Int 1: Governing Equations

In[1]:=
```
SetDirectory@NotebookDirectory[];
<< "../MMA library.m"
```

Electric Skateboards

In[20]:=
```
With[{context = "es`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[20]= p3`

Step 2: Parameters

In[125]:=
```
params = <| m → 1, g → 9.8, motor[t] → -r[t], i → 3 |>
```

Out[125]= 〈| m → 1, g → 9.8, motor[t] → -r[t], i → 3 |〉

Step 3: Define forces

In[126]:=
```
fn = normal[t] θhat;
fmotor = motor[t] rhat;
fg = m g RotationMatrix[θ[t]].(-rhat);
```

In[129]:=
```
(* Torques on ramp *)
tn = -r[t] normal[t];
```

Step 4: Equations of motion

In[130]:=
```
(* Mass motion *)
eq1 = fn + fmotor + fg == m polaraccel[];
(* Ramp rotation *)
eq2 = tn == i θ''[t]

eqns = Append[splitVectorEqn[eq1], eq2]
```

Out[131]= -normal[t] r[t] == i θ''[t]

Out[132]= {-g m Cos[θ[t]] + motor[t] == m (-r[t] θ'[t]^2 + r''[t]),
normal[t] - g m Sin[θ[t]] == m (2 r'[t] θ'[t] + r[t] θ''[t]), -normal[t] r[t] == i θ''[t]}

Step 5: Solve

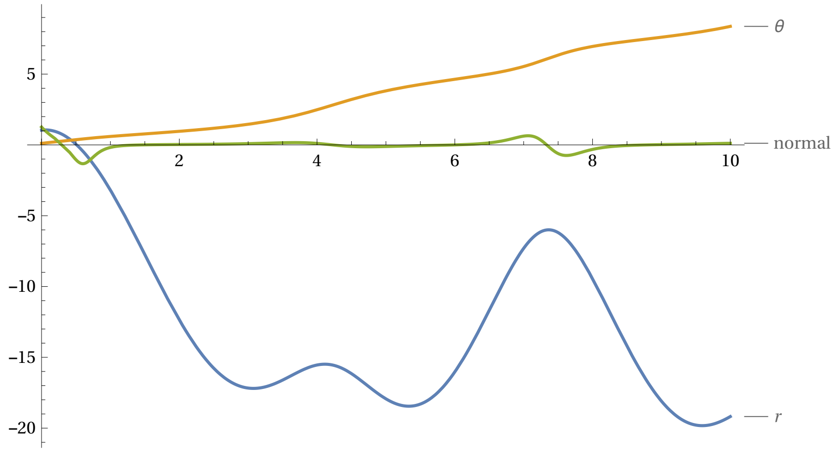
In[138]:=
```
sol = NDSolveValue[eqns /. params, {r[t], θ[t], normal[t]}, {t, 0, 10}];
```

NDSolveValue: Some of the functions have zero differential order, so the equations will be solved as a system of differential–algebraic equations.

NDSolveValue: Structural analysis indicates that 4 initial conditions are needed to fix the state of the system. Currently only 0 initial conditions are specified. NDSolve may return one of a family of solutions.

In[135]:=
```
Plot[Evaluate@sol, {t, 0, 10}, PlotLabels → {r, θ, normal}]
```

Out[135]=



Cleanup

In[139]:=
```
With[{context = "es`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[139]= p3`

Basic pendulum

In[140]:=
```
With[{context = "p1`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[140]= p3`

Step 2: Parameters

In[158]:=
```
params = <| m → 1, g → 9.8 , l → 2|>
```

Out[158]= ⟨|m → 1, g → 9.8, l → 2|⟩

Step 3: Define forces

In[182]:=
```
fg = m g RotationMatrix[θ[t]].(-rhat);
ftension = tension[t] rhat;
```

Step 4: Equations of motion

In[184]:=
```
(* Mass motion *)
eq1 = fg + ftension == m polaraccel[];
eqns = splitVectorEqn[eq1]
```

Out[185]= {-g m Cos[θ[t]] + tension[t] == m (-r[t] θ′[t]^2 + r″[t]),
 -g m Sin[θ[t]] == m (2 r′[t] θ′[t] + r[t] θ″[t])}

```
(* Constraints *)
```

In[194]:=
```
constraint = r[t] == l;
```

Step 5: Solve

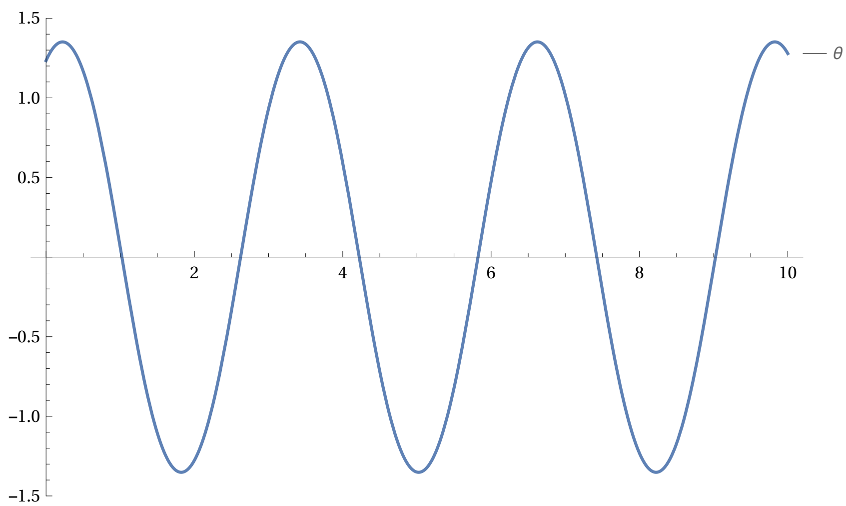
In[189]:=
```
sol = NDSolveValue[{eqns, constraint} /. params, {θ[t]}, {t, 0, 10},
  Method → {"IndexReduction" → {"Pantelides", "ConstraintMethod" → "Projection"}}]
```

NDSolveValue: Structural analysis indicates that 2 initial conditions are needed to fix the state of the system. Currently only 0 initial conditions are specified. NDSolve may return one of a family of solutions.

Out[189]=
```
{InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ][t]}
```

In[192]:=
```
Plot[Evaluate@sol, {t, 0, 10}, PlotLabels → {θ}]
```

Out[192]=



Cleanup

In[197]:=
```
With[{context = "p1`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[197]= p3`

Segway pendulum

In[3]:=
```
With[{context = "p2`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[3]= p3`

Step 2: Parameters

In[4]:=
```
params = <| m → 1, g → 9.8 , l → 2|>
```

Out[4]= ⟨| m → 1, g → 9.8, l → 2 |⟩

Step 3: Define forces

In[5]:=
```
xhat = RotationMatrix[-θ[t]].(-θhat);
```

In[6]:=
```
fg = m g RotationMatrix[θ[t]].(-rhat);
ftension = tension[t] rhat;
```

Step 4: Equations of motion

In[8]:=
```
(* Mass motion *)
eq1 = fg + ftension == m (polaraccel[] + x''[t] xhat);
eqns = splitVectorEqn[eq1]
```

Out[9]=
```
{-g m Cos[θ[t]] + tension[t] == m (-r[t] θ′[t]^2 + r″[t] - Sin[θ[t]] x″[t]),
 -g m Sin[θ[t]] == m (2 r′[t] θ′[t] - Cos[θ[t]] x″[t] + r[t] θ″[t])}
```

In[10]:=
```
(* Constraints *)
```

In[11]:=
```
constraint = r[t] == l;
```

In[12]:=
```
(* Driving functions *)
```

In[13]:=
```
driver = x[t] == Sin[2 t];
```

Step 5: Solve

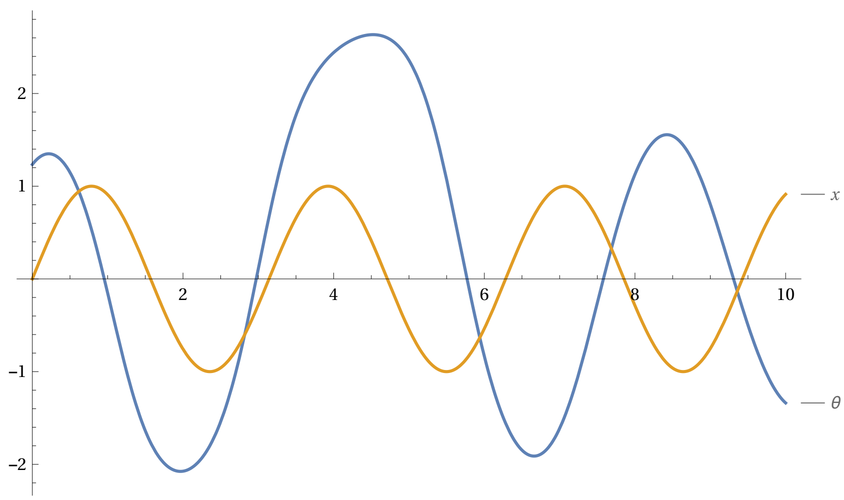
In[14]:=
```
sol = NDSolveValue[{eqns, constraint, driver} /. params, {θ[t], x[t]}, {t, 0, 10},
  Method → {"IndexReduction" → {"Pantelides", "ConstraintMethod" → "Projection"}}]
```

NDSolveValue: Structural analysis indicates that 2 initial conditions are needed to fix the state of the system. Currently only 0 initial conditions are specified. NDSolve may return one of a family of solutions.

Out[14]=
{InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar][t],

InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar][t]}

In[15]:=
```
Plot[Evaluate@sol, {t, 0, 10}, PlotLabels → {θ, x}]
```

Out[15]=


Cleanup

In[16]:=
```
With[{context = "p2`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[16]= p3`

Wheeled segway

In[39]:=
```
With[{context = "p3`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[39]=
```
p3`
```

Step 2: Parameters

In[40]:=
```
params = <| m1 → 1, m2 → 2, g → 9.8 , l → 2|>
```

Out[40]=
```
<|m1 → 1, m2 → 2, g → 9.8, l → 2|>
```

Step 3: Define forces

In[41]:=
```
xhat = RotationMatrix[-θ[t]].(-θhat);
```

In[42]:=
```
fg = m1 g RotationMatrix[θ[t]].(-rhat);
ftension = tension[t] rhat;
fdrive = drive[t] xhat;
```

Step 4: Equations of motion

In[74]:=
```
(* Mass motion *)
eq1 = fg + ftension == m1 (polaraccel[] + x''[t] xhat);
(* Base motion *)
eq2 = -ftension + fdrive == m2 x''[t] xhat;
eqns = Join[splitVectorEqn[eq1], splitVectorEqn[eq2]]
```

Out[76]=
```
{-g m1 Cos[θ[t]] + tension[t] == m1 (-r[t] θ'[t]^2 + r''[t] - Sin[θ[t]] x''[t]),
 -g m1 Sin[θ[t]] == m1 (2 r'[t] θ'[t] - Cos[θ[t]] x''[t] + r[t] θ''[t]),
 -drive[t] Sin[θ[t]] - tension[t] == -m2 Sin[θ[t]] x''[t],
 -Cos[θ[t]] drive[t] == -m2 Cos[θ[t]] x''[t]}
```

In[28]:=
```
(* Constraints *)
```

In[48]:=
```
constraint = r[t] == l;
```

In[30]:=
```
(* Driving functions *)
```

In[56]:=
```
driver = {x[t] == 0};
```

Step 5: Solve

In[95]:=
```
sol = NDSolveValue[{eqns, constraint, driver} /. params, {θ[t], x[t]}, {t, 0, 10},
  Method → {"IndexReduction" → {"Pantelides", "ConstraintMethod" → "Projection"}}]
```

NDSolveValue: There are fewer dependent variables, {r[t], tension[t], x[t], θ[t]}, than equations, so the system is overdetermined.

Out[95]=
```
NDSolveValue[{{{-9.8 Cos[θ[t]] + tension[t] == -r[t] θ'[t]^2 + r''[t] - Sin[θ[t]] x''[t],
    -9.8 Sin[θ[t]] == 2 r'[t] θ'[t] - Cos[θ[t]] x''[t] + r[t] θ''[t],
    -tension[t] == -2 Sin[θ[t]] x''[t], 0 == -2 Cos[θ[t]] x''[t]},
   r[t] == 2, {x[t] == 0}}, {θ[t], x[t]}, {t, 0, 10},
 Method → {IndexReduction → {Pantelides, ConstraintMethod → Projection}}]
```

In[58]:=
```
Plot[Evaluate@sol, {t, 0, 10}, PlotLabels → {θ}]
```

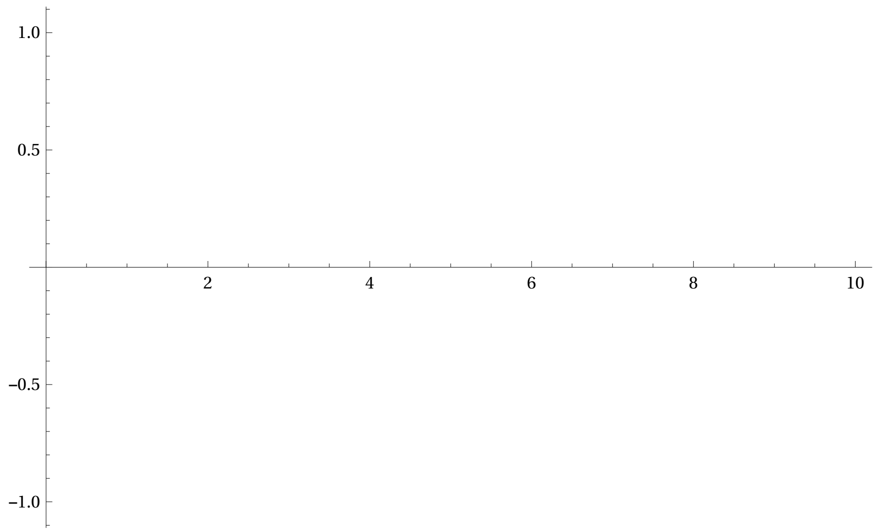
NDSolveValue: 0.0002042857142857143` cannot be used as a variable.

NDSolveValue: 0.0002042857142857143` cannot be used as a variable.

NDSolveValue: 0.20428591836734694` cannot be used as a variable.

General: Further output of NDSolveValue::dsvar will be suppressed during this calculation.

Out[58]=



Cleanup

In[34]:=
```
With[{context = "p3`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[34]= p3`

Scratch Work

In[96]:=
```
exportNotebookPDF[]
```

In[193]:=
```
Rotate[{0, 1}, Pi / 3]
```

Out[193]= {0, 1}

Step 5: Constraints

In[60]:=
```
constraint = r[t] == 1
```

Out[60]= r[t] == 1

In[67]:=
```
Module[{r = r},
 r[t_] := l;
 Simplify[eqns]
]
```

Out[67]= {g m Cos[θ] == motor[t] + l m θ′[t]^2, normal[t] == m (g Sin[θ] + l θ″[t])}