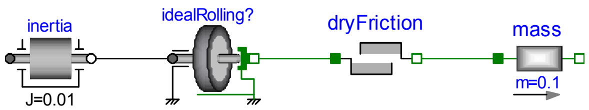
Virtual Physics
Equation-Based Modeling

TUM, November 21, 2017

1D-Mechanical Systems



Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Potential and Flow

Robotics and Mechatronics Centre

For each physical domain, there is a specific pair of effort / flow variables

Domain	Potential	Flow
Translational Mechanics	Velocity: *v* [m/s]	Force: *f* [N]
Rotational Mechanics	Angular Velocity: ω [1/s]	Torque: τ [Nm]
Electrics	Voltage Potential v [V]	Current i [A]
Magnetics	Magnetomotive Force: Θ [A]	Time-derivative of Magnetic Flux: $\dot{\Phi}$ [V]
Hydraulics	Pressure p [Pa]	Volume flow rate $\dot{V}$ [m³/s]
Thermal	Temperature T[K]	Entropy Flow Rate $\dot{S}$ [J/Ks]
Chemical	Chemical Potential: µ [J/mol]	Molar Flow Rate *ν* [mol/s]

Potential and Flow

Robotics and Mechatronics Centre

For the mechanical domain, the first two are relevant:

Domain	Potential	Flow
Translational Mechanics	**Velocity: *v* [m/s]**	**Force: *f* [N]**
Rotational Mechanics	**Angular Velocity: ω [1/s]**	**Torque: τ [Nm]**
Electrics	Voltage Potential v [V]	Current i [A]
Magnetics	Magnetomotive Force: Θ [A]	Time-derivative of Magnetic Flux: $\dot{\Phi}$ [V]
Hydraulics	Pressure p [Pa]	Volume flow rate $\dot{V}$ [m³/s]
Thermal	Temperature T[K]	Entropy Flow Rate $\dot{S}$ [J/Ks]
Chemical	Chemical Potential: µ [J/mol]	Molar Flow Rate *ν* [mol/s]

Potential and Flow

Robotics and Mechatronics Centre

- Each node was represented by a pair of variables

  A **potential** variable

  v (velocity for translational mechanics)

  ω (angular velocity for rotational mechanics)

  and a **flow** variable

  f (force for translational mechanics)

  τ (force for rotational mechanics)

Potential and Flow

- For one connection between a set of n nodes, n equations have to be generated.

- **n-1 equalities**

  Translational: $v_1 = v_2 = \ldots = v_n$

  Rotational: $\omega_1 = \omega_2 = \ldots = \omega_n$

- **1 balance equation**

  Translational: $f_1 + f_2 + \ldots + f_n = 0$

  Rotational: $\tau_1 + \tau_2 + \ldots + \tau_n = 0$

Holonomic Constraints

But the Modelica Standard Library supports different potential variables.

- Not the velocity v but the position s
- Not the angular velocity ω but the angle φ

```
connector Flange_a
  SI.Position s;
  flow SI.Force f
end Flange_a;
```

```
connector Flange_a
  SI.Angle phi;
  flow SI.Torque tau;
end Flange_a;
```

- Why is this? Is our table incorrect?

Holonomic Constraints

- No, the table is correct but the correct formulation of mechanical system adds another requirement:

The formulation of holonomic constraints!

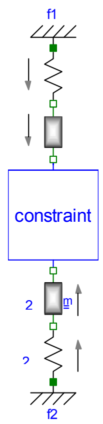
- Holonomic Constraints are algebraic constraints on the level of position.
- A rigid rod describes a given distance between two flanges. Here two positions are bound with one constraint equation.
- In order, to formulate such equations correctly, the position needs to be part of the connector.

Holonomic Constraints: Example

- Let us model a simple system:
- Two masses connected to springs.
- The position s_1 and s_2 are connected by the following holonomic constraint:

$$s_1 = |s_2| * s_2$$

- Such non-linear constraints are rare in 1D systems but common in multidimensional systems.



Holonomic Constraints: Example

Robotics and Mechatronics Centre

- The constraint on positional level:

$$s_1 = |s_2| \cdot s_2$$

- implies for the velocity:

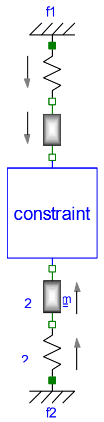
$$v_1 = 2|s_2| \cdot v_2$$

- Power conservation states

$$v_1 \cdot f_1 + v_2 \cdot f_2 = 0$$

- or

$$f_1 \cdot 2|s_2| + f_2 = 0$$



thanks to Edo Drenth for the power analysis

Holonomic Constraints: Example

Robotics and Mechatronics Centre

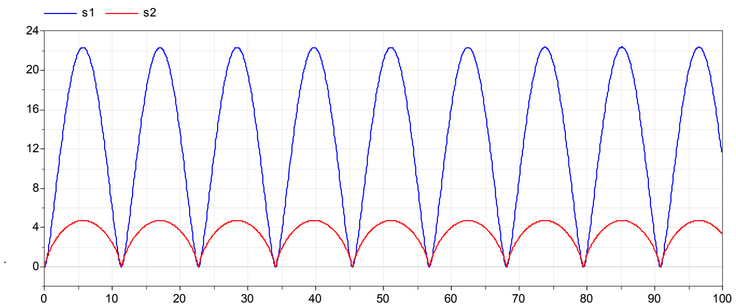
- Here is a handwritten Modelica-code for this example:
- The two variables **s1_int** and **s2_int** are used to formulate the constraints.
- On the next slide you see the simulation result (the positions of the two masses).

```
model TwoSpringsWithConstraint
  Real s1;
  Real s2;
  Real v1;
  Real v2;
  Real f;
  parameter Real m1 = 10;
  parameter Real m2 = 2;
  Real s1_int;
  Real s2_int;
equation
  v1 = der(s1);
  v2 = der(s2);
  -1*s1 + f =  m1*der(v1);
  -20*(s2-5) - f*abs(s2_int)*2
   = m2*der(v2);

  s1 = s1_int;
  s2 = s2_int;
  s1_int = abs(s2_int)*s2_int;
end TwoSpringsWithConstraint;
```


Holonomic Constraints: Example

Robotics and Mechatronics Centre

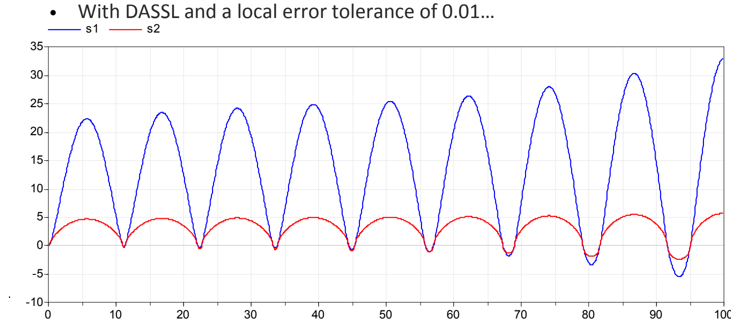
- With DASSL and a local error tolerance of 0.0001:



Holonomic Constraints: Example

Robotics and Mechatronics Centre

- With DASSL and a local error tolerance of 0.01…

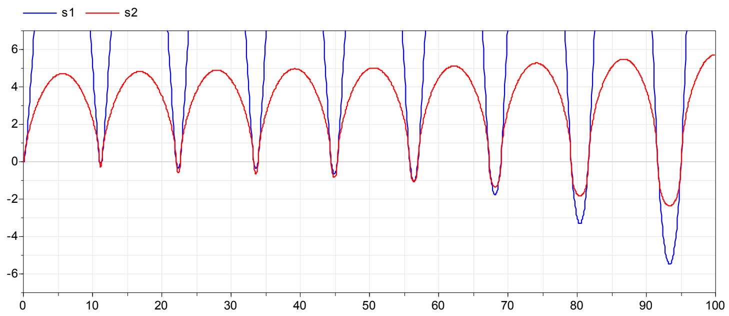


- … the conservation of energy is violated by numerical integration.

Holonomic Constraints: Example

Robotics and Mechatronics Centre

- With DASSL and a local error tolerance of 0.01…



- However, the holonomic constraint does hold up.

Holonomic Constraints: Example

Robotics and Mechatronics Centre

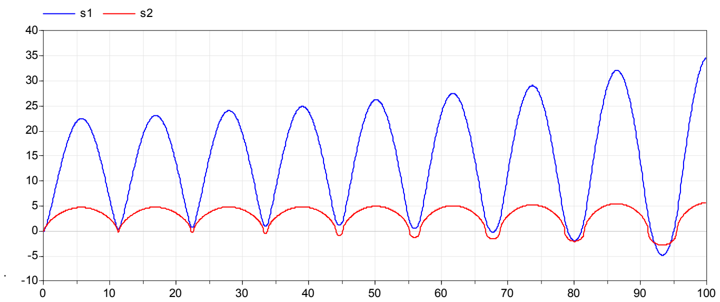
- But couldn't we formulate the same system, using just the velocities **v1** and **v2** instead of the positions **s1** and **s2**?
- We could formulate s1_int and s2_int as integrals for v1 and v2.
- Here is why not: (using DASSL with tolerance 0.01):

```
model TwoSpringsWithConstraint
  Real s1;
  Real s2;
  Real v1;
  Real v2;
  Real f;
  parameter Real m1 = 10;
  parameter Real m2 = 2;
  Real s1_int;
  Real s2_int;
equation
  v1 = der(s1);
  v2 = der(s2);
  -1*s1 + f =  m1*der(v1);
  -20*(s2-5) - f*abs(s2_int)*2
   = m2*der(v2);

  v1 = der(s1_int);
  v2 = der(s2_int);
  s1_int = abs(s2_int)*s2_int;
end TwoSpringsWithConstraint;
```


Holonomic Constraints: Example

Robotics and Mechatronics Centre

- With DASSL and a local error tolerance of 0.01…

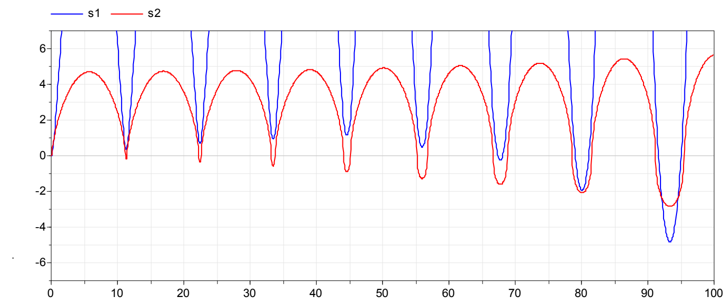


- … the error has gotten worse and…

Holonomic Constraints: Example

Robotics and Mechatronics Centre

- With DASSL and a local error tolerance of 0.01…



- …the holonomic constraint is lost.
- There is now a shift between the bodies.

Holonomic Constraints: Example

- What has happened? Why does the system behave differently?
- Since s1 and s1_int are not algebraically coupled, they are separately integrated.
- The same holds for s2 and s2_int.
- Hence, the holonomic constraints becomes subject to an increasing numerical integration error.
- This can drastically change the systems behavior.

```
model TwoSpringsWithConstraint
 Real s1;
  Real s2;
  Real v1;
  Real v2;
  Real f;
  parameter Real m1 = 10;
  parameter Real m2 = 2;
  Real s1_int;
  Real s2_int;
equation
  v1 = der(s1);
  v2 = der(s2);
  -1*s1 + f =  m1*der(v1);
  -20*(s2-5) - f*abs(s2_int)*2
   = m2*der(v2);

  v1 = der(s1_int);
  v2 = der(s2_int);
  s1_int = abs(s2_int)*s2_int;
end TwoSpringsWithConstraint;
```


Holonomic Constraints: Example

- What has happened? Why does the system behave differently?
- Since s1 and s1_int are not algebraically coupled, they are separately integrated.
- The same holds for s2 and s2_int.
- Hence, the holonomic constraints becomes subject to an increasing numerical integration error.
- This can drastically change the systems behavior.
- So… DON'T!

```
model TwoSpringsWithConstraint
 Real s1;
  Real s2;
  Real v1;
  Real v2;
  Real f;
  parameter Real m1 = 10;
  parameter Real m2 = 2;
  Real s1_int;
  Real s2_int;
equation
  v1 = der(s1);
  v2 = der(s2);
  -1*s1 + f =  m1*der(v1);
  -20*(s2-5) - f*abs(s2_int)*2
   = m2*der(v2);

  v1 = der(s1_int);
  v2 = der(s2_int);
  s1_int = abs(s2_int)*s2_int;
end TwoSpringsWithConstraint;
```


Holonomic Constraints

- For our mechanical components, this means that we have to use positions as potential variables:
- Each node was represented by a pair of variables

  A **potential** variable

  s (position for translational mechanics)

  φ (angle for rotational mechanics)

  and a **flow** variable

  f (force for translational mechanics)

  τ (force for rotational mechanics)

Potential and Flow

- We see that the new potential equations imply the old ones:
- **n-1 equalities**

  Translational: $s_1 = s_2 = \ldots = s_n$ implies $v_1 = v_2 = \ldots = v_n$

  Rotational: $\varphi_1 = \varphi_2 = \ldots = \varphi_n$ implies $\omega_1 = \omega_2 = \ldots = \omega_n$
- **1 balance equation**

  Translational: $f_1 + f_2 + \ldots + f_n = 0$

  Rotational: $\tau_1 + \tau_2 + \ldots + \tau_n = 0$

The information about the energy flow is still contained in our connector variables!

Dampers

Now we can model the components: The dampers

$\Delta v \cdot D = f$

$\Delta v = d(s_2 - s_1)/dt$

$f = f_2$

$0 = f_1 + f_2$

$\Delta\omega \cdot D = \tau$

$\Delta\omega = d(\varphi_2 - \varphi_1)/dt$

$\tau = \tau_2$

$0 = \tau_1 + \tau_2$

Dampers

Robotics and Mechatronics Centre

Now we can model the components: The dampers

$\Delta v \cdot D = f$

This is totally fine

$\Delta\omega \cdot D = \tau$

$\boxed{\Delta v = d(s_2 - s_1)/dt \qquad \Delta\omega = der(\varphi_2 - \varphi_1)/dt}$

$f = f_2$

$\tau = \tau_2$

$0 = f_1 + f_2$

$0 = \tau_1 + \tau_2$

The derivatives are computed symbolically not numerically

Springs

Robotics and Mechatronics Centre

The springs: Since the new formulation is based on the positions, the model does not own a derivative anymore.

$\Delta s \cdot C = f$

$\Delta s = (s_2 - s_1) - s0$

$f = f_2$

$0 = f_1 + f_2$

$\Delta\varphi \cdot C = \tau$

$\Delta\varphi = (\varphi_2 - \varphi_1) - \varphi_0$

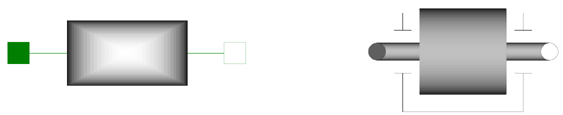
$\tau = \tau_2$

$0 = \tau_1 + \tau_2$

Mass and Inertia

Robotics and Mechatronics Centre

Whereas the spring components have lost their integrator, the mass and inertia have gained one:



$f = M \cdot dv/dt$

$v = ds_1/dt$

$s_2 = s_1$

$f = f_1 + f_2$

$\tau = I \cdot d\omega/dt$

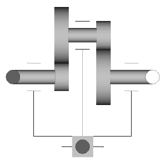
$\omega = d\varphi_1/dt$

$\varphi_2 = \varphi_1$

$\tau = \tau_1 + \tau_2$

Transformers

The transformer is represented by a gearbox. Its equation has hardly changed.

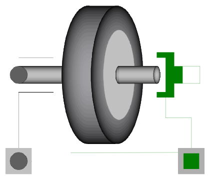


$$\varphi_2 = \text{Ratio} \cdot \varphi_1$$

$$\tau_1 = \text{Ratio} \cdot \tau_2$$

Transformers

An ideal rolling wheel represents a transformation between translational and rotational movement.



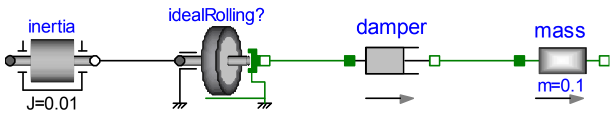
Ideal rolling means that the velocity of the virtual contact point is zero. The virtual contact point is located on the wheel.

$$\text{Radius} \cdot \varphi = s$$

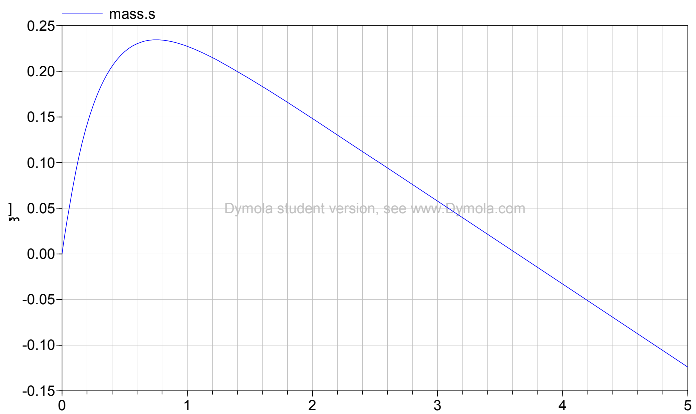
$$\tau = \text{Radius} \cdot f$$

Ball with counter spin

- Finally, let us model a simple mechanical system.
- A ball is placed on a table and propelled forwards with reverse spin. Eventually the spin will decelerate the ball and force him to roll backwards.
- Here is a first model of such a system.

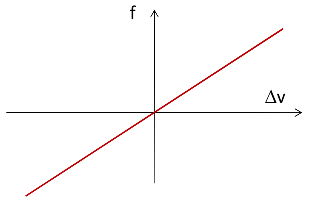


Ball with counter spin



Dry Friction

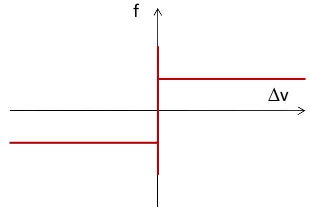
- The damper generates a friction force that is proportional to the difference in velocity.



- The damper is not a good friction model. It is too “smooth”.
- Instead we want to use a dry friction model instead.

Dry Friction

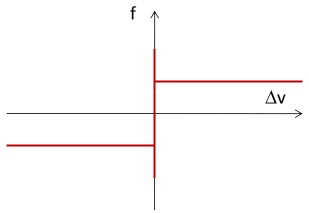
- The characteristic curve for dry friction is a multi-valued function and hence very tricky.



- The adhesive friction (“stiction”) is stronger than dry friction while sliding. The friction force always counteracts the movement.
- Hence, the curve contains discontinuities and represents infinite stiffness.
- The curve can also not be properly described by a mathematical function.

Dry Friction

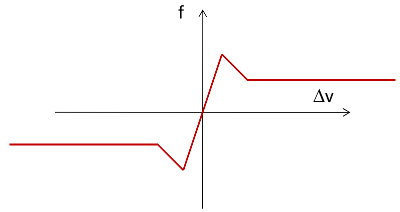
- Hence, the dry friction model of Modelica is pretty complicated and contains many language elements we do not know yet.



- Since we cannot cope with these discontinuities yet, we try to avoid them.
- We do so be regularizing the characteristic curve.

Dry Friction: Regularization

- To this end, we “stretch” the curve and transform it into a piecewise linear function.

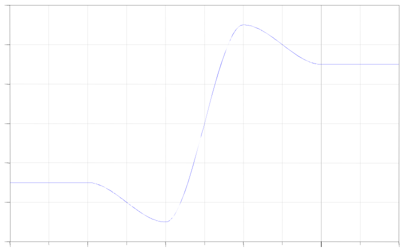


- The cost of this approach is: loss of precision and/or artificial stiffness.

Dry Friction: Regularization

Robotics and Mechatronics Centre

- Instead of generating a piecewise linear function, we can also compose the function using three S-functions and two constant functions.

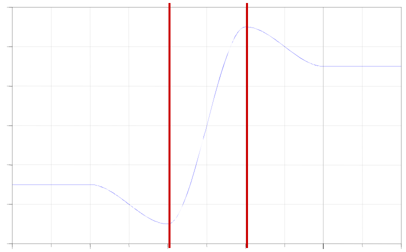


- The result is a nicely differentiable function.

Dry Friction: Regularization

Robotics and Mechatronics Centre

- Instead of generating a piecewise linear function, we can also compose the function using three S-functions.



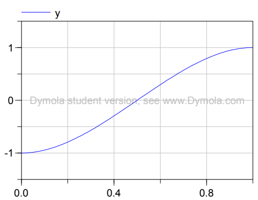
- The result is a nicely differentiable function.

Dry Friction: S-Function

Robotics and Mechatronics Centre

- For the S-Function, we use a polynomial:

$y = -x^3/2 + 3x/2$




```
function S_Func

  input Real x;
  output Real y;




algorithm

  if x > 1 then
    y := 1;
  elseif x < -1 then
    y := -1;
  else
    y := -0.5*x^3 + 1.5*x;
  end if;




end S_Func;
```


Dry Friction: S-Function

Robotics and Mechatronics Centre

- For the S-Function, we use a polynomial:

$y = -x^3/2 + 3x/2$

- Then, we provide inputs in order to scale the function to fit an arbitrary rectangle

(x_min, y_min, x_max, y_max)

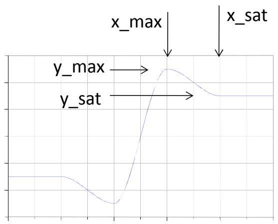
- The annotation tells Dymola that the function is differentiable once. So they are no discontinuities.
- This is important for the ODE-solver.

```
function S_Func "Models an S-Function"
  input Real x_min;
  input Real x_max;
  input Real y_min;
  input Real y_max;
  input Real x;
  output Real y;
protected
  Real x2;

algorithm
  x2 := x - x_max/2 - x_min/2;
  x2 := x2*2/(x_max-x_min);
  if x2 > 1 then
    y := 1;
  elseif x2 < -1 then
    y := -1;
  else
    y := -0.5*x2^3 + 1.5*x2;
  end if;
  y := y*(y_max-y_min)/2;
  y := y + y_max/2 + y_min/2;
  annotation(smoothOrder=1);
end S_Func;
```

Dry Friction: Triple S-Function

Robotics and Mechatronics Centre

- We may use the S-Function in order to compose the point-symmetric Triple S-Function:




```
function TripleS_Func

  input Real x_max;
  input Real x_sat;
  input Real y_max;
  input Real y_sat;

  input Real x;
  output Real y;

algorithm
  if x > x_max then
    y := S_Func(x_max,x_sat,
                y_max,y_sat,x);
  elseif x < -x_max then
    y := S_Func(-x_max,-x_sat,
                -y_max,-y_sat,x);
  else
    y := S_Func(-x_max,x_max,-
                y_max,y_max,x);
  end if;

  annotation(smoothOrder=1);
end TripleS_Func;
```


Dry Friction Model

Robotics and Mechatronics Centre

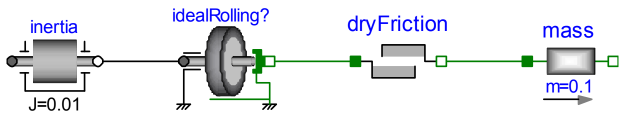
- Now we can model our own dry friction component:

```
model DryFriction
  extends Modelica.Mechanics.
    Translational.Interfaces.
    PartialCompliantWithRelativeStates;

  import SI = Modelica.SIunits;

  parameter SI.Force N
  "normal force";
  parameter SI.Velocity vAdhesion
  "adhesion velocity";
  parameter SI.Velocity vSlide
  "sliding velocity";
  parameter Real mu_A
  "friction coefficient at adhesion";
  parameter Real mu_S
  "friction coefficient at sliding";

equation
  f =  N*TripleS_Func(vAdhesion,vSlide,
                      mu_A,mu_S,v_rel);

end DryFriction;
```


Counter spin and dry friction

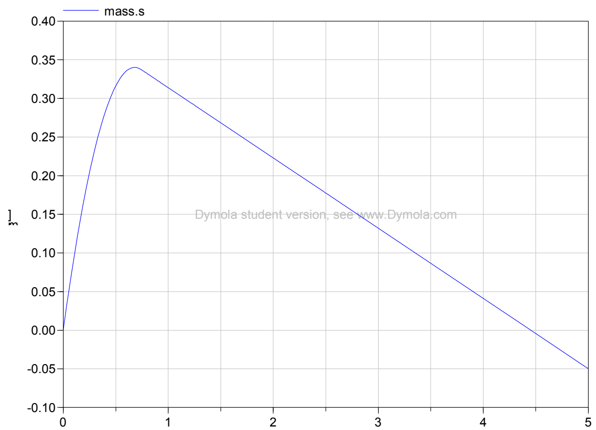
Robotics and Mechatronics Centre

- Here is the application of our dry-friction component.



Counter spin and dry friction

Robotics and Mechatronics Centre



Summary

Robotics and Mechatronics Centre

- Rotational and translational mechanics can be treated he same way.
- The proper formulation of mechanical systems requires the formulation of holonomic constraints.
- In order to enable this, positions and not velocities form the potential connector variables.
- Consequently, the derivatives are redistributed within the components.
- We learnt about dry friction and regularization.

Questions ?