Dr. Dirk Zimmer

Virtual Physics

21.10.2014

Exercise 2: Doing it the hard way

Solution

Task 1: Punch the equations into Modelica and simulate the model using Dymola

Here is the Modelica Code

- φ is replaced by phi
- ω is replaced by w
- α is replaced by z
- τ is replaced by t

```
model Exercise2
  parameter Real MS = 250;
  parameter Real MP = 70;
  parameter Real R = 2.5;
  parameter Real I = MP*R^2;
  parameter Real G = -9.81;
  Real s;
  Real v;
  Real aS;
  Real fS;
  Real phi;
  Real w;
  Real z;
  Real t;
  Real fn;
  Real fz;
  Real fP;
  Real aP;
initial equation
  s = 0;
  v = 0;
  phi = 1.25;
  w = 0;
equation
  aS = der(v);
  v = der(s);
  fS = MS*aS;
  z = der(w);
  w = der(phi);
  t = I * z;
  t = fn*R;
  fn = MP*(sin(phi)*G - cos(phi)*aP);
  fz = MP*R*w^2;
  fP + sin(phi)*fz - cos(phi)*fn - MP*aP =  0;
  aP = aS;
  fP + fS = 0;
end Exercise2;
```

Task 2: Generate the simulation code by yourself.

Now let us derive the state-space form. We can do that partly in Modelica. First, we separate the differential equations. By this, we see that s, v, phi, w form the states of our system. These variables can be supposed to be known. Hence, `fz` can be directly computed by the term: `MP*R*w^2`;

The remaining 7 equations can be simplified, we substitute `aS` and `aP` by a and remove the equation `aS = aP`. We also substitute `fP` by `–fS` and remove the equation: `fP + fS = 0`.

```
a = der(v);
v = der(s);
z = der(w);
w = der(phi);
fz = MP*R*w^2;

fS = MS*a;
t = I * z;
t = fn*R;
fn = MP*(sin(phi)*G - cos(phi)*a);
-fS + sin(phi)*fz - cos(phi)*fn - MP*a =  0;
```

This model still leads to the same simulation result. We can simplify it further. We substitute away all forces (except the already determined force `fz`). To this end, we replace `fS` by `MS*a` and `t` by `I*z`. It results that `I*z = fn*R`. Hence we can substitute any occurrence of `fn` by `I/R*z` or better: `MP*R*z`. Now a system of two equations remains to be solved for `a` and `z`.

```
a = der(v);
v = der(s);
z = der(w);
w = der(phi);
fz = MP*R*w^2;

R*z = sin(phi)*G - cos(phi)*a;
-MS*a + sin(phi)*fz - cos(phi)*MP*R*z - MP*a =  0;
```

If we substitute `R*z` by `(sin(phi)*G - cos(phi)*a)`, we get:

```
-MS*a + sin(phi)*fz - cos(phi)*MP* (sin(phi)*G - cos(phi)*a) - MP*a =  0;
```

This equation only depends on state-variables or from variables that can be directly derived out of the stage (`fz`). It can be solved for `a`:

```
a = (sin(phi)*fz - cos(phi)*sin(phi)*MP*G) / (MS + MP*(1-cos(phi)^2));
```

`z` is now simply determined by backward substitution as:

```
z = (sin(phi)*G - cos(phi)*a)/R;
```

We have transformed the equations into state-space form. Given the state-vector (`s,v,phi,w`), we can compute the derivatives by the following causal assignments:

```
fz := MP*R*w*w;
a := (sin(phi)*fz - cos(phi)*sin(phi)*MP*G) / (MS + MP*(1-cos(phi)*cos(phi)));
z := (sin(phi)*G - cos(phi)*a)/R;
der(v) := a;
der(s) := v;
der(w) := z;
der(phi) := w;
```

Applying the Forward Euler discretization scheme leads to the following Python code:

```python
#!/usr/bin/env python3
# Author Dirk Zimmer (c) 2011

from math import *

#Setting the parameters
MS = 250.0   #mass of the motocycle [kg]
MP = 70.0    #mass of the swing [kg]
R = 2.5      #Radius of the swing [m]
G = -9.81    #Gravity acceleration

phi0 = 1.25  #Initial elongation [rad]

h = 0.001    #time-step of forward Euler integration [s]
tStop = 5    #stop time [s]

#Setting the initial values
s = 0
v = 0
phi = phi0
w = 0
time = 0

#open file for ouput
fh = open("out.dat","w")

#perform time-integration
while time < tStop:
	fz = MP*R*w*w;
	a = (sin(phi)*fz-cos(phi)*sin(phi)*MP*G)/(MS+MP*(1-cos(phi)*cos(phi)));
	z = (sin(phi)*G - cos(phi)*a)/R;

	dv_dt = a
	ds_dt = v
	dw_dt = z
	dphi_dt = w

	v += h*dv_dt
	s += h*ds_dt
	w += h*dw_dt
	phi += h*dphi_dt

	time += h
	print(time,"\t",v,"\t",w,file=fh)

print("See out.dat for simulation result")

fh.close()
```