dimensionless Cd "Discharge coefficient across the edge";
```

```
  SI.Force Fflow "Flow forces";
  SI.Force Fh "hydrostatic force";
protected
  Types.Area Aefflux "Effective area used for computation";
equation
// mechanical ports equations
  flange_b.s = flange_a.s;
  flange_a.f + flange_b.f + (Fh - Fflow) * v_sgn = 0;
// xlap is x at the net of the underlap (xu = underlap at displacement = 0)
  xlap = xa_rel * v_sgn + xu;
// hydraulic ports equations
  Aefflux = if Ae < MaxDownstreamArea then Ae else MaxDownstreamArea;
  (q, w, Re, Cd) = Functions.orif3f(pA, pB, Aefflux, Dh, Cdmax, Rec);
end PartialHCDEdge;
```

Ora abbiamo tutto per poter iniziare la libreria dei blocchi HCD.

## 6.2 Leakage



E' fondamentale per simulare le fughe tra gli accoppiamenti cilindrici.
Consente di definire una lunghezza di ricoprimento e trattare il movimento come:
**Constant** (il ricoprimento è ritenuto costante anche se la spola si muove).
**Decrease** (se la spola si muove nella direzione della freccia, la lunghezza del ricoprimento decresce).
**Increase** (se la spola si muove nella direzione della freccia, la lunghezza del ricoprimento incrementa).

### 6.2.1 Parametri utilizzati nel modello

I principali parametri utilizzati nel modello sono:

- Diametro esterno della camicia (External bore diameter) – $d_{pe}$.
- Gioco diametrale (Clearance on diameter) – $d_c$.
- Lunghezza del contatto fisso (fixes contact length) - $lc_{fixed}$.
- Lunghezza del diametro a corsa 0 – $lc_0$.
- Eccentricità (Eccentricity between piston and sleeve) – ecc.

Si definisce $r_c$ come gioco radiale = $d_c$ / 2. L'eccentricità ecc deve essere minore o uguale a $r_c$.

### 6.2.2 Modello delle perdite per trafilamento

Un accoppiamento cilindrico, poiché usualmente il gioco radiale tra camicia e pistone è molto ridotto, può essere visto come due lastre parallele infinitamente estese.

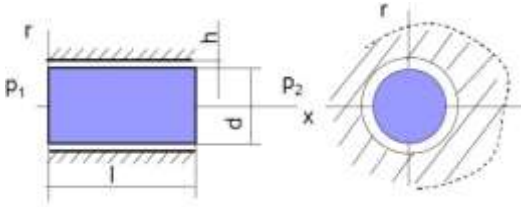



Figura 6-1: accoppiamento camicia-pistone

In realtà, con riferimento alla Figura 6-1, chiamiamo b l'estensione di tali lastre (larghezza) a patto che b sia molto maggiore di h, per cui $b = \pi \cdot d$ , ma poiché b >> h, allora possiamo utilizzare il modello delle lastre parallele, ad altezza costante, infinitamente estese.
Con riferimento ai nostri parametri d = $d_{pe}$ – $d_c$ e h = $r_c$.

### 6.2.2.1 In assenza di moto relativo tra le piastre

Partiamo dal problema di lastre parallele senza moto relativo tra esse.

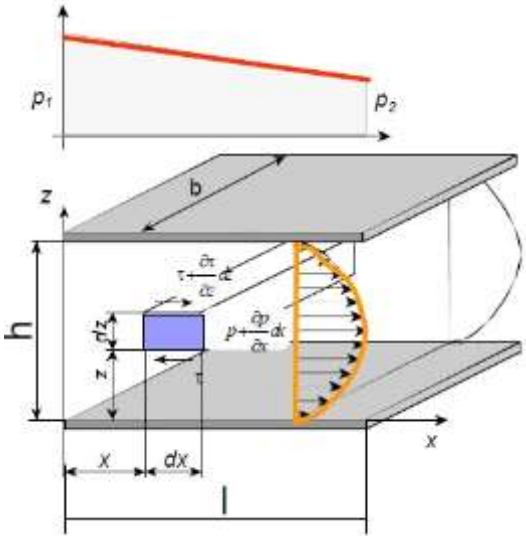



Figura 6-2: problema del flusso nelle lastre parallele infinitamente estese

Isoliamo un elemento di volume di larghezza b, altezza dz e lunghezza dx.
Alle facce normali a x agiscono la pressione p che spinge sulla faccia di sinistra, verso destra, e la pressione

$$p + \frac{\partial p}{\partial x} dx$$

Che agisce da destra, sulla superficie di destra e spinge l'elemento di fluido verso sinistra.
Sulle superfici inferiore e superiore agiscono le tensioni tangenziali T e

$$T + \frac{\partial T}{\partial z} dz$$

Quindi, applichiamo la seconda legge di Newton all'elemento di fluido:

$$p \cdot b \cdot dz - \left( p + \frac{\partial p}{\partial x} dx \right) \cdot b \cdot dz + T - \left( T + \frac{\partial T}{\partial z} dz \right) = 0 \rightarrow -\partial p \cdot b \cdot dz - \partial T = 0 \rightarrow b \cdot \partial p = -\frac{\partial T}{\partial z}$$

Poiché la forza tangenziale dovuta alla viscosità è:

$$T = -\mu \frac{du}{dz} \cdot b \cdot dx = -\tau \cdot b \cdot dx$$

6-2: forza d'attrito dovuta alla viscosità

Allora

$$\frac{\partial p}{\partial x} = \mu \cdot \frac{d^2 u}{dz^2}$$

Facciamo una doppia integrazione:

$$\frac{d^2 u}{d^2 z} = \frac{1}{\mu} \frac{\partial p}{\partial x}$$

Chiamo r =du/dz, quindi:

$$\frac{dr}{dz} = \frac{1}{\mu} \frac{\partial p}{\partial x} \rightarrow dr = \frac{1}{\mu} \frac{\partial p}{\partial x} dz \rightarrow r = \frac{1}{\mu} \frac{\partial p}{\partial x} z + c_1$$

Quindi

$$\frac{du}{dz} = \frac{1}{\mu} \frac{\partial p}{\partial x} z + c_1 \rightarrow du = \left( \frac{1}{\mu} \frac{\partial p}{\partial x} z + c_1 \right) dz \rightarrow u = \frac{1}{2\mu} \frac{\partial p}{\partial x} z^2 + c_1 z + c_2$$

6-3: moto entro piastre parallele, soluzionge generale

Se la piastra è ferma, allora le boundary conditions sono z = 0, u = 0 e z = h, u = 0.

Abbiamo così:

$$\begin{cases} c_2 = 0 \\ \frac{1}{2\mu}\frac{dp}{dx}h^2 + c_1 h = 0 \end{cases} \rightarrow \begin{cases} c_1 = -\frac{1}{2\mu}\frac{dp}{dx}h \\ c_2 = 0 \end{cases}$$

Quindi:

$$u(z) = \frac{1}{2\mu}\frac{dp}{dx}z^2 + c_1 z + c_2 = \frac{1}{2\mu}\frac{dp}{dx}z^2 - \frac{1}{2\mu}\frac{dp}{dx}hz = \frac{z^2 - hz}{2\mu}\frac{dp}{dx}$$

6-4: velocità del fluido nel moto tra due lastre parallele

La portata è data da:

$$Q = b \cdot \int_0^h u(z) \cdot dz = -\frac{1}{12\mu} \cdot b \cdot h^3 \cdot \frac{\partial p}{\partial x}$$

6-5: portata che scorre tra lastre parallele senza moto relativo

Il gradiente di pressione, se le lastre sono perfettamente parallele è lineare:




Quindi:

$$\frac{\partial p}{\partial x} = \frac{\Delta P}{l} = -\frac{p_1 - p_2}{l}$$

6-6: gradiente di pressione in lastre parallele ad altezza costante

e

$$p(x) = \frac{p_2 - p_1}{l} \cdot x + p_1$$

6-7: andamento della pressione tra lastre parallele al altezza costante

La forza d'attrito viscoso è:

$$F_v = \tau \cdot b \cdot l$$

Ma $\tau = \mu\frac{du}{dz} = \frac{1}{2}(2z - h)\frac{\partial p}{\partial x}$, quindi $\tau(0) = -\frac{h}{2}\frac{\partial p}{\partial x}$ e $\tau(h) = \frac{h}{2}\frac{\partial p}{\partial x}$.

Se immaginiamo z = 0 la camicia (ferma) e z = h il pistone che si muove, abbiamo:

$$F_v = \tau \cdot b \cdot l = \frac{h}{2}\frac{\partial p}{\partial x}bl = \Delta P\frac{h}{2}b = (p_2 - p_1) \cdot \pi \cdot (d_{pe} - d_c) \cdot \frac{r_c}{2}$$

6-8: forza d'attrito viscoso nel moto tra due piastre parallele senza moto relativo

### 6.2.2.2 *In presenza di moto relativo tra le piastre*

Consideriamo il fatto ora la piastra superiore (z = h) si muova a velocità $v_0$.

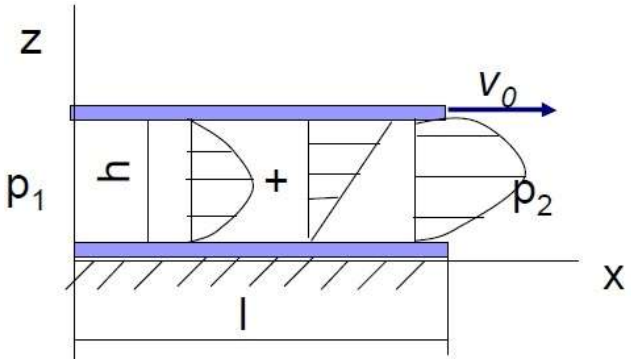



Figura 6-3: problema del flusso nelle lastre parallele infinitamente estese di cui una in moto relativo rispetto all'altra

Prendiamo la [6-3] e applichiamo le boundary conditions z = 0, v = 0 e z = h, v = $v_0$.

$$\begin{cases} c_1 = -\dfrac{1}{2\mu}\dfrac{\partial p}{\partial x}h + \dfrac{v_0}{h} \\ c_2 = 0 \end{cases}$$

Da cui:

$$u(z) = \frac{1}{2\mu}\frac{\partial p}{\partial x}\left(z^2 - hz\right) + \frac{v_0}{h}z$$

**6-9: velocità del fluido nel moto tra due lastre parallele, di cui una in moto relativo**

La portata risulta:

$$Q = b \cdot \int_0^h u(z) \cdot dz = -\frac{1}{12\mu} \cdot b \cdot h^3 \cdot \frac{\partial p}{\partial x} + v_0 \cdot \frac{h}{2} \cdot b$$

**6-10: portata che scorre tra lastre parallele di cui una in moto relativo**

Il coefficiente d'attrito viscoso è:

$$\tau = \mu\frac{du}{dz} = \frac{\partial p}{\partial x}\left(z - \frac{h}{2}\right) + \mu\frac{v_0}{h}$$

Quindi

$$\tau(h) = \frac{h}{2}\frac{\partial p}{\partial x} + \mu\frac{v_0}{h}$$

e

$$F_v = \tau \cdot b \cdot l = \left(\frac{h}{2}\frac{\partial p}{\partial x} + \mu\frac{v_0}{h}\right)bl = \left(p_2 - p_1\right) \cdot \pi \cdot \left(d_{pe} - d_c\right) \cdot \frac{r_c}{2} + \mu \cdot l \cdot v_0 \cdot \frac{\pi \cdot \left(d_{pe} - d_c\right)}{r_c}$$

**6-11: forza d'attrito viscoso nel moto tra due piastre parallele di cui una in moto relativo**

Si noti che se $v_0$ = 0, allora la [6-5] e la [6-8] sono casi particolari della [6-10] e della [6-11].

### 6.2.3 Codice

Ecco il codice:

```
model Leakage
  extends Interfaces.PartialHCD2PortsWithVolume;
  // Poppet parameters and variables
  type behavior = enumeration(Constant, Decrease, Increase);
  parameter Types.Diameter dpe = 20 "External bore diameter";
  parameter Types.Length dc = 0.01 "Clearance on diameter";
  parameter Types.Length lcfixed = 30 "Fixed contact length";
  parameter Types.Length lc0 = 30 "Contact length at zero displacement";
  parameter Types.Length maxctl = 1e+30 "Maximum contact length";
  parameter Types.Length ecc = 0 "Eccentricity between piston and sleeve";
  parameter behavior CL(start = behavior.Constant, fixed = true) "Constant = contact length (c.l.)
is fixed (default), Decrease = c.l. decrease as xlap increase, Increase = c.l. increase as xlap
increase";
  Types.Length lc "Contact length";
  SI.Force Fvisc "Viscous force";
  Types.Length rc;
  Types.DampingConstant c "Damping constant";
equation
// mechanical ports equations
  flange_b.s = flange_a.s;
  flange_a.f + flange_b.f - Fvisc * v_sgn = 0;
  VolSezA = 0;
  VolSezB = 0;
// flow equations
  if CL == behavior.Constant then
    lc = lcfixed;
  elseif CL == behavior.Decrease then
    lc = lc0 - xa_rel * v_sgn;
  else
    lc = lc0 + xa_rel * v_sgn;
```

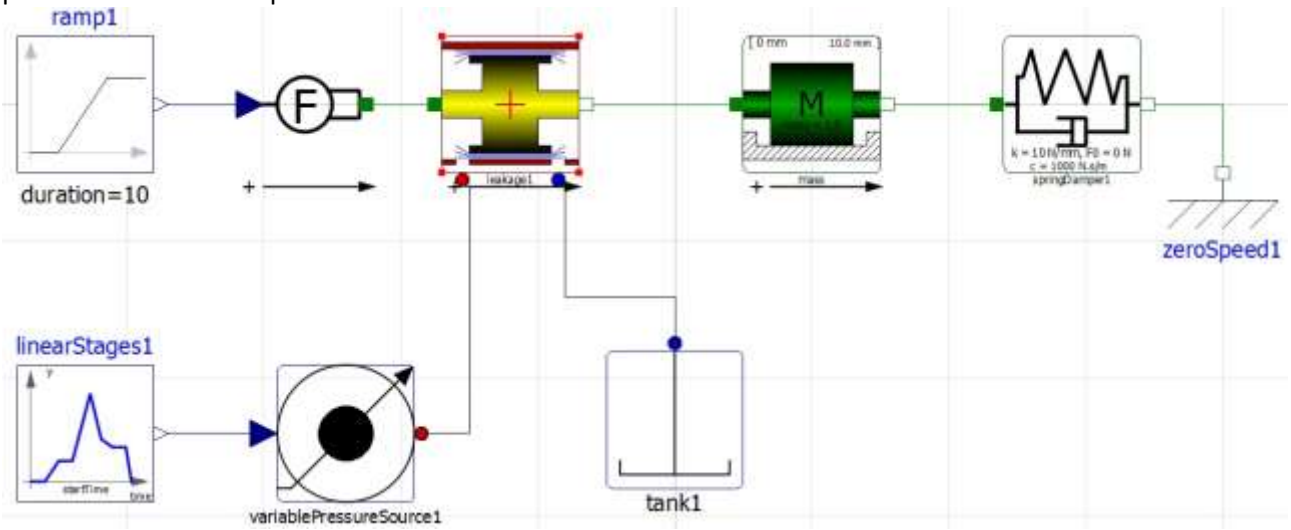
```
  end if;
  rc = dc / 2;
  q = Constants.pi * dpe * rc * (3 * lc * Constants.mu * va * v_sgn + 25 * dp * (2 * rc ^ 2 + 3 *
ecc ^ 2)) / (100 * lc * Constants.mu);
  Fvisc = -50.0 * Constants.pi * dp * 0.001*(dpe - dc) * rc + Constants.pi * (0.001 * dpe - 0.001 *
dc) * lc * Constants.mu * (+va) * v_sgn / rc;
  annotation(
    Icon(graphics = {Polygon(fillColor = {255, 255, 0}, fillPattern = FillPattern.HorizontalCylinder, points = {{-100, 20}, {-40,
20}, {-40, 60}, {-60, 60}, {-60, 70}, {60, 70}, {60, 60}, {40, 60}, {40, 20}, {100, 20}, {100, -20}, {40, -20}, {40, -60}, {60, -60},
{60, -70}, {-60, -70}, {-60, -60}, {-40, -60}, {-40, -20}, {-100, -20}, {-100, 20}}), Line(origin = {7, 78}, points = {{-87, 0}, {73,
0}}, color = {0, 0, 255}), Line(origin = {1.31, 71.6}, points = {{-81.312, -0.87741}, {-61.312, 3.12259}, {58.688, 3.12259}, {78.688,
-0.87741}}, color = {0, 0, 255}), Line(origin = {1, 64.81}, points = {{-81.0043, -4.80973}, {-61.0043, 7.19027}, {58.9957, 7.19027},
{78.9957, -4.80973}}, color = {0, 0, 255}), Line(origin = {0, -78}, points = {{-80, 0}, {80, 0}}, color = {0, 0, 255}), Line(origin =
{1.31, -70.89}, points = {{-81.3114, 1.85007}, {-61.3114, -4.14993}, {58.6886, -4.14993}, {78.6886, 1.85007}}, color = {0, 0, 255}),
Line(origin = {1, -65.81}, points = {{-81, 5.81401}, {-61, -6.18599}, {61, -6.18599}, {79, 5.81401}}, color = {0, 0, 255})},
coordinateSystem(initialScale = 0.1)));
end Leakage;
```

### 6.2.4 Verifica (confronto con AMESim)

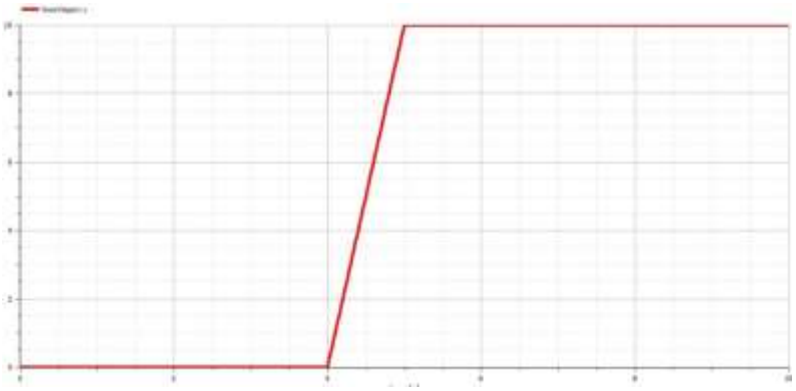
Potremmo come esercizio realizzare un caso e calcolare i valori utilizzando le formule [6-5, 6-8, 6-10 e 6-11] ma abbiamo già rubato tanto spazio alla teoria per questo esempio. Dando per scontata la correttezza di AMESim, ci inventiamo uno schema e lo confrontiamo sia con la FPLib che con la HCD di AMESim.

#### *6.2.4.1 Schema con blocco Leakage di FPLib*

Un pistone ha un gioco diametrale di $d_c$ = 0.01 mm e s'impegna in una camicia che ha $d_{pe}$ = 20 mm di diametro. E' accoppiato per una lunghezza di 30 mm che **decresce** con la corsa.
Un meccanismo fa compiere la corsa a velocità circa costante di 1 mm/s. Un sistema idralico mantiene la differenza di pressione tra le due camere 0 bar per 4 secondi, quindi in un secondo sale a 10 bar e quindi la pressione si mantiene per altri 5 secondi.

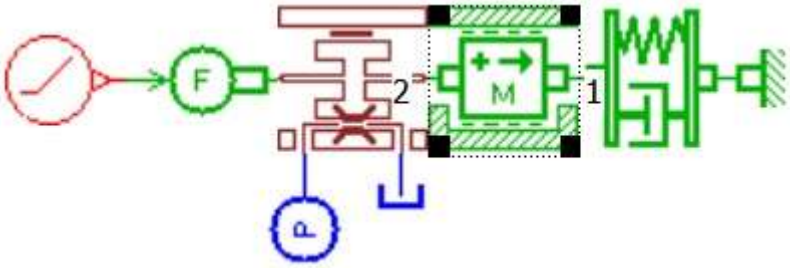



I parametri da modificare sono (per gli altri lasciare i valori di default):
Rampa: da 0 a 100 in 10s.
leakage: CL = FPLib.HCD.Components.Leakage.behavior.Decrease
mass: x1 = 0
springDamper: k = 10.
LinearStages = {0, 0, 4, 0, 10, 1, 10, 10, 5} (significa da 0 a 0 per 4s, da 0 a 10 in 1 s, da 10 a 10 in 5 s).
In pratica la pressione che variablePressureSource1 fornirà, avrà questo andamento temporale:



### 6.2.4.2 Schema con blocco Leakage di AMESim HCD

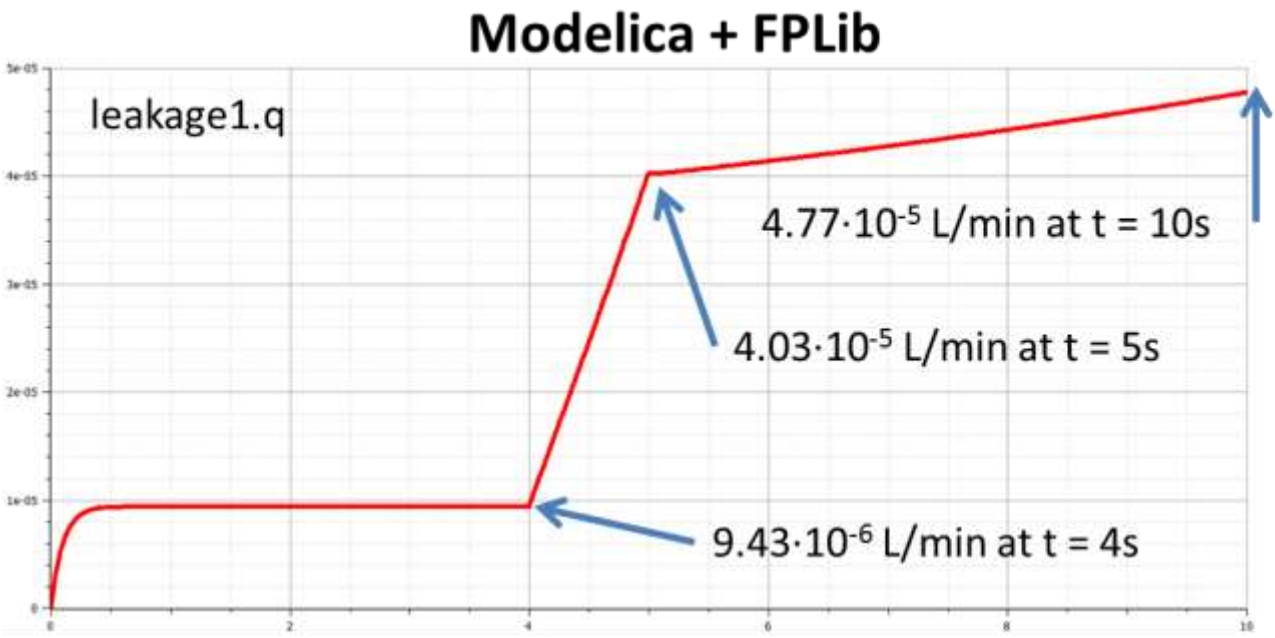
Ed ora lo stesso modello ma con AMESim

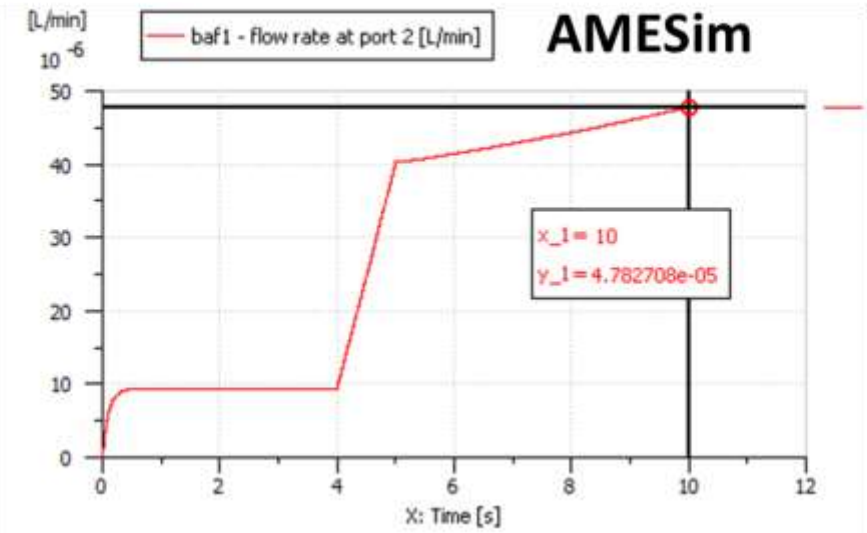



Sono stati impostati i parametri allo stesso modo.

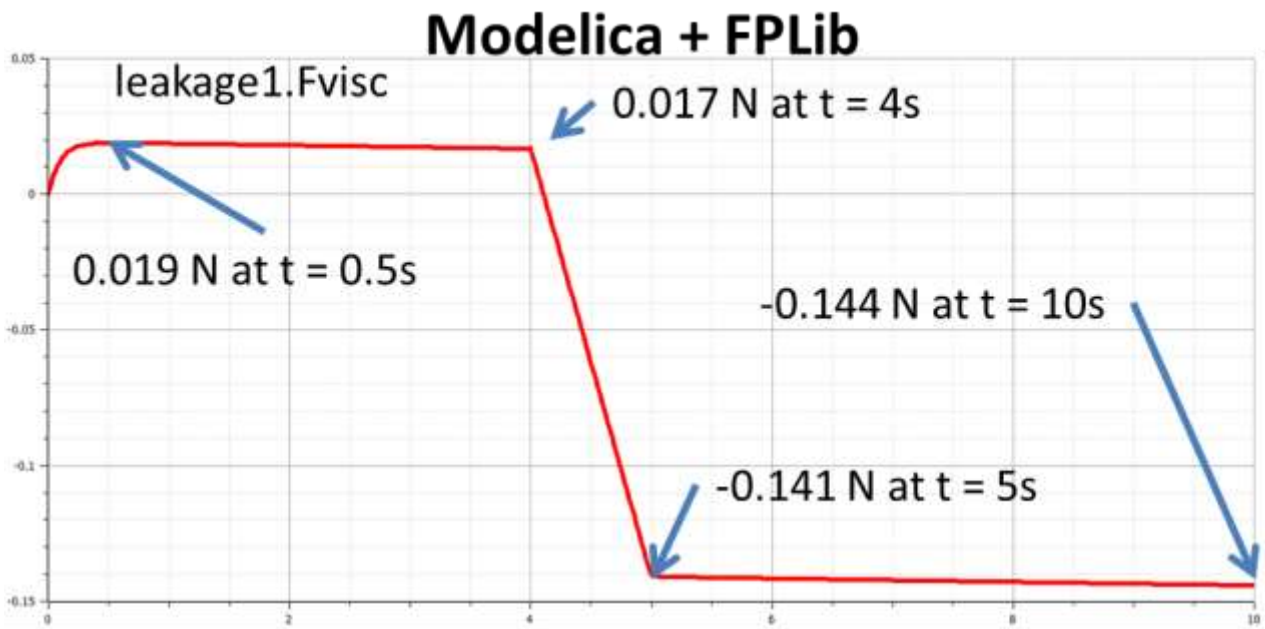
### 6.2.4.3 Il confronto

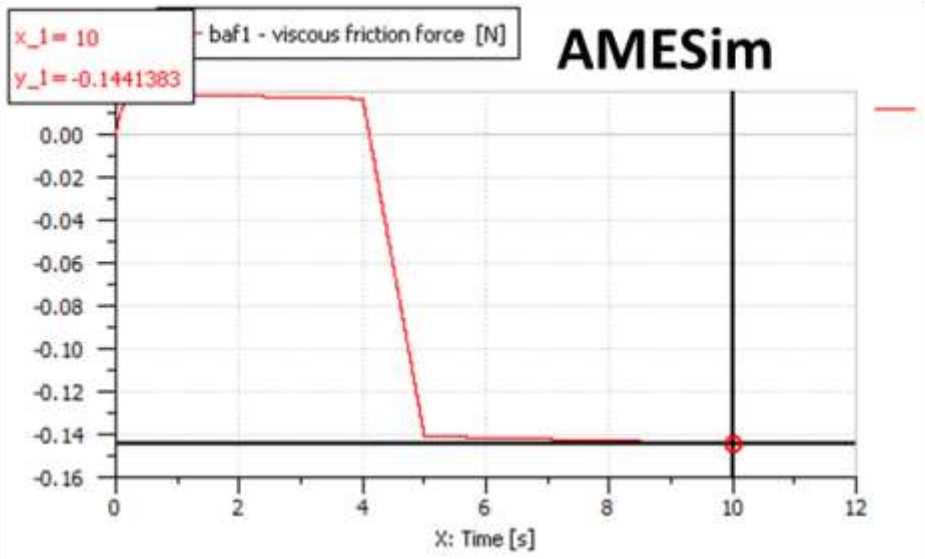
La portata di trafilamento (leakage).



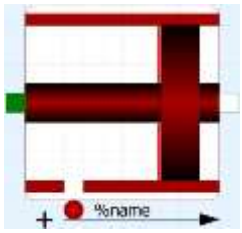

Ed ora la forza d'attrito viscosa




Possiamo dire che AMESim ha verificato il modello FPLib.

## 6.3 Pistone




E' il classico pistone delle valvole, utilizzato per trasformare la pressione in una forza che spinge nella direzione della freccia.
Se la freccia è concorde con il senso della massa, lasciare il flag FromLeft2Right = true.

### 6.3.1 Principali parametri e breve descrizione del modello

I principali parametri sono:

1. Diametro del pistone – $d_p$.
2. Diametro del perno – $d_r$.
3. Volume nella posizione iniziale – $V_0$.

Con questi si determina l'area attiva:

$$A_0 = \frac{\pi}{4}\left(d_p^{\,2} - d_r^{\,2}\right)$$

La forza che il pistone esercita è:

$$F_{hyd} = p \cdot A_0$$

Il volume dentro alla camera del pistone è:

$$V = V_0 + x_{rel} \cdot A_0 \cdot v_{sgn}$$

Dove $x_{rel}$ è una corsa relativa, rispetto alla posizione di partenza, e $v_{sgn}$ è -1 se FromLeft2Right = false, altrimenti è 1.
La camera del pistone è visto come un volume V che scambia olio con il sistema attraverso la porta idraulica A, quindi, per la porta A vale la [5-2] e per il volume vale la:

$$Q_{in} - Q_{out} = \frac{dV}{dt} + \frac{V}{\beta}\frac{dp}{dt}$$

**6-12: legge della pressione in un volume**

Dove $Q_{out} = 0$, $Q_{in}$ è la portata scambiata attraverso la porta A, V è il volume, β è il bulk modulus e p è la pressione.

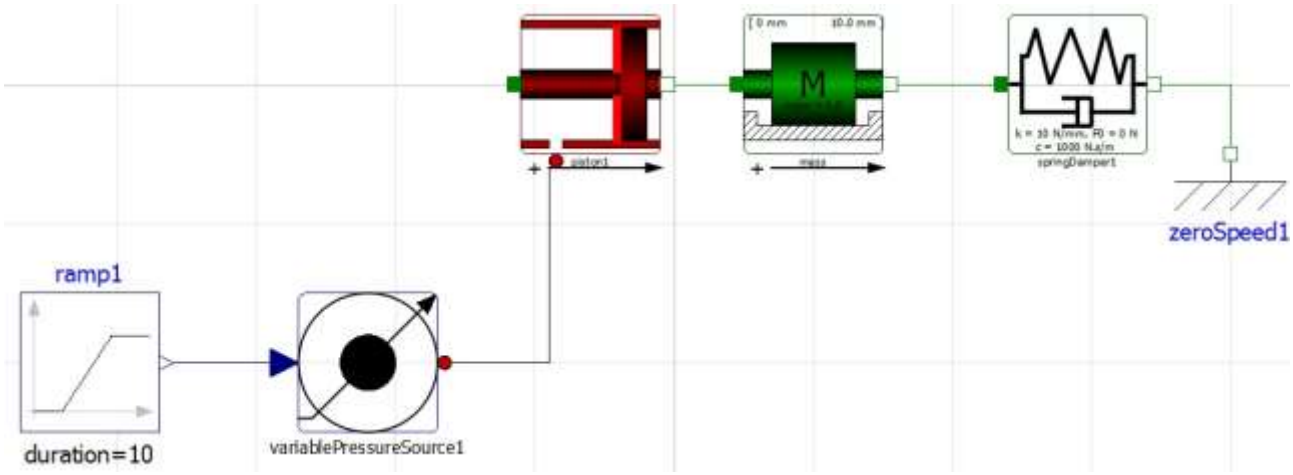
### 6.3.2 Codice

Ecco il codice:

```
model Piston
  extends Interfaces.PartialHCDBase;
  extends Mechanics.Translational.Interfaces.Partial2Ports;
  extends Mechanics.Translational.Interfaces.PartialOrientedReferenceSystemConfirmation;
  extends Mechanics.Translational.Interfaces.PartialRelativeMovements;
  FPLib.Hydraulic.Interfaces.FluidPort_a A annotation(
    Placement(visible = true, transformation(origin = {-134, -94}, extent = {{-10, -10}, {10, 10}},
rotation = 0), iconTransformation(origin = {-50, -110}, extent = {{-10, -10}, {10, 10}}, rotation =
0)));
  parameter Types.Diameter dp = 10 "Piston diameter";
  parameter Types.Diameter dr = 5 "Rod diameter";
  parameter Types.Diameter da = 100 "Inlet diameter at fluid port A" annotation(
    Dialog(tab = "Advanced"));
  parameter Types.Dimensionless Cd = 0.7 "discharge coefficient";
  parameter SI.ReynoldsNumber Rec = 1000 "critical Reynolds number";
  parameter Types.Volume V0 = 10 "Volume at initial position";
  Types.Pressure p(start = 0) "Pressure in the piston chamber";
  Types.Volume V(start = V0) "Volume in the piston chamber";
  Real Vdot(unit = "cm3/s", start = 0) "Variation of volume over time [cm3/s]";
  Types.Area A0 = Constants.pi * (dp ^ 2 - dr ^ 2) / 4 "Active area";
  Types.VolumeFlowRate qa(start = 0) "Inlet flow to the piston chamber";
```

```
  SI.ReynoldsNumber Re(start = 0) "Reynolds number";
  SI.Velocity wa(start = 0) "mean fluid velocity";
  Types.Dimensionless Cq(start = 0);
  Types.Area Aa = Constants.pi * da ^ 2 / 4 "Throttled area at the inlet of the piston volume";
equation
  V = V0 + 0.001 * xa_rel * A0 * v_sgn;
  Vdot = va * A0 * v_sgn;
// 1 m/s* 1 mm^2 = 1 cm3/s;
  assert(V > 0.0, "Volume must be positive (probably V0 is too low or orientation isn't good).",
AssertionLevel.error);
  (qa, wa, Re, Cq) = Functions.orif3f(A.p, p, Aa, da, Cd, Rec);
  A.q = qa;
  der(p) = Functions.hCapacitance(qa, 0, V, Vdot, p, Constants.bulk);
  flange_a.f + flange_b.f + 0.1 * p * A0 * v_sgn = 0;
  flange_b.s = flange_a.s;
  annotation(
    Icon(graphics = {Rectangle(origin = {30, -86}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-70, 6}, {70,
-6}}), Rectangle(origin = {-30, -86}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-70, 6}, {-30, -6}}),
Rectangle(origin = {30, 86}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-130, 6}, {70, -6}}),
Rectangle(origin = {60, 0}, fillColor = {170, 0, 0}, fillPattern = FillPattern.HorizontalCylinder, lineThickness = 0.5, extent = {{-
20, 80}, {20, -80}}), Rectangle(origin = {100, 0}, fillColor = {170, 0, 0}, fillPattern = FillPattern.HorizontalCylinder,
lineThickness = 0.5, extent = {{-20, 20}, {0, -20}}), Rectangle(origin = {-80, 0}, fillColor = {170, 0, 0}, fillPattern =
FillPattern.HorizontalCylinder, lineThickness = 0.5, extent = {{-20, 20}, {120, -20}}), Line(origin = {37.92, 48.08}, points = {{0,
30}, {0, -26}}, color = {255, 0, 0}, thickness = 2), Line(origin = {37.99, -52.21}, points = {{0, 30}, {0, -26}}, color = {255, 0,
0}, thickness = 2)}, coordinateSystem(initialScale = 0.1)));
end Piston;
```

### 6.3.3 Esempio: pistone utilizzato per muovere una massa




La rampa va da zero a 17 in 10 s. Il pistone è il componente di default (dp = 10 mm, dr = 5 mm).
La massa e lo springdamper sono gli stessi dell'esempio 6.2.4.1.
Per verificarlo, poiché costituisce una equazione lineare, possiamo utilizzare le trasformate di Laplace.

$$m\ddot{x} = p \cdot A_0 - c \cdot \dot{x} - kx \rightarrow ms^2 X(s) = P(s) \cdot A_0 - csX(s) - kX(s)$$

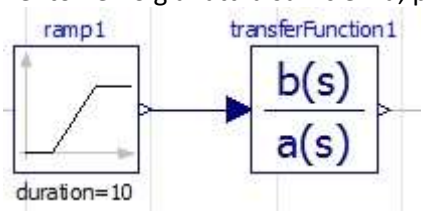
Da cui:

$$G(s) = \frac{X(s)}{P(s)} = \frac{A_0}{ms^2 + cs + k}$$

Ora abbiamo due strade, poiché p(t) = 1.7·t → P(s) = 1.7·$s^{-2}$ dobbiamo antitrasformare:

$$X(s) = P(s) \cdot G(s) = 1.7s^{-2} \cdot \frac{A_0}{ms^2 + cs + k}$$

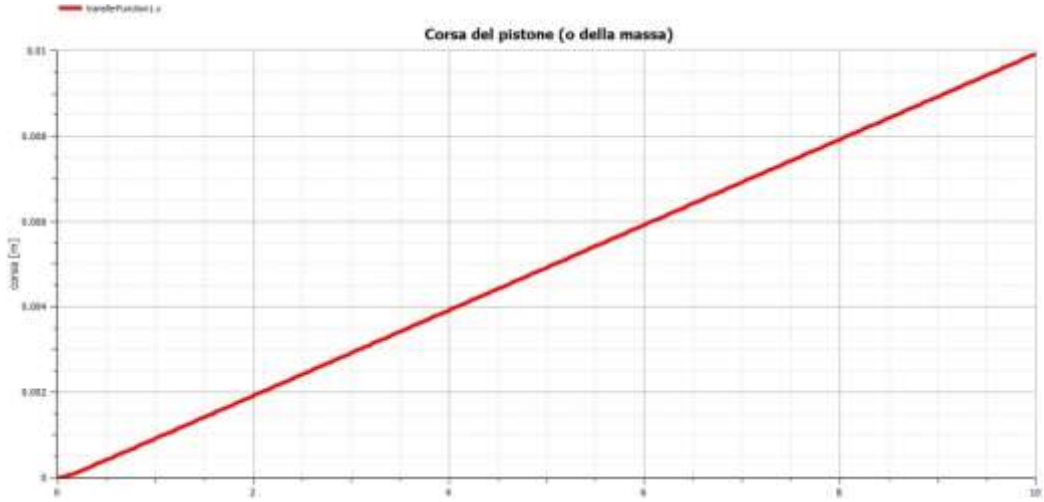
Oppure possiamo crare un modello con la rampa e un blocco Transfert funcion per far risolvere a Modelica l'equazione. Direi che di conti al momento ne ho già fatti a sufficienza, procediamo con il secondo metodo.




Ecco il codice, che possiamo mettere in un nuovo modello e chiamarlo ***temp***:

```
model temp
  parameter Modelica.SIunits.Area A0 = 5.8904862254808614e-5 "area";
  parameter Modelica.SIunits.Mass m = 1 "mass";
  parameter Modelica.SIunits.TranslationalSpringConstant k=10e+3 "spring rate";
  parameter Modelica.SIunits.TranslationalDampingConstant c=1000.0 "damping";
  Modelica.Blocks.Sources.Ramp ramp1(duration = 10, height = 17e+5)  annotation(
    Placement(visible = true, transformation(origin = {-72, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  Modelica.Blocks.Continuous.TransferFunction transferFunction1(a = {m, c, k}, b = {A0})
annotation(
    Placement(visible = true, transformation(origin = {-34, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
equation
  connect(ramp1.y, transferFunction1.u) annotation(
    Line(points = {{-61, 0}, {-46, 0}}, color = {0, 0, 127}));
  annotation(
    uses(Modelica(version = "3.2.2")));
end temp;
```

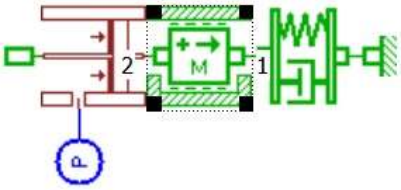
Ecco il risultato:



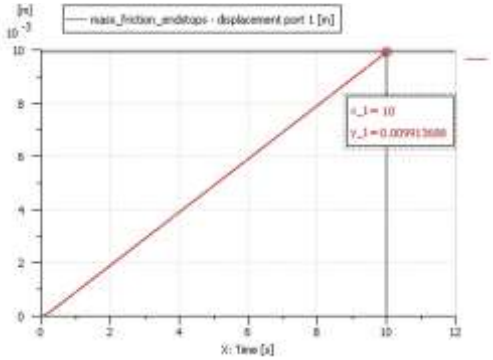

In pratica a 10 s la corsa arriva a 0.00991369 m, quindi circa 9.914 mm.

### 6.3.3.1 *Verifica con AMESim*

Ma noi non ci fidiamo, vogliamo vedere anche cosa dice AMESim (ovviamente con gli stessi parametri).

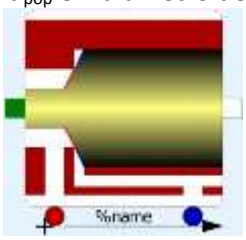



Ed ecco il grafico:



## 6.4 Otturatore conico (Poppet)

E' un elemento mobile a forma di cono, a tenuta su uno spigolo vivo (sede) ricavato tra due fori di diverso diametro, che si apre muovendosi per effetto della pressione o delle forze presenti alle porte meccaniche. Si distinguono il diametro della tenuta $d_s$ (il diametro del foro più piccolo, sul quale il cono appoggia quando la valvola non è in regolazione) e il diametro più grande dove il cono scorre $d_{pop}$ (può essere presente anche un gioco rilevante nelle costruzioni reali, in questo caso si può considerare il diametro cilindrico alla fine del cono oppure il diametro del foro: la scelta è facoltativa; la cosa importante è essere coerenti, specialmente nel caso in cui la contropressione agisca anche in chiusura, come vedremo nell'esempio 0, tramite un pistone. In questo caso il diametro del cono $d_{pop}$ e il diametro del pistone devono essere identici).




### 6.4.1 Principali parametri

1. Diametro del diametro massimo del cono (poppet) o del cilindro dove il cono finisce – $d_{pop}$.
2. Diametro della sede in cui il cono realizza la tenuta idraulica – $d_s$.
3. Diametro del perno (che impedisce che la superficie sia bagnata dall'olio) – $d_r$.
4. Semi-angolo del cono – alpha.
5. Coefficiente per il calcolo delle forze di flusso – $k_{jet}$ (default = 1)[3] ← Partial HCDEdge (6.1.2).

Non discutiamo il modello perché è piuttosto elementare, in pratica, i calcoli sulla sezione d'apertura da parte del cono sono sostanzialmente geometrici, le uniche due formule d'interesse sono la forza idraulica che agisce in apertura:

$$F_{hyd} = p_A \cdot A_{p,A} + p_B \cdot A_{p,B} = p_A \cdot \frac{\pi}{4} \cdot \left( {d_a}^2 - {d_r}^2 \right) + p_B \cdot \frac{\pi}{4} \cdot \left( {d_{pop}}^2 - {d_a}^2 \right)$$

**6-13: forza in apertura di un otturatore conico (poppet)**

E le forze di flusso:

$$F_{flow} = 2 \cdot k_{jet} \cdot C_d \cdot A_e \cdot \Delta P \cdot \cos\alpha$$

**6-14: forze di flusso su un otturatore conico (poppet)**

### 6.4.2 Codice

Ecco il codice:

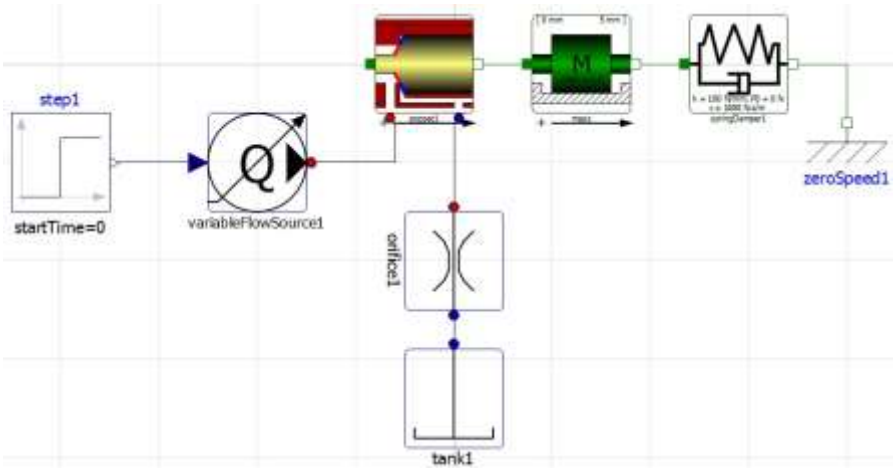
```
model Poppet
  extends FPLib.HCD.Interfaces.PartialHCDEdge(Cdmax = 0.7, Rec = 100, xu = 0);
  // Poppet parameters and variables
  parameter Types.Diameter dpop = 15 "Diameter of poppet";
  parameter Types.Diameter ds = 10 "Diameter of hole (seat)";
  parameter Types.Diameter dr = 5 "Diameter of rod";
  parameter Types.Angle alpha = 45 "Poppet half angle";
  Types.Diameter da "Active diameter of the poppet";
  Types.Area Apoppet(start = 0) "Area of the poppet";
protected
  Types.Area Amax = Constants.pi * (ds ^ 2 - dr ^ 2) / 4 "maximum area";
  SI.Angle alphar = alpha * Constants.pi / 180 "alpha in radiants";
  Types.Area ApA "Active section side A";
  Types.Area ApB "Active section side B";
equation
```

[3] Il calcolo delle forze di flusso è soltanto una stima. Un'analisi molto più realistica andrebbe fatta mediante un'analisi CFD e un successivo aggiustamente del fattore kjet. Senza quest'analisi, ci si può rendere conto della situazione, senza forze di flusso ($k_{jet}$ = 0) o con le più alte forze di flusso possibili ($k_{jet}$ = 1), oppure con qualunque valore intermedio da zero a 1. Non c'è alcun controllo su $k_{jet}$, quindi nulla vieta di mettere anche valori maggiori di uno.

```
// sections increase / decrease volumes (side A and side B)
  VolSezA = Constants.pi * (ds ^ 2 - dr ^ 2) / 4;
  VolSezB = Constants.pi * (dpop ^ 2 - dr ^ 2) / 4;
// active sections on which the pressures act (side A and side B)
// main outflow section
  assert(Apoppet >= 0, "Apoppet minore di zero, ocio!");
  if noEvent(xlap > 0) then
    Apoppet = Constants.pi * xlap * sin(alphar) * (ds - xlap * sin(alphar) * cos(alphar));
    if Apoppet <= Amax then
      Ae = Apoppet;
    else
      Ae = Amax;
    end if;
    Dh = 2 * xlap * sin(alphar);
    da = ds - 2 * xlap * sin(alphar) * cos(alphar);
  else
    Apoppet = 0;
    Ae = 0;
    Dh = 0;
    da = ds;
  end if;
// hydrostatic force
  ApA = Constants.pi * (da ^ 2 - dr ^ 2) / 4;
  ApB = Constants.pi * (dpop ^ 2 - da ^ 2) / 4;
  Fh = 0.1 * (pA * ApA + pB * ApB);
// Steady state flow force computation
  Fflow = 2 * kjet * Cd * (1e-6 * Ae) * (1e+5 * dp) * cos(alphar);
  annotation(
    Icon(graphics = {Polygon(fillColor = {255, 255, 127}, fillPattern =
FillPattern.HorizontalCylinder, points = {{-100, 20}, {-60, 20}, {-40, 60}, {100, 60}, {100, -60},
{-40, -60}, {-60, -20}, {-100, -20}, {-100, 20}}), Rectangle(origin = {25, -64}, fillColor = {170,
0, 0}, fillPattern = FillPattern.Solid, extent = {{-65, 4}, {75, -4}}), Polygon(origin = {-60, -58},
fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{0, 18}, {10, 18}, {10, -22},
{0, -22}, {0, 18}}), Polygon(origin = {-60, 62}, fillColor = {170, 0, 0}, fillPattern =
FillPattern.Solid, points = {{-40, 18}, {10, 18}, {10, -22}, {-40, -22}, {-40, 18}}), Polygon(origin
= {-10, 62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{-40, 18}, {110,
18}, {110, -2}, {-40, -2}, {-40, 18}}), Polygon(origin = {-100, -58}, fillColor = {170, 0, 0},
fillPattern = FillPattern.Solid, points = {{0, 18}, {20, 18}, {20, -22}, {0, -22}, {0, 18}}),
Line(origin = {-60.43, 29.42}, points = {{10, 10}, {0, -10}}, color = {255, 0, 0}, thickness = 1),
Line(origin = {-59.72, -31.44}, points = {{0, 12}, {10, -8}}, color = {255, 0, 0}, thickness = 1),
Line(origin = {-48.82, 49.04}, points = {{8, 10}, {0, -8}}, color = {0, 0, 255}, thickness = 1),
Line(origin = {-48.4394, -50.93}, points = {{0, 10}, {8, -8}}, color = {0, 0, 255}, thickness = 1)},
coordinateSystem(initialScale = 0.1)));
end Poppet;
```

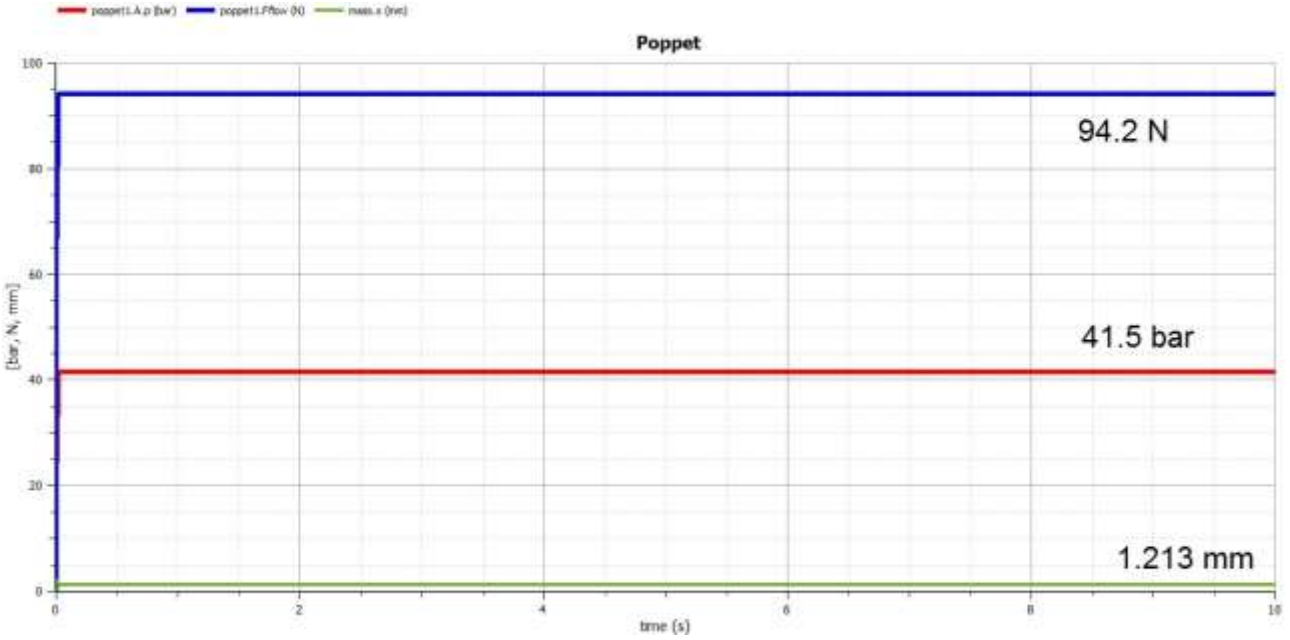
### 6.4.3 Esempio: schema minimale con otturatore conico

Realizziamo uno schema minimale per utilizzare un otturatore conico:



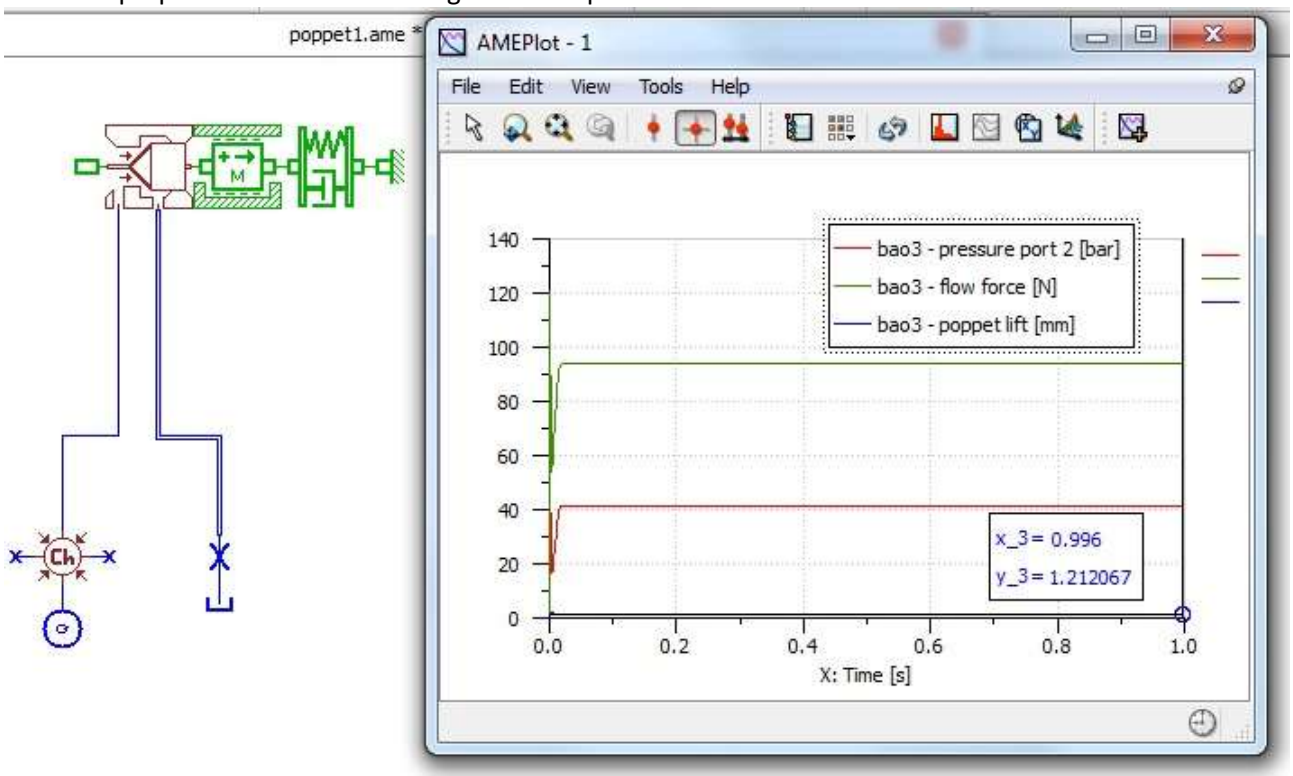

Per lo step impostiamo height = 100.
Per lo strozzatore, orifice1, mettere d = 10.
Per la massa mettere x1 = 0 e x2 = 5.
Per lo springdamper lasciare i valori di default (k = 100, F0 = 0, c = 1000).
Per l'otturatore conico lasciare i valori di default ($d_{pop}$ = 15, $d_s$ = 10, $d_r$ = 5, alpha = 45, $k_{jet}$ = 1).

Lanciamo la simulazione:




### 6.4.3.1 *Verifica con AMESim*

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:




I risultati sono molto vicini.

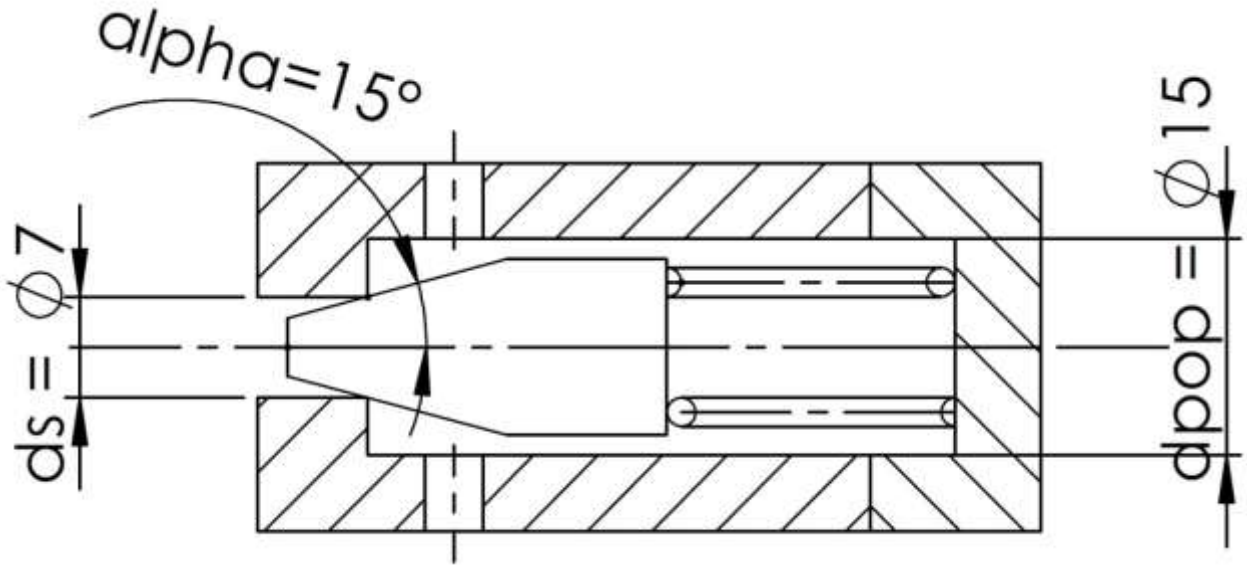
### 6.4.4 Esempio: valvola antiurto ad azione diretta

Vogliamo dimensionare una valvola limitatrice di pressione per circa 40 L/min, tarata a 350 bar, e intendiamo utilizzare un cono che fa tenuta su un diametro ds = 7 mm. Abbiamo a disposizione una molla con k = 100 N/mm.

#### *6.4.4.1 Determinare*

1. La forza di precarico della molla per aprire a circa 350 bar.
2. La strozzatura a valle dell'elemento mobile della valvola (in pratica sono i fori radiali verso T).
3. Mostrare le curve Q-ΔP ottimali per questa soluzione.

#### *6.4.4.2 Schema e dati*




Diametro totale del cono $d_{pop}$ = 15 mm
Diametro di tenuta $d_s$ = 7 mm
Q = 40 L/min
Costante elastica della molla k = 100 N/mm

#### *6.4.4.3 Decisioni*

Simuliamo la serie di fori radiali con un unico strozzatore equivalente.
Decidiamo infine di mettere mano al coefficiente $k_{jet}$: *tutte le valvole limitatrici di pressione (di cui le valvola antiurto fanno parte) hanno un deflettore ad una certa distanza dal punto di tenuta, solidale con il cono. Il compito di tale deflettore è quello di compensare le forze di flusso, per tanto, un angolo di 15° al fine del calcolo delle forze di flusso appare irrealistico. Un deflettore che funziona perfettamente è in grado d'espellere il fluido con un angolo superiore ai 90° e quindi di trarre beneficio dalle forze di flusso, operando addirittura in apertura. Un deflettore che espelle il flusso a 90° annulla le forze di flusso*. Un deflettore può tuttavia essere progettato per espellere il flusso a un certo angolo θ (circa), in questo caso, il valore corretto per $k_{jet}$:

$$k_{jet} = \frac{\cos\theta}{\cos\alpha}$$

**6-15: determinazione corretta del coefficiente $k_{jet}$ quando angolo del cono e angolo d'efflusso sono disgiunti**

*Nel nostro caso decidiamo che la valvola che dobbiamo studiare disponga di un deflettore capace d'annullare le forze di flusso $k_{jet} = 0$.*

#### *6.4.4.4 Ipotesi*

Non conosciamo il coefficiente di smorzamento viscoso. In genere i sistemi idraulici hanno un coefficiente molto alto. In prima approssimazione possiamo fare le seguenti considerazioni.

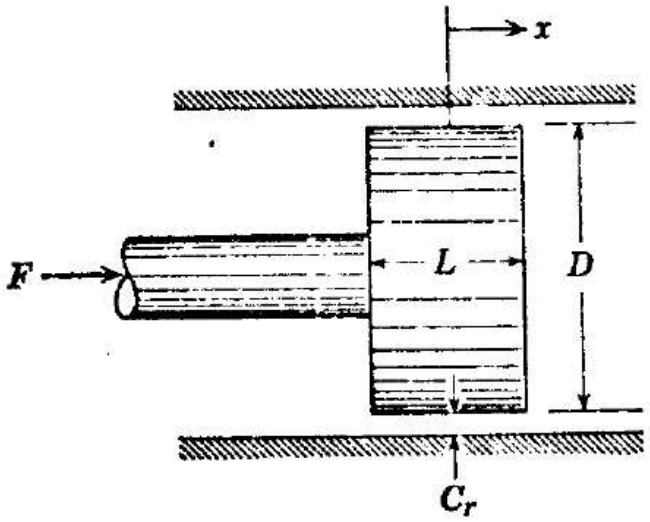



Figura 6-4:Pistone accoppiato sul diametro D su camicia per un tratto L con gioco radiale $C_r$.

Con $A_T = \pi \cdot D \cdot L$ si ha:

$$F = \mu \cdot A_T \cdot \frac{\dot{x}}{C_r} = \frac{\pi \cdot D \cdot L \cdot \mu}{C_r} \cdot \frac{dx}{dt}$$

La forza d'attrivo viscoso è pertanto:

$$F_v = -\frac{\pi \cdot D \cdot L \cdot \mu}{C_r} \cdot \frac{dx}{dt} = -\kappa \cdot \frac{dx}{dt}$$

6-16: attrito viscoso di un pistone immerso in un fluido

con $\kappa = \frac{\pi \cdot D \cdot L \cdot \mu}{C_r}$ = coefficiente d'attrito viscoso.

Con μ = 0.051 Pa·s, D = $15 \cdot 10^{-3}$ m, possiamo decidere di usare L = $60 \cdot 10^{-3}$ m e $C_r$ = $0.01 \cdot 10^{-3}$ m (due centesimi di gioco diametrale), risulta c = 14 Ns/m.
Questo è soltanto una parte del coefficiente di smorzamento reale e di solito è ininfluente.
A questo va aggiunto lo **smorzamento dovuto al fluido che lascia la camera della molla per spostarsi verso la camera dov'è presente il cono**, quando il pistone arretra, o al contrario quando il pistone avanza.
In prima approssimazione questa è dato da:

$$\chi = \frac{A_p^{\,3}}{K_{lam}^{\,2} A_0^{\,2}}$$

6-17: coefficiente di smorzamento viscoso dato dal travaso di fluido

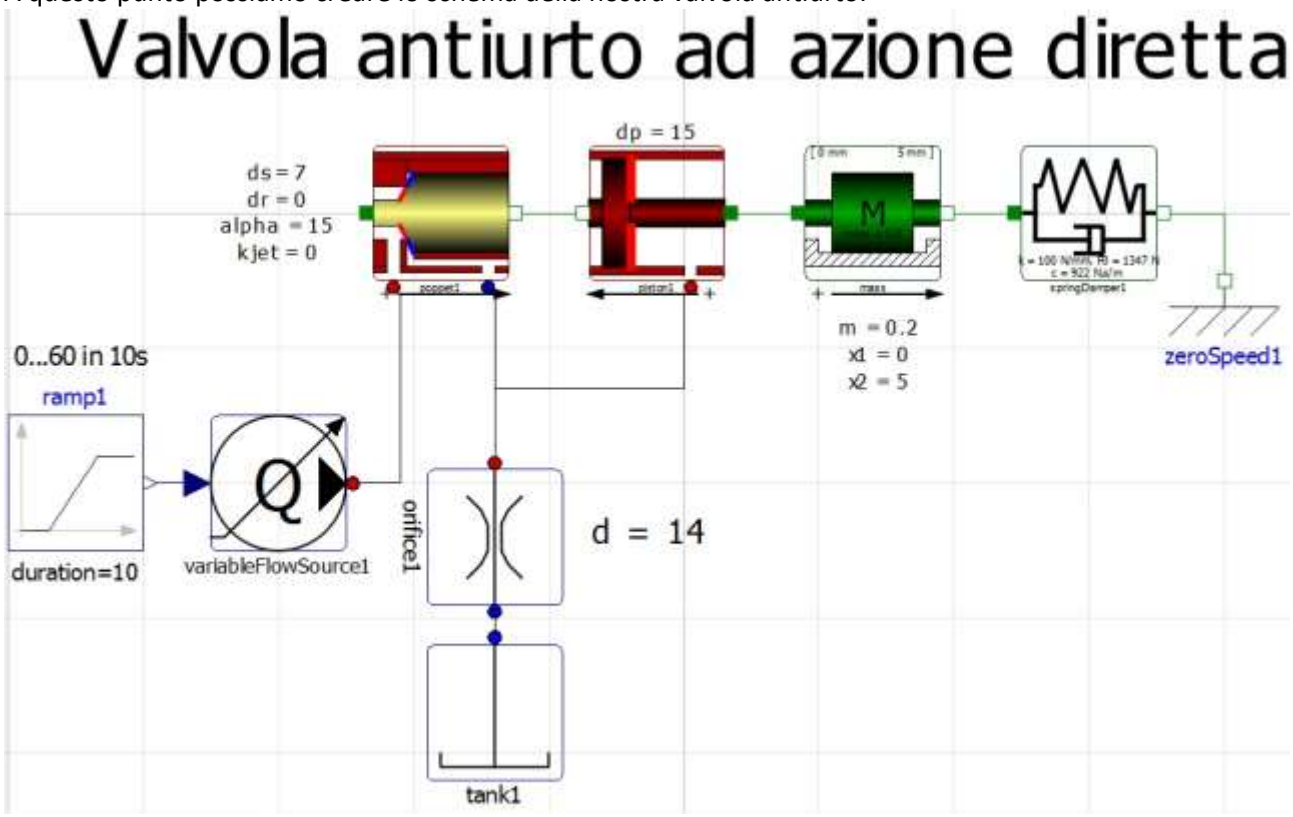
Dove $A_p$ è la sezione del pistone che scorre, $K_{lam}$ è il coefficiente d'efflusso laminare e $A_0$ è la sezione di passaggio del travaso.
Possiamo supporre un $K_{lam}$ = 0.1 e quindi, con $A_p = \pi \cdot 15^2/4 \cdot 10^{-6} = 1.767 \cdot 10^{-4}$ m$^2$, $A_0$ (equivalene) = $\pi \cdot 1^2/4 \cdot 10^{-6}$ = $7.854 \cdot 10^{-7}$ m$^2$, quindi χ = 894 Ns/m.
Ne consegue che il coefficiente di smorzamento, presunto, è c = κ + χ ≈ 910.

### *6.4.4.5 Analisi*

Iniziamo l'analisi considerando la forza di precarico teorica che deve avere la molla.
Con un area As relativa al diametro di tenuta $d_s$, la forza necessaria per equilibrare una pressione $P_{tar}$ = 350 bar è data da:

$$F = P_{tar} \cdot A_s = P_{tar} \cdot \frac{\pi}{4} \cdot d_s^{\,2} = 350 \cdot 10^5 \cdot \frac{\pi}{4} \cdot 7^2 \cdot 10^{-6} = 1347 N$$

A questo punto possiamo creare lo schema della nostra valvola antiurto.



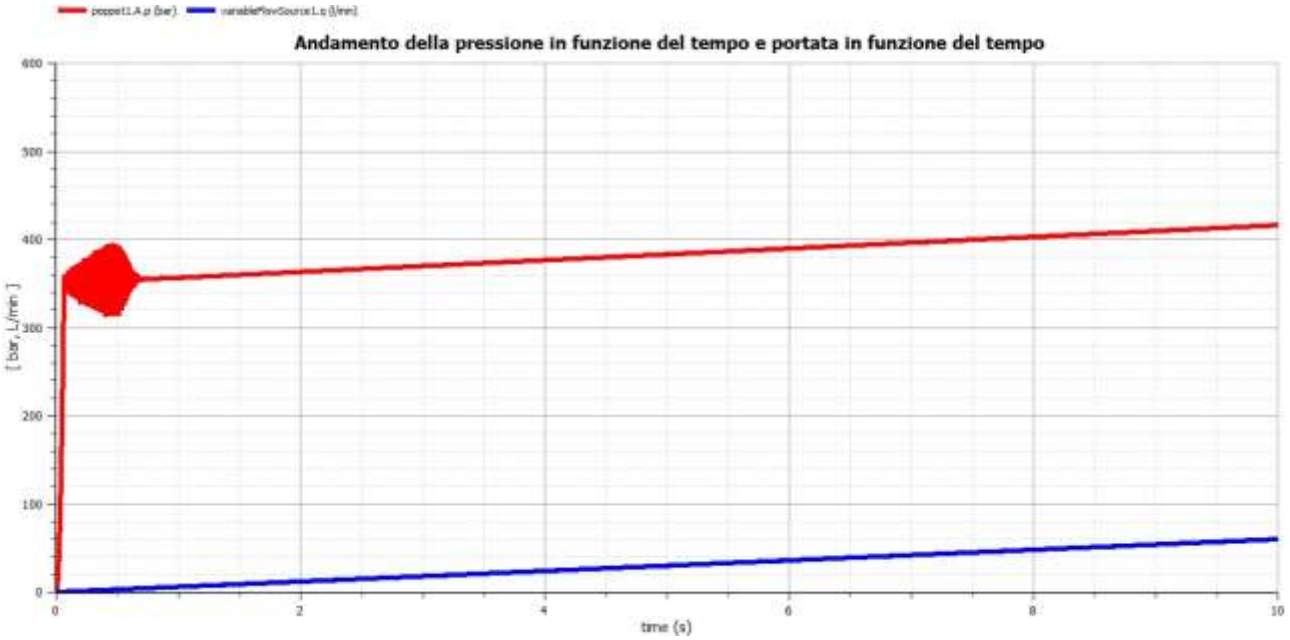

Se lanciamo la simulazione otteniamo:




In pratica otteniamo una portata Q = 40 L/min a circa 6.7 s e in corrispondenza di questo tempo ho ΔP ≈ 394 bar.

### 6.4.4.6 Commenti

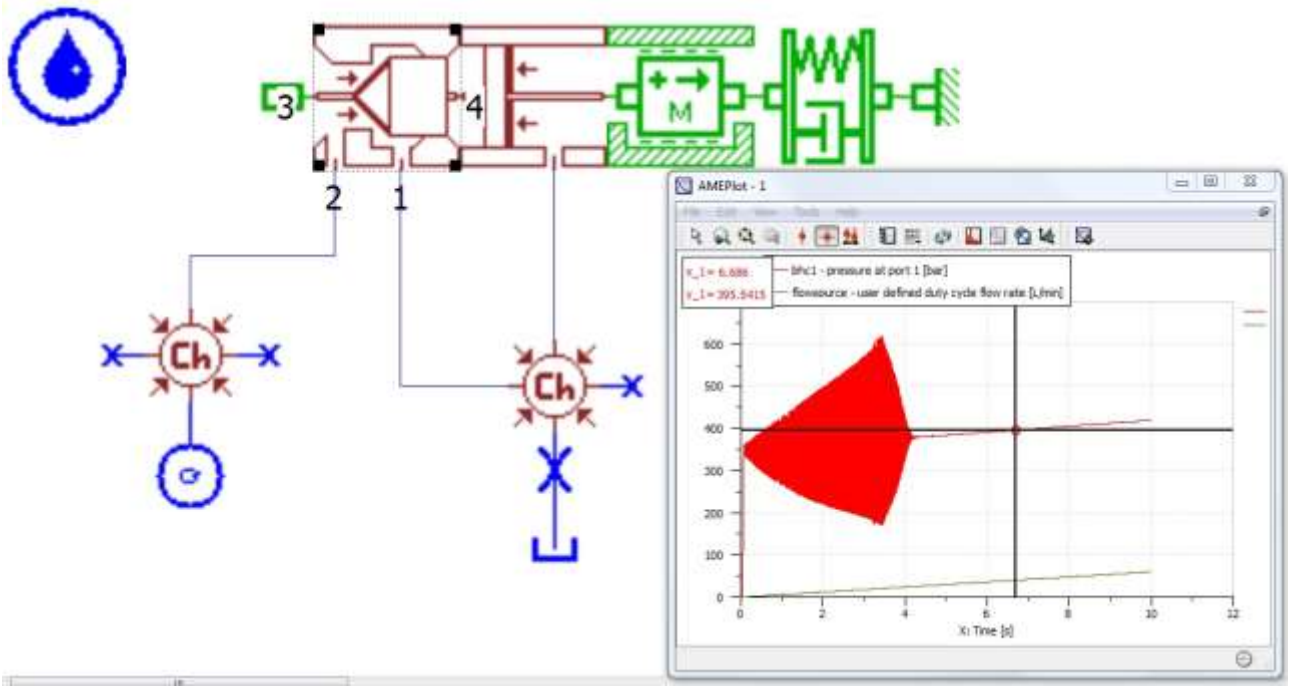
Si nota una certa instabilità a basse portate. Questo può essere tollerato in una valvola antiurto, che ha un funzionameno intermittente. Se dovessimo progettare invece una vera e propria limitatrice di pressione, dovremmo aumentare il coefficiente di smorzamento.
Un coefficiente di smorzamento circa doppio ( c = 2000 Ns/m) consente di limitare gran parte delle vibrazioni.



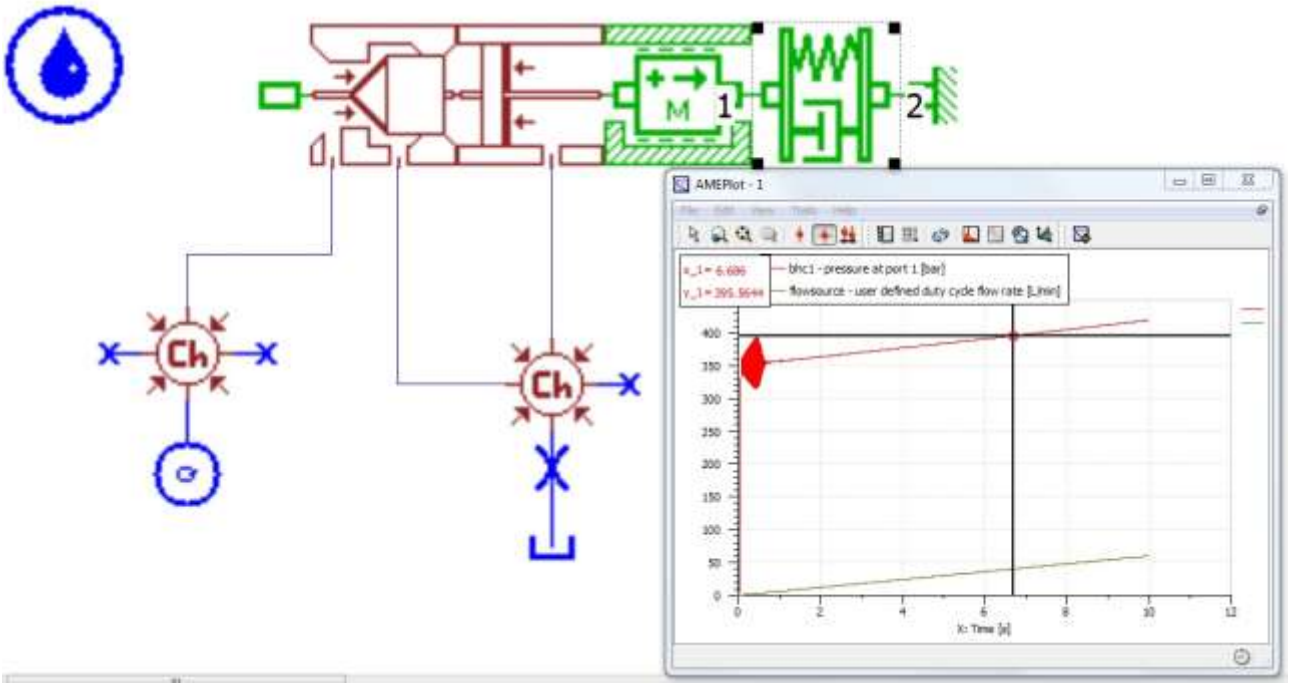

*Per questo motivo il design delle valvole antiurto spesso prevede il solo cono, così come l'abbiamo modellato, mentre le valvole limitatrici di pressione hanno uno smorzatore progettato appositamente (l'otturatore conico presenta un codulo che s'impegna con gioco ridotto in una camicia ricavata nel corpo valvola: quando l'otturatore si muove, il volume chiuso tra la camicia e il codulo deve fuoriuscire da una strozzatura calibrata ricavata nel codulo)*.

### 6.4.4.7 Verifica con AMESim

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:

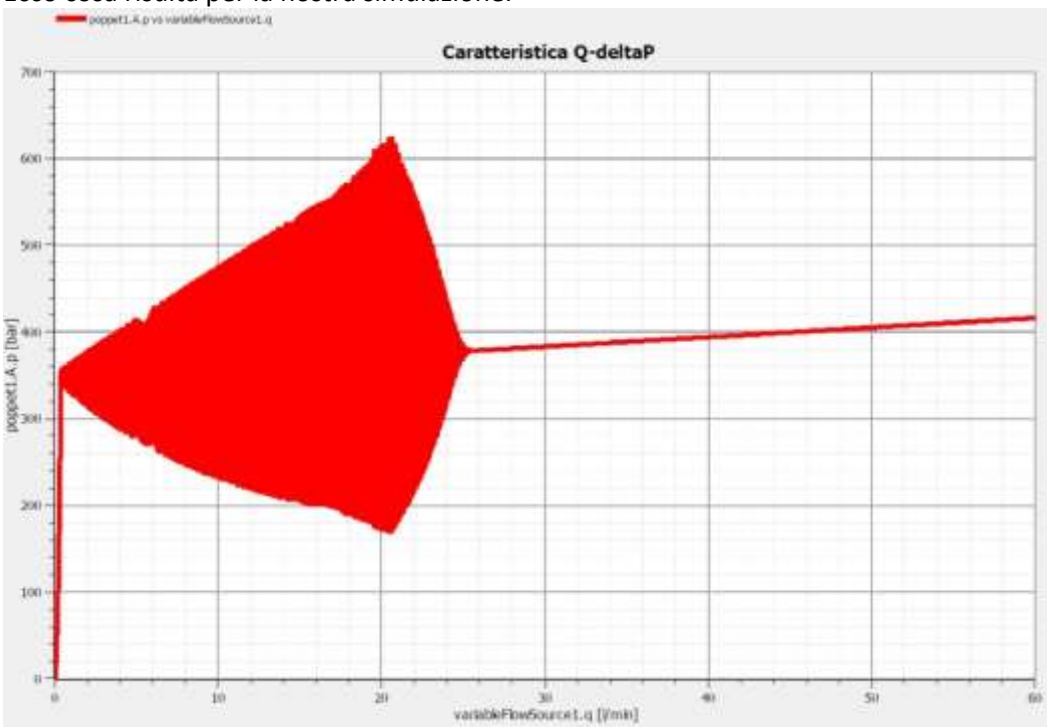



AMESim fornisce a 6.7 s (e quindi Q = 40 L/min) una pressione di 395 bar.
Se impostiamo il damping ratio a 2000, anche AMESim trova che il rumore venga ridotti in gran parte:



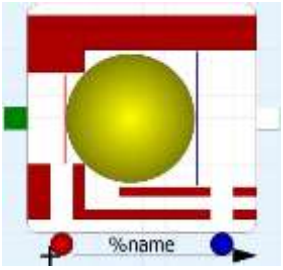

### *6.4.4.8 Curva Q-ΔP*

In realtà non abbiamo fornito una curva Q-ΔP esplicita: abbiamo fornito Q(t) e ΔP(t).
Per vedere un grafico con una variabile indipendente x e una variabile dipendente y si deve aprire una **finestra di plot parametrica** (barra dei comandi in alto a sinistra, accando a new plot), quindi **selezionare per prima la variabile da utilizzare come asse delle x e poi la variabile da usare come asse delle y**.
Ecco cosa risulta per la nostra simulazione:



## 6.5 Ball Valve

A differenza del cono, la tenuta è fatta con una sfera. E' ideale per realizzare valvole unidirezionali per piccole portate (max. 20 L/min, mentre in un funzionamento intermittente anche 40-50 L/min), perché è una costruzione semplice ed economica.




A differenza dell'otturatore conico, dove nel blocco la contropressione non agisce posteriormente in richiusura e se necessario bisogna esplicitarlo connettendo un pistone dietro al blocco poppet, in questo caso la pressione della porta B (blu), agisce in richiusura.

### 6.5.1 Principali parametri

1. Diametro della sede in cui la sfera realizza la tenuta idraulica – $d_s$.
2. Diametro della sfera – $d_b$.
3. Diametro di un perno lato molla (se > 0 impedisce che la superficie sia bagnata dall'olio) – $d_{rp}$.
4. Diametro di un perno dal lato sede (se > 0 impedisce che la superficie sia bagnata dall'olio) – $d_{rs}$.
5. Coefficiente per il calcolo delle forze di flusso – $k_{jet}$ (default = 1) ←Partial HCDEdge (6.1.2).

Non discutiamo il modello perché è piuttosto elementare, in pratica, i calcoli sulla sezione d'apertura da parte della sfera sono sostanzialmente geometrici, le uniche due formule d'interesse sono la forza idraulica che agisce in apertura:

$$F_{hyd} = p_A \cdot A_{p,A} + p_B \cdot A_{p,B} = p_A \cdot \frac{\pi}{4} \cdot \left( {d_a}^2 - {d_{rs}}^2 \right) - p_B \cdot \frac{\pi}{4} \cdot \left( {d_a}^2 - {d_{rp}}^2 \right)$$

6-18: forza in apertura di un otturatore a sfera (ball valve)

E le forze di flusso si calcolano ancora con la [6-14]. A cambiare sono infatti le sezioni effettive aperte $A_e$, che dipendono dalle geometrie differenti.

### 6.5.2 Codice

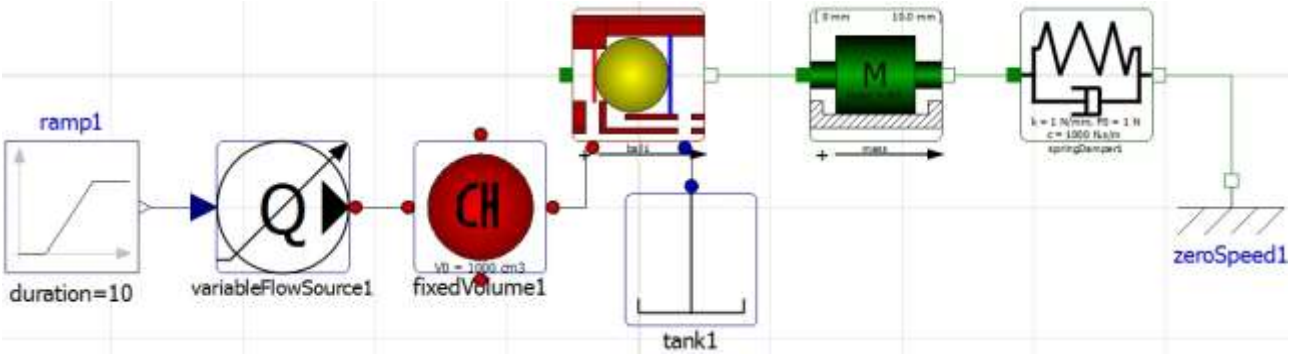
Ecco il codice:

```
model Ball
  extends FPLib.HCD.Interfaces.PartialHCDEdge(Cdmax = 0.7, Rec = 100, xu = 0);
  // Ball parameters and variables
  parameter Types.Diameter ds = 5 "Seat diameter";
  parameter Types.Diameter db = 10 "Ball diameter";
  parameter Types.Diameter drp = 0 "Rod diameter (opposite to seat)";
  parameter Types.Diameter drs = 0 "Rod diameter (seat side)";
  Types.Diameter da "Active diameter of the poppet";
  // Types.Area Aball(start = 0) "Area of the poppet";
protected
  Types.Area Amax = Constants.pi * (ds ^ 2 - drs ^ 2) / 4 "maximum area";
  SI.Angle alpha "seat angle";
  SI.Angle theta "jet force angle";
  Types.Area ApA "Active section side A";
  Types.Area ApB "Active section side B";
equation
// sections increase / decrease volumes (side A and side B)
  VolSezA = Constants.pi * (ds ^ 2 - drs ^ 2) / 4;
  VolSezB = Constants.pi * (db ^ 2 - drp ^ 2) / 4;
// main outflow section
  if noEvent(xlap > 0) then
//Aball = Constants.pi * ((-db ^ 2 / ds ^ 2) + 1 / tan(alpha) ^ 2 + 1) * ds ^ 2 / (4 * sqrt(1 /
tan(alpha) ^ 2 + 1));
    Ae = Functions.bilimit(Constants.pi * ((-db ^ 2 / ds ^ 2) + 1 / tan(alpha) ^ 2 + 1) * ds ^ 2 /
(4 * sqrt(1 / tan(alpha) ^ 2 + 1)), 0, Amax);
    1 / tan(alpha) = 2 * xlap / ds + sqrt(db ^ 2 / ds ^ 2 - 1);
    Dh = 4 * Ae / (Constants.pi * (ds + da));
    da = db / sqrt(1 + 1 / tan(alpha) ^ 2);
```

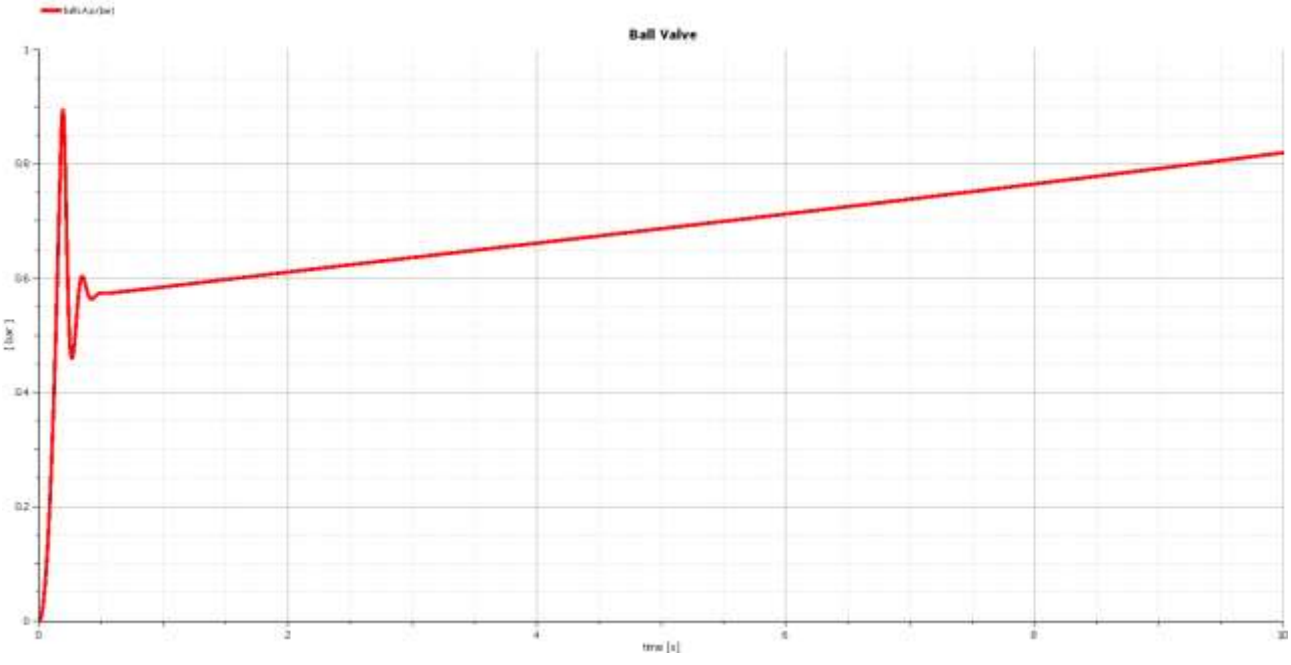
```
  else
    alpha = asin(ds / db);
    Ae = 0;
    Dh = 0;
    da = ds;
  end if;
// hydrostatic force
  ApA = Constants.pi * (da ^ 2 - drs ^ 2) / 4;
  ApB = Constants.pi * (da ^ 2 - drp ^ 2) / 4;
  Fh = 0.1 * (pA * ApA - pB * ApB);
// Steady state flow force computation
  theta = acos(da / db);
  Fflow = 2 * kjet * Cd * (1e-6 * Ae) * (1e+5 * dp) * cos(theta);
  annotation(
    Icon(graphics = {Rectangle(origin = {25, -64}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-45, 4}, {35,
-4}}), Polygon(origin = {-60, -58}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{0, 18}, {10, 18}, {10, -22},
{0, -22}, {0, 18}}), Polygon(origin = {-60, 62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{-40, 18}, {10,
18}, {10, -22}, {-40, -22}, {-40, 18}}), Polygon(origin = {-10, 62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points
= {{-40, 18}, {110, 18}, {110, -2}, {-40, -2}, {-40, 18}}), Polygon(origin = {-100, -58}, fillColor = {170, 0, 0}, fillPattern =
FillPattern.Solid, points = {{0, 18}, {20, 18}, {20, -22}, {0, -22}, {0, 18}}), Line(origin = {-67.0939, 24.9875}, points = {{0, 12},
{0, -64}}, color = {255, 0, 0}, thickness = 1), Ellipse(origin = {-16, -4}, fillColor = {255, 255, 0}, fillPattern =
FillPattern.Sphere, extent = {{-50, 60}, {62, -52}}, endAngle = 360), Rectangle(origin = {125, -64}, fillColor = {170, 0, 0},
fillPattern = FillPattern.Solid, extent = {{-45, 4}, {-25, -4}}), Line(origin = {48.07, 45.89}, points = {{0, 12}, {0, -104}}, color
= {0, 0, 255}, thickness = 1)}, coordinateSystem(initialScale = 0.1)));
end Ball;
```

### 6.5.3 Esempio: ball valve (minimal)

Ecco uno schema minimale per una check valve a sfera.

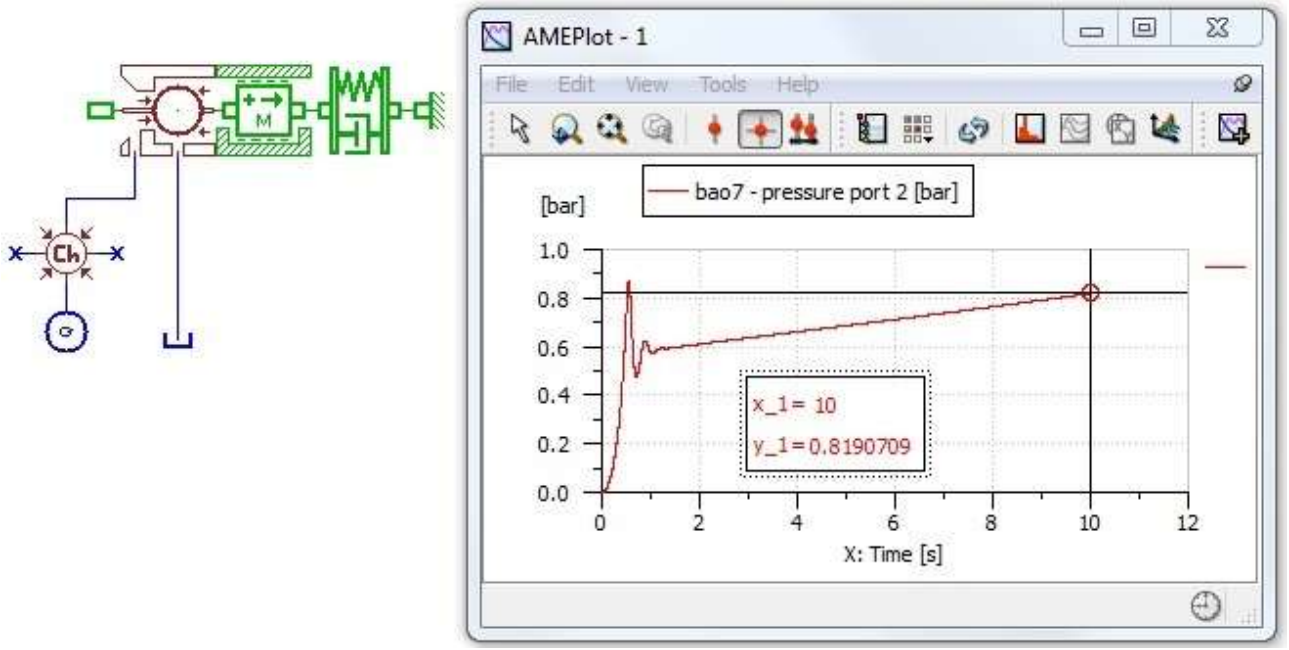



Il blocco ball1 è lasciato di defualt ($d_b$ = 10, $d_s$ = 5, $k_{jet}$ = 1). La rampa va da 0 a 2 in 10s. Il volume è di 1000 $cm^3$. La massa è 0.2 kg, $x_1$ = 0, $x_2$ = 10. Lo springdamper ha k = 1, $F_0$ = 1 e c = 1000.
Se lanciamo la simulazione osserviamo quanto segue:



### 6.5.3.1 *Verifica con AMESim*

Abbiamo preparato con AMESim il seguente semplice modello da usare come verifica:



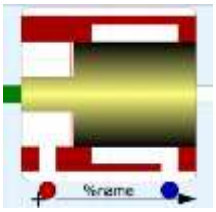

Entrambi forniscono gli stessi ΔP a parità di portata ma una differenza tra i due calcoli è presente.
AMESim ha un transitorio più lungo rispetto a quello calcolato con Modelica.
Per semplificare i calcoli nella mia FPLib ho assunto un bulk modulus costante, mentre AMESim lo ricalcola continuamente, tenendo conto anche di altri fattori come ad esempio le pareti che contengono il fluido.
Se volessimo ottenere lo stesso effetto dovremmo andare a ridurre la costante

```
final constant SI.BulkModulus bulk = 1.7e+9 "bulk modulus";
```

oppure potremmo aumentare il volume della camera, portandolo ad esempio a $V_0 = 5000\ cm^3$.

## 6.6 AnularOrificeSpool

E' il classico spigolo pilotante di una spola.




### 6.6.1 Principali parametri

1. Diametro della spola - $d_{spool}$.
2. Diametro del colletto - $d_r$.
3. Jet angle - theta.
4. Ricoprimento (se < 0 → positivo, se = 0 → critico, se > 0 → negativo) - $x_u$.
5. Coefficiente per il calcolo delle forze di flusso – $k_{jet}$ (default = 1) ← Partial HCDEdge (6.1.2).

Non discutiamo il modello perché è piuttosto elementare, in pratica, i calcoli sulla sezione d'apertura sono sostanzialmente geometrici, le uniche due formule d'interesse sono la forza idraulica che agisce in apertura:

$$F_{hyd} = p_A \cdot \frac{\pi}{4} \cdot \left( {d_{spool}}^2 - {d_r}^2 \right)$$

6-19: forza in apertura nel risalto di una spola

Le forze di flusso si calcolano ancora con la [6-14]. A cambiare sono infatti le sezioni effettive aperte $A_e$, che dipendono dalle geometrie differenti.

### 6.6.2 Definire il ricoprimento

Il parametro $x_u$ definisce il ricoprimento negativo quando la spola è nella posizione x = 0.
La presenza di ricoprimento negativo o **underlap** (definita da $\mathbf{x_u > 0}$) significa che nella posizione di riposo (posizione in cui x = 0), è già presente una apertura corrispondente a $x_u$.
Per contro, se $\mathbf{x_u < 0}$ si assume che siamo in presenza di **overlap** o ricoprimento positivo.
In pratica può essere utile questo schemino:

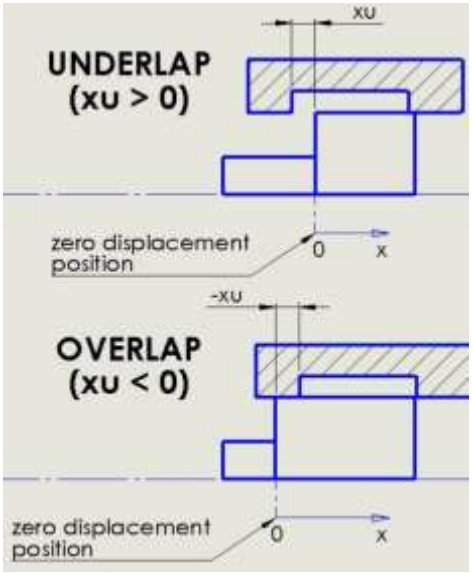



Figura 6-5: schema per l'inserimento del ricoprimento nelle spole

Chiaramete un **ricoprimento "critico"** si ottiene con $\mathbf{x_u = 0}$ (che è il valore di default).

### 6.6.3 Codice

Ecco il codice:






```
model AnularOrificeSpool
  extends FPLib.HCD.Interfaces.PartialHCDEdge(Cdmax = 0.7, Rec = 100);
  // Poppet parameters and variables
  parameter Types.Diameter dspool = 10 "Spool diameter";
  parameter Types.Diameter dr = 5 "Rod diameter";
  parameter Types.Angle theta = 69 "Jet angle";
  Types.Area Aspool(start = 0) "Area opened by the spool";
protected
  Types.Area Amax = Constants.pi * (dspool ^ 2 - dr ^ 2) / 4 "maximum area";
equation
// sections increase / decrease volumes (side A and side B)
  VolSezA = Constants.pi * (dspool ^ 2 - dr ^ 2) / 4;
  VolSezB = 0;
// active sections on which the pressures act (side A and side B)
// main outflow section
  assert(Aspool >= 0, "Apoppet minore di zero, ocio!");
  if noEvent(xlap > 0) then
    Aspool = Constants.pi * dspool * xlap;
    if Aspool <= Amax then
      Ae = Aspool;
    else
      Ae = Amax;
    end if;
    Dh = 2 * xlap;
  else
    Aspool = 0;
    Ae = 0;
    Dh = 0;
  end if;
// hydrostatic force
  Fh = 0.1 * pA * Amax;
// Steady state flow force computation
  Fflow = 2 * kjet * Cd * (1e-6 * Ae) * (1e+5 * dp) * cos(theta * Constants.pi / 180);
  annotation(
    Icon(graphics = {Polygon(fillColor = {255, 255, 127}, fillPattern = FillPattern.HorizontalCylinder, points = {{-100, 20}, {-40,
20}, {-40, 60}, {100, 60}, {100, -60}, {-40, -60}, {-40, -20}, {-100, -20}, {-100, 20}}), Rectangle(origin = {145, -64}, fillColor =
{170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-65, 4}, {-45, -16}}), Polygon(origin = {-60, -58}, fillColor = {170, 0, 0},
fillPattern = FillPattern.Solid, points = {{0, -2}, {40, -2}, {40, -22}, {0, -22}, {0, -2}}), Polygon(origin = {-60, 62}, fillColor =
{170, 0, 0}, fillPattern = FillPattern.Solid, points = {{-40, 18}, {40, 18}, {40, -2}, {-40, -2}, {-40, 18}}), Polygon(origin = {-10,
62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{90, 18}, {110, 18}, {110, -2}, {90, -2}, {90, 18}}),
Polygon(origin = {-100, -58}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{0, -2}, {20, -2}, {20, -22}, {0, -
22}, {0, -2}}), Line(origin = {-138.59, -24.3866}, points = {{96, 84}, {96, 46}}, color = {255, 0, 0}, thickness = 1), Line(origin =
{-138.208, -104.799}, points = {{96, 84}, {96, 46}}, color = {255, 0, 0}, thickness = 1)}, coordinateSystem(initialScale = 0.1)));
end AnularOrificeSpool;
```

### 6.6.4 Esempio: verifica di una semplice valvola con spola e pistone di pilotaggio

Vogliamo studiare il sistema costituito dai blocchi seguenti:

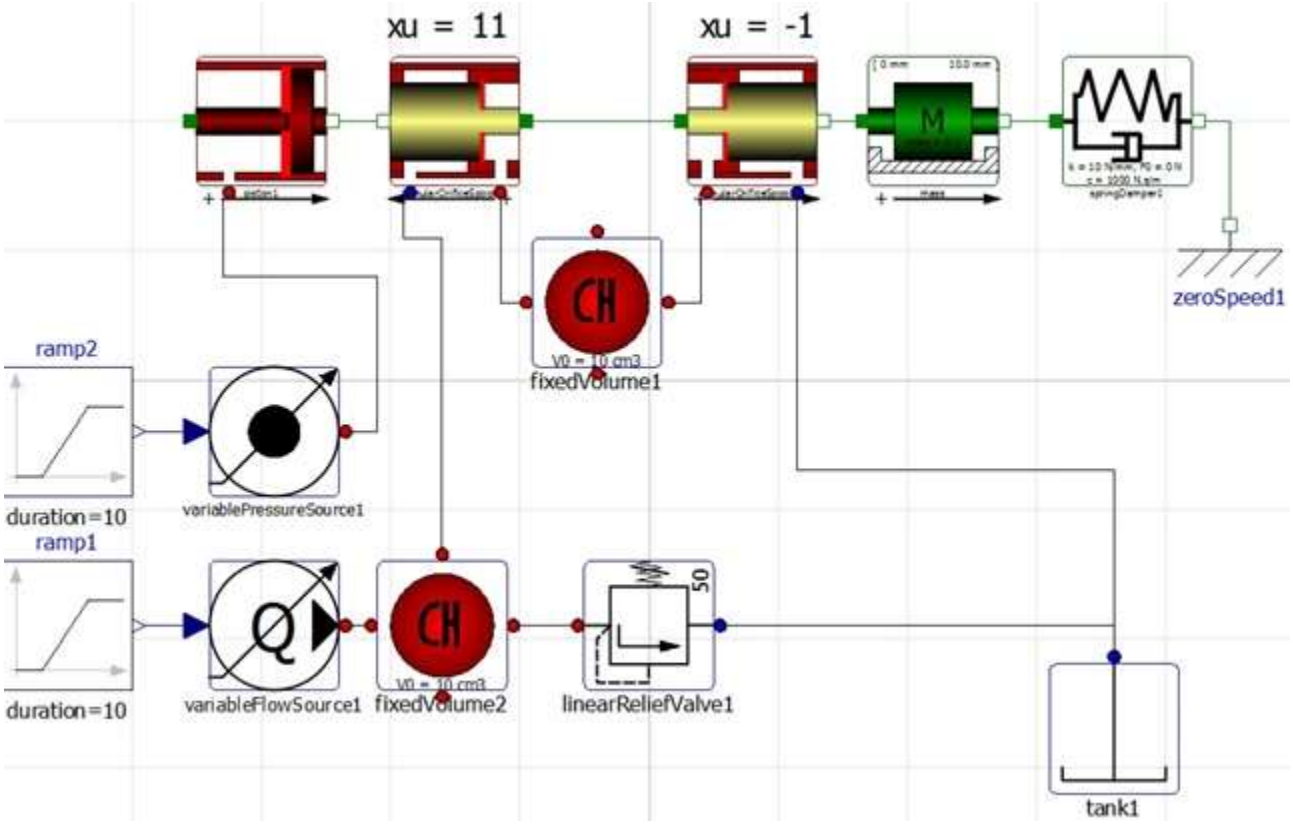



Figura 6-6: valvola d'esempio con spola e pistone di pilotaggio

I due blocchi contrapposti AnularOrificeSpool hanno gli stessi parametri di default eccetto che quello a sinistra (AnularOrificeSpool2) ha **FromLeft2Right = false**, perché è girato in contrapposizione alla freccia della massa, che definisce l'orientamento del sistema e ha $x_u = 11$, che significa un underlap di 11 mm, vale a dire che si trova già aperto di 11 millimetri. L'AnularOrificeSpool di destra (AnularOrificeSpool1) invece ha $x_u = -1$, il che significa che è un overlap e che quindi parte chiuso di 1 mm. In entrambi i blocchi sono stati impostati i kjet a zero, perché al momento non siamo interessati alle forze di flusso. In questa condizione, la pressione sulle sezioni attive dovrebbe bilanciare la forza sulla spola.

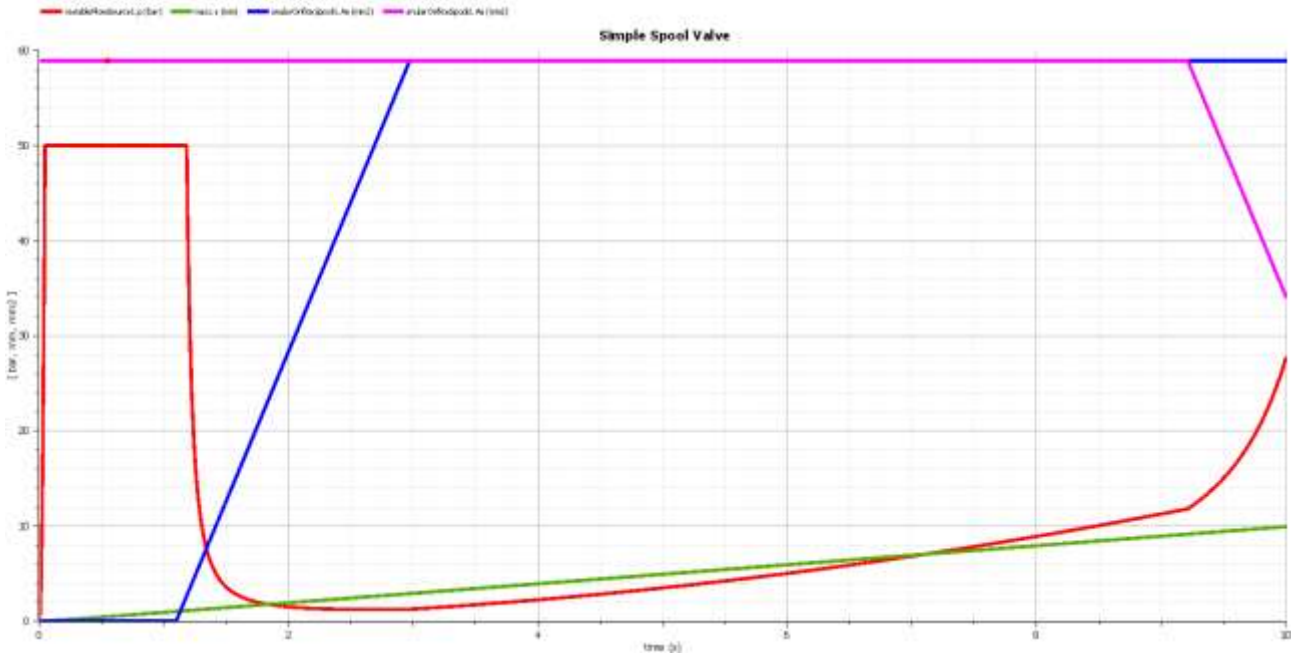
Il pistone di sinistra ha tutti i parametri di default, mentre la massa e lo springdamper sono gli stessi dell'esempio 6.2.4.1. *Abbiamo già visto questa configurazione di pistone, massa e springdamper nell'esempio 6.3.3: la utilizziamo spesso perché abbiamo visto che è in grado di far compiere una corsa alla massa di circa 10 mm in 10 s, ad una velocità circa costante*.

L'idea sarebbe quella di poter azionare la spola costituita dai due spigoli pilotanti per far passare l'olio dalla variableFlowSource1 a dentro il primo spigolo pilotante (quello a sx), quindi dentro al secondo e poi a scarico.

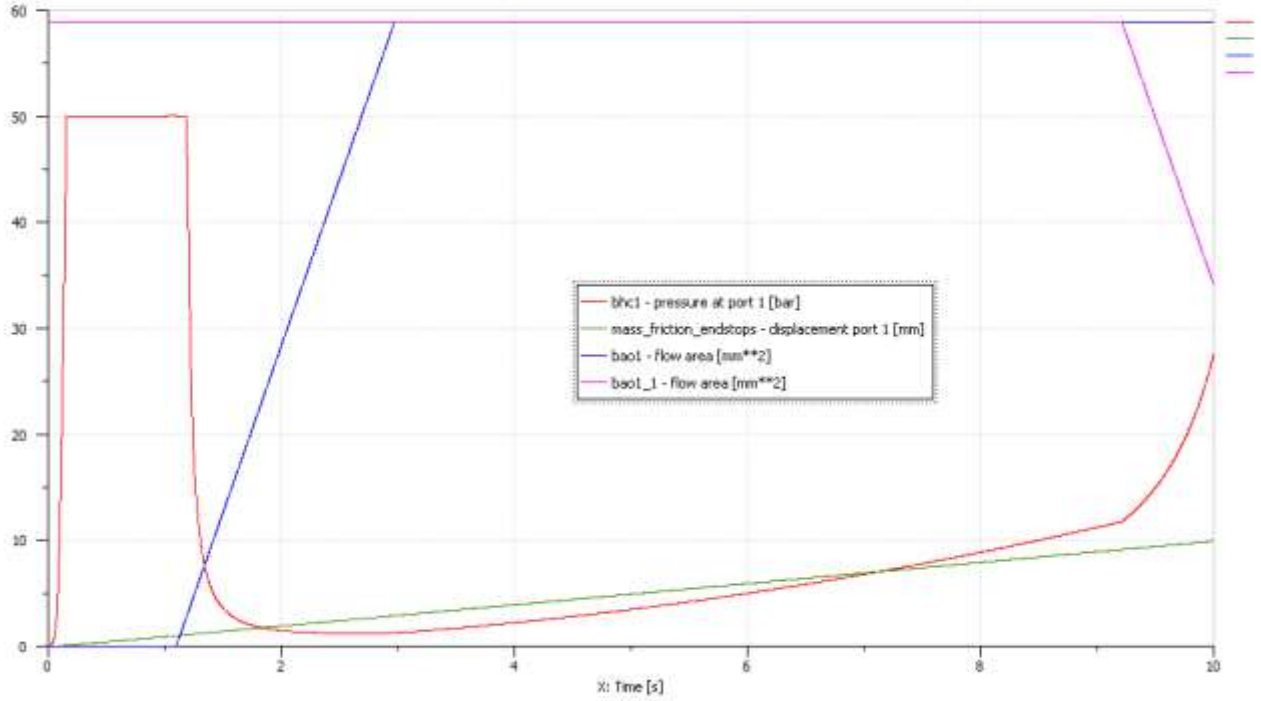
Poiché AnularOrificeSpool1 è inizialmente chiuso (overlap, $x_u = -1$), è necessaria una valvola limitatrice di pressione dato che nei momenti iniziali la pompa manda su una linea chiusa.

La prima rampa va da 0 a 100 in 10s e diviene portata nel blocco variableFlowSource, la seconda va da 0 a 17 (sempre in 10s) e diventa pressione nel variablePressureSource.

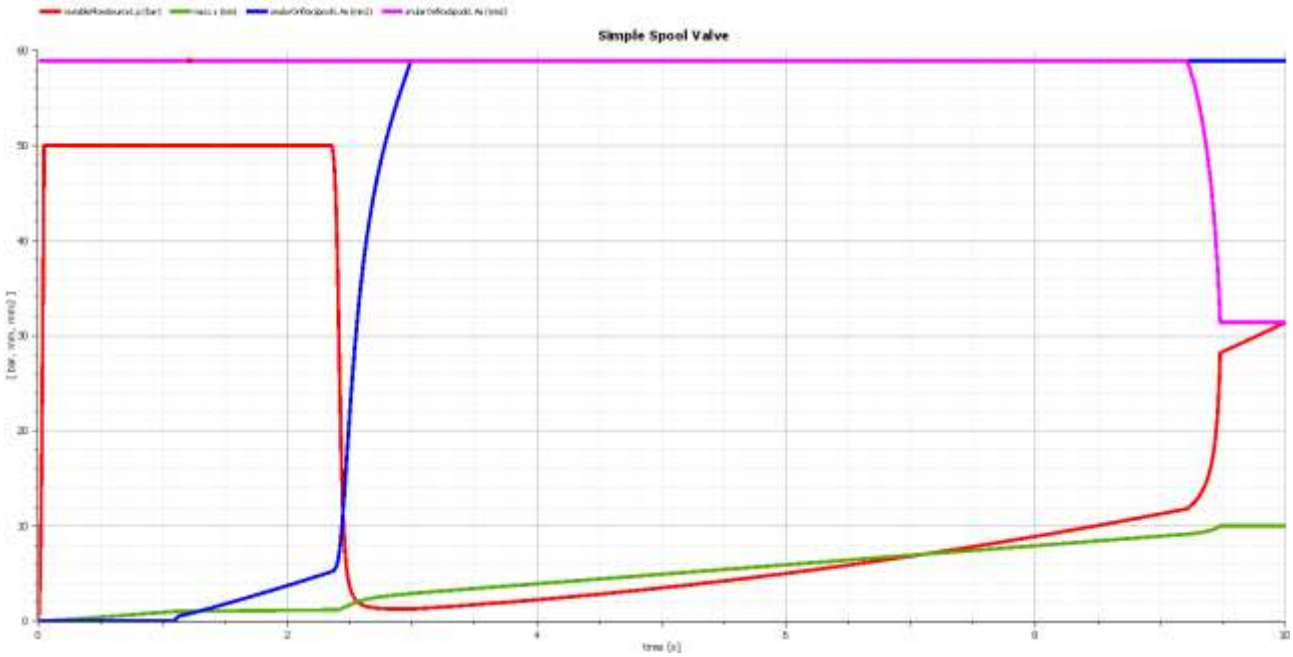
Se lanciamo la simulazione, vediamo quello ci potevamo aspettare.



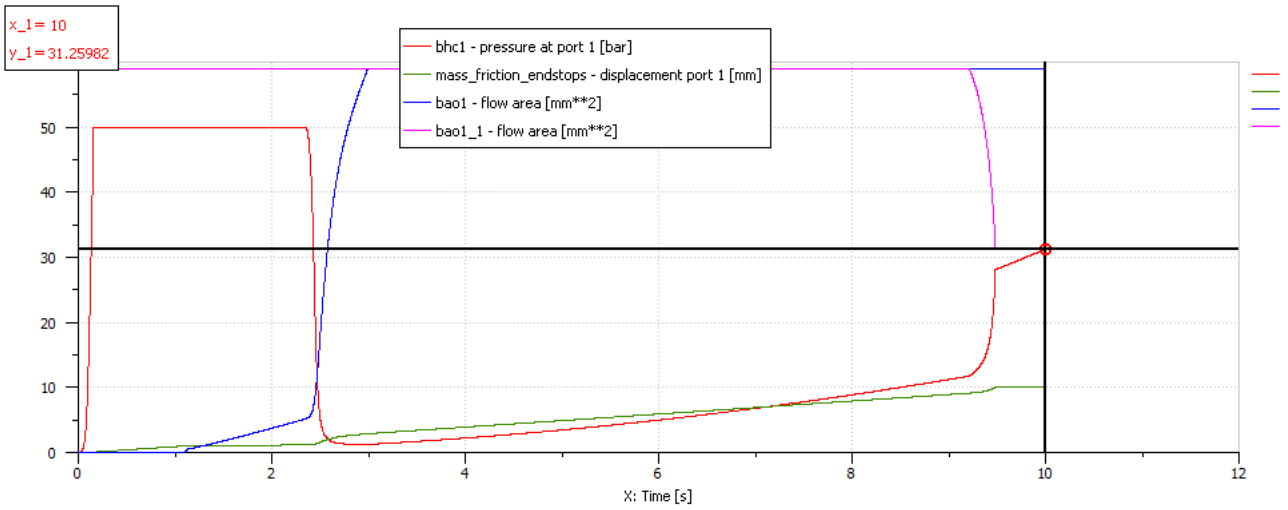

Si vede cioè che la limitatrice entra in regolazione per il primo secondo, poi appena AnularSpoolOrifice1 apre (curva verde, sottile), la pressione si abbassa. Alla fine la pressione si alza perché AnularSpoolOrifice2 inizia a richiudere (curva violetta sottile).
Possiamo verificare che AMESim con gli stessi parametri fornisca lo stesso risultato:



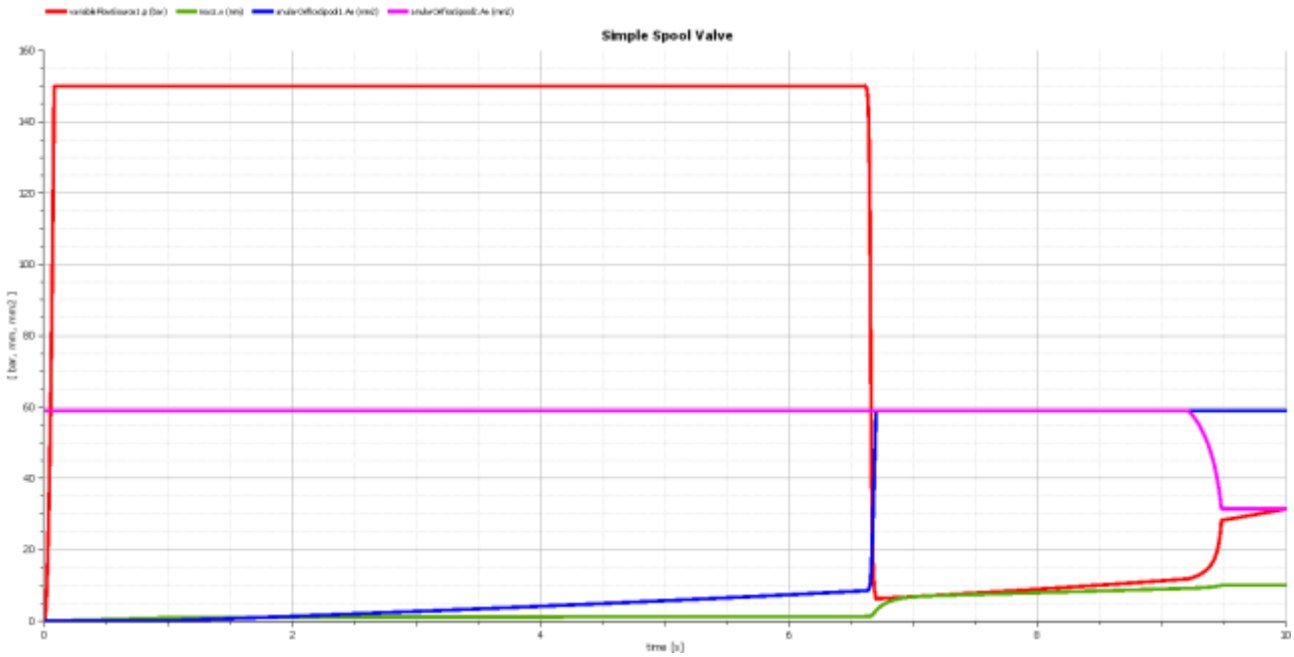

Naturalmente, senza forze di flusso, posso anche tarare la limitatrice di pressione a 350 bar e tutto funziona.
Ora consideriamo le forze di flusso, quindi, in entrambi i blocchi AnularOrificeSpool impostiamo $k_{jet} = 1$.
(*Io lo imposto anche nel modello AMESim dato che dovrò fare il confronto*).
Se ora rilancio la simulazione ottengo:




E lo stesso dicasi per AMESim.



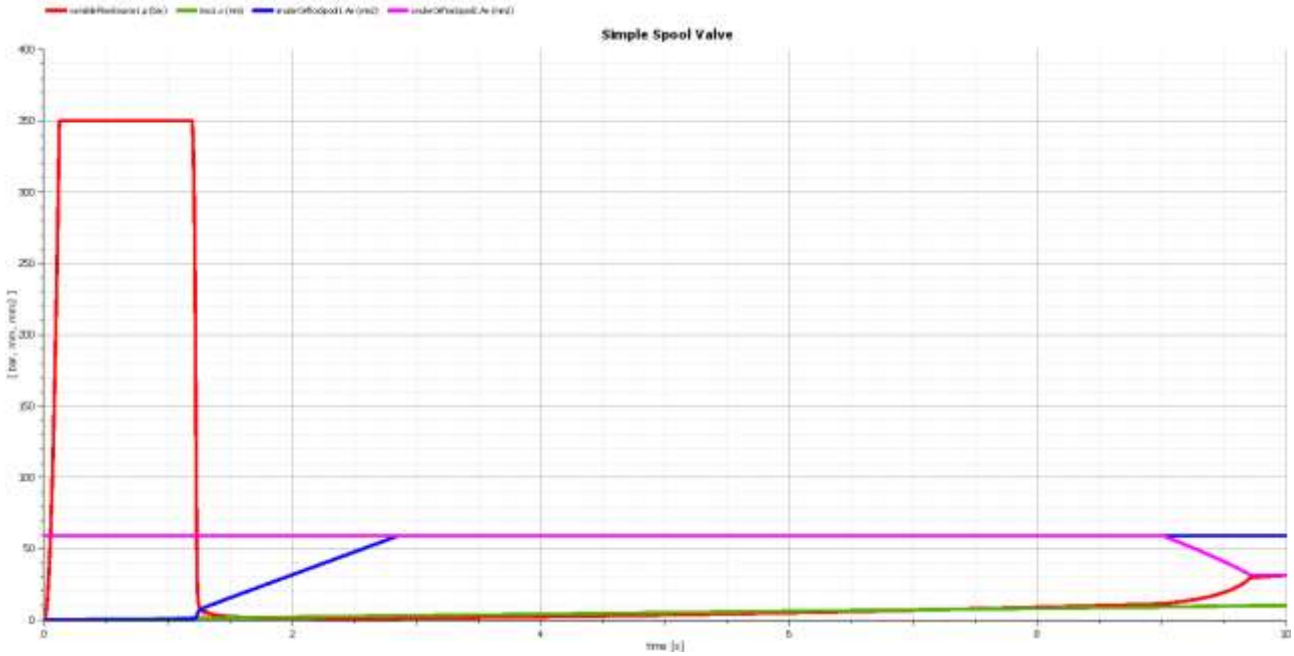

Tarando la VL a 150 bar (ometto il grafico di AMESim per non allungare ulteriormente questo testo ma i risultati sono identici) accade che la limitatrice lavora per circa 6.6 s e poi, a fatica il pistone riesce a muovere la spola.



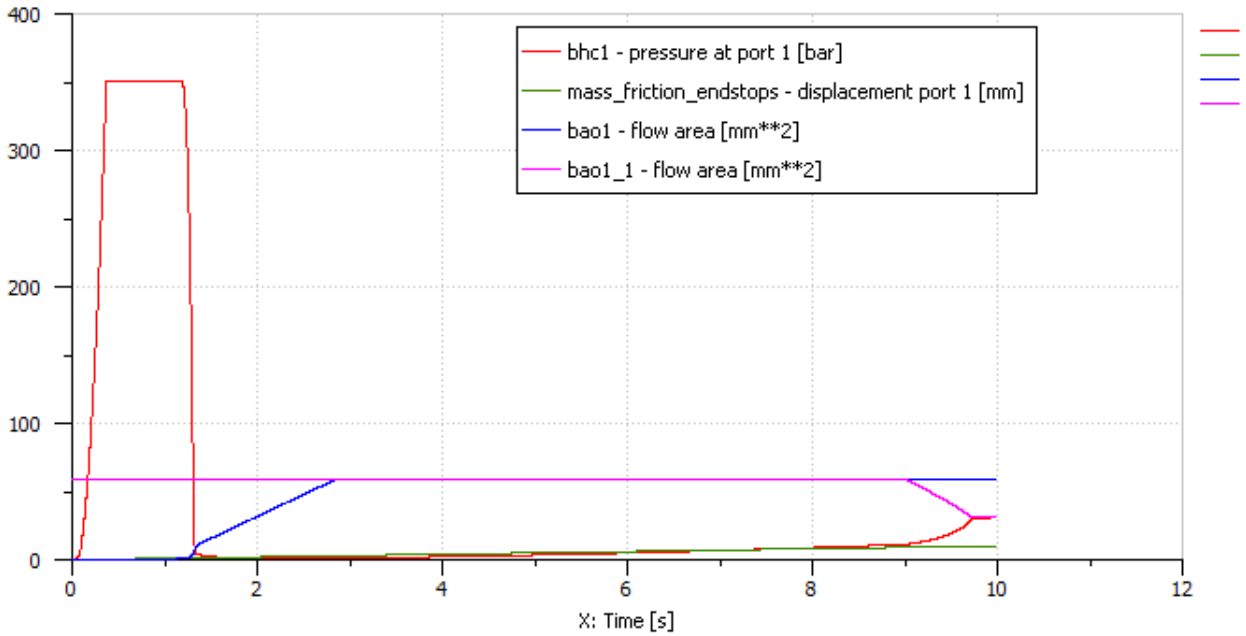

Tarando la VL a 250 bar la spola si muove di appena 1.26 mm e a 350 bar 1.17 mm (stessi risultati per FPLib e AMESim) e il sistema è sempre in valvola di massima, quindi non va bene!

A questo punto grazie alla simulazione ci si pone un problema, prima che questo possa arrivare in sala prove… diciamo che non è un risultato scontato. Ho lavorato per anni in un'azienda che produce valvole direzionali e in passato, senza l'utilizzo di questi strumenti, erano tanti i progettisti che cadevano in questi tranelli dovuti alle forze di flusso.

Poiché la forza di flusso è un disturbo, aumentiamo la forza necessaria per il posizionamento, in modo da ridurre di 10 volte l'influenza di questo disturbo, come misure: mettiamo una molla 10 volte più rigida, quindi k = 100 N/mm, chiaramente occorre una forza maggiore, per esercitarla, decidiamo di aumentare il diametro del pistone dato che solitamente non si vuole modificare la pressione di pilotaggio.
A questo punto serve che a P = 1.7 MPa, un pistone con un perno di 5 mm e un diametro esterno x, spinga per 1000 N, quindi x = 27.8. Quindi useremo un pistone con diametro Ø28.
Modifichiamo dunque la costante elastica della molla in k=100 N/mm e per il pistone $d_p$ = 28 e $d_r$ = 5 (*non lo modifichiamo*).
Rilanciamo la simulazione e godiamoci lo spettacolo:




AMESim fornisce lo stesso risultato (questa volta lo pubblico):




**Questo è un buon esempio di come la simulazione possa aiutare la progettazione**.

## 6.7 GenericSpool

E' una spola con una legge d'area generica, fornita attraverso uno o più array.




Questi blocchi sono molto flessibili perché consentono di modellare tantissimi tipi d'efflusso, con leggi d'area customizzate, fornite mediante una tabella.

### 6.7.1 Principali parametri

I principali parametri sono:

1. Diametro della spola - $d_{spool}$.
2. Diametro del colletto - $d_r$.
3. Array x_array[:] – vettore che indica le x di riferimento per la legge d'area, parametro obbligatorio
4. Array A_array[:] – Sezioni d'apertura corrispondenti a x_array, A = $f_{area}(x)$, parametro obbligatorio.
5. Array Dh_array[:] – Diametri equivalenti nei punti di x_array, Dh = $f_{Dh}(x)$, parametro facoltativo.
6. Array theta_array[:] – angoli del getto del fluido nei punti x_array, theta = $f_{theta}(x)$, facoltativo.
7. Ricoprimento (se < 0 → positivo, se = 0 → critico, se > 0 → negativo) - $x_u$.
8. Coefficiente per il calcolo delle forze di flusso – $k_{jet}$ (default = 1) ←Partial HCDEdge (6.1.2).

Gli array x_array[:] e A_array[:] definiscono una tabella in cui una legge d'area è fornita per punti:

```
x_array[:] = { x0, x1, …, xn-1 }
A_array[:] = { A0, A1, …, An-1 }
```

**L'area viene determinata tramite un algoritmo d'interpolazione lineare a tratti**.

L'algoritmo d'interpolazione per $x < x_0$ ritorna sempre $A_0$, per $x>x_{n-1}$ ritorna sempre $A_{n-1}$, garantendo quindi che i limiti non vengano superati né in un senso né nell'altro.

Ad esempio, per fornire una legge del tipo A = x2, da 0 a 5, con passo = 0.5 scriviamo:

```
x_array[:] = { 0, 0.5, 1, 1.5, 2, 2.5, 3 , 3.5, 4, 4.5, 5 }
A_array[:] = { 0, 0.25, 1, 2.25, 4, 6.25, 9, 12.25, 16, 20.25, 25 }
```

Ovviamente nei parametri mettiamo soltanto il testo in grassetto, quindi quello tra le parentesi graffe, comprese.

Se voglio inserire, per gli stessi punti di x_array, il diametro idraulico equivalente, posso definirlo nello stesso modo:

```
x_array[:] =  { x0, x1, …, xn-1 }
Dh_array[:] = { D0, D1, …, Dn-1 }
```

Se Dh_array viene lasciato al valore di default ( { 0 , 0 } ), un diametro idraulico equivalente viene calcolato con la formula:

$$D_h = 2 \cdot \sqrt{\frac{A_e}{\pi}}$$

6-20: diametro idraulico equivalente per un orificio

*Da un punto di vista pratico, non assegnare un diametro idraulico equivalente e utilizzare il valore calcolato per l'orificio non è un gran problema poiché negli spigoli pilotanti $R_{e,c} \approx 100$, quindi il moto sarà sempre turbolento e quindi $C_q$ coinciderà sempre con $C_{d,max}$. Dal punto di vista formale, invece non è corretto e andrebbe fornito*. L'errore potrebbe esserci se le portate fossero veramente molto basse, cosa che accade di rado in questi blocchi. A tal proposito si può verificare con la [5-4] che Re sia effettivamente superiore a 100. Se è minore o uguale a 100 allora è necessario provvedere un Dh_array corretto.

Allo stesso modo posso definire un angolo per il getto del fluido:

```
x_array[:] =      { x0, x1, …, xn-1 }
theta_array[:] = { θ0, θ1, …, θn-1 }
```

Lasciando il valore di default ( { 69 } ) verrà assunto 69° per tutto il range di corsa.

Non discutiamo il modello perché è piuttosto elementare.
Le uniche due formule d'interesse sono la forza idraulica che agisce in apertura:

$$F_{hyd} = p_A \cdot \frac{\pi}{4} \cdot \left( {d_{spool}}^2 - {d_r}^2 \right)$$

**6-21: forza in apertura nel risalto di una spola**

Le forze di flusso si calcolano ancora con la [6-14]. A cambiare sono infatti le sezioni effettive aperte $A_e$, che dipendono dalle geometrie differenti.

### 6.7.2 Strategie per la parametrizzazione dell'area

La legge d'area è una funzione della corsa che restituisce la sezione aperta (e nel nostro caso anche Dh e theta). La legge d'area ha sempre una forma:

$$x_array[:] = \{ x_0, x_1, \ldots, x_{n-1} \}$$
$$A_array[:] = \{ A_0, A_1, \ldots, A_{n-1} \}$$

Possono capitare diversi casi,

#### *6.7.2.1 Strategia per il ricoprimento critico*

In tale circostanza, ad uno spostamento molto piccolo δx verso la direzione che ne determina l'apertura, la sezione passa da A(0) = 0, ad A(δx) > 0.
In questo caso la migliore strategia è quella di far coincidere l'apertura dalla x = 0, e quindi, come nell'esempio della precedente pagina:

```
x_array[:] = { 0, 0.5, 1, 1.5, 2, 2.5, 3 , 3.5, 4, 4.5, 5 }
A_array[:] = { 0, 0.25, 1, 2.25, 4, 6.25, 9, 12.25, 16, 20.25, 25 }.
```

Quando il ricoprimento è critico, per la [6-1], $x_{lap}$ coincide in valore assoluto con $x_{a,rel}$ che corrisponde allo spostamento relativo della spola. All'inizio della simulazione (a meno di non forzarne il valore, scelta non consigliata), $x_{a,rel}$ è zero, quindi ad uno spostamente positivo (se FromLeft2Right = true) che fa aumentare x, corrisponde direttamete A_array di quella x, mentre ad uno spostamento negativo (se FromLeft2Right = false) che da diminuire x (x < 0), corrisponde direttamente A_array di –x (x cambiata di segno).

#### *6.7.2.2 Underlap/Overlap gestito con $x_u$*

Possiamo far coincidere l'inizio apertura $A_0$ = 0 (A_Array[0] = 0) con $x_0$ = 0 (x_array[0] = 0), come se fosse un ricoprimento critico (6.7.2.1) , e gestire l'overlap ($x_u$ < 0) o l'underlap ($x_u$ > 0).
Funziona poiché la [6-1] trasla $x_{lap}$ (**la sezione è determinata nella simulazione sempre in funzione di $x_{lap}$**) rispetto a $x_{a,rel}$ di $x_u$. Ad esempio con *FromLeft2Rigt = true*, se $x_u$ = 1 (underlap di 1 = spola già aperta di 1 mm) e la legge d'area è (ad esempio) x_array[:] = { 0, 1, 2 } e A_array[:] = { 0, 4, 5 } abbiamo $x_{lap} = x_{a,rel} + x_u$, quindi con $x_{a,rel}$ = 0 (alla partenza della simulazione) ho già $x_{lap}$ = 1, e l'area di $x_{lap}$ = 1 è 4 $mm^2$ (cioè è già la spola aperta il gisuto). Se fosse invece $x_u$ = -1 (overlap di 1 = spola ancora chiusa di 1 mm) allora abbiamo $x_{lap} = x_{a,rel} + x_u$, quindi con $x_{a,rel}$ = 0 (alla partenza della simulazione) ho già $x_{lap}$ = -1, quindi la sezione è nulla.
Quindi, in pratica, la legge d'area parte dal primo elemento di x_array=0 e dal primo elemento di A_array = 0 ma già dalla x successiva la sezione è aperta. Il ricoprimento è affidato alla variabile $x_u$.

#### *6.7.2.3 No Underlap/Overlap, quindi $x_u = 0$*

L'**overlap** semplicemente si crea con una distanza in cui la sezione sia nulla, es. spola con overlap di 1 mm:

```
x_array[:] = { 0, 1, 2, 3, 4, 5 }
A_array[:] = { 0, 0, 1, 2, 3, 4 }.
```

**Quindi l'overlap inizia sempre con $x_0 = 0$ e $A_0 = 0$.**

L'**underlap** è più complicato. Dobbiamo pensare che se una spola ha ricoprimento negativo, significa che rispetto alla freccia propria (e non a quella della massa), la sezione sarà aperta a x = dimensione del ricoprimento (chiamiamolo **U**, ed è numero positivo perché è una lunghezza), se è aperta in questo punto, vuole dire che è chiusa a $\mathbf{x_0 = -U}$, e quindi $\mathbf{A_0 = 0}$. Quindi, uno step dopo, a $x_1$, sarà aperta $A_1 > A_0$, …, e a $x_{n-1}$ sarà aperta al valore finale sarà aperta $A_{n-1} > A_{n-2}$, es. spola underlap di 5 mm, $x_0$ = -5 e $A_0$ = 0, e poi ad ogni step successivo la sezione aumenterà, quindi:

```
x_array[:] = { -5,-4, -3, -2, -1, 0 }
A_array[:] = { 0, 1, 2, 3, 4, 5, 6  }.
```

**Quindi l'underlap inizia sempre con $x_0$ = -underlap e $A_0 = 0$.**

In corrispondenza di $x_0$ la sezione deve essere zero, *perché l'interpolazione a sinistra di $x_0$ ($x < x_0$) restituisce sempre $A_0$ , e se fosse >0 allora la spola sarebbe sempre aperta*.
In pratica, si mette $x_u$ = 0, e il ricoprimento si gestisce direttamente sulla legge d'area.

Se quindi il ricoprimento è critico, allora la soluzione più naturale è la 1), negli altri casi, sarebbe preferibile provare prima con la 2), perché è più semplice e con una unica legge d'area si possono gestire diversi underlap e overlap modificando $x_u$ al volo. L'altro metodo resta comunque una opportunità, in particolare per un overlap è pure molto semplice.

### 6.7.3 Codice

Ecco il codice:

```
da aggiornare…
model GenericSpool
  extends FPLib.HCD.Interfaces.PartialHCDEdge(Cdmax = 0.7, Rec = 100);
  // Poppet parameters and variables
  parameter Types.Diameter dspool = 10 "Spool diameter";
  parameter Types.Diameter dr = 5 "Rod diameter";
  parameter Types.Position x_array[:] = {0, 0} "x-array = points where area is given";
  parameter Types.Area A_array[:] = {0, 0} "Area as function of x";
  parameter Types.Diameter Dh_array[:] = {0, 0} "Hydraulic diameter as function of x";
  parameter Types.Angle theta_array[:] = {69} "Jet angle as function of x";
  Types.Area Aspool(start = 0) "Area opened by the spool";
  Types.Angle theta "Jet force angle";
protected
  Types.Area Amax = Constants.pi * (dspool ^ 2 - dr ^ 2) / 4 "maximum area";
  Types.Diameter Dh_t;
  Types.Angle theta_t;
equation
// sections increase / decrease volumes (side A and side B)
  VolSezA = Constants.pi * (dspool ^ 2 - dr ^ 2) / 4;
  VolSezB = 0;
// active sections on which the pressures act (side A and side B)
// main outflow section
  assert(Aspool >= 0, "Apoppet minore di zero, ocio!");
  if noEvent(xlap > 0) then
    Aspool = Functions.linterp(xlap, x_array, A_array);
    if size(Dh_array, 1) == size(x_array, 1) then
      Dh_t = Functions.linterp(xlap, x_array, Dh_array);
      if Aspool > Constants.epsilon and Dh_t > Constants.epsilon then
        Dh = Dh_t;
      else
        Dh = 2 * sqrt(Aspool / Constants.pi);
      end if;
    else
      Dh_t = 0;
      Dh = 2 * sqrt(Aspool / Constants.pi);
    end if;
    if size(theta_array, 1) == size(x_array, 1) then
      theta_t = Functions.linterp(xlap, x_array, theta_array);
      if Aspool > Constants.epsilon and theta_t > Constants.epsilon then
        theta = theta_t;
      else
        theta = theta_array[1];
      end if;
    else
      theta_t = 0;
      theta = theta_array[1];
    end if;
    if Aspool <= Amax then
      Ae = Aspool;
    else
      Ae = Amax;
    end if;
  else
    Dh_t = 0;
    theta_t = 0;
    theta = 0;
    Aspool = 0;
    Ae = 0;
    Dh = 0;
  end if;
// hydrostatic force
  Fh = 0.1 * pA * Amax;
// Steady state flow force computation
  Fflow = 2 * kjet * Cd * (1e-6 * Ae) * (1e+5 * abs(dp)) * cos(theta * Constants.pi / 180);
  annotation(
    Icon(graphics = {Rectangle(origin = {145, -64}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, extent = {{-65, 4}, {-
45, -16}}), Polygon(origin = {-60, -58}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{0, -2}, {40, -2}, {40,
-22}, {0, -22}, {0, -2}}), Polygon(origin = {-60, 62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid, points = {{-40, 18},
```

```
{40, 18}, {40, -2}, {-40, -2}, {-40, 18}}), Polygon(origin = {-10, 62}, fillColor = {170, 0, 0}, fillPattern = FillPattern.Solid,
points = {{90, 18}, {110, 18}, {110, -2}, {90, -2}, {90, 18}}), Polygon(origin = {-100, -58}, fillColor = {170, 0, 0}, fillPattern =
FillPattern.Solid, points = {{0, -2}, {20, -2}, {20, -22}, {0, -22}, {0, -2}}), Line(origin = {-151.415, -24.7073}, points = {{96,
84}, {96, 46}}, color = {255, 0, 0}, thickness = 1), Line(origin = {-150.712, -105.12}, points = {{96, 84}, {96, 46}}, color = {255,
0, 0}, thickness = 1), Polygon(fillColor = {255, 255, 0}, fillPattern = FillPattern.HorizontalCylinder, points = {{-24, 60}, {-40,
54}, {-50, 40}, {-54, 20}, {-100, 20}, {-100, -20}, {-54, -20}, {-50, -40}, {-40, -54}, {-24, -60}, {100, -60}, {100, 60}, {-24,
60}}), Polygon(origin = {16, 3}, fillPattern = FillPattern.Solid, points = {{16, 21}, {20, 21}, {22, 31}, {16, 39}, {6, 41}, {0, 37},
{-6, 29}, {-8, 17}, {-12, 3}, {-16, -19}, {-22, -41}, {-16, -41}, {-8, -11}, {-2, 9}, {2, 23}, {6, 33}, {12, 31}, {16, 27}, {16,
21}}), Polygon(origin = {7, -1}, fillPattern = FillPattern.Solid, points = {{-13, 7}, {13, -1}, {9, -7}, {-13, 1}, {-13, 7}}),
Polygon(origin = {29, -25}, fillPattern = FillPattern.Solid, points = {{-13, -11}, {-3, 1}, {-9, 13}, {-3, 13}, {1, 5}, {7, 13}, {13,
13}, {5, 1}, {9, -13}, {5, -13}, {1, -3}, {-1, -3}, {-9, -13}, {-13, -11}})}, coordinateSystem(initialScale = 0.1)));
end GenericSpool;
```

### 6.7.4 Esempio 6.6.4 ma utilizzando GenericSpool e Underlap/Overlap gestito con $x_u$

Nello schema precedente (Figura 6-6), intendiamo rimpiazzare i due blocchi AnularOrificeSpool con due blocchi GenericSpool, al fine di prendere confidenza con questi blocchi.

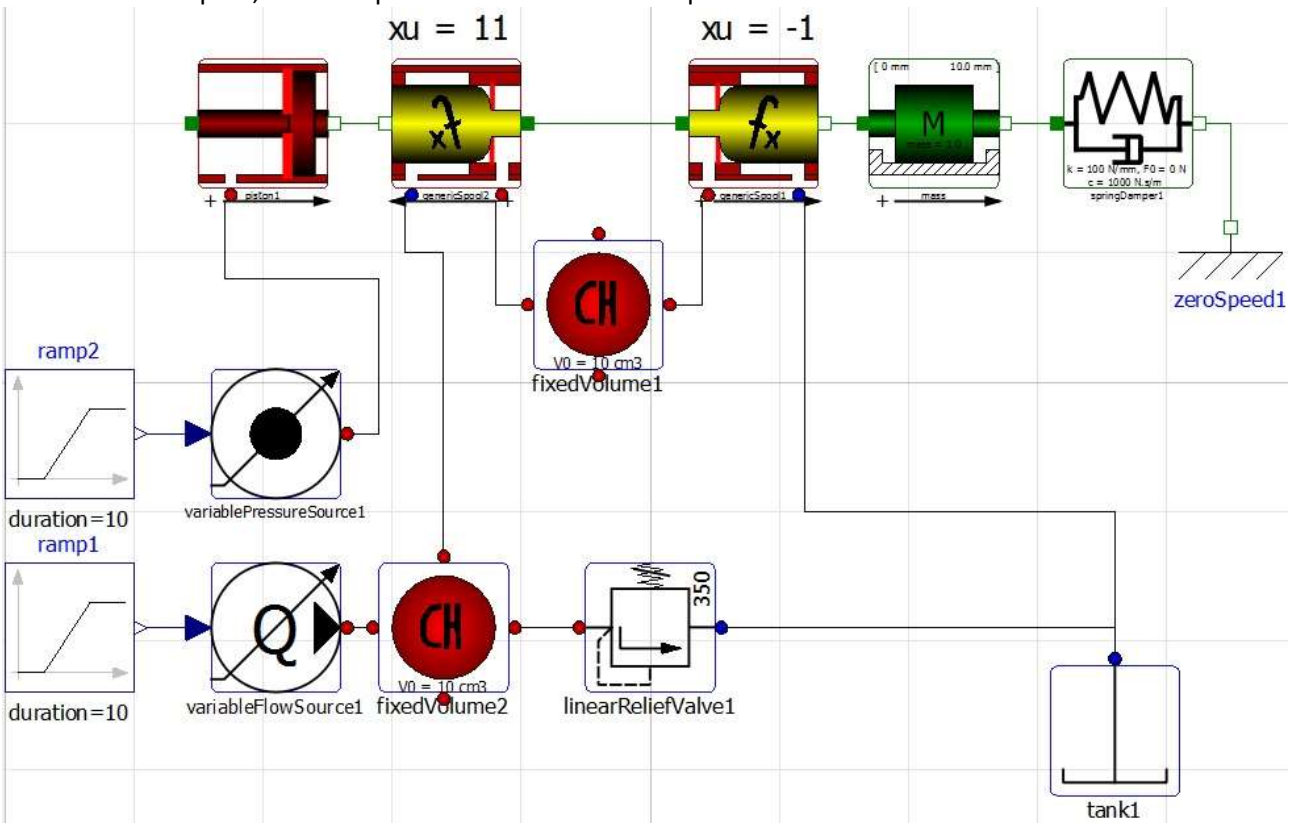



**Figura 6-7: valvola d'esempio con spola generica (GenericSpool) e pistone di pilotaggio**

Chiaramente tutti i parametri dei vari blocchi non cambiano, ad eccezione dei due blocchi anularOrificeSpool che diventano genericSpool1 e genericSpool2:

L'anularOrificeSpool1 di Figura 6-6 (quella a destra) aveva: $d_{spool}$ = 10, $d_r$ = 5, theta = 69°, $x_u$ = -1 (overlap di 1 mm), quindi, ora definiamo genericSpool1 e per l'overlap usiamo il metodo Underlap/Overlap gestito con $x_u$ che consiste nel trattare la legge d'area come se fosse un ricoprimento critico, in questo caso:
La corsa va da 0 a 10 e quindi **`x_array[:] = { 0, 10 }`**.
La sezione varia da $A_0$ = 0 a $A_1 = \pi \cdot d_{spool} \cdot x_1$ = 314.1593 e quindi **`A_Array[:] = { 0, 314.1593 }`**[4].
Per il ricoprimento impostiamo il parametro $x_u$: overlap → segno negativo; è di 1 mm, quindi $x_u$ = -1.
La freccia è nella stessa direzione di quella della massa, quindi **FromLeft2Right = true** (valore di default)

L'anularOrificeSpool2 di Figura 6-6 (quella a sinistra) aveva: $d_{spool}$ = 10, $d_r$ = 5, theta = 69°, $x_u$ = 11 (underlap di 11 mm), quindi, ora definiamo genericSpool2 e per l'overlap usiamo il metodo Underlap/Overlap gestito con $x_u$ che consiste nel trattare la legge d'area come se fosse un ricoprimento critico, in questo caso:
La corsa va da 0 a 10 e quindi **`x_array[:] = { 0, 10 }`**.

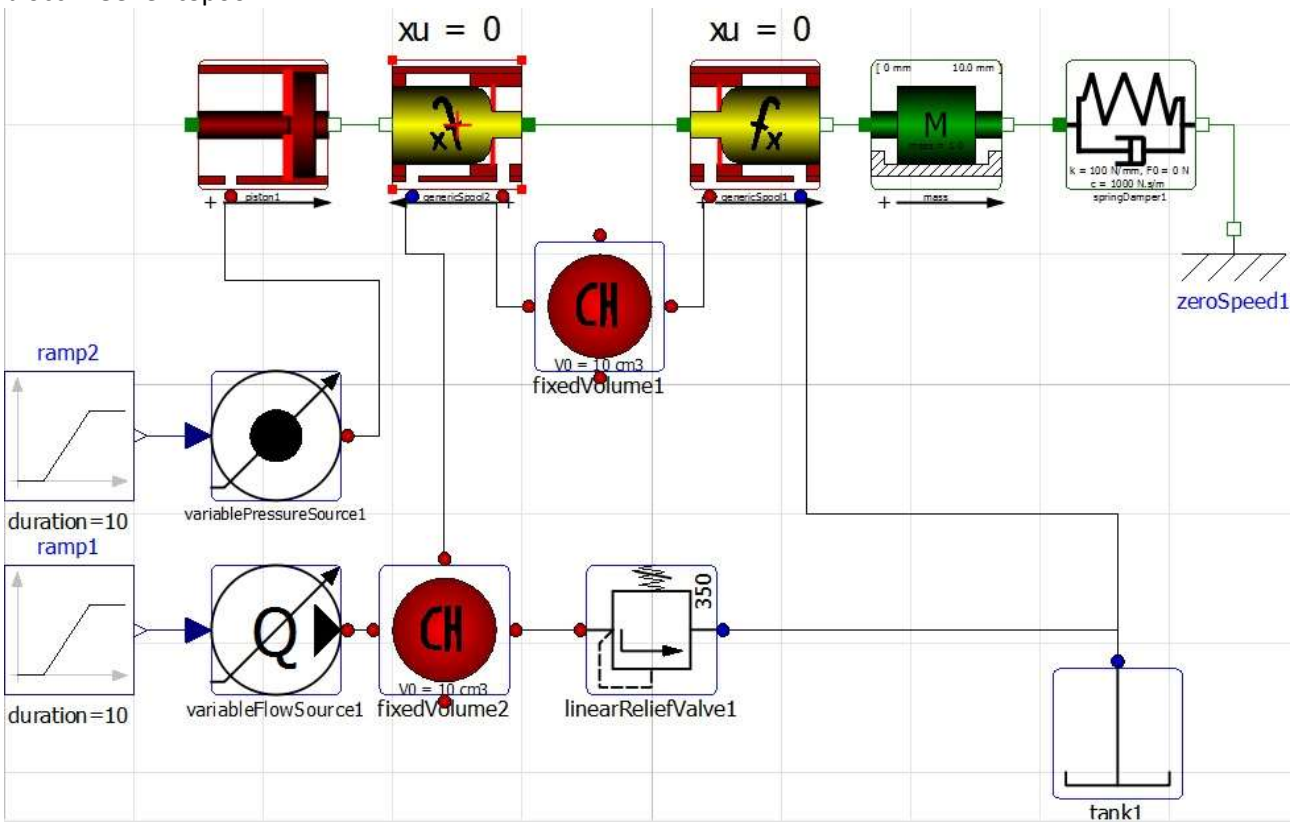
[4] Grazie all'interpolazione lineare, per una legge lineare, basta inserire i due punti estremi.

La sezione varia da $A_0 = 0$ a $A_1 = \pi \cdot d_{spool} \cdot x_1 = 314.1593$ e quindi **`A_Array[:] = { 0, 314.1593 }`**[5].
Per il ricoprimento impostiamo il parametro $x_u$: underla → segno positivo; è di 11 mm, quindi $x_u = 11$.
Poiché la freccia è opposta a quella della massa necessita di impostare **FromLeft2Right = false**.

Ora possiamo lanciare la simulazione e controllare che i risultati siano identici a quelli del paragrafo 6.6.4.

### 6.7.5 Esempio 6.6.4 ma utilizzando GenericSpool e No Underlap/Overlap, quindi xu = 0

Nello schema precedente (Figura 6-7), intendiamo modificare la strategia per impostare i ricoprimenti. Essenzialmente lo schema è identico al precedente, cambia soltanto il modo con il quale definiamo i due blocchi GenericSpool.




Volendo utilizzare la tratega No Underlap/Overlap, quindi $x_u = 0$, intanto settiamo $x_u$ a zero nei due blocchi. Il blocco a destra (genericSpool1) è un overlap di 1 mm, quindi abbiamo visto (6.7.2.3) che è molto facile definire un overlap con questa tecnica, è sufficiente traslare la legge d'area, in questo caso di 1 mm, e quindi:

```
x_array[:] = { 0, 1, 11 }.
A_Array[:] = { 0, 0, 314.1593 }
```

Il blocco a sinistra (genericSpool2) è un underlap di 11 mm, quindi abbiamo visto (6.7.2.3) che è più complicato (ma solo concettualmente), ora abbiamo l'underlap di 11, quindi $x_0 = -11$, poi la corsa è 10 quindi $x_1 = -11 + 10 = -1$. La sezione a $x_0$ è sempre 0, quindi $A_0 = 0$, e uno step dopo (in questo caso tutta la corsa perché è lineare) avrò la sezione più grande e quindi:

```
x_array[:] = { -11, -1, }.
A_Array[:] = { 0, 314.1593 }
```

Ora possiamo lanciare la simulazione e controllare che i risultati siano identici a quelli del paragrafo 6.6.4.

[5] In pratica il metodo Underlap/Overlap gestito con $x_u$ consente di tenere gli stessi x_array e A_array se la legge d'area è ovviamente la stessa, e di lavorare soltanto su $x_u$ per modificare il ricoprimento. Per questo motivo è molto pratico ed è il metodo da preferire per l'impostazione dei ricoprimenti: sarà molto facile modificarli.

# 7 Listato completo

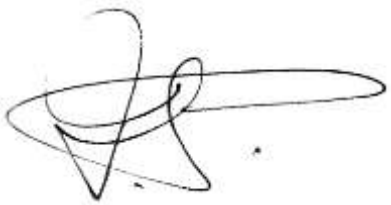
Questo documento è in preview, il listato completo sarà disponibile presto nella sezione per gli utenti registrati. Scusate ma c'è ancora del lavoro da fare…

# 8 Conclusione

Abbiamo visto come usare **Modelica**:

1) Come semplice programma per risolvere sistemi d'equazioni DAE (**Differential-Algebraic Equations**).
2) Come strumento simile ad AMESim per simulare sistemi complessi, composti da vari componenti interconnessi, con **modelli a parametri concentrati**.

Mentre Modelica è ben corredato da librerie di blocchi per la meccanica, la termodinamica, l'elettromagnetismo, purtroppo difetta per la presenza di una buona libreria per l'oleodinamica.
Per questo motivo ho voluto far vedere come sia possibile in modo relativamente semplice costruire una libreria con le proprie mani.
Spero tanto d'aver stimolato la vostra fantasia, con questa lettura.
Grazie a tutti per l'attenzione.



Paolo Ferraresi
pferraresi.box@gmail.com