Solid Pendulum with a Spring Mass System

Initialization Code (optional)

Manipulate

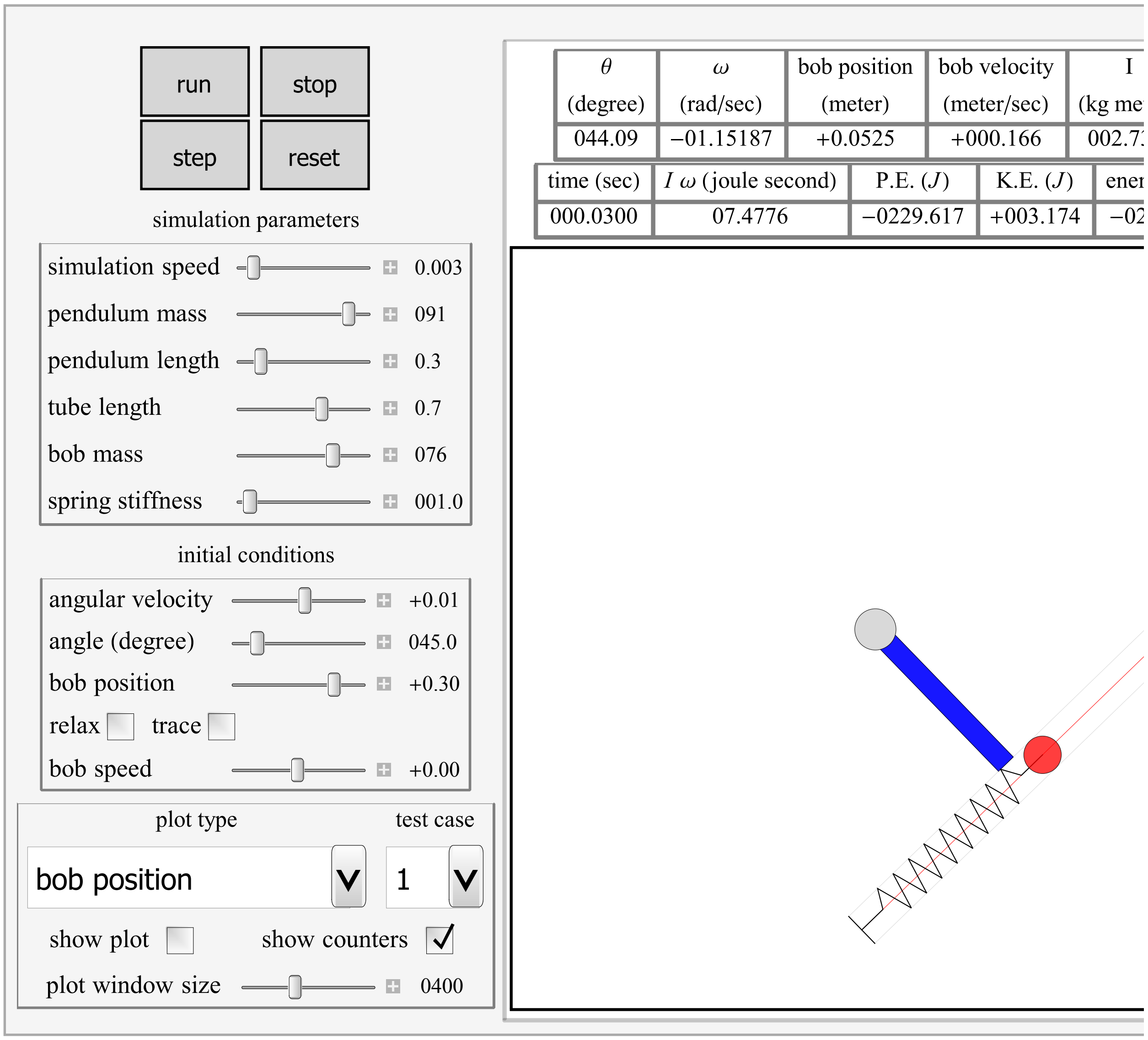
```
Manipulate[
 gTick;
 Text@Module[{g1, wasHit = False},
   If[setIC,
    setIC = False;
    list@setIC[{initialBobX, initialBobSpeed, initialTheta, initialThetaDot, r0, L0}];
    isRelaxedSpring = If[Abs[initialBobX] ≤ $MachineEpsilon, True, False]
   ];

   If[runningState == "RUNNING" || runningState == "STEP",
    wasHit = solver@step[m0, L0, k, M0, delT, w, r0];
    If[wasHit, (*animate a hit on the edge unless last time it was no hit*)
     If[wasHitLastTime == True,
      wasHit = False,
      wasHitLastTime = True
     ],
     wasHitLastTime = False
    ];

    If[runningState == "STEP", runningState = "STOP", gTick += del]
   ];

   g1 = display@makeDisplay[plotType, m0, M0, L0, r0, wasHit, k, showPlots, showCounters, trace, w];
   FinishDynamic[];
   g1
  ],
 (*---------- control layout ------------*)
 Text@Grid[{
    {Grid[{{
        Button[Style["run", 12], {If[Not[plotType == "velocity/acceleration"],
            runningState = "RUNNING"]; gTick += del}, ImageSize -> {55, 35}],
        Button[Style["stop", 12], {If[Not[plotType == "velocity/acceleration"], runningState = "STOP"];
          gTick += del}, ImageSize -> {55, 35}]
       },
       {
        Button[Style["step", 12], {If[Not[plotType == "velocity/acceleration"],
            runningState = "STEP"]; gTick += del}, ImageSize -> {55, 35}],

        Button[Style["reset", 12], (*bring simulation back to initial conditions*)
         {
          setIC = True;
          runningState = "STOP";
          gTick += del
         }, ImageSize -> {55, 35}]
       }}, Spacings → {0.5, .2}
     ]},
{Style["simulation parameters", 12]},
{Grid[
  {
   {"simulation speed",
    Manipulator[Dynamic[delT, {delT = #} &],
     {0.001, 0.05, 0.001}, ImageSize → Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[delT, {3, 3}], 11]
   },
   {"pendulum mass",
    Manipulator[Dynamic[m0,
      {m0 = Chop@#;
        setIC = True;
        gTick += del
       } &], {1, 100, 1}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[m0, 3], 11]
   },
   {"pendulum length",
    Manipulator[Dynamic[L0,
      {L0 = Chop@#;
        setIC = True;
        gTick += del
       } &], {0.1, 2, 0.1}, ImageSize -> Tiny, ContinuousAction → False],
    Style[Dynamic@padIt2[L0, {2, 1}], 11]
   },
   {"tube length",
    Manipulator[Dynamic[w,
      {(w = #) &,
       (
         w = Chop@#;
         If[initialBobX > (w / 2 - r0),
           initialBobX = w / 2 - r0
          ,
          If[initialBobX < (-w / 2 + r0),
           initialBobX = -w / 2 + r0
          ]
         ];
         setIC = True;
         gTick += del
        ) &
      }], {0.1, 1, 0.1}, ImageSize -> Tiny, ContinuousAction → False],
    Style[Dynamic@padIt2[w, {1, 1}], 11]
   },

   {"bob mass",
    Manipulator[Dynamic[M0,
      {M0 = Chop@#;
        setIC = True;
        gTick += del
       } &], {1, 100, 1}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[M0, 3], 11]
   },

   {"spring stiffness",
    Manipulator[Dynamic[k,
      {k = Chop@#;
        setIC = True;
        gTick += del
       } &], {0, 500, .1}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[k, {4, 1}], 11]
   }
  }, Spacings → {.6, .5}, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
 ]
},
{Style["initial conditions", 12]},
{Grid[{
   {"angular velocity",
    Manipulator[Dynamic[initialThetaDot,
      {initialThetaDot = Chop@#;
        setIC = True;
        gTick += del
       } &], {-0.1, 0.1, .01}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt1[initialThetaDot, {2, 2}], 11]
   },
   {"angle (degree)",
    Manipulator[Dynamic[initialTheta,
      {initialTheta = Chop@#;
        setIC = True;
        gTick += del
       } &], {0, 2 Pi, 2 Pi / 100.}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt2[initialTheta * 180.0 / Pi, {4, 1}], 11]
   },
   {"bob position",
    Manipulator[Dynamic[initialBobX,
      {(initialBobX = #) &,
       (
         initialBobX = Chop[#];
         If[initialBobX > (w / 2 - r0),
           initialBobX = w / 2 - r0
          ,
          If[initialBobX < (-w / 2 + r0),
           initialBobX = -w / 2 + r0
          ]
         ];
         setIC = True;
         gTick += del
        ) &
      }], {-0.47, 0.47, 0.01}, ImageSize -> Tiny, ContinuousAction → False],
    Style[Dynamic@padIt1[initialBobX, {2, 2}], 11]
   },
   {
    Grid[{{
       Row[{"relax", Spacer[2],
         Checkbox[Dynamic[isRelaxedSpring,
           {isRelaxedSpring = #;
             initialBobX = 0;
             setIC = True;
             gTick += del
            } &]]}],
       Row[{"trace", Spacer[2],
         Checkbox[Dynamic[trace,
           {trace = #;
             If[#, list@clearTrace[]];
             gTick += del
            } &]]}]
      }}, Spacings → {0.4, 0}], SpanFromLeft
   },
   {"bob speed",
    Manipulator[Dynamic[initialBobSpeed,
      {initialBobSpeed = Chop@#;
        setIC = True;
        gTick += del
       } &], {-0.1, 0.1, 0.01}, ImageSize -> Tiny, ContinuousAction -> False],
    Style[Dynamic@padIt1[initialBobSpeed, {2, 2}], 11]
   }
  }, Spacings → {.6, .3}, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
 ]
},
{Grid[{
```

```
{Style["plot type", 12], Style["test case", 12]},
{
 PopupMenu[Dynamic[plotType, {plotType = #;
     If[plotType == "velocity/acceleration",
      If[runningState == "RUNNING",
       runningState = "SUSPEND_RUNNING"
      ]
      ,
      If[runningState == "SUSPEND_RUNNING", runningState = "RUNNING"]
     ]; gTick += del} &],
  { "disk angular velocity" → Style["disk angular velocity", 12],
   "disk angle" → Style["disk angle", 12],
   "bob position" → Style["bob position", 12],
   "bob velocity" → Style["bob velocity", 12]
  }, ImageSize -> All, ContinuousAction -> False, Enabled → Dynamic[showPlots]]
 ,
 PopupMenu[Dynamic[testCase, {testCase = #;
     runningState = "STOP";
     Which[testCase == 1,
      (
       isRelaxedSpring = False; delT = 0.01; m0 = 40; k = 1; w = .7;
       initialThetaDot = 0.01; initialTheta = 45 Degree;
       initialBobX = 0.3; initialBobSpeed = 0; M0 = 10; trace = False;
       setIC = True;
       runningState = "STOP";
       plotType = "bob position"
      ),
      testCase == 2,
      (
       isRelaxedSpring = False; delT = 0.02; m0 = 16; k = 500; w = 1;
       initialThetaDot = 0.01; initialTheta = 45 Degree;
       initialBobX = -.2; initialBobSpeed = 0; M0 = 30; trace = True;
       setIC = True;
       runningState = "STOP";
       plotType = "disk angular velocity"
      ),
      testCase == 3,
      (
       isRelaxedSpring = True; delT = 0.03; m0 = 50; k = 0.5; M0 = 100; trace = False; w = .3;
       initialThetaDot = 0; initialTheta = 270 Degree; initialBobX = .10; initialBobSpeed = 0.1;
       setIC = True;
       runningState = "STOP";
       plotType = "disk angular velocity"
      )
     ];
     gTick += del} &],
  {
   1 → Style["1", 12], 2 → Style["2", 12], 3 → Style["3", 12]
  }, ImageSize -> All, ContinuousAction -> False]
},
{Grid[{
   {
    Row[{"show plot ", Spacer[4],
      Checkbox[Dynamic[showPlots, {showPlots = #; gTick += del} &]]}], Spacer[12],
    Row[{"show counters ", Spacer[4], Checkbox[Dynamic[showCounters,
         {showCounters = #; gTick += del} &]]}], SpanFromLeft
   }
  }
 ], SpanFromLeft
},
{Grid[{
   {"plot window size",
    Manipulator[Dynamic[xScale,
               {xScale = #;
                 list@setSize[xScale];
                 list@add[{0, initialBobX, initialBobSpeed, initialTheta, initialThetaDot}];
                 setIC = True;
                 gTick += del
                } &], {10, 1000, 1},
              ImageSize -> Tiny, ContinuousAction -> False, Enabled → Dynamic[showPlots]],
             Style[Dynamic@padIt2[xScale, 4], 11]
            }
           }], SpanFromLeft
          }
         }, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
        ]
       }
      }, Spacings → {0, {2 → 0.7, 4 → 0.7, 6 → 0.5, 8 → 0}}, Alignment → Center, Frame → None],
    (*----control variables--*)
    {{m0, 25}, None},
    {{M0, 40}, None},
    {{L0, 0.5}, None},
    {{delT, 0.01}, None},
    {{k, 1}, None},
    {{w, .5}, None},

    {{initialThetaDot, 0.09}, None},
    {{initialTheta, 45 Degree}, None},
    {{initialBobX, 0.05}, None},
    {{initialBobSpeed, 0}, None},
    {{isRelaxedSpring, False}, None},
    {{plotType, "bob position"}, None},
    {{testCase, 1}, None},
    {{showPlots, False}, None},
    {{showCounters, True}, None},
    {{xScale, 400}, None},
    {{trace, False}, None},
    {{setIC, False}, None},

    (*----- refresh control ---*)
    {{gTick, 0}, ControlType → None},
    {{del, $MachineEpsilon}, None},

    (*----- state variables for sim---*)
    {{runningState, "STOP"}, None}, (*"RUNNING","STEP","SUSPEND","STOP"*)
    {{wasHitLastTime, False}, None},
    {{r0, .03}, None},

    TrackedSymbols :→ {gTick},
    ControlPlacement → Left,
    SynchronousUpdating → False,
    SynchronousInitialization → True,
    ContinuousAction → False,
    Alignment -> Center,
    ImageMargins → 0,
    FrameMargins → 0,
    Paneled → True,
    Frame → False,

    Initialization :→
     {
      (*definitions used for parameter checking*)
      integerStrictPositive = (IntegerQ[#] && # > 0 &);
      integerPositive = (IntegerQ[#] && # ≥ 0 &);
      numericStrictPositive = (Element[#, Reals] && # > 0 &);
      numericPositive = (Element[#, Reals] && # ≥ 0 &);
```

```
numericStrictNegative = (Element[#, Reals] && # < 0 &);
numericNegative = (Element[#, Reals] && # ≤ 0 &);
bool = (Element[#, Booleans] &);
numeric = (Element[#, Reals] &);
integer = (Element[#, Integers] &);

(* This list object is used to store and the manage the time series generated during running
 of the simulation so that display object can use it to make plots and the animation *)

listClass[$size_?integer(*maximum size of the list*),
  $r0_?numericStrictPositive,
  $L0_?numericStrictPositive] :=
 Module[{size, lists, r0, L0, self},
  (*lists has the structure {idx,{  {t,x,v,θ,ω,{xTrace,yTrace}}},...   ,....}*)

  self@getSize[] := size;
  self@setSize[n_] := ( size = n ; lists = {0, Table[Table[0, {6}], {n}] });
  self@add[{time_?numericPositive, (* current time*)
      x_?numeric, (*bob position*)
      v_?numeric, (*bob speed*)
      θ_?numeric, (*disk angle*)
      ω_?numeric(*disk angular speed*)
     }
    ] := Module[{},
    lists[[1]] ++;
    If[lists[[1]] > size, lists[[1]] = 1];
    lists[[ 2, lists[[1]], 1 ]] = time;
    lists[[ 2, lists[[1]], 2 ]] = x;
    lists[[ 2, lists[[1]], 3 ]] = v;
    lists[[ 2, lists[[1]], 4 ]] = θ;
    lists[[ 2, lists[[1]], 5 ]] = ω;
    lists[[ 2, lists[[1]], 6 ]] = {(L0 + r0) Sin[θ] + x Cos[θ], -(L0 + r0) Cos[θ] + x Sin[θ]}
   ];

  self@setIC[{x_?numeric, v_?numeric, θ_?numeric,
      ω_?numeric, rr0_?numericStrictPositive, LL0_?numericStrictPositive}] := (
    list@clear[];
    r0 = rr0;
    L0 = LL0;
    list@add[{0, x, v, θ, ω}]
   );

  self@getCurrent[] := lists[[  2, lists[[1]] ]];
  self@clear[] := lists[[1]] = 0;
  self@getPositionList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 2}]] ] ;
  self@getSpeedList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 3}]] ] ;
  self@getThetaList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 4}]] ] ;
  self@getOmegaList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 5}]] ] ;
  self@getTraceData[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, 6]] ] ;

  (*--- constructor---*)
  self@setSize[$size];
  r0 = $r0;
  L0 = $L0;
  self
 ];
(*--------------------------------------------------------*)
solverClass[] :=
 Module[{solve, self},
  (*--------------------------------------------*)
  solve[M0_?numericStrictPositive,
    m0_?numericPositive,
    L0_?numericStrictPositive,
    eq1_,
    eq2_,
    t_,
    x_,
    θ_,
    w_?numericStrictPositive,
    delT_?numericStrictPositive(*time length to integrate over for NDSolve*),
    r0_?numericStrictPositive] := Module[{wasHit = False, currentSolution, currentX, currentV, currentθ,
     currentOmega, sol, ic, $x, $v, $θ, $ω, simTime, trace, newV, newOmega, eqq1, eqq2},

    {simTime, $x, $v, $θ, $ω, trace} = list@getCurrent[];

    ic = {x[0] == $x, x'[0] == $v, θ[0] == Mod[$θ, 2 Pi], θ'[0] == $ω};
    currentSolution = Quiet@NDSolve[Flatten@{eq1, eq2, ic},
       {x, x', θ, θ'}, {t, 0, delT}, Method → {"BDF"}, MaxSteps → Infinity];

    currentSolution = First@currentSolution;
    currentX = x[delT] /. currentSolution;
    currentV = x'[delT] /. currentSolution;
    currentθ = θ[delT] /. currentSolution;
    currentOmega = θ'[delT] /. currentSolution;

       (* if bob hits the edge of the disk or the base of the spring,
       fix speed and reset things. Assume perfect elastic edge. Solve such that the total
     moment of momuntum of the system before and after collision remain the same*)
    If[currentX > (w / 2 - r0) || currentX < (-w / 2 + r0),
     wasHit = True;
     eqq1 = newV == -currentV;
     eqq2 = (m0 L0^2 / 3) currentOmega + M0 (currentV) L0 + M0 (L0^2 currentOmega) + M0 currentX^2
          currentOmega == (m0 L0^2 / 3) newOmega + M0 (newV) L0 + M0 (L0^2 newOmega) + M0 currentX^2 newOmega;
     sol = {newV, newOmega} /. NSolve[{eqq1, eqq2}, {newV, newOmega}];

     If[Length[sol] > 1,
      If[currentV > 0,
       sol = Select[sol, First[#] ≤ 0 &],
       sol = Select[sol, First[#] > 0 &]
      ]
     ];

     sol = First[sol];
     currentV = sol[[1]];
     currentOmega = sol[[2]]
    ];

    list@add[{simTime + delT, currentX, currentV, currentθ, currentOmega}];
    wasHit
   ];
  (*-----------------------------------------------*)
  self@step[
     m0_?numericPositive, (*pendulum mass*)
     L0_?numericStrictPositive, (*pendulum length*)
     k_?numericPositive, (*spring k*)
     M0_?numericStrictPositive, (*bob mass*)
     delT_?numericStrictPositive, (*time length to integrate over for NDSolve*)
     w_?numericPositive,
     r0_?numericStrictPositive
    ] := Module[{eq1, eq2, t, x, θ, currentθ, wasHit = False,
     $x, $v, $θ, $ω, simTime, trace, g = 9.81, solidPendulum, springAttached},
    {simTime, $x, $v, $θ, $ω, trace} = list@getCurrent[];
```

```
      solidPendulum = Abs[m0] > $MachineEpsilon;
      springAttached = Abs[k] > $MachineEpsilon;

      If[solidPendulum,
       If[springAttached,
        eq1 = g M0 Sin[θ[t]] + k x[t] - M0 x[t] θ'[t]^2 + M0 (x''[t] + L0 θ''[t]) == 0;
        eq2 = 1/2 g L0 m0 Sin[θ[t]] - g M0 (-L0 Sin[θ[t]] - Cos[θ[t]] x[t]) +
           1/3 L0^2 m0 θ''[t] + 1/2 M0 (4 x[t] x'[t] θ'[t] + 2 x[t]^2 θ''[t] + 2 L0 (x''[t] + L0 θ''[t])) == 0;
        wasHit = solve[M0, m0, L0, eq1, eq2, t, x, θ, w, delT, r0]
        ,
        eq1 = g M0 Sin[θ[t]] - M0 x[t] θ'[t]^2 + M0 (x''[t] + L0 θ''[t]) == 0;
        eq2 = 1/2 g L0 m0 Sin[θ[t]] - g M0 (-L0 Sin[θ[t]] - Cos[θ[t]] x[t]) +
           1/3 L0^2 m0 θ''[t] + 1/2 M0 (4 x[t] x'[t] θ'[t] + 2 x[t]^2 θ''[t] + 2 L0 (x''[t] + L0 θ''[t])) == 0;
        wasHit = solve[M0, m0, L0, eq1, eq2, t, x, θ, w, delT, r0]
       ]
       ,
       If[springAttached,
        eq1 = g M0 Sin[θ[t]] + k x[t] - M0 x[t] θ'[t]^2 + M0 (x''[t] + L0 θ''[t]) == 0;
        eq2 = -g M0 (-L0 Sin[θ[t]] - Cos[θ[t]] x[t]) +
           1/2 M0 (4 x[t] x'[t] θ'[t] + 2 x[t]^2 θ''[t] + 2 L0 (x''[t] + L0 θ''[t])) == 0;
        wasHit = solve[M0, m0, L0, eq1, eq2, t, x, θ, w, delT, r0]
        ,
        eq1 = g M0 Sin[θ[t]] - M0 x[t] θ'[t]^2 + M0 (x''[t] + L0 θ''[t]) == 0;
        eq2 = -g M0 (-L0 Sin[θ[t]] - Cos[θ[t]] x[t]) +
           1/2 M0 (4 x[t] x'[t] θ'[t] + 2 x[t]^2 θ''[t] + 2 L0 (x''[t] + L0 θ''[t])) == 0;
        wasHit = solve[M0, m0, L0, eq1, eq2, t, x, θ, w, delT, r0]
       ]
      ];

      wasHit
     ];

    self
   ];
  (*--------- displayClass ---------------------------------------*)
  displayClass[] := Module[{makeCounters, makeDiskDiagram, makePlot, self},

    (*--------- private ---------------------------------------*)
    makeCounters[
      m0_?numericPositive, (*pendulum arm mass*)
      M0_?numericStrictPositive, (*bob mass*)
      L0_?numericStrictPositive, (*pendulum length mass*)
      k_?numericPositive] := Module[{h1, h2, currentMomentOfInertia, x, v, θ, ω,
       PE, KE, trace, g = 9.81, simTime, angularMomentum, solidPendulum, springAttached},

      solidPendulum = Abs[m0] > $MachineEpsilon;
      springAttached = Abs[k] > $MachineEpsilon;
      {simTime, x, v, θ, ω, trace} = list@getCurrent[];
      If[solidPendulum,
       KE = 1/6 L0^2 m0 ω^2 + 1/2 M0 (x^2 ω^2 + (v + L0 ω)^2);
       currentMomentOfInertia = m0 * L0 ^ 2 / 3;
       angularMomentum = m0 L0 ^ 2 / 3 * ω + M0 (L0 v + (L0^2 + x^2) ω);
       If[springAttached,
        PE = -1/2 g L0 m0 Cos[θ] + 1/2 k x^2 - g M0 (L0 Cos[θ] - Sin[θ] x),
        PE = -1/2 g L0 m0 Cos[θ] - g M0 (L0 Cos[θ] - Sin[θ] x)
       ],
       KE = 1/2 M0 (x^2 ω^2 + (v + L0 ω)^2);
       currentMomentOfInertia = 0;
       angularMomentum = M0 (L0 v + (L0^2 + x^2) ω);
       If[springAttached,
        PE = 1/2 k x^2 - g M0 (L0 Cos[θ] - Sin[θ] x),
        PE = -g M0 (L0 Cos[θ] - Sin[θ] x)
       ]
      ];

      h1 = Style[Grid[{
          {"θ", "ω", "bob position", "bob velocity", Style["I", Italic]
          },
          {"(degree)", "(rad/sec)", "(meter)", "(meter/sec)", "(kg meter^2)"
          },
          { padIt2[180. / Pi * θ, {5, 2}], padIt1[ω, {7, 5}],
           padIt1[x, {5, 4}], padIt1[v, {6, 3}], padIt2[currentMomentOfInertia, {7, 4}]
          }
         },
         Frame → {All, None, { {{1, 2}, {1, 5}} → True , {{3, 3}, {1, 5}} → True}},
         FrameStyle → Gray,
         Spacings → 1,
         ItemSize → {{All, 2 ;; -1} → 5},
         Alignment → Center], 12];

      h2 = Style[Grid[{
          {"time (sec)", Row[{Style["I", Italic], " ω (joule second)"}], Row[{"P.E. (", Style["J", Italic],
             ")"}], Row[{"K.E. (", Style["J", Italic], ")"}], Row[{"energy (", Style["J", Italic], ")"}]
          },
          { padIt2[Mod[simTime, 1000], {7, 4}], padIt2[angularMomentum, {6, 4}],
           padIt1[PE, {7, 3}], padIt1[KE, {6, 3}], padIt1[PE + KE, {6, 2}]
          }},
         Frame → All,
         FrameStyle → Gray,
         Spacings → 1,
         ItemSize → {{All, 2 ;; -1} → 6},
         Alignment → Center], 12];

      Grid[{{h1}, {h2}}, Spacings → {0, .1}]
     ];
    (*---------------private----------------------------------*)

    makeDiskDiagram[k_?numericPositive,
      m0_?numericStrictPositive,
      M0_?numericStrictPositive,
      L0_?numericStrictPositive,
      r0_?numericStrictPositive,
      wasHit_?bool,
       {ContentSizeW_?numericStrictPositive, ContentSizeH_?numericStrictPositive},
      trace_?bool,
      w_] := Module[{a = 2 r0, b = 1.1 r0, x, y, splash, xx, v, θ, ω, traceData,
       currentTrace, spring, pin, bob, ext, simTime, frame0, frame1, frame2, frame3, frame4},

       {simTime, xx, v, θ, ω, currentTrace} = list@getCurrent[];
      traceData = list@getTraceData[];

      If[wasHit,
       splash = {
         { Dotted, Red,
          Rotate[ Line[{{L0 + r0, Sign[xx] w / 2}, {L0 + r0, Sign[xx] (w / 2 + 2 r0)}}], -(Pi / 2) + θ, {0, 0}]},
         { Dotted, Red, Rotate[Line[{{L0 + r0, Sign[xx] w / 2}, {L0 + 2 r0, Sign[xx] (w / 2 + 2 r0)}}],
           -(Pi / 2) + θ, {0, 0}]},
         { Dotted, Red, Rotate[Line[{{L0 + r0, Sign[xx] w / 2}, {L0 + 3 r0, Sign[xx] (w / 2 + 2 r0)}}],
           -(Pi / 2) + θ, {0, 0}]},
         { Dotted, Red, Rotate[Line[{{L0 + r0, Sign[xx] w / 2}, {L0 - r0, Sign[xx] (w / 2 + 2 r0)}}],
           -(Pi / 2) + θ, {0, 0}]},
         { Dotted, Red, Rotate[Line[{{L0 + r0, Sign[xx] w / 2}, {L0 - 2 r0, Sign[xx] (w / 2 + 2 r0)}}],
           -(Pi / 2) + θ, {0, 0}]},
         { Dotted, Red, Rotate[Line[{{L0 + r0, Sign[xx] w / 2}, {L0 - 3 r0, Sign[xx] (w / 2 + 2 r0)}}],
           -(Pi / 2) + θ, {0, 0}]}
        };
       If[xx < 0, xx = -w / 2 + r0, xx = w / 2 - r0]
       ,
       splash = Sequence @@ {};
      ];

      x = xx * Cos[θ];
      y = xx * Sin[θ];
      traceData = list@getTraceData[];
      bob = {Red, Opacity[M0 / 100], EdgeForm[Thin], Disk[{L0 + r0, xx}, r0]};
      pin = {LightGray, EdgeForm[Thin], Disk[{0, 0}, b]};
      ext = 1.1 Sqrt[(L0 + 2 r0) ^ 2 + (w / 2) ^ 2];
      If[k == 0,
       spring = Sequence @@ {},
       spring = Line@makeSpring[L0 + r0, -w / 2, L0 + r0, xx, r0]
      ];

      frame0 = {Thin, LightGray,
        Line[{ {L0, 0}, {L0 , -w / 2}, {L0 + a, -w / 2}, {L0 + a, w / 2}, {L0 + a, w / 2}, {L0, w / 2}, {L0, 0}}]};
      frame1 = {Blue, Opacity[m0 / 100], EdgeForm[Thin], Polygon[
          { {0, -b / 2}, {L0 , -b / 2}, {L0, b / 2}, {0, b / 2}}]}];
      frame2 = {Black, Line[{ {L0 , -w / 2}, {L0 + a, -w / 2}}]};
      frame3 = {Black, Line[{ {L0 + a , w / 2}, {L0, w / 2}}]};
      frame4 = {Thin, Red, Line[{ {L0 + r0 , -w / 2}, {L0 + r0, w / 2}}]};

      Framed@Graphics[{
         Rotate[frame0, -(Pi / 2) + θ, {0, 0}],
         Rotate[{White, frame1}, -(Pi / 2) + θ, {0, 0}],
         Rotate[frame2, -(Pi / 2) + θ, {0, 0}],
         Rotate[frame3, -(Pi / 2) + θ, {0, 0}],
         Rotate[frame4, -(Pi / 2) + θ, {0, 0}],
         Rotate[{Black, spring}, -(Pi / 2) + θ, {0, 0}],
         Rotate[bob, -(Pi / 2) + θ, {0, 0}],
         Rotate[pin, -(Pi / 2) + θ, {0, 0}],
         splash,
         If[trace, {Thin, Darker@Red, Line[traceData]}, Sequence @@ {}]
        },
        Axes → False,
        PlotRange → {{-ext, ext}, {-ext , ext}},
        ImageSize → {ContentSizeW, 1.01 ContentSizeH},
        ImagePadding → {{5, 5}, {5, 5}},
        ImageMargins → 0,
        AspectRatio → 1
       ]
     ];
    (*-------------------- private ----------------------------*)
    makePlot[plotType_String,
      w_?numericStrictPositive] := Module[{plotTitle, data},

      Which[plotType == "disk angular velocity",
       data = list@getOmegaList[];
       If[Length[data] == 0, data = {{0, 0}}];
       plotTitle =
        {{Style["ω (rad/sec)", 12], None}, {Style["time"], Style["disk angular velocity vs. time", 12]}};
       plotRange = {{data[[1, 1]], data[[-1, 1]]}, All}
       ,
       plotType == "disk angle",
       data = list@getThetaList[];
       data[[All, 2]] = Map[180.0 / Pi * # &, data[[All, 2]]];
       If[Length[data] == 0, data = {{0, 0}}];
       plotTitle = {{Style["θ (deg)", 12], None}, {Style["time"], Style["disk angle vs. time", 12]}};
       plotRange = {{data[[1, 1]], data[[-1, 1]]}, {0, 360}}
       ,
       plotType == "bob position",
       data = list@getPositionList[];
       If[Length[data] == 0, data = {{0, 0}}];
       plotTitle = {{Style[Row[{"position (", Style["m", Italic], ")"}], 12], None},
          {Style["time"], Style["bob position vs. time", 12]}};
       plotRange = {{data[[1, 1]], data[[-1, 1]]}, {-w / 2, w / 2}}
       ,
       plotType == "bob velocity",
       data = list@getSpeedList[];
       If[Length[data] == 0, data = {{0, 0}}];
       plotTitle = {{Style[Row[{"velocity (", Style["m/s", Italic], ")"}], 12], None},
          {Style["time"], Style["bob velocity vs. time", 12]}};
       plotRange = {{data[[1, 1]], data[[-1, 1]]}, All}

      ];

      ListPlot[ data,
       PlotRange → plotRange,
       AxesOrigin → {0, 0},
       Joined → True,
       Frame → True,
       GridLines → Automatic,
       ImageSize → {330, 160},
       ImagePadding → {{60, 10}, {30, 25}},
       ImageMargins -> {{2, 1}, {1, 1}},
       Axes → False,
       FrameLabel → plotTitle,
       PlotStyle → Red,
       AspectRatio → 0.27]
     ];
    (*------- public ----------------------------------*)
    self@makeDisplay[plotType_String,
       m0_?numericPositive,
       M0_?numericStrictPositive,
       L0_?numericStrictPositive,
       r0_?numericStrictPositive,
       wasHit_?bool,
       k_?numericPositive,
       showPlots_?bool,
       showCounters_?bool,
       trace_?bool,
       w_] :=
     If[showPlots,
      If[showCounters,
       Grid[{
          {makeCounters[m0, M0, L0, k]},
          {makePlot[plotType, w]},
          {makeDiskDiagram[k, m0, M0, L0, r0, wasHit, {1.4 ContentSizeW, 0.515 ContentSizeH}, trace, w]}
         }, Spacings → 0]
       ,
       Grid[{
          {makePlot[plotType, w]},
          {makeDiskDiagram[k, m0, M0, L0, r0, wasHit, {1.4 ContentSizeW, 0.787 ContentSizeH}, trace, w]}
         }, Spacings → 0]
      ],
      If[showCounters,
       Grid[{
          {makeCounters[m0, M0, L0, k]},
          {makeDiskDiagram[k, m0, M0, L0, r0, wasHit, {1.4 ContentSizeW, 0.93 ContentSizeH}, trace, w]}
         }, Spacings → 0]
       ,
       Grid[{
          {makeDiskDiagram[k, m0, M0, L0, r0, wasHit, {1.4 ContentSizeW, 1.2 ContentSizeH}, trace, w]}
         }, Spacings → 0]
      ]
     ];

    self
   ];
 (*--------------------------------------------*)
 (* helper function for formatting            *)
 (*--------------------------------------------*)
 padIt1[v_?numeric, f_List] :=
  AccountingForm[Chop[v] , f, NumberSigns → {"-", "+"}, NumberPadding → {"0", "0"}, SignPadding → True];
 (*--------------------------------------------*)
 (* helper function for formatting            *)
 (*--------------------------------------------*)
 padIt2[v_?numeric, f_List] :=
  AccountingForm[Chop[v] , f, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}, SignPadding → True];
 padIt2[v_?numeric, f_Integer] := AccountingForm[Chop[v] , f, NumberSigns → {"", ""},
   NumberPadding → {"0", "0"}, SignPadding → True];

 (*This function is called to make spring,based on code by Arpad Kosa from WRI demo*)
 (*at Wolfram web site modified by me.This returns a Line which is the spring*)
 (*szel, larger number means bigger spring width*)
 makeSpring[xFirst_?numeric, yFirst_?numeric, xEnd_?numeric, yEnd_?numeric, szel_?numeric] :=
  Module[{hx, veghossz, hossz, hy, dh, tbl},
   hx = xEnd - xFirst;
   If[Abs[hx] ≤ $MachineEpsilon, hx = 10^-6];
   hy = yEnd - yFirst;
   If[Abs[hy] ≤ $MachineEpsilon, hy = 10^-6];

   veghossz = 0.03;
   hossz = Sqrt[hx^2 + hy^2];
   dh = (hossz - 2 * veghossz) / 20;
   tbl = Table[If[OddQ[i], {xFirst + hx * (i * dh + veghossz) / hossz + hy * szel / hossz,
        yFirst + hy * (i * dh + veghossz) / hossz - hx * szel / hossz}, {xFirst + hx * (i * dh + veghossz) / hossz -
         hy * szel / hossz, yFirst + hy * (i * dh + veghossz) / hossz + hx * szel / hossz}], {i, 2, 18}];
   {{xFirst, yFirst}} ~ Join ~ {{xFirst + hx * (dh + veghossz) / hossz, yFirst + hy * (dh + veghossz) / hossz}} ~
```

```
        Join ~ tbl ~ Join ~ {{xFirst + hx * (19 * dh + veghossz) / hossz, yFirst + hy * (19 * dh + veghossz) / hossz}} ~
        Join ~ {{xEnd, yEnd}}
     ];

    (*--- constant parameters size and width of display ---*)
    ContentSizeW = 260;
    ContentSizeH = 405;

    (* objects used by the simulation. These must be here in the initialization section *)
    list = listClass[500, 0.02, 0.5];
    list@add[{0, 0.05, 0.0, Pi / 4, 0.09}];
    solver = solverClass[];
    display = displayClass[];
   }
]
```



simulation parameters

simulation speed	0.003
pendulum mass	091
pendulum length	0.3
tube length	0.7
bob mass	076
spring stiffness	001.0

initial conditions

angular velocity	+0.01
angle (degree)	045.0
bob position	+0.30
relax ☐	trace ☐
bob speed	+0.00

plot type: bob position
test case: 1
show plot ☐ show counters ✓
plot window size 0400

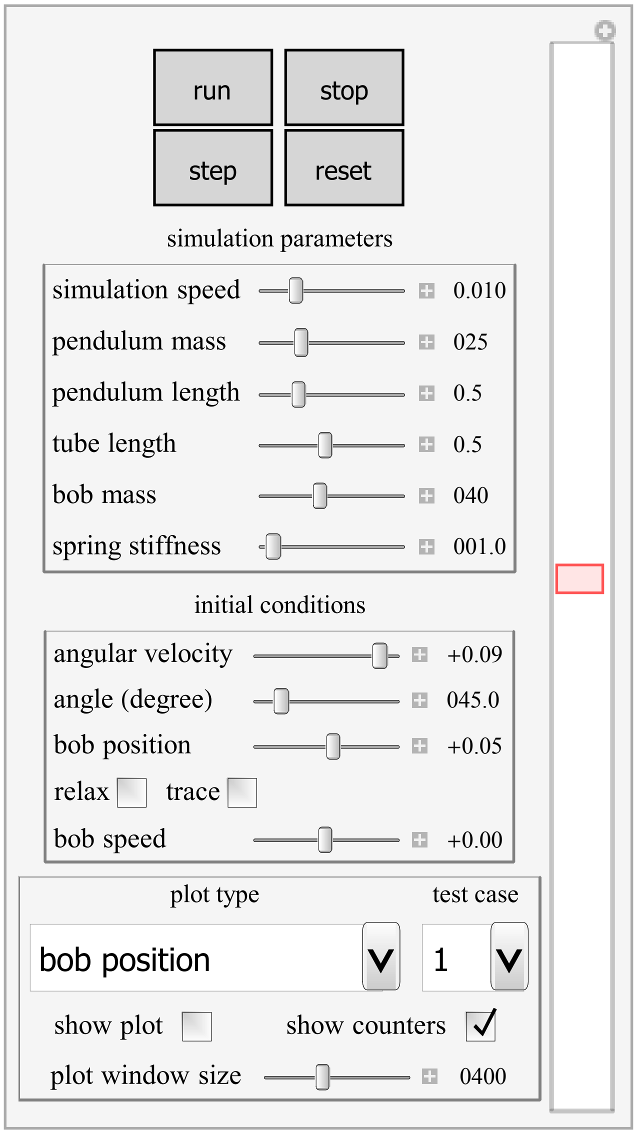
θ (degree)	ω (rad/sec)	bob position (meter)	bob velocity (meter/sec)	I (kg me
044.09	−01.15187	+0.0525	+000.166	002.7

time (sec)	$I\ \omega$ (joule second)	P.E. (J)	K.E. (J)	ener
000.0300	07.4776	−0229.617	+003.174	−02

Caption

This Demonstration describes the dynamics of a solid pendulum in which a bob mass slides back and forth over a massless and smooth thin tube attached to the bottom of a pendulum bar. The bob mass is attached to the end of a spring. When the bob hits the ends of the tube it bounces back; the ends are assumed to be perfectly elastic. The two equations of motions for the bob and the pendulum bar are derived using the Lagrangian method which is solved numerically by *Mathematica*'s built-in function `NDSolve`. The spring is anchored to the left edge of the tube and assumed to be massless and can only move in one direction along the length of the tube. Setting the spring stiffness to zero effectively removes the spring leaving the bob to oscillate freely.

Thumbnail



simulation parameters

simulation speed 0.010
pendulum mass 025
pendulum length 0.5
tube length 0.5
bob mass 040
spring stiffness 001.0

initial conditions

angular velocity +0.09
angle (degree) 045.0
bob position +0.05
relax ☐ trace ☐
bob speed +0.00

plot type: bob position
test case: 1
show plot ☐ show counters ☑
plot window size 0400

Details (optional)

description of the user interface

The top four buttons are used to control the Demonstration. The reset button brings the simulation back to the initial conditions. The control variables below these are used to change the initial conditions for the pendulum and the bob. The "relax" check box is used to put the spring into the relaxed initial position, which is at half the length of the tube. You can see a trace of the bob motion using the "trace" check box. You can change the length of the pendulum bar and the tube length as well as their masses.

The result has three parts. The counters at the top show the current state. The field labeled "energy" is the sum of the potential energy (P.E.) and the kinetic energy (K.E.). The field labeled "I" is the mass moment of inertia of the pendulum bar, defined as $m \frac{L^2}{3}$, where m is the mass of the pendulum bar and L is the length of the bar. The field labeled $I\,\omega$ is the total angular momentum of the system. The middle part of the display shows different plots. The plot shown is selected using the pop-up menu labeled "plot type". The third part shows the pendulum itself as it swings. You can select the parts to see using the check boxes to the left.
A pop-up menu labeled "test case" is used to select one of three preconfigured simulation parameters. Click the run button after selecting a test case to start the Demonstration.
The Demonstration runs with no throttling by using a single dynamic variable called "`gTick`" that is updated (or ticked) at the end of each `Manipulate` expression evaluation, so that the `Manipulate` expression is reevaluated again immediately. This lets the Demonstration run at the maximum speed. You can use a slider labeled "simulation speed" to adjust the speed of the simulation as needed.
The action runs continuously until you stop it. The time counter rewinds each 1000 seconds. All units used as assumed to be in SI.

derivation of equations of motion

The equations of motions are derived using the Lagrangian method. The bob has one degree of freedom: the distance x is along the length of the tube; $x = 0$ is the middle of the tube which is also the location where the pendulum bar is attached to the tube. The pendulum likewise has one degree of freedom: the angle of rotation θ. For more information on the derivation of the equations of motion, please see the author's report.
References

[1] D. Morin, *Introduction to Classical Mechanics: With Problems and Solutions,* New York: Cambridge University Press, 2008.

[2] The Editors of REA, *The Mechanics Problem Solver,* New Jersey: Research and Education Association, 2002.

[3] D. A. Wells, *Lagrangian Dynamics*, New York: Schaums' Outline, 1967.

Control Suggestions (optional)

✓ Resize Images

✓ Rotate and Zoom in 3D

☐ Drag Locators

☐ Create and Delete Locators

✓ Slider Zoom

✓ Gamepad Controls

✓ Automatic Animation

☐ Bookmark Animation

Search Terms (optional)

angular momentum
mass spring
vibration
moment of momentum
pendulum

Related Links (optional)

angular momentum
NDSolve

Authoring Information

Contributed by: Nasser M. Abbasi