# Numerical HW 3

```
Clear["Global`*"]
```

---

## Setup

- ### Write out the Lagrange equations of motion

- **Start with position information**

```
x1 := L1 Sin[th[t]]
y1 := -L1 Cos[th[t]]
x2 := x1 + L2 Sin[ph[t]]
y2 := y1 - L2 Cos[ph[t]]
pos1 := {x1, y1};
pos2 := {x2, y2};
```

```
pos1
```

$$\{L1\,\text{Sin}[th[t]],\ -L1\,\text{Cos}[th[t]]\}$$

```
pos2
```

$$\{L2\,\text{Sin}[ph[t]] + L1\,\text{Sin}[th[t]],\ -L2\,\text{Cos}[ph[t]] - L1\,\text{Cos}[th[t]]\}$$

- **Have mathematica calculate the velocity for you.**

```
x1dot := D[x1, t]
y1dot := D[y1, t]
x2dot := D[x2, t]
y2dot := D[y2, t]
```

- **Get the potential, kinetic energy and then lagrangian.**

```
U := FullSimplify[g * (M1 * y1 + M2 * y2)]
K := FullSimplify[1/2 M1 (x1dot^2 + y1dot^2) + 1/2 M2 (x2dot^2 + y2dot^2)]
L := FullSimplify[K - U]
```

```
U
```

$$-g\,(L2\,M2\,\text{Cos}[ph[t]] + L1\,(M1 + M2)\,\text{Cos}[th[t]])$$

```
K
```

$$\frac{1}{2}\left(L2^2\,M2\,ph'[t]^2 + 2\,L1\,L2\,M2\,\text{Cos}[ph[t] - th[t]]\,ph'[t]\,th'[t] + L1^2\,(M1 + M2)\,th'[t]^2\right)$$

- **Solve the Lagrange equations of motion using matehatica, and then solve for the second time derivatives.**

```
LagrangeSolver[L_] := (
  theq = (Simplify[∂th[t] L - ∂t (∂th'[t] L)] == 0);
  pheq = (Simplify[∂ph[t] L - ∂t (∂ph'[t] L)] == 0);
  Solve[{theq, pheq}, {ph''[t], th''[t]}][[1]]
 )
```

- **Remove the functional dependence to get it into a form that fits better with putting in numbers**

```
tRemoverRule := {th[t] → th, th'[t] → thd, ph[t] → ph, ph'[t] → phd};
```

```
Clear[t]
```

- **Make functions for putting in numbers**

```
thdd[th_, thd_, ph_, phd_] =
 FullSimplify[th''[t] /. LagrangeSolver[L] /. tRemoverRule]
phdd[th_, thd_, ph_, phd_] =
 FullSimplify[ph''[t] /. LagrangeSolver[L] /. tRemoverRule]
kin[th_, thd_, ph_, phd_] = FullSimplify[U /. LagrangeSolver[L] /. tRemoverRule]
pot[th_, thd_, ph_, phd_] = FullSimplify[K /. LagrangeSolver[L] /. tRemoverRule]
```

$$\left(2\,L2\,M2\,phd^2\,\text{Sin}[ph-th]+L1\,M2\,thd^2\,\text{Sin}[2\,(ph-th)]+g\,M2\,\text{Sin}[2\,ph-th]-g\,(2\,M1+M2)\,\text{Sin}[th]\right)\Big/\left(2\,L1\left(M1+M2-M2\,\text{Cos}[ph-th]^2\right)\right)$$

$$-\left(\left(L2\,M2\,phd^2\,\text{Cos}[ph-th]+(M1+M2)\left(L1\,thd^2+g\,\text{Cos}[th]\right)\right)\text{Sin}[ph-th]\right)\Big/\left(L2\left(M1+M2-M2\,\text{Cos}[ph-th]^2\right)\right)$$

$$-g\,(L2\,M2\,\text{Cos}[ph]+L1\,(M1+M2)\,\text{Cos}[th])$$

$$\frac{1}{2}\left(L2^2\,M2\,phd^2+L1^2\,(M1+M2)\,thd^2+2\,L1\,L2\,M2\,phd\,thd\,\text{Cos}[ph-th]\right)$$

```
thdd[th, thd, ph, phd] /. M1 → 1 /. M2 → 1 /. L1 → l /. L2 → l // FullSimplify
```

$$\left(-2\,l\left(phd^2+thd^2\,\text{Cos}[ph-th]\right)\text{Sin}[ph-th]+g\,(-\text{Sin}[2\,ph-th]+3\,\text{Sin}[th])\right)\Big/\left(l\,(-3+\text{Cos}[2\,(ph-th)])\right)$$

# Global Parameters

- **Put in the parameters for the problem**

```
pParams = {
    g → 9.8, (*m/s^2*)
    L2 → 0.25, (*m*)
    L1 → 0.25, (*m*)
    M1 → 3, (*kg*)
    M2 → 3(*kg*)
   };
```

# Calculation Modules

- **This calculates the angular positions as a function of time.**

```
doublePendulumMod[initialParams_, physicalParams_, tMax_, dt_, print_] :=
 Module[{
   iMax = (Floor[tMax / dt] + 1),
   lastI
  },
  (*Initialize my tables*)
  thArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  thdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  thddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  phArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  phdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  phddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  kinArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  potArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  eTotArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
  (*Set my initial conditions*)
  thArr[[1, 2]] = th0 * Pi / 180 /. initialParams;
  thdArr[[1, 2]] = thd0 * Pi / 180 /. initialParams;
  phArr[[1, 2]] = ph0 * Pi / 180 /. initialParams;
  phdArr[[1, 2]] = phd0 * Pi / 180 /. initialParams;
  (*Set the acceleration guys taking into account
   constants so we don't have to do it each time in the loop*)
  thdd2[th_, thd_, ph_, phd_] = thdd[th, thd, ph, phd] /. physicalParams;
  phdd2[th_, thd_, ph_, phd_] = phdd[th, thd, ph, phd] /. physicalParams;
  kin2[th_, thd_, ph_, phd_] = kin[th, thd, ph, phd] /. physicalParams;
  pot2[th_, thd_, ph_, phd_] = pot[th, thd, ph, phd] /. physicalParams;
  (*Go through iterations keeping track of last iteration number*)
  lastI = 1;

  Do[
```

```
  lastI = i;
  kinArr[[i, 2]] =
   kin2[thArr[[i, 2]], thdArr[[i, 2]], phArr[[i, 2]], phdArr[[i, 2]]];
  potArr[[i, 2]] = pot2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  eTotArr[[i, 2]] = kinArr[[i, 2]] + potArr[[i, 2]];
  thddArr[[i, 2]] =
   thdd2[thArr[[i, 2]], thdArr[[i, 2]], phArr[[i, 2]], phdArr[[i, 2]]];
  phddArr[[i, 2]] = phdd2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  thdArr[[i + 1, 2]] = thdArr[[i, 2]] + thddArr[[i, 2]] * dt;
  phdArr[[i + 1, 2]] = phdArr[[i, 2]] + phddArr[[i, 2]] * dt;
  thArr[[i + 1, 2]] =
   thArr[[i, 2]] + thdArr[[i, 2]] * dt + 0.5 * thddArr[[i, 2]] * dt^2;
  phArr[[i + 1, 2]] = phArr[[i, 2]] + phdArr[[i, 2]] * dt + 0.5 * phddArr[[i, 2]] * dt^2;
  , {i, 1, iMax - 1}];
 lastI
]
```

- **This is an improvement on the preceding module. It does exactly the same thing, but runs 10 times as fast.**

```
doublePendulumCmp = Compile[
   {
    {th0, _Real},
    {thd0, _Real},
    {ph0, _Real},
    {phd0, _Real},
    {tMax, _Real},
    {dt, _Real}
   },
   Module[{
     iMax = (Floor[tMax / dt] + 1),
     lastI
    },
    (*Initialize my tables*)
    thArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    thdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    thddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    phArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    phdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    phddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    kinArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    potArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    eTotArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    eErrArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
    (*Set my initial conditions*)
    thArr[[1, 2]] = th0 * Pi / 180;
    thdArr[[1, 2]] = thd0 * Pi / 180;
```

```
 phArr[[1, 2]] = ph0 * Pi / 180;
 phdArr[[1, 2]] = phd0 * Pi / 180;
 (*Set the acceleration guys taking into account
  constants so we don't have to do it each time in the loop*)
 thdd2[th_, thd_, ph_, phd_] = thdd[th, thd, ph, phd] /. pParams;
 phdd2[th_, thd_, ph_, phd_] = phdd[th, thd, ph, phd] /. pParams;
 kin2[th_, thd_, ph_, phd_] = kin[th, thd, ph, phd] /. pParams;
 pot2[th_, thd_, ph_, phd_] = pot[th, thd, ph, phd] /. pParams;
 (*Go through iterations keeping track of last iteration number*)
 lastI = 1;

 Do[
  lastI = i;
  kinArr[[i, 2]] =
   kin2[thArr[[i, 2]], thdArr[[i, 2]], phArr[[i, 2]], phdArr[[i, 2]]];
  potArr[[i, 2]] = pot2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  eTotArr[[i, 2]] = kinArr[[i, 2]] + potArr[[i, 2]];
  eErrArr[[i, 2]] =
   100 * Abs[(eTotArr[[i, 2]] - eTotArr[[1, 2]]) / eTotArr[[1, 2]]];
  thddArr[[i, 2]] = thdd2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  phddArr[[i, 2]] = phdd2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  If[i < iMax,
   thdArr[[i + 1, 2]] = thdArr[[i, 2]] + thddArr[[i, 2]] * dt;
   phdArr[[i + 1, 2]] = phdArr[[i, 2]] + phddArr[[i, 2]] * dt;
   thArr[[i + 1, 2]] =
    thArr[[i, 2]] + thdArr[[i, 2]] * dt + 0.5 * thddArr[[i, 2]] * dt^2;
   phArr[[i + 1, 2]] = phArr[[i, 2]] + phdArr[[i, 2]] * dt +
     0.5 * phddArr[[i, 2]] * dt^2;
  ];
  , {i, 1, iMax}];
  lastI
 ]
];
```

- **This uses a Runge-Kutta method (4th order in dt) for numeric integration. I don't have it working as of yet. I might finish it some day.**

```
doublePendulumRK = Compile[
   {
    {th0, _Real},
    {thd0, _Real},
    {ph0, _Real},
    {phd0, _Real},
    {tMax, _Real},
    {dt, _Real}
```

```
},
Module[{
  iMax = (Floor[tMax / dt] + 1),
  lastI
 },
 (*Initialize my tables*)
 thArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 thdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 thddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 phArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 phdArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 phddArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 kinArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 potArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 eTotArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 eErrArr = Table[{dt * (i - 1), 0}, {i, 1, iMax}];
 (*Set my initial conditions*)
 thArr[[1, 2]] = th0 * Pi / 180;
 thdArr[[1, 2]] = thd0 * Pi / 180;
 phArr[[1, 2]] = ph0 * Pi / 180;
 phdArr[[1, 2]] = phd0 * Pi / 180;
 (*Set the acceleration guys taking into account
  constants so we don't have to do it each time in the loop*)
 thdd2[th_, thd_, ph_, phd_] = thdd[th, thd, ph, phd] /. pParams;
 phdd2[th_, thd_, ph_, phd_] = phdd[th, thd, ph, phd] /. pParams;
 kin2[th_, thd_, ph_, phd_] = kin[th, thd, ph, phd] /. pParams;
 pot2[th_, thd_, ph_, phd_] = pot[th, thd, ph, phd] /. pParams;
 (*Go through iterations keeping track of last iteration number*)
 lastI = 1;

 Do[
  lastI = i;
  kinArr[[i, 2]] =
   kin2[thArr[[i, 2]], thdArr[[i, 2]], phArr[[i, 2]], phdArr[[i, 2]]];
  potArr[[i, 2]] = pot2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  eTotArr[[i, 2]] = kinArr[[i, 2]] + potArr[[i, 2]];
  eErrArr[[i, 2]] =
   100 * Abs[(eTotArr[[i, 2]] - eTotArr[[1, 2]]) / eTotArr[[1, 2]]];
  thddArr[[i, 2]] = thdd2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  phddArr[[i, 2]] = phdd2[thArr[[i, 2]], thdArr[[i, 2]],
    phArr[[i, 2]], phdArr[[i, 2]]];
  k1Th = thddArr[[i, 2]];
  k1Ph = phddArr[[i, 2]];
  thdTmp = thdArr[[i, 2]] + 0.5 * k1Th;
  thTmp = thArr[[i, 2]] + 0.5 * (thdTmp + thdArr[[i, 2]]) * dt;
```

```
    phdTmp = phdArr[[i, 2]] + 0.5 * k1Ph;
    phTmp = phArr[[i, 2]] + 0.5 * (phdTmp + phdArr[[i, 2]]) * dt;
    k2Th = thdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    k2Ph = phdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    thdTmp = thdArr[[i, 2]] + 0.5 * k2Th;
    thTmp = thArr[[i, 2]] + 0.5 * (thdTmp + thdArr[[i, 2]]) * dt;
    phdTmp = phdArr[[i, 2]] + 0.5 * k2Ph;
    phTmp = phArr[[i, 2]] + 0.5 * (phdTmp + phdArr[[i, 2]]) * dt;
    k3Th = thdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    k3Ph = phdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    thdTmp = thdArr[[i, 2]] + k3Th;
    thTmp = thArr[[i, 2]] + 0.5 * (thdTmp + thdArr[[i, 2]]) * dt;
    phdTmp = phdArr[[i, 2]] + k3Ph;
    phTmp = phArr[[i, 2]] + 0.5 * (phdTmp + phdArr[[i, 2]]) * dt;
    k4Th = thdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    k4Ph = phdd2[thTmp, thdTmp, phTmp, phdTmp] * dt;
    If[i < iMax,
     thdArr[[i + 1, 2]] = thdArr[[i, 2]] + (k1Th + 2 k2Th + 2 k3Th + k4Th) / 6;
     phdArr[[i + 1, 2]] = phdArr[[i, 2]] + (k1Ph + 2 k2Ph + 2 k3Ph + k4Ph) / 6;
     thArr[[i + 1, 2]] = thArr[[i, 2]] + 0.5 (thdArr[[i, 2]] + thdArr[[i + 1, 2]]) * dt;
     phArr[[i + 1, 2]] = phArr[[i, 2]] + 0.5 (phdArr[[i, 2]] + phdArr[[i + 1, 2]]) * dt;
    ];
    , {i, 1, iMax}];
   lastI
  ]
 ];
```

- **This runs the previous compiled module over and over perturbing the initial conditions showing how for small initial angles, the final result is in fact stable, but it ends up chaotic after some time.**

```
finalPos = Compile[{
    {th00, _Real},
    {th0Max, _Real},
    {dTh, _Real},
    {perturbTh, _Real},
    {nPerturb, _Integer},
    {tMax, _Real},
    {dt, _Real}},
   Module[{
     phi0 = 2 * th00,
     iMax = Floor[(th0Max - th00) / dTh] + 1
    },
    (*Initialize tables to hold final answers*)
    thFinalArr =
     Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0}, {j, 1, nPerturb + 1}, {i, 1, iMax}];
    thdFinalArr = Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0},
      {j, 1, nPerturb + 1}, {i, 1, iMax}];
    thddFinalArr = Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0},
      {j, 1, nPerturb + 1}, {i, 1, iMax}];
    phFinalArr = Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0},
      {j, 1, nPerturb + 1}, {i, 1, iMax}];
    phdFinalArr = Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0},
      {j, 1, nPerturb + 1}, {i, 1, iMax}];
    phddFinalArr = Table[{dTh * (i - 1) + (j - 1) * perturbTh, 0},
      {j, 1, nPerturb + 1}, {i, 1, iMax}];
    (*Perform the loop to grab the final values of those functions*)
    Do[
     th0 = thFinalArr[[j, i, 1]];
     ph0 = 2 * th0;
     lastI = doublePendulumCmp[th0, 0, ph0, 0, tMax, dt];
     thFinalArr[[j, i, 2]] = thArr[[lastI, 2]];
     thdFinalArr[[j, i, 2]] = thdArr[[lastI, 2]];
     thddFinalArr[[j, i, 2]] = thddArr[[lastI, 2]];
     phFinalArr[[j, i, 2]] = phArr[[lastI, 2]];
     phdFinalArr[[j, i, 2]] = phdArr[[lastI, 2]];
     phddFinalArr[[j, i, 2]] = phddArr[[lastI, 2]];
     NotebookDelete[printOut];
     printOut = PrintTemporary[StringJoin[ToString[i], " of ",
        ToString[iMax], ", ", ToString[j], " of ", ToString[nPerturb + 1]]];
     , {j, 1, nPerturb + 1}, {i, 1, iMax}]
   ]
  ];
```

# Plotting functions

- **Here are some viewing things. The first will be used to plot an animation of the motion of the object. Reall there for cool effect rather than for science.**

```
posViewer[th1_, ph1_, params_] := (
  Clear[t];
  vpos1 = pos1 /. {th[t] → th1, ph[t] → ph1} /. params;
  vpos2 = pos2 /. {th[t] → th1, ph[t] → ph1} /. params;
  Graphics1 = {Red, Disk[vpos1, 0.03]};
  Graphics2 = {Blue, Disk[vpos2, 0.03]};
  GraphicsLine1 = {Thick, Black, Line[{{0, 0}, vpos1, vpos2}]};
  scale = 1.2;
  Graphics[Flatten[{GraphicsLine1, Graphics1, Graphics2}],
   PlotRange → ({{-(L1 + L2) * scale, (L1 + L2) * scale},
        {-(L1 + L2) * scale, (L1 + L2) * scale}} /. params)]
 )
```

- **Plots of the functions of time**

```
thPlot := ListPlot[thArr, PlotRange → All,
  Joined → True, AxesLabel → {"time[s]", "θ[radians]"}]
thdPlot := ListPlot[thdArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "dθ/dt[radians/sec]"}]
thddPlot := ListPlot[thddArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "d²θ/dt²[radians/sec²]"}]
phPlot := ListPlot[phArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "ϕ[radians]"}]
phdPlot := ListPlot[phdArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "dϕ/dt[radians/sec]"}]
phddPlot := ListPlot[phddArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "d²ϕ/dt²[radians/sec²]"}]
ePlot := ListPlot[{kinArr, potArr, eTotArr}, PlotRange → All,
  Joined → True, AxesLabel → {"time[s]", "Energies[J]"}]
eErrPlot := ListPlot[eErrArr, PlotRange → All, Joined → True,
  AxesLabel → {"time[s]", "% error in total energy"}]
```

- **Plots of the final items as a function of time**

```
thFPlot := ListPlot[thFinalArr, PlotRange → All, Joined → True]
thdFPlot := ListPlot[thdFinalArr, PlotRange → All, Joined → True]
thddFPlot := ListPlot[thddFinalArr, PlotRange → All, Joined → True]
phFPlot := ListPlot[phFinalArr, PlotRange → All, Joined → True]
phdFPlot := ListPlot[phdFinalArr, PlotRange → All, Joined → True]
phddFPlot := ListPlot[phddFinalArr, PlotRange → All, Joined → True]
```

# Somes checks for the calculation and some general comments

- **I wrote this originally using a module, and then as a compiled function. Note that the compiled function runs more than 10 times faster than the bare module. I stopped using the module at some point so there are some things in the compiled version that aren't in the module.**

```
Timing[
 doublePendulumMod[{th0 → 10, thd0 → 0, ph0 → 20, phd0 → 0}, pParams, 10, .001, 0]]
```

```
{66.625, 10 000}
```

```
Timing[doublePendulumCmp[10, 0, 20, 0, 10, 0.001]]
```

```
{3.469, 10 001}
```

```
Timing[doublePendulumCmp[10, 0, 20, 0, 0.5, 0.001]]
```

```
{0.234, 501}
```

```
Timing[doublePendulumRK[10, 0, 20, 0, 10, 0.01]]
```

```
{1.406, 1001}
```

```
thArr[[5]]
```

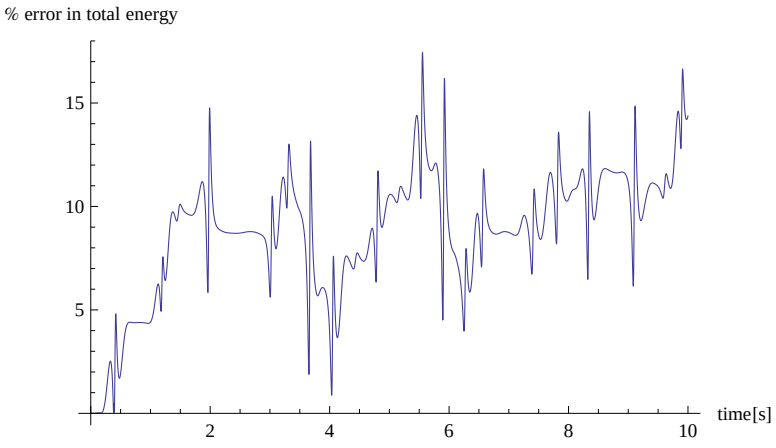
```
{0.004, 0.175809}
```

- **Check energy conservation and the animation for an array of different starting situations (different starting velocities and angular positions). Make sure the animation looks like it should for the first bit of time (it matches the initial conditions) and make sure energy is relatively well conserved. For the more chaotic motions, you'll have a hard time keeping that error below 15%.**

```
doublePendulumCmp[90, 0, 180, 0, 10, 0.001]
```

```
10 001
```

```
eErrPlot
```

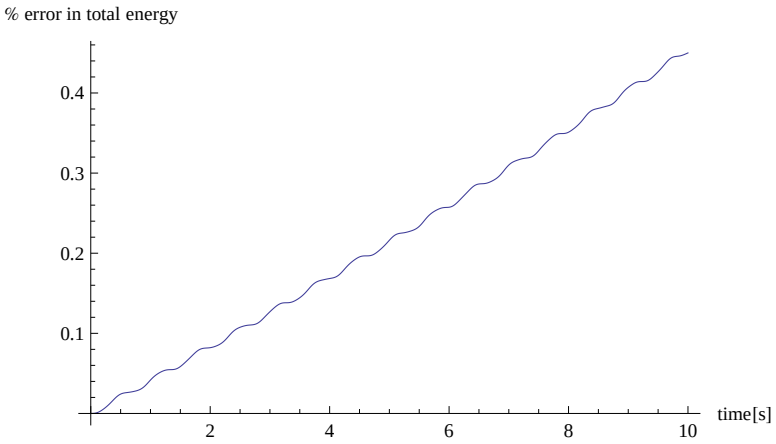



# Results (the stuff you're graded on)

- **a) Plot the % error as a function of time in the total energy of the system using dt=0.001, tMax=10 sec and solve for the final error in energy for the following:**

**th0=10 degrees and ph0=20 degrees (initial velocities are zero)**
**th0=30 degrees and ph0=60 degrees (initial velocities are zero)**
**th0=90 degrees and ph0=180 degrees (initial velocities are zero)**
**th0=0 degrees and ph0=0 degrees and phd0=50 degrees/sec.**
**th0=0 degrees and ph0=0 degrees and phd0=100 degrees/sec.**
**th0=0 degrees and ph0=0 degrees and phd0=500 degrees/sec.**

```
lastI = doublePendulumCmp[10, 0, 20, 0, 10, 0.001];
eErrPlot
```




```
lastI = doublePendulumCmp[10, 0, 20, 0, 10, 0.001];
eErrArr[[lastI, 2]]
lastI = doublePendulumCmp[30, 0, 60, 0, 10, 0.001];
eErrArr[[lastI, 2]]
lastI = doublePendulumCmp[90, 0, 180, 0, 10, 0.001];
eErrArr[[lastI, 2]]
lastI = doublePendulumCmp[0, 0, 0, 50, 10, 0.001];
eErrArr[[lastI, 2]]
lastI = doublePendulumCmp[0, 0, 0, 100, 10, 0.001];
eErrArr[[lastI, 2]]
lastI = doublePendulumCmp[0, 0, 0, 500, 10, 0.001];
eErrArr[[lastI, 2]]
```

```
0.450017
```

```
5.25594
```

```
14.3712
```

```
0.0786237
```

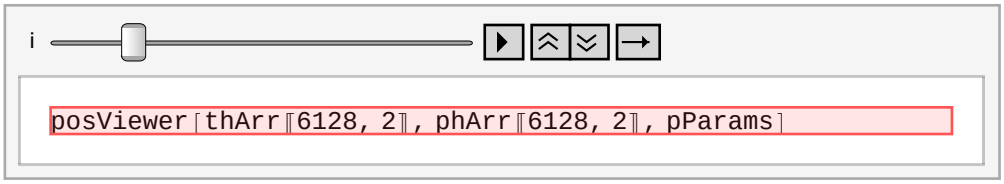
```
0.311803
```

```
8.20533
```

- **b) Animate the case where th0=10 degrees, ph0=20 degrees, and also for th0=90 degrees and ph0=180 degrees. Give a snapshot of the final time for the second run to prove you did it.**

```
lastI = doublePendulumCmp[181, 0, 180, 0, 40, 0.001];
```

```
Animate[posViewer[thArr[[i, 2]], phArr[[i, 2]], pParams], {i, 1, Length[thArr], 1}]
```



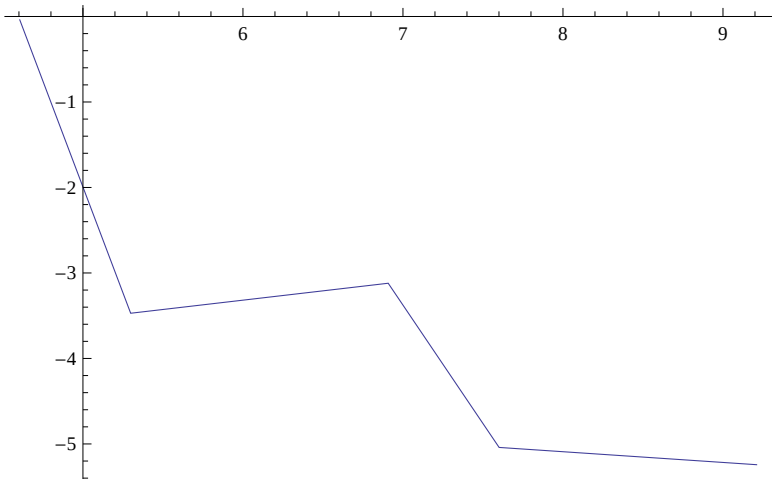

- **I added this in because the animator automatically changes when I run stuff below.**

```
posViewer[thArr[[lastI, 2]], phArr[[lastI, 2]], pParams]
```

- **c) Do a convergence study. Plot the final th with respect to dt for th0=20 degrees and ph0=40 degrees for dt=0.1, dt=0.01, 0.05, 0.001,0.0005, and 0.0001 for a tMax of 1. Plot -log(dt) vs log(chnage in final theta) and show that the change in final theta is approaching zero, log goes to minus infinity.**

```
finalTh = {{0.01, 0}, {0.005, 0}, {0.001, 0}, {0.0005, 0}, {0.0001, 0}};
finalThPlot = Table[0, {i, 5}];
Do[
  finalI = doublePendulumCmp[20, 0, 40, 0, 1, finalTh[[i, 1]]];
  finalTh[[i, 2]] = thArr[[finalI, 2]];
  finalThPlot[[i]] = {-Log[finalTh[[i, 1]]], If[i == 1, finalTh[[i, 2]],
     Log[Abs[(finalTh[[i, 2]] - finalTh[[i - 1, 2]]) / finalTh[[i - 1, 2]]]]]};
  NotebookDelete[printOut];
  printOut = PrintTemporary[i];
  , {i, 1, Length[finalTh]}];
```

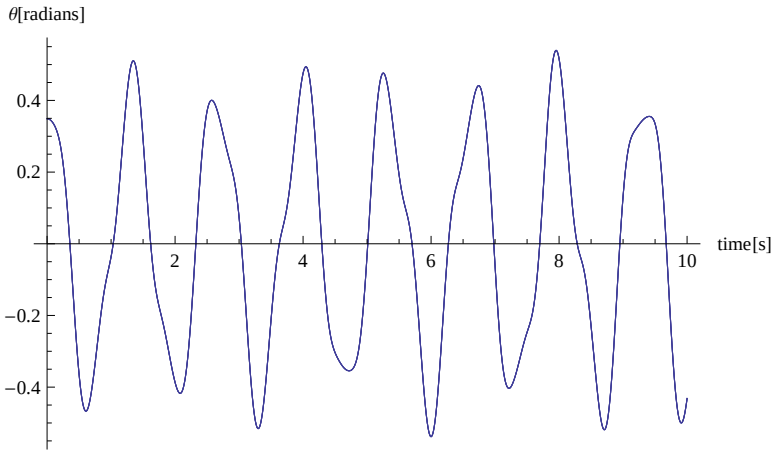
```
ListPlot[finalThPlot, Joined → True]
```




- **d) Plot th as a function of time using a dt of 0.001 and for th0=20 and for ph0=40, 40.01, 40.02, 40.03, tMax=10.**

```
finalI = doublePendulumCmp[20, 0, 40, 0, 10, 0.001];
thplt1 = thPlot;
finalI = doublePendulumCmp[20, 0, 40.01, 0, 10, 0.001];
thplt2 = thPlot;
finalI = doublePendulumCmp[20, 0, 40.02, 0, 10, 0.001];
thplt3 = thPlot;
finalI = doublePendulumCmp[20, 0, 40.03, 0, 10, 0.001];
thplt4 = thPlot;
```

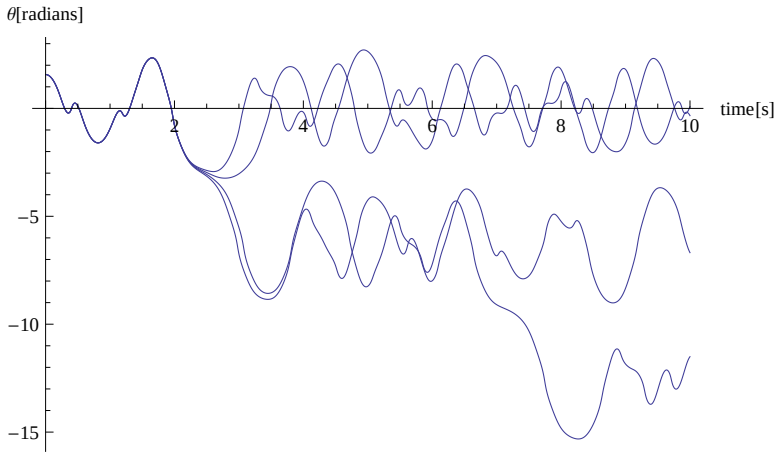
```
Show[thplt1, thplt2, thplt3, thplt4]
```



- **e) Plot th as a function of time using a dt of 0.001 and for th0=90 and for ph0=180, 180.01, 180.02, 180.03**

```
finalI = doublePendulumCmp[90, 0, 180, 0, 10, 0.001];
thplt1 = thPlot;
finalI = doublePendulumCmp[90, 0, 180.01, 0, 10, 0.001];
thplt2 = thPlot;
finalI = doublePendulumCmp[90, 0, 180.02, 0, 10, 0.001];
thplt3 = thPlot;
finalI = doublePendulumCmp[90, 0, 180.03, 0, 10, 0.001];
thplt4 = thPlot;
```

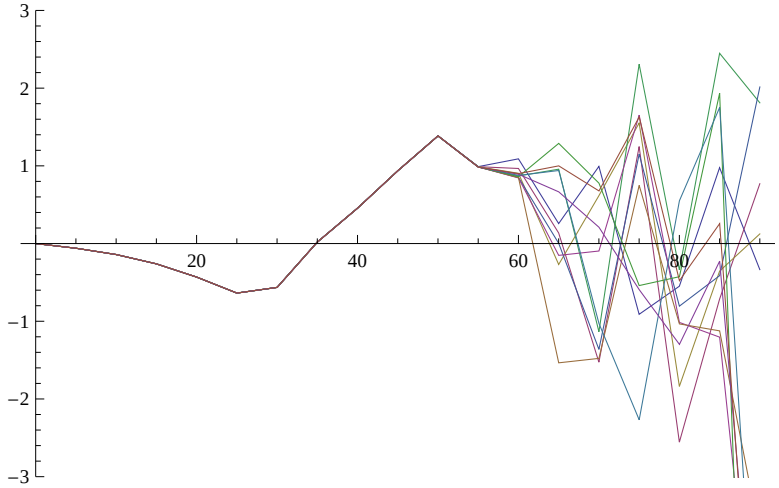
```
Show[thplt1, thplt2, thplt3, thplt4]
```




- **f) Extra credit: Study the final theta position at 10 seconds making ph0=2*th0 and the initial velocities 0. Change th by 5 degrees at a time starting at 0 going to 90 degrees. Run 20 perturbations about the starting th0 in perturbation steps of about 0.001. Can you make a statement from this about where it looks like the motion becomes chaotic?**

```
finalPos[0, 90, 5, 0.001, 10, 10, 0.001]
```

```
Show[thFPlot, PlotRange → {-3, 3}]
```




It appears that the motion becomes significantly chaotic on this time scale by about 60 degrees.