Heavy Spring with Double Pendulum

Initialization Code (optional)

Manipulate

```
In[1]:= Manipulate[
         gTick;
         Module[{g1, $x, $xv, $y, $yv, $θ, $ω, $trace, cm, initialYEffective, g = 9.81, L0},

          L0 = initialRelaxedLength[len0, m0, M0, k1, g];
          initialYEffective = initialY + L0;

          If[setIC,
           setIC = False;
           cm = calculateCenterOfMass[m0, M0, initialYEffective, initialTheta Degree, initialBobX ];
           list@setIC[{initialBobX, initialBobSpeed,
               initialY, initialYSpeed, initialTheta Degree, initialThetaDot, cm, r0, L0}];
           isRelaxedSpring = If[Abs[initialBobX] ≤ $MachineEpsilon, True, False];
           isRelaxedTopSpring = If[Abs[initialY - L0] ≤ $MachineEpsilon, True, False]
          ];

          If[runningState == "RUNNING" || runningState == "STEP",
           solver@step[m0, len0, k1, M0, delT];
           If[runningState == "STEP", runningState = "STOP", gTick += del]
          ];

          g1 = display@makeDisplay[plotType, m0, M0, len0, r0, k1, showPlots, showCounters, traceCM, traceBob, w];
          {simTime, $x, $xv, $y, $yv, $θ, $ω, cm, $trace} = list@getCurrent[];
          FinishDynamic[];
          Text@g1
         ],
         (*---------- control layout ------------*)

         Item[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}]
             },
             {Grid[{
                {Style["time (sec)", 11]},
                {Style[Dynamic@padIt2[Mod[simTime, 1000], {7, 4}], 11]}
               }, Spacings → {0, 0.5}]
             }
           }, Spacings → {0.1, .7}
          ], ControlPlacement → Right],
```

```
Grid[{
  {Grid[
    {
     {Row[{"speed", Spacer[15], "(slow)"}],
      Manipulator[Dynamic[delT, {delT = #} &],
       {0.001, 0.05, 0.001}, ImageSize → Tiny, ContinuousAction -> False],
      "(fast)"
     },
     {Row[{"display zoom", Spacer[10], "(in)"}],
      Manipulator[Dynamic[w,
         {w = Chop@#; gTick += del} &], {0.05, 3, 0.01}, ImageSize → Tiny, ContinuousAction -> False],
      "(out)"
     }
    }, Spacings → {.6, .5}, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
   ]
  },
  {Style["top spring initial conditions", 12]},
  {Grid[{
     {"spring mass",
      Manipulator[Dynamic[m0,
         {m0 = Chop@#; setIC = True; gTick += del} &],
       {1, 5, 0.1}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt2[m0, {2, 2}], 11]
     },
     {"spring stiffness",
      Manipulator[Dynamic[k1,
         {k1 = Chop@#; setIC = True; gTick += del} &],
       {300, 10 000, 10}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt2[k1, 4], 11]
     },
     {"angular velocity",
      Manipulator[Dynamic[initialThetaDot,
         {initialThetaDot = #; setIC = True; gTick += del} &],
       {-0.5, 0.5, .1}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt1[initialThetaDot, {2, 2}], 11]
     },
     {"angle (degree)",
      Manipulator[Dynamic[initialTheta,
         {initialTheta = #; setIC = True; gTick += del} &],
       {-45, 45, 1}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt1[initialTheta, 3], 11]
     },
     {"initial extension",
      Manipulator[Dynamic[initialY,
         {(initialY = #) &, (initialY = Chop[#]; setIC = True; gTick += del) &}],
       {-0.25, 0.25, 0.01}, ImageSize -> Tiny, ContinuousAction → False],
      Style[Dynamic@padIt1[initialY, {2, 2}], 11]
     }
    }, Spacings → {.6, .3}, Alignment → Left, Frame → True, FrameStyle -> Directive[Thickness[.005], Gray]
   ]
  },

  {Style["bottom spring initial conditions", 12]},
  {Grid[{
     {"spring stiffness",
      Manipulator[Dynamic[k, {k = Chop@#; setIC = True; gTick += del} &],
       {300, 3000, 10}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt2[k, 4], 11]
     },
     {"bob mass",
      Manipulator[Dynamic[M0, {M0 = Chop@#; setIC = True; gTick += del} &],
       {0, 50, 1}, ImageSize -> Tiny, ContinuousAction -> False],
      Style[Dynamic@padIt2[M0, 2], 11]
     },
```

```
    {"bob initial position",
     Manipulator[Dynamic[initialBobX, {initialBobX = #; setIC = True; gTick += del} &],
      {-0.5, 0.5, 0.1}, ImageSize -> Tiny, ContinuousAction → False],
     Style[Dynamic@padIt1[initialBobX, {1, 1}], 11]
    },
    {
     Grid[{{
         Row[{"relax", Spacer[2],
           Checkbox[Dynamic[isRelaxedSpring,
             {isRelaxedSpring = #; initialBobX = 0; setIC = True; gTick += del} &]]}], Spacer[9],
         Row[{"trace bob", Spacer[2],
           Checkbox[Dynamic[traceBob, {traceBob = #; gTick += del} &], Enabled → Dynamic[M0 > 0]]}],
         Spacer[9],
         Row[{"show CM", Spacer[4],
           Checkbox[Dynamic[traceCM, {traceCM = #; gTick += del} &]]}]
        }}, Spacings → {0.4, 0}], SpanFromLeft
    },
    {"bob initial 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],
        "vertical displacement" → Style["vertical displacement", 12]
      }, ImageSize -> All, ContinuousAction -> False, Enabled → Dynamic[showPlots]]
     ,
     PopupMenu[Dynamic[testCase, {testCase = #;
          runningState = "STOP";
          Which[testCase == 1,
           (isRelaxedSpring = False; delT = 0.006; m0 = 1; k = 3000; k1 = 10 000; w = 1.2;
            initialThetaDot = 0; initialTheta = 0;
            initialBobX = -0.5; initialBobSpeed = 0; M0 = 50; traceCM = False; traceBob = False;
            initialY = 0.25; setIC = True; runningState = "STOP"; plotType = "bob position"
           ),
           testCase == 2,
           (isRelaxedSpring = False; delT = 0.05; m0 = 5; k = 500; w = 0.9; k1 = 10 000;
            initialThetaDot = 0.0; initialTheta = 15; initialBobX = -.2; initialBobSpeed = 0; M0 = 0; traceCM =
             False; traceBob = False; setIC = True; runningState = "STOP"; plotType = "disk angular velocity"
           ),
           testCase == 3,
           (isRelaxedSpring = True; delT = 0.03;
            m0 = 5; k = 3000; k1 = 10 000; M0 = 5; traceCM = False; traceBob = False; w = 1.1;
            initialThetaDot = 0; initialTheta = 0; initialBobX = 0; initialBobSpeed = 0;
            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]; 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, 1}, None}, (*mass of top spring*)
{{M0, 10}, None},
{{len0, 0.6}, None}, (*relaxed length, fixed does not change*)
{{delT, 0.01}, None},
{{k, 500}, None},
{{k1, 500}, None},
{{initialThetaDot, 0.0}, None},
{{initialTheta, 15}, None},
{{initialBobX, -0.35}, None},
{{initialY, .1}, None},
{{initialYDot, 0}, None},
{{initialBobSpeed, 0}, None},
{{initialYSpeed, 0}, None},
{{isRelaxedSpring, False}, None},
{{isRelaxedTopSpring, False}, None},
{{plotType, "bob position"}, None},
{{testCase, 1}, None},
{{showPlots, False}, None},
{{showCounters, True}, None},
{{xScale, 400}, None},
{{traceBob, False}, None},
{{traceCM, 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, .05}, None},
{{w, 1.3}, None},
{{simTime, 0}, None},
```

```
TrackedSymbols :→ {gTick},
ControlPlacement → Left,
SynchronousUpdating → False,
SynchronousInitialization → False,
ContinuousAction → False,
Alignment -> Center,
ImageMargins → 0,
FrameMargins → 0,
Paneled → True,
Frame → False,
AutorunSequencing → {1},
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,vx,y,vy,θ,ω,{cmX,cmY},{xTrace,yTrace}}},...   ,....}*)

    self@getSize[] := size;
    self@setSize[n_] := ( size = n ; lists = {0, Table[Table[0, {9}], {n}] });
    self@add[{time_?numericPositive, (* current time*)
        x_?numeric, (*bob position*)
        vx_?numeric, (*bob speed*)
        y_?numeric, (*arm spring position*)
        vy_?numeric, (*arm spring speed*)
        θ_?numeric, (*disk angle*)
        ω_?numeric, (*disk angular speed*)
        cm_List(*center of mass coordinates*)
       }
      ] := 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 ]] = vx;
      lists[[ 2, lists[[1]], 4 ]] = y;
      lists[[ 2, lists[[1]], 5 ]] = vy;
      lists[[ 2, lists[[1]], 6 ]] = θ;
      lists[[ 2, lists[[1]], 7 ]] = ω;
      lists[[ 2, lists[[1]], 8 ]] = cm;
      lists[[ 2, lists[[1]], 9 ]] = {(L0 + y + r0) Sin[θ] + x Cos[θ], -(L0 + y + r0) Cos[θ] + x Sin[θ]}
     ];

    self@setIC[{x_?numeric, vx_?numeric, y_?numeric, vy_?numeric, θ_?numeric,
        ω_?numeric, cm_List, rr0_?numericStrictPositive, LL0_?numericStrictPositive}] := (
```

```
                                                                                                                --
      list@clear[];
      r0 = rr0;
      L0 = LL0;
      list@add[{0, x, vx, y, vy, θ, ω, cm}]
    );

  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@getYList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 4}]] ] ;
  self@getYSpeedList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 5}]] ] ;
  self@getThetaList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 6}]] ] ;
  self@getOmegaList[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, {1, 7}]] ] ;
  self@getBobTraceData[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, 9]] ] ;
  self@getCMTraceData[] := Module[{len = lists[[1]]}, lists[[ 2, 1 ;; len, 8]] ] ;

  (*--- constructor---*)
  self@setSize[$size];
  r0 = $r0;
  L0 = $L0;
  self
 ];
(*-----------------------------------------------------------*)
solverClass[] :=
 Module[{solve, self},
  (*----------------------------------------------*)
  solve[M0_?numericStrictPositive,
    m0_?numericPositive,
    len0_?numericStrictPositive,
    eq1_,
    eq2_,
    eq3_,
    t_,
    x_,
    y_,
    θ_,
    delT_?numericStrictPositive(*time length to integrate over for NDSolve*),
    k1_?numericStrictPositive] := Module[{currentSolution, currentX, currentXV, currentY, currentYV,
     currentθ, currentOmega, ic, $x, $xv, $y, $yv, $θ, $ω, simTime, trace, g = 9.81, cm, L0},

    L0 = initialRelaxedLength[len0, m0, M0, k1, g];

    {simTime, $x, $xv, $y, $yv, $θ, $ω, cm, trace} = list@getCurrent[];
    ic = {x[0] == $x, x'[0] == $xv, y[0] == $y, y'[0] == $yv, θ[0] == Mod[$θ, 2 Pi], θ'[0] == $ω};
    currentSolution = Quiet@NDSolve[Flatten@{eq1, eq2, eq3, ic},
        {x, x', y, y', θ, θ'}, {t, 0, delT}, Method → {"BDF"}, MaxSteps → Infinity];

    currentSolution = First@currentSolution;
    currentX = x[delT] /. currentSolution;
    currentXV = x'[delT] /. currentSolution;
    currentY = y[delT] /. currentSolution;
    currentYV = y'[delT] /. currentSolution;
    currentθ = θ[delT] /. currentSolution;
    currentOmega = θ'[delT] /. currentSolution;
    cm = calculateCenterOfMass[m0, M0, L0 + currentY, currentθ, currentX];
    list@add[{simTime + delT, currentX, currentXV, currentY, currentYV, currentθ, currentOmega, cm}]
   ];
  (*----------------------------------------------*)
  solve[
    m0_?numericPositive,
    len0_?numericStrictPositive,
```

```
  eq2_,
  eq3_,
  t_,
  y_,
  θ_,
  delT_?numericStrictPositive(*time length to integrate over for NDSolve*),
  k1_?numericStrictPositive] := Module[{currentSolution, currentY, currentYV,
   currentθ, currentOmega, ic, $x, $xv, $y, $yv, $θ, $ω, simTime, trace, g = 9.81, cm, L0},

  L0 = initialRelaxedLength[len0, m0, M0, k1, g];

  {simTime, $x, $xv, $y, $yv, $θ, $ω, cm, trace} = list@getCurrent[];

  ic = {y[0] == $y, y'[0] == $yv, θ[0] == Mod[$θ, 2 Pi], θ'[0] == $ω};
  currentSolution = Quiet@NDSolve[Flatten@{eq2, eq3, ic},
     {y, y', θ, θ'}, {t, 0, delT}, Method → {"BDF"}, MaxSteps → Infinity];

  currentSolution = First@currentSolution;
  currentY = y[delT] /. currentSolution;
  currentYV = y'[delT] /. currentSolution;
  currentθ = θ[delT] /. currentSolution;
  currentOmega = θ'[delT] /. currentSolution;

  cm = {(L0 + currentY)/2 Sin[currentθ], -(currentY + L0)/2 Cos[currentθ]};
  list@add[{simTime + delT, $x, $xv, currentY, currentYV, currentθ, currentOmega, cm}]
 ];

(*----------------------------------------------*)
self@step[
   m0_?numericPositive, (*pendulum mass*)
   len0_?numericStrictPositive,
   k1_?numericPositive, (*pendulum arm k*)
   M0_?numericPositive, (*bob mass*)
   delT_?numericStrictPositive(*time length to integrate over for NDSolve*)
  ] := Module[{eq1, eq2, eq3, t, y, x, θ, currentθ, g = 9.81},

  (*these are the equations of motion for the case of bob having mass and not*)
  (*these were derived using
   Lagrnagian method. See my report for how this was done, link is below*)

  If[M0 > $MachineEpsilon,
   eq1 = g M0 Sin[θ[t]] + k x[t] - M0 θ'[t] (-y'[t] + x[t] θ'[t]) +
      M0 (y'[t] θ'[t] + x''[t] + (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) θ''[t]) == 0;
   eq2 = 1/2 g m0 Sin[θ[t]] (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) + g M0 (Cos[θ[t]] x[t] +
         Sin[θ[t]] (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t])) + 2/3 m0 (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) y'[t] θ'[t] +
      1/3 m0 (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t])^2 θ''[t] + 1/2 M0 (2 x'[t] (-y'[t] + x[t] θ'[t]) +
         2 y'[t] (x'[t] + (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) θ'[t]) + 2 x[t] (x'[t] θ'[t] - y''[t] + x[t] θ''[t]) +
         2 (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) (y'[t] θ'[t] + x''[t] + (len0 + (g m0)/(3 k1) + (g M0)/k1 + y[t]) θ''[t])) == 0;
```

```
      eq3 = 1/2 g m0 (1 - Cos[θ[t]]) + g M0 (1 - Cos[θ[t]]) + k1 y[t] - 1/3 m0 (len0 + g m0/(3 k1) + g M0/k1 + y[t]) θ′[t]^2 - M0 θ′[
            t] (x′[t] + (len0 + g m0/(3 k1) + g M0/k1 + y[t]) θ′[t]) + 1/3 m0 y″[t] - M0 (x′[t] θ′[t] - y″[t] + x[t] θ″[t]) == 0;
      solve[M0, m0, len0, eq1, eq2, eq3, t, x, y, θ, delT, k1]
      ,
      eq2 = 1/2 g m0 Sin[θ[t]] (len0 + g m0/(3 k1) + y[t]) +
          2/3 m0 (len0 + g m0/(3 k1) + y[t]) y′[t] θ′[t] + 1/3 m0 (len0 + g m0/(3 k1) + y[t])^2 θ″[t] == 0;
      eq3 = 1/2 g m0 (1 - Cos[θ[t]]) + k1 y[t] - 1/3 m0 (len0 + g m0/(3 k1) + y[t]) θ′[t]^2 + 1/3 m0 y″[t] == 0;
      solve[m0, len0, eq2, eq3, t, y, θ, delT, k1]
     ]
    ];

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

  (*--------- private ----------------------------------------*)
  makeCounters[
    m0_?numericPositive, (*pendulum arm mass*)
    M0_?numericPositive, (*bob mass*)
    len0_?numericStrictPositive, (*pendulum length mass*)
    k_?numericPositive,
    k1_?numericPositive] := Module[{h1, h2, currentMomentOfInertia,
     x, xv, y, yv, θ, ω, PE, KE, trace, g = 9.81, simTime, angularMomentum, cm, L0},

    {simTime, x, xv, y, yv, θ, ω, cm, trace} = list@getCurrent[];

    (*these are the  KE and PE for the system. See my report for how this was done, link is below*)
    If[M0 > $MachineEpsilon,
      L0 = len0 + m0 g/(3 k1) + M0 g/k1;
      KE = 1/6 m0 yv^2 + 1/6 m0 (len0 + g m0/(3 k1) + g M0/k1 + y)^2 ω^2 + 1/2 M0 ((-yv + x ω)^2 + (xv + (len0 + g m0/(3 k1) + g M0/k1 + y) ω)^2);
      PE = 1/2 k x^2 + 1/2 k1 y^2 +
         1/2 g m0 (1 - Cos[θ]) (len0 + g m0/(3 k1) + g M0/k1 + y) + g M0 (Sin[θ] x + (1 - Cos[θ]) (len0 + g m0/(3 k1) + g M0/k1 + y))
      ,
      L0 = len0 + m0 g/(3 k1);
      KE = 1/6 m0 yv^2 + 1/6 m0 (len0 + g m0/(3 k1) + y)^2 ω^2;
      PE = 1/2 k1 y^2 + 1/2 g m0 (1 - Cos[θ]) (len0 + g m0/(3 k1) + y)
    ];

    currentMomentOfInertia = m0 (L0 + y) ^ 2 / 3;
```

```
  angularMomentum = m0 (L0 + y) ^2 / 3 * ω + M0 xv (L0 + y)  + M0 (L0 + y) ^2 ω + M0 (x ω - yv) x;

  h1 = Style[Grid[{
       {"θ", "ω", "bob position",
        "bob velocity", Row[{Style["y", Italic], "(", Style["t", Italic], ")"}]},
       {"(degree)", "(rad/sec)", "(meter)", "(meter/sec)", "(meter)"},
       { padIt2[180. / Pi * θ, {5, 2}],
        padIt1[ω, {6, 3}], padIt1[x, {5, 3}], padIt1[xv, {5, 3}], padIt1[y, {4, 3}]}
      },
      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[{
       {Row[{Style["I", Italic], " ω (J sec)"}],
        Row[{"P.E. (J)"}],
        Row[{"K.E. (J)"}],
        Row[{"energy (J)"}]
       },
       { padIt2[angularMomentum, {7, 4}],
        padIt1[PE, {7, 3}],
        padIt1[KE, {6, 3}],
        padIt1[PE + KE, {8, 4}]
       }},
      Frame → All,
      FrameStyle → Gray,
      Spacings → 1,
      ItemSize → {{All, 2 ;; -1} → 6},
      Alignment → Center], 12];

  Grid[{{h1}, {h2}}, Spacings → {0, .1}]
 ];
(*---------------private----------------------------------*)
makeDiskDiagram[m0_?numericStrictPositive,
  M0_?numericPositive,
  r0_?numericStrictPositive,
  {width_?numericStrictPositive, height_?numericStrictPositive},
  traceBob_?bool,
  traceCM_?bool,
  w_?numericStrictPositive,
  k1_?numericStrictPositive,
  len0_?numericStrictPositive
 ] := Module[{a = 2 r0, b = 0.5 r0, x, y, xx, xv, yy, yv, θ, ω, traceData, traceOn = False, spring, pin,
   pin0, bob, simTime, frame0, frame1, springArm, bobHasMass, cm, cg, g = 9.81, L0, options},

  bobHasMass = M0 > $MachineEpsilon;
  If[bobHasMass, L0 = len0 + m0 g / (3 k1) + M0 g / k1, L0 = len0 + m0 g / (3 k1)];

  {simTime, xx, xv, yy, yv, θ, ω, cm, currentTrace} = list@getCurrent[];
  cg = Text[Style["⊕", 14], cm];

  If[traceBob,
   traceData = list@getBobTraceData[];
   traceOn = True
  ];

  If[bobHasMass,
   x = xx * Cos[θ];
```

```
    y = xx * Sin[θ];
    bob = {Red, Opacity[M0 / 0.1], EdgeForm[Thin], Disk[{L0 + yy + r0, xx}, r0]};
    (*bob spring*)
    spring = {Gray, Line@makeSpring[L0 + yy + r0, 0, L0 + yy + r0, xx, r0, 14]};
    (*tube frame*)
    frame0 = {Blue, Line[{ {L0 + yy , -w / 2}, {L0 + yy, w / 2}}]};
    frame1 = {Blue, Line[{ {L0 + yy + a , -w / 2}, {L0 + yy + a, w / 2}}]};
    pin0 = {Blue, EdgeForm[Thin], Disk[{L0 + yy + r0, 0}, b / 2]};
   ];

   pin = {Blue, EdgeForm[Thin], Disk[{0, 0}, b / 2]};
   (*pendulum arm*)
   springArm = {Thickness[0.004], Line@makeSpring[0, 0, L0 + yy, 0, r0, 24]};
   options = {Axes → False,
     PlotRange → {{-w, w}, {-w , w / 2}},
     ImageSize → {width, height},
     ImagePadding → All,
     ImageMargins → 0,
     AxesOrigin → {0, 0},
     PlotRangeClipping → False,
     PlotRangePadding → None,
     Frame → True,
     AspectRatio → 1.15,
     GridLines → Automatic,
     GridLinesStyle → Directive[Gray, Dashed]};

   If[bobHasMass,
    Graphics[{
      Rotate[frame0, -(Pi / 2) + θ, {0, 0}],
      Rotate[frame1, -(Pi / 2) + θ, {0, 0}],
      Rotate[{Black, spring}, -(Pi / 2) + θ, {0, 0}],
      Rotate[{Black, springArm}, -(Pi / 2) + θ, {0, 0}],
      Rotate[bob, -(Pi / 2) + θ, {0, 0}],
      Rotate[pin, -(Pi / 2) + θ, {0, 0}],
      Rotate[pin0, -(Pi / 2) + θ, {0, 0}],
      If[traceCM, cg, Sequence @@ {}],
      If[traceOn, {Thin, Magenta, Line[traceData]}, Sequence @@ {}]
     }, options
    ],
    Graphics[{
      Rotate[{Black, springArm}, -(Pi / 2) + θ, {0, 0}],
      Rotate[pin, -(Pi / 2) + θ, {0, 0}],
      If[traceCM, cg, Sequence @@ {}]
     }, options
    ]
   ]
  ];
(*-------------------- 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 (sec)", 12], 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 (sec)", 12], Style[Row[{"disk angle θ(", Style["t", Italic], ") 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["position (meter)", 12], None},
      {Style["time (sec)", 12], 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 (sec)", 12], Style["bob velocity vs. time", 12]}};
    plotRange = {{data[[1, 1]], data[[-1, 1]]}, All}
    ,
    plotType == "vertical displacement",
    data = list@getYList[];
    If[Length[data] == 0, data = {{0, 0}}];
    plotTitle = {{Style[Row[{Style["y", Italic], " (meter)"}], 12], None},
      {Style["time (sec)", 12], Style[Row[{"vertical displacement ",
          Style["y", Italic], "(", Style["t", Italic], ") 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 → {300, 160},
    ImagePadding → {{50, 10}, {35, 20}},
    ImageMargins -> {{2, 1}, {1, 1}},
    Axes → False,
    FrameLabel → plotTitle,
    PlotStyle → Red,
    AspectRatio → 0.38]
  ];
 (*------- public ----------------------------------*)
 self@makeDisplay[plotType_String,
    m0_?numericPositive,
    M0_?numericPositive,
    len0_?numericStrictPositive,
    r0_?numericStrictPositive,
    k1_?numericPositive,
    showPlots_?bool,
    showCounters_?bool,
    traceCM_?bool,
    traceBob_?bool,
    w_] :=
  If[showPlots,
   If[showCounters,
    Grid[{
      {makeCounters[m0, M0, len0, k, k1]},
      {makePlot[plotType, w]},
      {makeDiskDiagram[m0, M0, r0, {contentSizeW, 0.695 contentSizeW}, traceBob, traceCM, w, k1, len0]}
     }, Spacings → 0, Frame → None]
    ,
    Grid[{
      {makePlot[plotType, w]},
      {makeDiskDiagram[m0, M0, r0, {contentSizeW, 1.02 contentSizeW}, traceBob, traceCM, w, k1, len0]}
```

```
      }, Spacings → 0, Frame → None]
    ],
    If[showCounters,
     Grid[{
       {makeCounters[m0, M0, len0, k, k1]},
       {makeDiskDiagram[m0, M0, r0, {contentSizeW, 1.25 contentSizeW}, traceBob, traceCM, w, k1, len0]}
      }, Spacings → 0, Frame → None]
     ,
     Grid[{
       {makeDiskDiagram[m0, M0, r0,
         {1.01 contentSizeW, 1.575 contentSizeW}, traceBob, traceCM, w, k1, len0]}
      }, Spacings → 0, Frame → None]
    ]
   ];

  self
 ];

(*-------------------------*)
initialRelaxedLength[len0_?numericStrictPositive, m0_?numericStrictPositive, M0_?numericPositive,
  k_?numericStrictPositive, g_?numericStrictPositive] := (len0 + m0 g/(3 k) + M0 g/k);
(*-------------------------*)
calculateCenterOfMass[m0_?numericStrictPositive,
  M0_?numericPositive, y_?numeric, θ_?numeric, x_?numeric] :=
 {1/(m0 + M0) (m0 y/2 Sin[θ] + M0 (y Sin[θ] + x Cos[θ])), 1/(m0 + M0) (-m0 y/2 Cos[θ] + M0 (-y Cos[θ] + x Sin[θ]))};

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

  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, nTurns - 2}
     ];

    {{xFirst, yFirst}} ~ Join ~ {{xFirst + hx * (dh + veghossz) / hossz, yFirst + hy * (dh + veghossz) / hossz}} ~
     Join ~ tbl ~ Join ~ {{xFirst + hx * ((nTurns - 1) * dh + veghossz) / hossz,
       yFirst + hy * ((nTurns - 1) * dh + veghossz) / hossz}} ~ Join ~ {{xEnd, yEnd}}
   ];

  (*--- constant parameters size and width of display ---*)
  contentSizeW = 300;
  contentSizeH = 405;

  (* objects used by the simulation. These must be here in the initialization section *)
  (*listClass[size,r0,L0], where L0=len0+m0 g/(3 k1)+M0 g/k1*)
  list = listClass[500, 0.05, 0.6 + (1 × 9.81)/(3 × 500) + (10 × 9.81)/500];
  (*list@add[{time,x,vx,y,vy,θ,ω,cm*)
  list@add[{0, -0.35, 0.0, 0.1, 0.0, 15 Degree, 0.0, {0, 0}}];

  solver = solverClass[];
  display = displayClass[];
 }
]
```

speed (slow) (fast)

display zoom (in) (out)

top spring initial conditions

spring mass 1.00

spring stiffness 0500

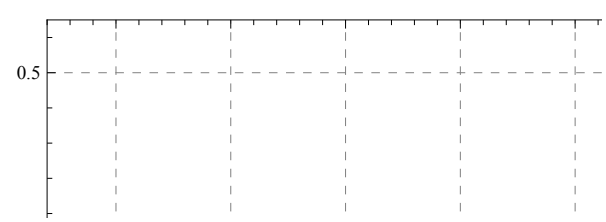
angular velocity +0.00

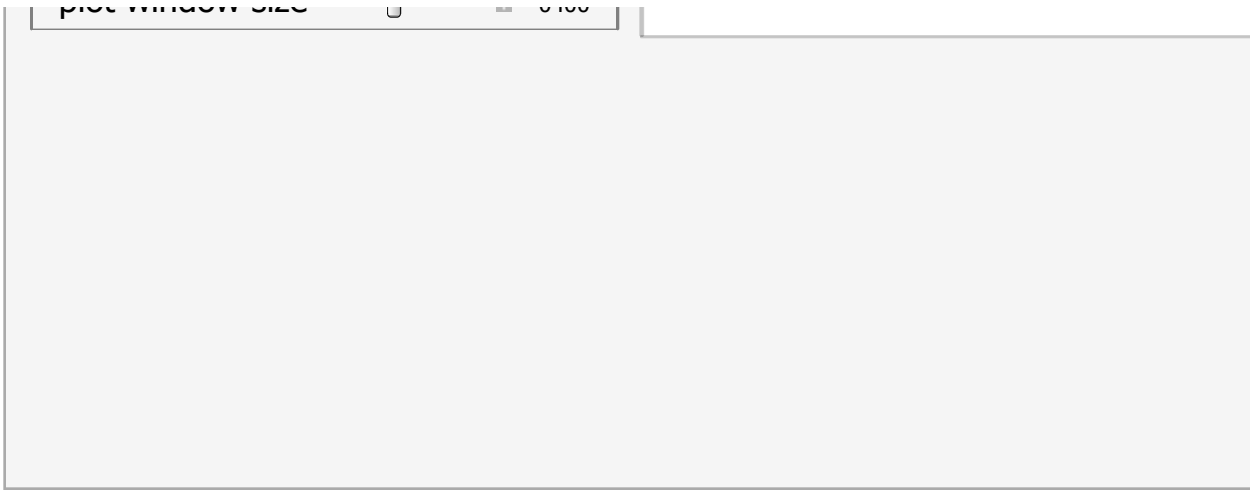
angle (degree) +015

initial extension +0.10

θ (degree)	ω (rad/sec)	bob position (meter)	bob velocity (meter/sec)	$y($ (met
015.00	+000.000	−00.350	+00.000	+0.1

$I\ \omega$ (J sec)	P.E. (J)	K.E. (J)	energy (J)
000.0000	+0027.407	+000.000	+0027.4069





Details (optional)

description of the user interface

The four buttons on the right side are used to control the Demonstration. The reset button brings the Demonstration back to the initial conditions. There are two main sets of control variables. One set is for setting the initial conditions for the top spring, and the second set is used for setting the initial conditions for the lower spring. The "relax" check box for the lower spring is used to put the spring into the relaxed position. For the top spring the relaxed position is set at $60\,\mathrm{cm}$, while the lower spring is assumed to have zero relaxed length located at the point where the two springs are joined. You can see a trace of the bob motion using the "trace" check box. You can see the location of the center of mass (CM) of the whole system using the "show CM" checkbox.
The result has three parts. The counters 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 $I\,\omega$ is the total angular momentum of the system. The field labeled $y(t)$ is the current extension of the top spring measured from its relaxed position. You can see different plots by selecting the plot type using the pop-up menu labeled "plot type". The main display shows the pendulum itself as it swings.
A pop-up menu labeled "test case" is used to select one of three preconfigured 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 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. A slider labeled "speed" can be used to adjust the speed of the Demonstration as needed.
The Demonstration runs continuously until you stop it. All units used are in SI.

derivation of equations of motion

The equations of motion are derived using the Lagrangian procedure. Raleigh's method is used for the top spring such that the effective mass used is $\frac{1}{3}$ of the top spring mass. This effective mass is added to the bob mass to account for the vertical displacement of the top spring. The bob has three degrees of freedom: the distance $x(t)$ is along the length of lower spring where $x = 0$ is the location where the top pendulum attached to the lower pendulum, the angle of rotation $\theta(t)$ and $y(t)$, the vertical displacement of the top spring. For more information on the derivation of the equations of motion, please see the author's report.
References

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

[2] REA's *Problem Solver's Mechanics Static and Dynamics,* New Jersey: Research and Education Association, 2002.

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

[4] S. Timoshenko, *Vibration Problems In Engineering,* New York: D. Van. Nostrand, 1937.

[5] Wikipedia page on effective mass for spring-mass system

Control Suggestions (optional)

- ☑ Resize Images
- ☑ Rotate and Zoom in 3D