Differential Equations&*Mathematica*

Produced by Bruce Carpenter Published by Math Everywhere, Inc.
www.matheverywhere.com

DE.08 Linearizations Basics

B.1) Approximating a nonlinear diffeq system by linearizing at equilibrium points:

How to do it and how to use it

□B.1.a.i)

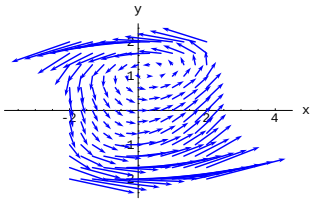
Here's a nonlinear diffeq system:

```
Clear[m, n, x, y, t]
m[x_, y_] = 0.4 x - 0.9 y^3 + 1.2;
n[x_, y_] = x - 0.4 Sin[y];
diffeqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[diffeqsystem]]
```

$$x'[t] == 1.2 + 0.4\,x[t] - 0.9\,y[t]^3$$
$$y'[t] == -0.4\,\text{Sin}[y[t]] + x[t]$$

And a flow plot:

```
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
{xlow, xhigh} = {-2, 2};
{ylow, yhigh} = {-2, 2};
jump = (xhigh - xlow)/12;
flowplot = Table[Arrow[Field[x, y], Tail → {x, y},
   VectorColor → Blue, ScaleFactor → 0.25, HeadSize → 0.2],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
nonlinearflow = Show[flowplot, Axes → True, AxesLabel → {"x", "y"}];
```



You can spot an equilibrium point in this plot.
Plot it.

□Answer:

Saying that a point {xeq, yeq} is an equilibrium point is the same as saying that there is no flow out of {xeq, yeq}, and this is the same as saying:

{m[xeq, yeq], n[xeq, yeq]} = {0, 0}.

Some folks use the alternate term "fixed point" to describe an equilibrium point.

This tells you that {xeq, yeq} lies at the intersection of the curve defined by

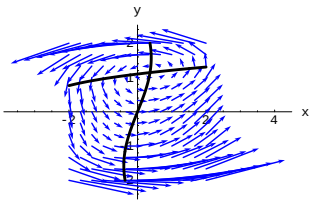
m[x, y] = 0

and the curve defined by

n[x, y] = 0.

Here are these two curves shown with the flow plot:

```
mcontourplot = ContourPlot[m[x, y], {x, xlow, xhigh},
  {y, ylow, yhigh}, Contours → {0}, ContourStyle → Thickness[0.01],
  PlotPoints → 50, ContourSmoothing → Automatic,
  ContourShading → False, DisplayFunction → Identity];
ncontourplot = ContourPlot[n[x, y], {x, xlow, xhigh},
  {y, ylow, yhigh}, Contours → {0}, ContourStyle → Thickness[0.01],
  PlotPoints → 50, ContourSmoothing → Automatic,
  ContourShading → False, DisplayFunction → Identity];
Show[nonlinearflow, mcontourplot, ncontourplot,
 DisplayFunction → $DisplayFunction];
```



You can see {xeq, yeq} near {0.2, 1.3}.

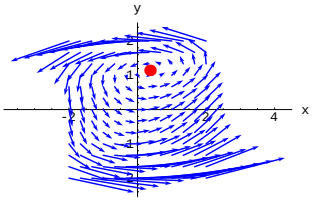
To nail {xeq, yeq} down, do this:

```
{xeq, yeq} =
 {x, y} /. FindRoot[{m[x, y] == 0, n[x, y] == 0}, {x, 0.2}, {y, 1.3}]
```

{0.364031, 1.14347}

See the equilibrium point with the flow:

```
equilibriumplot = Graphics[{Red, PointSize[0.04], Point[{xeq, yeq}]}];

Show[nonlinearflow, equilibriumplot];
```



Grab both plots, align and animate.

□B.1.a.ii) Using the linearization matrix to linearize a nonlinear diffeq system

Stay with the same nonlinear diffeq system as in part i);

```
Clear[m, n, x, y, t]
m[x_, y_] = 0.4 x - 0.9 y^3 + 1.2;
n[x_, y_] = x - 0.4 Sin[y];
diffeqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[diffeqsystem]]
```

$$x'[t] == 1.2 + 0.4\,x[t] - 0.9\,y[t]^3$$
$$y'[t] == -0.4\,\text{Sin}[y[t]] + x[t]$$

The linearization matrix A[x, y] at {x, y} for this system is:

Some folks call this matrix by the name "Jacobian."

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {D[m[x, y], x], D[m[x, y], y]};
gradn[x_, y_] = {D[n[x, y], x], D[n[x, y], y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} 0.4 & -2.7\,y^2 \\ 1 & -0.4\,\text{Cos}[y] \end{pmatrix}$$

The gradient of m[x, y] sits in the top horizontal row.
The gradient of n[x, y] sits in the bottom horizontal row.

How do you use the linearization matrix A[xeq, yeq] to linearize this system at {xeq, yeq}?

□Answer:

Very easily.

You go with the linearization matrix A[xeq, yeq] and use it for the coefficient matrix of this system:

{x′[t], y′[t]} = A[xeq, yeq] . {x[t] − xeq, y[t] − yeq}:

```
linearized =
({x'[t], y'[t]} ==
   Chop[Expand[A[xeq, yeq] . {x[t] - xeq, y[t] - yeq}]]);
ColumnForm[Thread[linearized]]
```

$$x'[t] == 3.89123 + 0.4\,x[t] - 3.53033\,y[t]$$
$$y'[t] == -0.174473 + 1.\,x[t] - 0.165775\,y[t]$$

Done.

You can see A[xeq, yeq] sitting in the system:

```
MatrixForm[A[xeq, yeq]]
```

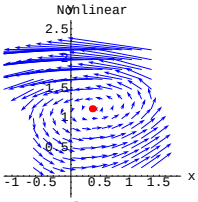
$$\begin{pmatrix} 0.4 & -3.53033 \\ 1 & -0.165775 \end{pmatrix}$$

□B.1.b)

Here is the flow for the original system:

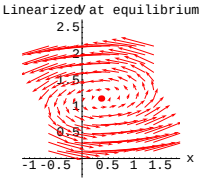
```
h = 1.0;
{xlow, xhigh} = {xeq - h, xeq + h};
{ylow, yhigh} = {yeq - h, yeq + h};
```

```
jump = (xhigh - xlow)/12;
scalefactor = 0.2;
sizer = 0.1;
nonlinearflow = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
nonlinearflowplot = Show[nonlinearflow, equilibriumplot,
  PlotRange → {{xeq - 1.5 h, xeq + 1.5 h}, {yeq - 1.5 h, yeq + 1.5 h}},
  Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Nonlinear"];
```



Here is the corresponding flow of the linearized system:

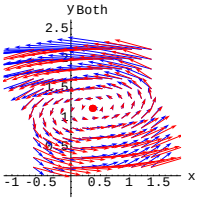
```
Clear[linearizedField]
linearizedField[x_, y_] = A[xeq, yeq] . {x - xeq, y - yeq};
linearizedflow = Table[Arrow[linearizedField[x, y], Tail → {x, y},
    VectorColor → Red, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
linearizedflowplot = Show[linearizedflow, equilibriumplot, PlotRange →
    {{xeq - 1.5 h, xeq + 1.5 h}, {yeq - 1.5 h, yeq + 1.5 h}}, Axes → True,
  AxesLabel → {"x", "y"}, PlotLabel → "Linearized at equilibrium"];
```



Grab both plots, align and animate slowly.

Here are both flows together:

```
both =
 Show[nonlinearflowplot, linearizedflowplot, PlotLabel → "Both"];
```

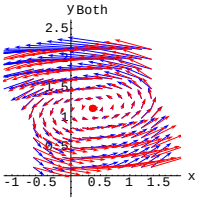


What fundamental relation between the orginal nonlinear system and its linearization at the equilibrium point is illustrated by these plots?

□ **Answer:**

Take another look:

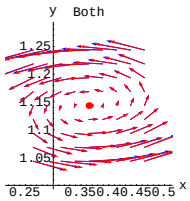
```
Show[both];
```



Near the equilibrium point, the flow of the linearization is a deadringer for the original nonlinear flow. As you go further from the equilbrium point, the quality of the approximation begins to deteriorate.

See what is happening in the immediate vicinty of the equilibrium point:

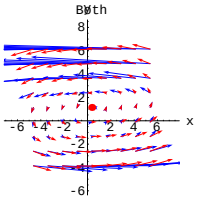
```
h = 0.1;
{xlow, xhigh} = {xeq - h, xeq + h};
{ylow, yhigh} = {yeq - h, yeq + h};
jump = (xhigh - xlow)/8;
scalefactor = 0.2;
sizer = 0.01;
nonlinearflowplot = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
linearizedflowplot = Table[Arrow[linearizedField[x, y], Tail → {x, y},
    VectorColor → Red, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
Show[nonlinearflowplot, linearizedflowplot, equilibriumplot,
 PlotRange → {{xeq - 1.5 h, xeq + 1.5 h}, {yeq - 1.5 h, yeq + 1.5 h}},
 Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Both"];
```



Near the equilibrium point, you get a great approximation of the nonlinear flow by the linearized flow.

See what happens when you go far away:

```
h = 5;
{xlow, xhigh} = {xeq - h, xeq + h};
{ylow, yhigh} = {yeq - h, yeq + h};
jump = (xhigh - xlow)/8;
scalefactor = 0.1;
sizer = 0.5;
nonlinearflowplot = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
linearizedflowplot = Table[Arrow[linearizedField[x, y], Tail → {x, y},
    VectorColor → Red, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
Show[nonlinearflowplot, linearizedflowplot, equilibriumplot,
 PlotRange → {{xeq - 1.5 h, xeq + 1.5 h}, {yeq - 1.5 h, yeq + 1.5 h}},
 Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Both"];
```



The further you go from the equilibrium point, the lower the quality of the approximation of the original nonlinear system by the linearized system.

□B.1.c) Using the linearization matrix to predict the behavior of a nonlinear system

What advantage do you get by approximating this nonlinear system with its linearized version?

□ **Answer:**

You can look at the eigenvalues of the linearization matrix:

```
Eigenvalues[A[xeq, yeq]]
{0.117113 + 1.8575 I, 0.117113 - 1.8575 I}
```

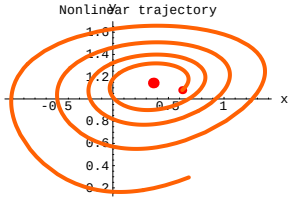
Propelling swirlers.

Because the linerized flow approximates the original nonlinear flow so well, this information guarantees that trajectories in the original nonlinear system that start near (but not on) the equilibrium point spiral away from the equilibrium point never to return to the vicinity of equilibrium point.

Watch it happen for some random starting points near the equilibrium point:

```
h = 0.3;
{xstarter, ystarter} =
 {Random[Real, {xeq - h, xeq + h}], Random[Real, {yeq - h, yeq + h}]};
```

```
endtime = 14;
starterpoint = {xstarter, ystarter};
Clear[x, y]
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t]
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};

trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];

starterplot = Graphics[{Red, PointSize[0.03], Point[starterpoint]}];

Show[starterplot, trajectoryplot,
 equilibriumplot, PlotRange → All, Axes → True,
 AxesLabel → {"x", "y"}, PlotLabel → "Nonlinear trajectory",
 DisplayFunction → $DisplayFunction];
```



Rerun a couple of times.

Just as predicted by the eigenvalues of the linearization matrix (Jacobian matrix) at the equillibrium point:

```
Eigenvalues[A[xeq, yeq]]
```

{0.117113 + 1.8575 I, 0.117113 - 1.8575 I}

□Repellor rules:

If the eigenvalues of the linearization matrix A[xeq, yeq] are both positive real numbers, then trajectories of the nonlinear system that start near the equilibrium point are propelled more or less directly from the equilibrium point.

If the eigenvalues of the linearization matrix A[xeq, yeq] are

$p + I\,q$ and $p - I\,q$ with $p > 0$ and $q \neq 0$,

then trajectories of the nonlinear system that start near the equilibrium point spiral away from the equilibrium point.

If either of these happens, most folks say the equilbrium point is an repellor.

□Saddle rule:

If the eigenvalues of the linearization matrix A[xeq, yeq] consist of one positive real number and one negative real number, then trajectories of the nonlinear system that start near the equilibrium are either propelled directly away from the equilibrium point or move toward the equilibrium point and then move away.

If either of these happens, most folks say the equilbrium point is an saddle.

B.2) Lyapunov's rules

□B.2.a) Lyapunov's

Given a nonlinear system, when can you trust the prediction you get from the eigenvalues of the linearization matrix A[xeq, yeq] at an equilibrium point {xeq, yeq}?

□Answer:

A juicy piece of advanced mathematics covers this. A brillant Russian fellow named Lyapunov proved conclusively the following rules:

□Attractor rules:

If the eigenvalues of the linearization matrix A[xeq, yeq] are both negative real numbers, then trajectories of the nonlinear system that start near the equilibrium point are sucked more or less directly to the equilibrium point.

If the eigenvalues of the linearization matrix A[xeq, yeq] are

$p + I\,q$ and $p - I\,q$ with $p < 0$ and $q \neq 0$,

then trajectories of the nonlinear system that start near the equilibrium point spiral in and stall at the equilibrium point.

If either of these happens, most folks say the equilbrium point is an attractor.

Fancy folks say that an equilibrium point is asymptotically stable if it is an attractor.
Some other folks call attractors by the name "sink."

□No information rule:

If the equilibrium point is a neither a repellor nor an attractor nor a saddle, then you are walking on thin ice if you trust the information you get from the linearization at the equilibrium point.

For instance, if the the eigenvectors of the linearization are pure swirlers, it is sometimes wrong to conclude that trajectories in the original nonlinear system starting near the equilbrium point are closed curves.

□B.2.b)

Give some illustrations of Lyapunov's rules.

□Answer:

Here's one:

```
Clear[m, n, x, y, t]
m[x_, y_] = -1.3 x + 0.17 y - 0.2 x^2 y;
n[x_, y_] = 0.50 x - 0.29 y + 0.7;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```

$x'[t] == -1.3\,x[t] + 0.17\,y[t] - 0.2\,x[t]^2\,y[t]$
$y'[t] == 0.7 + 0.5\,x[t] - 0.29\,y[t]$

The linearization matrix (Jacobian) at a point {x, y} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} -1.3 - 0.4\,x\,y & 0.17 - 0.2\,x^2 \\ 0.5 & -0.29 \end{pmatrix}$$

The equilibrium points are:

```
equisols = ColumnForm[Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}]]
```

{y → 0.914599 - 2.86101 I, x → -0.869532 - 1.65939 I}
{y → 0.914599 + 2.86101 I, x → -0.869532 + 1.65939 I}
{y → 2.99839, x → 0.339065}

The only equilbrium point is:

```
{xeq, yeq} = {0.339065, 2.99839}
```

```
{0.339065, 2.99839}
```

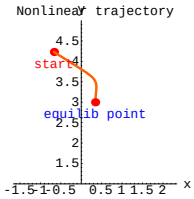
The eigenvalues of the linearization matrix at {xeq, yeq} are:

```
Eigenvalues[A[xeq, yeq]]
{-1.75677, -0.239888}
```

Both negative. The equilibrium point is an attractor.

See a random trajectory that starts near the equilibrium point head right for the equilibrium point:

```
h = 2;
{xstarter, ystarter} =
 {Random[Real, {xeq - h, xeq + h}], Random[Real, {yeq - h, yeq + h}]};
endtime = 15;
starterpoint = {xstarter, ystarter};

Clear[x, y]
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];

Clear[x, y, t]
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};

trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.05], Point[starterpoint]}];
equilibriumplot = Graphics[{Red, PointSize[0.05], Point[{xeq, yeq}]}];
starterpointlabel =
 Graphics[{Red, Text["start", starterpoint, {0, 1.5}]}];
equiliblabel =
 Graphics[{Blue, Text["equilib point", {xeq, yeq}, {0, 1.5}]}];

Show[starterplot, trajectoryplot, equilibriumplot, equiliblabel,
 starterpointlabel, PlotRange → {{xeq - h, xeq + h}, {yeq - h, yeq + h}},
 Axes → True, AxesLabel → {"x", "y"},
 AspectRatio → 1, PlotLabel → "Nonlinear trajectory",
 DisplayFunction → $DisplayFunction];
```



Rerun a couple of times.

Down the drain every time.

Here's another:

```
Clear[m, n, x, y, t]
m[x_, y_] = 1.3 y - 8.3 x^2;
n[x_, y_] = -1.6 x;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```

$$x'[t] == -8.3\,x[t]^2 + 1.3\,y[t]$$
$$y'[t] == -1.6\,x[t]$$

The linearization matrix at a point {x, y} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} -16.6\,x & 1.3 \\ -1.6 & 0 \end{pmatrix}$$

The equilibrium points are:

```
equisols = ColumnForm[Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}]]
{y → 0., x → 0.}
```

The equilbrium point is {0, 0} :

```
{xeq, yeq} = {0, 0}
{0, 0}
```

The eigenvalues of the linearization matrix at {xeq, yeq} are:

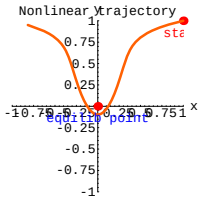
```
Eigenvalues[A[xeq, yeq]]
{0. + 1.44222 I, 0. - 1.44222 I}
```

The linearization predicts trajectories that oscillate around {xeq, yeq}.

But Lypunov's "No Information" rule says not to trust this.

See a trajectory that starts near {xeq, yeq}:

```
{xstarter, ystarter} = {xeq, yeq} + {1, 1};
endtime = 6.2;
starterpoint = {xstarter, ystarter};

Clear[x, y]
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];

Clear[x, y, t]
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};

trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.05], Point[starterpoint]}];
equilibriumplot = Graphics[{Red, PointSize[0.05], Point[{xeq, yeq}]}];
starterpointlabel =
 Graphics[{Red, Text["start", starterpoint, {0, 1.5}]}];
equiliblabel =
 Graphics[{Blue, Text["equilib point", {xeq, yeq}, {0, 1.5}]}];

Show[starterplot, trajectoryplot,
 equilibriumplot, equiliblabel, starterpointlabel,
 PlotRange → {{-1, 1}, {-1, 1}}, Axes → True, AxesLabel → {"x", "y"},
 AspectRatio → 1, PlotLabel → "Nonlinear trajectory",
 DisplayFunction → $DisplayFunction];
```



The linearization predicts trajectories that oscillate around {xeq, yeq}.

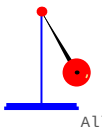
But this time, the information from the linearization matrix was useless.

This is in harmony with Lyapunov's rules.

B.3) The pendulum oscillator: Damped and undamped

Put your eyes on this snapshot of a pendulum:

```
L = 10;
setup = Show[pendulum[L, π/6], DisplayFunction → $DisplayFunction];
```

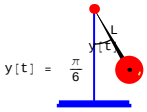


All pendulum graphics are by Ben Halperin.
His code is in the initialization cells.

It's just like the pendulum that hangs out the bottom of a cuckoo clock, or the bigger pendulum that hangs in that old clock in your grandmother's hallway.
To come up with a tractable model for the motion of the pendulum, you have to make a few assumptions that may be hard to swallow at first.
They are these:
→ The pendulum hangs from a frictionless pivot.

→ The pendulum moves in a vacuum so that there is no air resistance.
→ The arm of the pendulum which holds the bob at the end is both massless and perfectly rigid.
With those assumptions under your belt, look at this:

```
Clear[y, t, s]
L = 10;
y[t] = π/6;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
Show[calibratedpendulum[L, y[t]], angle, labels,
 DisplayFunction → $DisplayFunction];
```



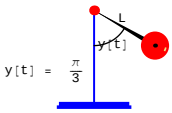
L stands for the length of the pendulum rod measured from the swivel to the center of the bob.

y[t] is the indicated angle measured counterclockwise from the vertical support.

In the specific grahpics here, the sample value $L = 10$ is used.

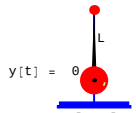
Here's what happens when you go with $y[t] = \frac{\pi}{3}$:

```
Clear[y, t, s]
y[t] = π/3;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
Show[calibratedpendulum[L, y[t]], angle, labels,
 DisplayFunction → $DisplayFunction];
```



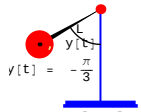
Here's what happens when you go with $y[t] = 0$:

```
Clear[y, t, s]
y[t] = 0;
Clear[s]
labels = Graphics[
  {Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
Show[calibratedpendulum[L, y[t]], labels,
 DisplayFunction → $DisplayFunction];
```



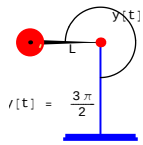
Here's what happens when you go with $y[t] = -\frac{\pi}{3}$:

```
Clear[y, t, s]
y[t] = -π/3;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
Show[calibratedpendulum[L, y[t]], angle, labels,
 DisplayFunction → $DisplayFunction];
```



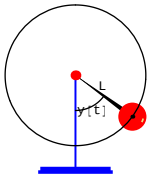
Here's what happens when you go with $y[t] = \frac{3\pi}{2}$:

```
Clear[y, t, s]
y[t] = 3π/2;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
Show[calibratedpendulum[L, y[t]], angle, labels,
 DisplayFunction → $DisplayFunction];
```



Get it? Got it? Good!
To get ready to come up with the differential equation y[t] solves, look at this set up:

```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
Clear[y, t, s]
L = 10;
y[t] = 0.3 π;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
setup = Show[pendulum[L, y[t]], bobpath, angle, labels,
  DisplayFunction → $DisplayFunction];
```



That circle is the path the center of bob moves on.

□B.3.a.i) The pendulum oscillator diffeq

$$y''[t] + \frac{g\,\mathrm{Sin}[y[t]]}{L} = 0.$$

Use the picture and Newton's law

Force = Mass × Acceleration

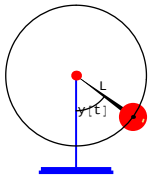
to come up with the differential equation that models the motion of this pendulum.

□Answer:

Take another look at the set up together with the circular path on which the center of the lump swings:

```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
```

```
Clear[y, t, s]
L = 10;
y[t] = 0.3 π;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
setup = Show[pendulum[L, y[t]], bobpath, angle, labels,
  DisplayFunction → $DisplayFunction];
```



Agree that the length of the rod is L.

You need a circular parameterization of the circle in terms of y[t].

Here's one that works very well:

```
Clear[position, L, y, t];
position[t_] = L {Sin[y[t]], -Cos[y[t]]}
```

```
{L Sin[y[t]], -L Cos[y[t]]}
```

This is the parametrization of the circle you want to make the pendulum hang down in the vertical position when the angle y[t] = 0:

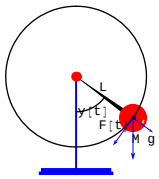
```
position[t] /. y[t] → 0
```

```
{0, -L}
```

Take a new look with all the goodies thrown in:

```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
Clear[y, t, s]
L = 10;
y[t] = 0.3 π;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
```

```
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}],
   Text["M g", {L Sin[y[t]], -L Cos[y[t]]} + {0, -0.3 L} + {0.15 L, 0}],
   Text["F[t]", 0.85 {L Sin[y[t]], -L Cos[y[t]]} +
     0.5 {0, -0.6 L} . {Cos[y[t]], Sin[y[t]]} {Cos[y[t]], Sin[y[t]]}]}];
overallforce = Arrow[{0, -0.6 L}, Tail → {L Sin[y[t]], -L Cos[y[t]]}];
tangentialforce =
 Arrow[{0, -0.6 L} . {Cos[y[t]], Sin[y[t]]} {Cos[y[t]], Sin[y[t]]},
  Tail → {L Sin[y[t]], -L Cos[y[t]]}];
factor = √((0.6 L)^2 - ({0, -0.6 L} . {Cos[y[t]], Sin[y[t]]})^2);
tension = Arrow[
  factor {Sin[y[t]], -Cos[y[t]]}, Tail → {L Sin[y[t]], -L Cos[y[t]]}];
all = Show[pendulum[L, y[t]], angle, labels, bobpath, overallforce,
  tangentialforce, tension, DisplayFunction → $DisplayFunction];
```



Agree that:

→ The mass of the bob on the end is M.

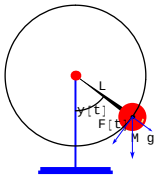
→ The vertical force on the bob is M(ass) g(ravity).

→ The component of force pointing in the direction of the pendulum rod can only try to stretch the unstretchable rod,

so the only part of the force that actually moves the pendulum is the tangential component F[t].

Take another look:

```
Show[all];
```



In this snapshot, assume the bob is moving to from left to right so that

$$y'[t] > 0.$$

And look at:

```
Clear[L, t, y]
position'[t]
```

```
{L Cos[y[t]] y'[t], L Sin[y[t]] y'[t]}
```

Because

$$y'[t] > 0,$$

the unit tangent vector pointing in the direction of the path of the center of the bob is:

```
Clear[unittan];
unittan[t_] = {Cos[y[t]], Sin[y[t]]}
```

```
{Cos[y[t]], Sin[y[t]]}
```

The unit tangent points opposite to the tangential acceleration.

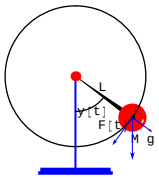
Consequently the tangential component of the force on the bob, F[t], is given by:

```
Clear[F, M];
F[t_] = Simplify[-M position''[t] . unittan[t]]
```

```
-L M y''[t]
```

Take one final peek at the picture:

```
Show[all];
```



The picture tells you that F[t] is also given by:

```
Clear[sameF, g]
sameF[t_] = M g Sin[y[t]]
```

```
g M Sin[y[t]]
```

The result is:

```
pendulummodel = F[t] == sameF[t]
```

```
-L M y''[t] == g M Sin[y[t]]
```

Solve this for y"[t]:

```
Solve[pendulummodel, y''[t]]
```

```
{{y''[t] → -(g Sin[y[t]]/L)}}
```

This gives:

```
pendulumoscillator = (y''[t] + g Sin[y[t]]/L == 0)
```

```
g Sin[y[t]]/L + y''[t] == 0
```

```
The behavior of this pendulum does not depend on the size of the mass.
                    Galileo is redeemed.
```

There it is!

A very famous differential equation.

```
                    This differential equation came about under
                the asssumption that the pendulum is swinging from left to right.
            If you want to be really anal-retentive about this and want to see how
              you get the same differential equation under the assumption that the
                  pendulum is swinging from right to left, open the next cells.
```

Look again at:

```
position'[t]
```

```
{L Cos[y[t]] y'[t], L Sin[y[t]] y'[t]}
```

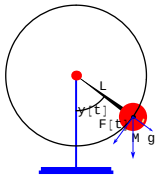
When you assume the bob is moving from right to left, then you have to say that y′[t] < 0.

In this case, the unit tangent vector pointing in the direction of the path of the center of the bob is:

```
Clear[unittan];
unittan[t_] = -{Cos[y[t]], Sin[y[t]]}
```
```
{-Cos[y[t]], -Sin[y[t]]}
```

Take another look at the snapshot:

```
Show[all];
```



Because the bob is moving from right to left, the unit tangent above points in the same direction as the tangential component of the acceleration. This gives

```
Clear[F, M];
F[t_] = Simplify[M position''[t] . unittan[t]]
```
```
-L M y″[t]
```

On the otherhand, the picture still tells you that F[t] is also given by:

```
Clear[sameF, g]
sameF[t_] = M g Sin[y[t]]
```
```
g M Sin[y[t]]
```

The result is:

```
pendulummodel = F[t] == sameF[t]
```
```
-L M y″[t] == g M Sin[y[t]]
```
```
Solve[pendulummodel, y″[t]]
```
$$\{\{y''[t] \to -\frac{g\,\text{Sin}[y[t]]}{L}\}\}$$

This gives you:

```
pendulumoscillator = y″[t] + g Sin[y[t]] / L == 0
```
$$\frac{g\,\text{Sin}[y[t]]}{L} + y''[t] == 0$$

Just as you got in the case that the pendulum is moving from left to right.

Going either way, the pendulum oscillator is a winner.

□B.3.a.ii) The damped pendulum oscillator

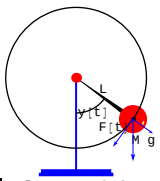
$$\mathbf{y''[t] + \frac{b\,y'[t]}{L\,M} + \frac{g\,\text{Sin}[y[t]]}{L} = 0.}$$

What differential equation do you get when you include a friction term?

□Answer:

You do exactly as you did above at the start.

```
Show[all];
```




```
Clear[position, L, x, t];
position[t_] = L {Sin[y[t]], -Cos[y[t]]}
```
```
{L Sin[y[t]], -L Cos[y[t]]}
```
```
position′[t]
```
```
{L Cos[y[t]] y′[t], L Sin[y[t]] y′[t]}
```

Assume the bob is moving from left to right so that y′[t] > 0.

Accordingly the unit tangent vector pointing in the direction of the path of the center of the bob is:

```
Clear[unittan];
unittan[t_] = {Cos[y[t]], Sin[y[t]]}
```
```
{Cos[y[t]], Sin[y[t]]}
```

Without friction, you put:

```
Clear[F, M];
F[t_] = Simplify[-M position″[t] . unittan[t]]
```
```
-L M y″[t]
```

With friction, you put:

```
Clear[F, M, b];
F[t_] = (F[t_] = Simplify[-M position″[t] . unittan[t]]) - b y′[t]
```
```
-b y′[t] - L M y″[t]
```

The reason for the extra $(-b\,y'[t])$ term is that the retarding force due to friction at the hub is proportional to the anglular velocity $y'[t]$.

Now you go on as in part i):

```
Clear[sameF, g]
sameF[t_] = M g Sin[y[t]]
```
```
g M Sin[y[t]]
```

The result is:

```
dampedpendulummodel = F[t] == sameF[t]
```
```
-b y′[t] - L M y″[t] == g M Sin[y[t]]
```
```
Solve[dampedpendulummodel, y''[t]]
```
$$\{\{y''[t] \to -\frac{g\,M\,\text{Sin}[y[t]] + b\,y'[t]}{L\,M}\}\}$$

A little rearranging and cancelling gives:

```
dampedpendulumoscillator = y″[t] + b y′[t] / (L M) + g Sin[y[t]] / L == 0
```
$$\frac{g\,\text{Sin}[y[t]]}{L} + \frac{b\,y'[t]}{L\,M} + y''[t] == 0$$

There it is!

A very famous nonlinear differential equation.

As you might have expected, the damping term $\frac{b\,y'[t]}{(L\,M)}$ is very small when L and M are large.

B.4) The ordinary linear oscillator

$$\mathbf{y''[t] + \frac{g\,y[t]}{L} = 0}$$

is a linearization of the pendulum oscillator

$$\mathbf{y''[t] + \frac{g\,\text{Sin}[y[t]]}{L} = 0}$$

□B.4.a)

The undamped pendulum oscillator diffeq is:

```
Clear[y, g, L, t]
pendulumoscillator = y″[t] + g Sin[y[t]] / L == 0
```
$$\frac{g\,\text{Sin}[y[t]]}{L} + y''[t] == 0$$

The ordinary undamped linear oscillator diffeq is:

```
Clear[y, g, L, t]
ordinaryoscillator = y″[t] + g y[t] / L == 0
```
$$\frac{g\,y[t]}{L} + y''[t] == 0$$

Explain why the ordinary linear oscillator arrives as a linearization of the pendulum oscillator.

□Answer:

Look at:

```
pendulumoscillator
```
$$\frac{g\,\text{Sin}[y[t]]}{L} + y''[t] == 0$$

Make this second order differential equation into a system of two first order differential equations, by putting

$$x[t] = y'[t]$$

and then replacing y′[t]with x[t] and replacing y″[t]with x′[t]:

```
Clear[x]
ColumnForm[Thread[{new = (y'[t] == x[t]),
pendulumoscillator /. {y'[t] -> x[t], y''[t] -> x'[t]}}]]
```

$y'[t] == x[t]$

$\frac{g \text{Sin}[y[t]]}{L} + x'[t] == 0$

Clean this up:

```
Clear[m, n, x, y, t]
m[x_, y_] = -g Sin[y]/L;
n[x_, y_] = x;
pendulumsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[pendulumsystem]]
```

$x'[t] == -\frac{g \text{Sin}[y[t]]}{L}$

$y'[t] == x[t]$

The linearization matrix at {0, 0} is:

```
{xeq, yeq} = {0, 0};
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[xeq, yeq]]
```

$\begin{pmatrix} 0 & -\frac{g}{L} \\ 1 & 0 \end{pmatrix}$

The linearization at {0, 0} is:

```
linearized =
({x'[t], y'[t]} == Expand[A[xeq, yeq].{x[t] - xeq, y[t] - yeq}]);
ColumnForm[Thread[linearized]]
```

$x'[t] == -\frac{g\, y[t]}{L}$

$y'[t] == x[t]$

Remembering that x[t] = y′[t], you see that this is the same as:

```
y''[t] + g y[t]/L == 0
```

$\frac{g\, y[t]}{L} + y''[t] == 0$

The upshot:

The ordinary linear oscillator

$$y''[t] + \frac{g\, y[t]}{L} = 0$$

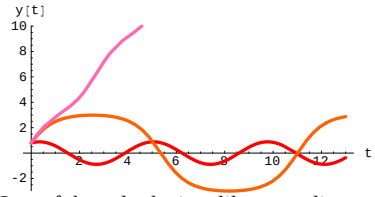
is a linearization of the pendulum oscillator

$$y''[t] + \frac{g \text{Sin}[y[t]]}{L} = 0.$$

□B.4.b.i) Using the linearization to see why the pendulum oscillator sometimes behaves like an ordinary linear oscillator and sometimes it doesn't

Here are three solutions of the pendulum oscillator corresponding to g = 9.8 and L = 5:
They all start with y[0] = 0.8, but one starts with a reasonably small y′[0], the next one starts with a bigger y′[0] and the last starts with an even bigger y′[0]:

```
Clear[y, yslow, yfaster, yreallyfast]
pendulumoscillator = y''[t] + g Sin[y[t]]/L == 0;
g = 9.8;
L = 5;
endtime = 13;
slowstart = NDSolve[{pendulumoscillator,
   y[0] == 0.8, y'[0] == 0.5}, y[t], {t, 0, endtime}];
yslow[t_] = y[t] /. slowstart[[1]];
fasterstart = NDSolve[{pendulumoscillator,
   y[0] == 0.8, y'[0] == 2.57}, y[t], {t, 0, endtime}];
yfaster[t_] = y[t] /. fasterstart[[1]];
reallyfaststart = NDSolve[{pendulumoscillator,
   y[0] == 0.8, y'[0] == 2.9}, y[t], {t, 0, endtime}];
yfaster[t_] = y[t] /. fasterstart[[1]];
yreallyfast[t_] = y[t] /. reallyfaststart[[1]];
Plot[{yslow[t], yfaster[t], yreallyfast[t]},
 {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.01]},
   {CadmiumOrange, Thickness[0.01]}, {HotPink, Thickness[0.01]}},
 AxesLabel → {"t", "y[t]"}, PlotRange → {-3, 10}, AspectRatio → 1/2];
```



One of these looks just like an ordinary oscillator.
Another looks something like an ordinary oscillator.
The third doesn't look like an ordinary oscillator at all.
Use linearizations to help explain why this happened.

□Answer:

The flow field for the nonlinear pendulum oscillator is:

```
Clear[m, n, x, y, t, Field]
m[x_, y_] = -g Sin[y]/L;
n[x_, y_] = x;
Field[x_, y_] = {m[x, y], n[x, y]}
{-1.96 Sin[y], x}
```

The linearization matrix at {0, 0} is:

```
{xeq, yeq} = {0, 0};
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[xeq, yeq]]
```

$\begin{pmatrix} 0 & -1.96 \\ 1 & 0 \end{pmatrix}$

The linearization at {0, 0} is:

```
linearized =
({x'[t], y'[t]} == Expand[A[xeq, yeq].{x[t] - xeq, y[t] - yeq}]);
ColumnForm[Thread[linearized]]
```

$x'[t] == -1.96\, y[t]$

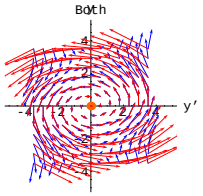
$y'[t] == 1.\, x[t]$

The flow field for the linearization is:

```
Clear[linearizedField]
linearizedField[x_, y_] = A[xeq, yeq].{x - xeq, y - yeq}
{-1.96 y, 1. x}
```

See the linearized flow and the nonlinear pendulum flow:

```
h = 3.5;
{xlow, xhigh} = {xeq - h, xeq + h};
{ylow, yhigh} = {yeq - h, yeq + h};
jump = (xhigh - xlow)/12;
scalefactor = 0.35;
sizer = 0.3;
nonlinearflowplot = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
linearizedflowplot = Table[Arrow[linearizedField[x, y], Tail → {x, y},
    VectorColor → Red, ScaleFactor → scalefactor, HeadSize → sizer],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
equilibriumplot =
 Graphics[{CadmiumOrange, PointSize[0.05], Point[{xeq, yeq}]}];
both = Show[nonlinearflowplot, equilibriumplot, linearizedflowplot,
  PlotRange → {{xeq - 1.5 h, xeq + 1.5 h}, {yeq - 1.5 h, yeq + 1.5 h}},
  Axes → True, AxesLabel → {"y'", "y"}, PlotLabel → "Both"];
```



The nonlinear pendulum flow is plotted in red.
The linearized flow is plotted in blue.

Near {0, 0}, the two flows are similar - almost copies of each other.

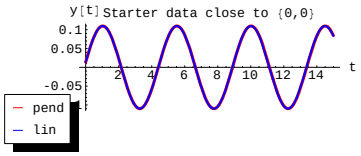
The upshot:

When the starter data on y[0] and y′[0]are close to {0, 0}, the nonlinear pendulum oscillator behaves very much like the ordinary linear oscillator that comes from its linearization.

Watch it happen for random starter data close to {0, 0}:

```
h = 0.5;
Clear[y, ypend, yordinary, t]
{startery, starteryprime} =
 {Random[Real, {-h, h}], Random[Real, {-h, h}]}
pendulumoscillator = y''[t] + g Sin[y[t]] / L == 0;
linearizedoscillator = y''[t] + g y[t] / L == 0;
g = 9.8;
L = 5;
endtime = 15;
pendulum = NDSolve[{pendulumoscillator,
   y[0] == startery, y'[0] == starteryprime}, y[t], {t, 0, endtime}];
ypend[t_] = y[t] /. pendulum[[1]];
ordinary = NDSolve[{linearizedoscillator,
   y[0] == startery, y'[0] == starteryprime}, y[t], {t, 0, endtime}];
yordinary[t_] = y[t] /. ordinary[[1]];
Plot[{ypend[t], yordinary[t]}, {t, 0, endtime},
 PlotStyle → {{Red, Thickness[0.012]}, {Blue, Thickness[0.008]}},
 AxesLabel → {"t", "y[t]"},
 PlotLabel → "Starter data close to {0,0}", AspectRatio → 1/3,
 PlotLegend → {"pend", "lin"}, LegendSize → 0.4];
```

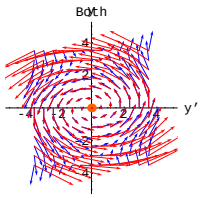
```
{0.0138436, 0.152305}
```



The point above the graph is the starter point.
Rerun many times.

Take another look at both flows near {0, 0}:

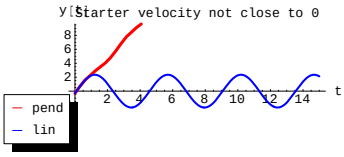
```
Show[both];
```



The plot indicates that when you go with starter data on {y[0], y′[0]}not so close to {0, 0}, the pendulum oscillator behaves much differently from the ordinary oscillator coming from its linearization. See what happens when you go with some random starter data not so close to {0, 0}:

```
Clear[y, ypend, yordinary, t]
{startery, starteryprime} =
 {Random[Real, {-1.0, 1.0}], Random[Real, {1.5, 4.0}]}
pendulumoscillator = y''[t] + g Sin[y[t]] / L == 0;
linearizedoscillator = y''[t] + g y[t] / L == 0;
g = 9.8;
L = 5;
endtime = 15;
pendulum = NDSolve[{pendulumoscillator,
   y[0] == startery, y'[0] == starteryprime}, y[t], {t, 0, endtime}];
ypend[t_] = y[t] /. pendulum[[1]];
ordinary = NDSolve[{linearizedoscillator,
   y[0] == startery, y'[0] == starteryprime}, y[t], {t, 0, endtime}];
yordinary[t_] = y[t] /. ordinary[[1]];
Plot[{ypend[t], yordinary[t]}, {t, 0, endtime},
 PlotStyle → {{Red, Thickness[0.012]}, {Blue, Thickness[0.008]}},
 AxesLabel → {"t", "y[t]"},
 PlotLabel → "Starter velocity not close to 0", AspectRatio → 1/3,
 PlotLegend → {"pend", "lin"}, LegendSize → 0.4];
```

```
{-0.308107, 3.25938}
```



The point above the graph is the starter point.
Rerun many times.

B.5) Linearizations and gradients:

Getting an idea about why linearizations give good approximations

□B.5.a)

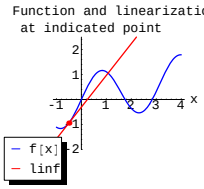
The linearization of a function f[x] at a point x_0 on the x – axis is:

```
Clear[f, x, linearf, x0]
linearf[x_] = f[x0] + f'[x0] (x - x0)
```

```
f[x0] + (x - x0) f'[x0]
```

See the plot of a function f[x] together with the plot of its linearization at a random point x_0 on the x – axis:

```
f[x_] = 0.2 x + Sin[2 x];
{xlow, xhigh} = {-1, 4};
x0 = Random[Real, {xlow, xhigh}];

Plot[{f[x], linearf[x]}, {x, xlow, xhigh},
 PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.01]}},
 AxesLabel → {"x", ""}, PlotRange → {-2, 2.5},
 PlotLabel → "Function and linearization \n at indicated point",
 PlotLegend → {"f[x]", "linf[x]"}, LegendSize → 0.6,
 Epilog → {Red, PointSize[0.04], Point[{x0, f[x0]}]}];
```



Rerun a couple of times.

That's right!

The linearization of f[x] at x_0 is nothing but the tangent

$$y = f[x_0] + f'[x_0]\,(x - x_0)$$

at x_0.

It gives a really good approximation of f[x] near x_0, the point of linearization.

How do you use gradients to carry this off for functions of two variables?

□Answer:

One of the great drivers of mathematics is the power of analogy.

To linearize a function f[x] of one variable at x_0, you go with the derivative

$$f'[x]$$

and write

$$\text{linf}[x] = f[x_0] + f'[x_0]\,(x - x_0).$$

To linearize a function f[x, y] of two variables at $\{x_0, y_0\}$, you go with the gradient

$$\nabla f[x, y] = \{\partial f[x, y]/\partial x,\ \partial f[x, y]/\partial y\}$$

If you're running these lessons through Windows,
the funny looking characters above are supposed to be partial
derivative symbols.

which is the same as

$$\nabla f[x, y] = \{\partial_x f[x, y], \partial_y f[x, y]\}$$

and write

$$\text{linf}[x, y] = f[x_0, y_0] + \nabla f[x_0, y_0] . \{x - y_0, y - y_0\}$$

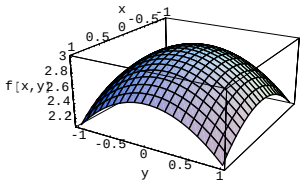
which is the same as

$$\text{linf}[x, y] = f[x_0, y_0] + \text{gradf}[x_0, y_0] . \{x - y_0, y - y_0\}$$

That's a dot product.

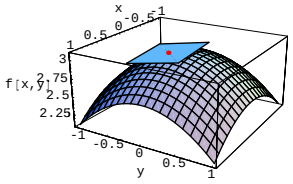
To help you stick your mental tongs into the idea behind using the gradient for linearization, look at the following plot of a sample function f[x, y]:

```
Clear[f, x, y]
f[x_, y_] = 3 - 0.4 x^2 - 0.5 y^2;
actualsurfaceplot =
 ParametricPlot3D[{x, y, f[x, y]}, {x, -1, 1}, {y, -1, 1},
  Axes → Automatic, AxesLabel → {"x", "y", "f[x,y]"}, PlotRange → All,
  ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```



Now look at plots of linearizations at the indicated points:

```
{x0, y0} = {Random[Real, {0, 1}], Random[Real, {-1, 1}]};
Clear[gradf, linearf, point];
gradf[x_, y_] = {∂x f[x, y], ∂y f[x, y]};
linearf[x_, y_] = f[x0, y0] + gradf[x0, y0] . {x - x0, y - y0};
h = 0.4;
linearizationplot =
 ParametricPlot3D[{x, y, linearf[x, y]}, {x, x0 - h, x0 + h},
  {y, y0 - h, y0 + h}, PlotPoints → {2, 2}, DisplayFunction → Identity];
linpoint =
 Graphics3D[{Red, PointSize[0.02], Point[{x0, y0, f[x0, y0]}]}];
Show[actualsurfaceplot, linearizationplot, linpoint];
```



Rerun several times.

That flat sheet hugging onto the actual surface plot is a plot of the linearization at the indicated point.

In fact, that sheet is tangent to the surface and approximates the actual surface very well close to the indicated point.

Some calculous courses call the flat sheet by the name "tangent plane."

□B.5.b)

Here's a nonlinear diffeq system:

```
Clear[m, n, x, y]
m[x_, y_] = 0.4 x - 0.1 x y;
n[x_, y_] = -0.3 y + 0.5 y x - 0.5;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```

```
x'[t] == 0.4 x[t] - 0.1 x[t] y[t]
y'[t] == -0.5 - 0.3 y[t] + 0.5 x[t] y[t]
```

The equilibrium points are:

```
equisols = Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}];
Clear[xeq, yeq]
{xeq[1], yeq[1]} = {x, y} /. equisols[[1]]
{xeq[2], yeq[2]} = {x, y} /. equisols[[2]]
```

```
{0., -1.66667}
{0.85, 4.}
```

The linearized system at {xeq[2], yeq[2]} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {D[m[x, y], x], D[m[x, y], y]};
gradn[x_, y_] = {D[n[x, y], x], D[n[x, y], y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};

linearized2 =
({x'[t], y'[t]} ==
   Chop[Expand[A[xeq[2], yeq[2]] . {x[t] - xeq[2], y[t] - yeq[2]}]]);
ColumnForm[Thread[linearized2]]
```

```
x'[t] == 0.34 - 0.085 y[t]
y'[t] == -2.2 + 2. x[t] + 0.125 y[t]
```

How is this linearized system related to the linearizations of m[x, y] and n[x, y] at {xeq[2], yeq[2]}?
How does the answer explain why the linearized system approximates the original system very well near {xeq[2], yeq[2]}?

□**Answer:**

Very intimately.

To get the idea, linearize m[x, y] and linearize n[x, y] at {xeq[2], yeq[2]}:

```
Clear[linearm, linearn];
linearm[x_, y_] = Chop[
  m[xeq[2], yeq[2]] + gradm[xeq[2], yeq[2]] . {x - xeq[2], y - yeq[2]}]

linearn[x_, y_] = Chop[
  n[xeq[2], yeq[2]] + gradn[xeq[2], yeq[2]] . {x - xeq[2], y - yeq[2]}]
```

```
-0.085 (-4. + y)
2. (-0.85 + x) + 0.125 (-4. + y)
```

Use linearm[x, y] and linearn[x, y] to make a new diffeq system:

```
newDiffEqsystem =
({x'[t], y'[t]} ==
   Expand[{linearm[x[t], y[t]], linearn[x[t], y[t]]}]);
ColumnForm[Thread[newDiffEqsystem]]
```

```
x'[t] == 0.34 - 0.085 y[t]
y'[t] == -2.2 + 2. x[t] + 0.125 y[t]
```

This is the same as the linearized system you got via the linearization matrix:

```
ColumnForm[Thread[linearized2]]
```

```
x'[t] == 0.34 - 0.085 y[t]
y'[t] == -2.2 + 2. x[t] + 0.125 y[t]
```

It works this way everytime.

And that's why linearized systems approximate the original system very, very well at the point of linearization.

□B.5.c)

In what other mathematical issues is the idea of linearization important?

□**Answer:**

The idea of linearization sits at the heart of the chain rule for functions of more than one variable.

And the idea of linearization sits at the heart of the way you change variables in double and triple integrals.

It's around all over the place.

DE.08 Linearizations Tutorials

T.1) Using formulas resulting from the linearization of a pendulum oscillator to estimate the amplitude and frequency of a pendulum oscillator

Because linear {differential] equations are easy to solve and study, the theory of linear oscillation is the most highly developed area of mechanics.
In many non-linear problems, linearization produces a satisfactory approximate solution. Even when this is not the case, the study of the linear part of a problem is often a first step, to be followed by the study of the relation between motions in a nonlinear system and its linear model.
-------World DiffEq Master V. I. Arnold
writing in his advanced book Mathematical Methods of Classical Mechanics (Springer-Verlag, New York, 1989)

T.1.a.i) Approximate formulas: The good news

The truth is that nothing, including *Mathematica*, has ever succeeded in coming up with an exact formula for the actual pendulum oscillator. Try it in the sample case L = 7.5 and g = 9.8 with starter data y[0] = 0.4 and y′[0] = −0.3:

```
L = 7.5;
g = 9.8;
starterangle = 0.4;
startervelocity = -0.3;
Clear[t, y]
pendulumoscillator = y''[t] + g Sin[y[t]] / L == 0
```

```
1.30667 Sin[y[t]] + y''[t] == 0
```

```
actual = DSolve[
  {pendulumoscillator, y[0] == starterangle, y'[0] == startervelocity},
  y[t], t]
```

```
DSolve[{1.30667 Sin[y[t]] + y''[t] == 0, y[0] == 0.4, y'[0] == -0.3},
 y[t], t]
```

Out of luck. Formulas for the solution do not exist.
Lots of formula-hungry folks turn to the linearized version for a quick formula:

```
linpendulumoscillator = y''[t] + g y[t] / L == 0
Clear[approxformulay];
linearized = Chop[DSolve[{linpendulumoscillator,
    y[0] == starterangle, y'[0] == startervelocity}, y[t], t]];
approxformulay[t_] = N[ComplexExpand[y[t] /. linearized[[1]]], 6]
```

```
1.30667 y[t] + y''[t] == 0
```

```
0.4 Cos[1.1431 t] - 0.262445 Sin[1.1431 t]
```

How do you know whether or not to trust this approximate formula?

□ **Answer:**

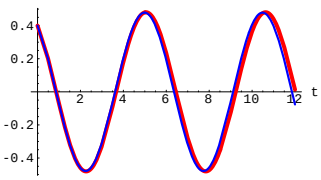
Look at the starter data:

```
{starterangle, startervelocity}
```

```
{0.4, -0.3}
```

The starter point is not far from the point of linearization at {0, 0}; so you can expect this formula to give good results. Give it the acid test:

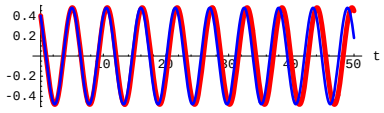
```
endtime = 12;
realthing = NDSolve[{pendulumoscillator, y[0] == starterangle,
   y'[0] == startervelocity}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
Plot[{yreal[t], approxformulay[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1 / GoldenRatio];
```



Nailed it fairly well. You could probably use this formula for t between 0 and 12.

See more:

```
endtime = 50;
realthing = NDSolve[{pendulumoscillator, y[0] == starterangle,
   y'[0] == startervelocity}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
Plot[{yreal[t], approxformulay[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1 / GoldenRatio];
```



Whether to use the formula for a long time interval is a question of your own judgment.

□ T.1.a.ii)

Look at the approximate formula for the pendulum oscillator you got in part i) above:

```
approxformulay[t]
```

```
0.4 Cos[1.1431 t] - 0.262445 Sin[1.1431 t]
```

Use it to estimate the amplitude, period and frequency of this pendulum oscillator.

□ **Answer:**

Look at the approximate formula again:

```
approxformulay[t]
```

```
0.4 Cos[1.1431 t] - 0.262445 Sin[1.1431 t]
```

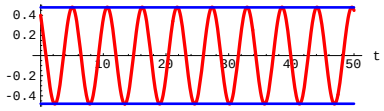
Write down the amplitude:

```
approxamplitude = Sqrt[0.4^2 + (-0.262445)^2]
```

```
0.478411
```

Confirm with a plot:

```
amplitudeplot = Plot[{yreal[t], approxamplitude, -approxamplitude},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.01], Red},
    {Thickness[0.008], Blue}, {Thickness[0.008], Blue}},
  AxesLabel → {"t", ""}, AspectRatio → 1 / GoldenRatio];
```



Estimate the period of the pendulum oscillator:

```
approxformulay[t]
```

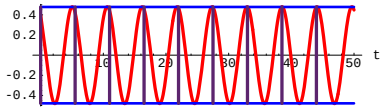
```
0.4 Cos[1.1431 t] - 0.262445 Sin[1.1431 t]
```

```
approxperiod = Solve[1.1431 t == N[2 π], t]
```

```
{{t → 5.49662}}
```

This pendulum oscillator repeats itself approximately every 5.5 time units.

```
approxperiod = 5.5;
periodlines = Table[Graphics[{UltramarineViolet, Thickness[0.01],
    Line[{{t, -approxamplitude}, {t, approxamplitude}}]}],
  {t, 0, 8 approxperiod, approxperiod}];
Show[amplitudeplot, periodlines];
```



Lots of folks like to talk about frequency. The approximate frequency of this pendulum oscillator is:

```
appoxfrequency = 1 / approxperiod
```

```
0.181818
```

This oscillator goes through about 0.18 complete oscillations per unit of time.

□T.1.a.iii) Approximate formulas: The bad news

Can you always use the approximate formula you get from the linearization to estimate amplitude, period, and frequency of the pendulum oscillator?

□**Answer:**

For crying out loud, this sounds like a question that wuss Calculus Cal would ask (if he had the brains to come up with it).

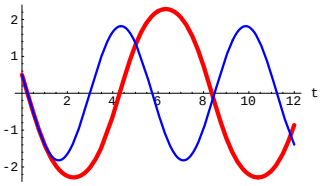
The quality of estimates you get from the linearization depends on the quality of the approximation resulting from the linearization. When you blindly trust the stuff you get from linearizations, you are running the risk of walking in deep do-do.

Look at this one:

```
L = 7.5;
g = 9.8;
Clear[t, y]
pendulumoscillator = y''[t] + g Sin[y[t]] / L == 0
starterangle = 0.5;
startervelocity = -2.0;
simplependulumoscillator = y''[t] + g y[t] / L == 0
Clear[approxformulay];
linearized = Chop[DSolve[{simplependulumoscillator,
    y[0] == starterangle, y'[0] == startervelocity}, y[t], t]];
approxformulay[t_] = N[y[t] /. linearized[[1]], 7]
```

```
1.30667 Sin[y[t]] + y''[t] == 0
1.30667 y[t] + y''[t] == 0
0.5 Cos[1.143095 t] - 1.749636 Sin[1.143095 t]
```

```
endtime = 12;
realthing = NDSolve[{pendulumoscillator, y[0] == starterangle,
   y'[0] == startervelocity}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
Plot[{yreal[t], approxformulay[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1 / GoldenRatio];
```



The "approximate formula" resulting from the linearization here is:

```
approxformulay[t]
```

```
0.5 Cos[1.143095 t] - 1.749636 Sin[1.143095 t]
```

Any one who would try to use this approximate formula to estimate anything about the actual pendulum oscillator should look for the clue phone or run the risk of getting the reputation of a total dork.

In fact the only DiffEq&*Mathematica* student who might be tempted to use this formula to estimate anything is Calculus Cal himself.

Cal, get yourself and your brain-rot out of here right now.

T.2) Why most folks don't care about linearizing at points other than equilibrium points

□T.2.a)

Given a nonlinear system, folks calculate the equilibrium points and then check out the eigenvalues of the linearization matrix to determine the nature of the equilibrium points.
Why don't folks do this at points other than equilibrium points?

□**Answer:**

It doesn't give you useful information.

To see why, go with

x′[t] = m[x[t], y[t]]

y′[t] = n[x[t], y[t]].

If{xnoteq, ynoteq} is not an equilibrium point, then the flow at {xnoteq, ynoteq} is given by

{m[xnoteq, ynoteq], n[xnoteq, ynoteq]},

And because {xnoteq, ynoteq} is not an equilibrium point, this flow is not {0, 0}.

So:

The flow at {xnoteq, ynoteq} goes through {xnoteq, ynoteq} in the direction of the vector:

{m[xnoteq, y noteq], n[xnoteq, y noteq]}.

As a result,

→{xnoteq, ynoteq} cannot be an attractor.

→{xnoteq, ynoteq} cannot be a repellor.

Check it out in this example:

The system:

```
Clear[m, n, x, y, t]
m[x_, y_] = 0.4 x^2 - 0.6 y^2 + 1.2;
n[x_, y_] = x - 0.4 y;
diffeqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[diffeqsystem]]
```

```
x'[t] == 1.2 + 0.4 x[t]^2 - 0.6 y[t]^2
y'[t] == x[t] - 0.4 y[t]
```

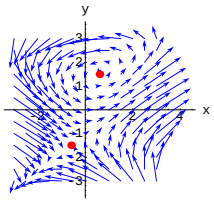
The equilbrium points:

```
equisols = Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}];
Clear[xeq, yeq]
{xeq[1], yeq[1]} = {x, y} /. equisols[[1]]
{xeq[2], yeq[2]} = {x, y} /. equisols[[2]]
```

```
{-0.598506, -1.49626}
{0.598506, 1.49626}
```

The flow with the equilibrium points:

```
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
{xlow, xhigh} = {-3, 3};
{ylow, yhigh} = {-3, 3};
jump = (xhigh - xlow) / 12;
flowplot = Table[Arrow[Field[x, y], Tail → {x, y},
   VectorColor → Blue, ScaleFactor → 0.3, HeadSize → 0.3],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
equilibriumplots = Table[Graphics[
   {Red, PointSize[0.04], Point[{xeq[k], yeq[k]}]}], {k, 1, 2}];
setup =
 Show[flowplot, equilibriumplots, Axes → True, AxesLabel → {"x", "y"}];
```

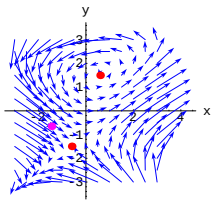


Away from the equilibrium points, the flow is just like "Old Man River." The flow just keeps on rolling along.

Throw in a random point that is not an equilibrium point:

```
{xnoteq, ynoteq} = {Random[Real, {-2, 2}], Random[Real, {-1, 1}]}
notequilibriumplot =
 Graphics[{Magenta, PointSize[0.04], Point[{xnoteq, ynoteq}]}];
Show[setup, notequilibriumplot];
```

```
{-1.44827, -0.650604}
```



Rerun a couple of times.

The flow goes right through the new point.

T.3) Using Lyapunov's rules to investigate the predator-prey model

□T.3.a.i)

The predator-prey model is

$$x'[t] = a\,x[t] - b\,x[t]\,y[t],$$
$$y'[t] = -c\,y[t] + d\,y[t]\,x[t].$$

with a, b, c and d all positive.

Here x[t] is the prey population at time t.
And y[t] is the predator population at time t.

When you feed extra prey in at a rate of r units per time unit the model becomes

$$x'[t] = a\,x[t] - b\,x[t]\,y[t] + r$$
$$y'[t] = -c\,y[t] + d\,y[t]\,x[t].$$

Go with the sample case of

$$a = 0.7,\ b = 0.3,\ c = 0.44 \text{ and } d = 0.08$$

and calculate the equilibrium points as functions of r.
Discuss the results.

□Answer:

Set it up with cleared a, b, c, d and r:

```
Clear[m, n, x, y, a, b, c, d, r, t]
m[x_, y_] = a x - b x y + r;
n[x_, y_] = -c y + d y x;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```

```
x'[t] == r + a x[t] - b x[t] y[t]
y'[t] == -c y[t] + d x[t] y[t]
```

The equilibrium points are:

```
equisols = ColumnForm[Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}]]
```

$$\{y \to 0,\ x \to -\frac{r}{a}\}$$
$$\{y \to \frac{a\,c+d\,r}{b\,c},\ x \to \frac{c}{d}\}$$

The only equilbrium point of any physical interest is:

```
Clear[xeq, yeq, r]
{xeq[r_], yeq[r_]} = {c/d, (a c + d r)/(b c)}
```

$$\left\{\frac{c}{d},\ \frac{a\,c+d\,r}{b\,c}\right\}$$

Now incorporate the given values of a, b, c and d to get the equilibrium points as functions of r:

```
a = 0.7;
b = 0.3;
c = 0.44;
d = 0.08;
{xeq[r], yeq[r]}
```

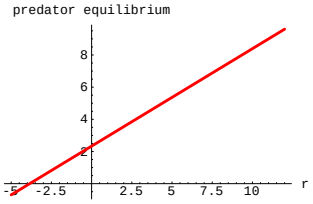
$$\{5.5,\ 7.57576\,(0.308 + 0.08\,r)\}$$

This indicates that the equilibrim prey population is 5.5

REGARDLESS of what r is.

The equilibrium predator population varies with r this way:

```
Plot[yeq[r], {r, -5, 12}, PlotStyle → {{Red, Thickness[0.01]}},
 AxesLabel → {"r", "predator equilibrium"}];
```



If you feed in prey at a rate of r prey units per time unit and you make r more negative than:

```
Solve[yeq[r] == 0, r]
```

$$\{\{r \to -3.85\}\}$$

Then the equilibrium predator population becomes negative.

This is not good.

□T.3.a.ii)

Keep everything the same as in part i) and keep everything in part i) live in your computer's memory.
The equilibrium points as functions of r are:

```
{xeq[r], yeq[r]}
```

$$\{5.5,\ 7.57576\,(0.308 + 0.08\,r)\}$$

Use Lyapunov's rules to determine the nature of the equilibrium points.

□Answer:

The linearization (Jacobian) matrix at a point {x, y} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} 0.7 - 0.3\,y & -0.3\,x \\ 0.08\,y & -0.44 + 0.08\,x \end{pmatrix}$$

Calulate the eigenvalues of the linearization matrix (Jacobian) at {xeq[r], yeq[r]}:

```
{eigenvalue1[r_], eigenvalue2[r_]} =
 Chop[Eigenvalues[A[xeq[r], yeq[r]]]]
```

$$\left\{\frac{1}{2}\left(-0.181818\,r - 0.181818\,\sqrt{-12.6306 + r}\,\sqrt{2.95061 + r}\right),\right.$$
$$\left.\frac{1}{2}\left(-0.181818\,r + 0.181818\,\sqrt{-12.6306 + r}\,\sqrt{2.95061 + r}\right)\right\}$$

Check a few out:

```
ColumnForm[Table[{r, eigenvalue1[r], eigenvalue2[r]}, {r, -2, 16, 2}]]
```

```
{-2, 0.181818 - 0.339031 I, 0.181818 + 0.339031 I}
{0, 0. - 0.554977 I, 0. + 0.554977 I}
{2, -0.181818 - 0.659501 I, -0.181818 + 0.659501 I}
{4, -0.363636 - 0.704108 I, -0.363636 + 0.704108 I}
{6, -0.545455 - 0.700342 I, -0.545455 + 0.700342 I}
{8, -0.727273 - 0.64736 I, -0.727273 + 0.64736 I}
{10, -0.909091 - 0.530616 I, -0.909091 + 0.530616 I}
{12, -1.09091 - 0.279137 I, -1.09091 + 0.279137 I}
{14, -1.71072, -0.834738}
{16, -2.18098, -0.728114}
```

All but the large r's.

Look at the oh-so important real number that determines whether

{xeq[r], yeq[r]}

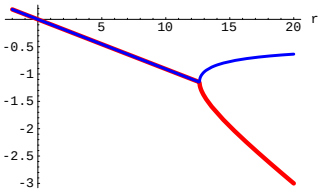
is an attractor or a repellor:

```
ColumnForm[
 Chop[Table[{r, (eigenvalue1[r] + Conjugate[eigenvalue1[r]]) / 2,
    (eigenvalue2[r] + Conjugate[eigenvalue2[r]]) / 2},
   {r, -2, 16, 2}]]]
```

```
{-2, 0.181818, 0.181818}
{0, 0, 0}
{2, -0.181818, -0.181818}
{4, -0.363636, -0.363636}
{6, -0.545455, -0.545455}
{8, -0.727273, -0.727273}
{10, -0.909091, -0.909091}
{12, -1.09091, -1.09091}
{14, -1.71072, -0.834738}
{16, -2.18098, -0.728114}
```

Plot as a function of r:

```
eigenvalplot = Plot[{1/2 (eigenvalue1[r] + Conjugate[eigenvalue1[r]]),
   1/2 (eigenvalue2[r] + Conjugate[eigenvalue2[r]])}, {r, -2, 20},
  PlotStyle → {{Red, Thickness[0.015]}, {Blue, Thickness[0.01]}},
  AspectRatio → 1/GoldenRatio, AxesLabel → {"r", ""}];
```



The plot siganals that:

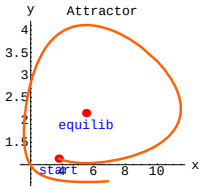
-> The equillibrium points {xeq[r], yeq[r]} are attractors for r > 0.

-> The equillibrium points{xeq[r], yeq[r]} are repellors for r < 0.

Here's a sample for positive r:

```
r = 2.5

h = 2;
{xstarter, ystarter} = {Random[Real, {xeq[r] - h, xeq[r] + h}],
      Random[Real, {yeq[r] - h, yeq[r] + h}]};
endtime = 15;
starterpoint = {xstarter, ystarter};

Clear[x, y];
xdiffeq = (x'[t] == m[x[t], y[t]]);
ydiffeq = (y'[t] == n[x[t], y[t]]);

xstartereqn = (x[0] == xstarter);
ystartereqn = (y[0] == ystarter);

approxsolutions = NDSolve[{xdiffeq, ydiffeq,
  xstartereqn, ystartereqn}, {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} = {x[t] /. approxsolutions[[1]],
        y[t] /. approxsolutions[[1]]};
```

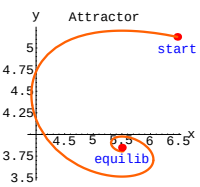
```
Clear[x, y];
xdiffeq = (x'[t] == m[x[t], y[t]]);
ydiffeq = (y'[t] == n[x[t], y[t]]);

xstartereqn = (x[0] == xstarter);
ystartereqn = (y[0] == ystarter);

approxsolutions = NDSolve[{xdiffeq, ydiffeq,
  xstartereqn, ystartereqn}, {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} = {x[t] /. approxsolutions[[1]],
        y[t] /. approxsolutions[[1]]};

trajectoryplot = ParametricPlot[{x[t], y[t]}, {t, 0, endtime},
   PlotStyle -> {{CadmiumOrange, Thickness[0.015]}},
      DisplayFunction -> Identity];

starterplot = Graphics[{Red, PointSize[0.05],
         Point[starterpoint]}];
equilibriumplot = Graphics[{Red, PointSize[0.05],
         Point[{xeq[r], yeq[r]}]}];

starterpointlabel =
 Graphics[{Blue, Text["start", starterpoint, {0, 1.5}]}];

equiliblabel =
 Graphics[{Blue, Text["equilib", {xeq[r], yeq[r]}, {0, 1.5}]}];

Show[starterplot, trajectoryplot,
 equilibriumplot, starterpointlabel, equiliblabel ,
PlotRange -> All,
 Axes -> True, AxesLabel -> {"x", "y"}, AspectRatio -> 1,
  PlotLabel -> "Attractor", DisplayFunction -> $DisplayFunction];
```



Rerun a couple of times.

```
trajectoryplot = ParametricPlot[{x[t], y[t]}, {t, 0, endtime},
   PlotStyle -> {{CadmiumOrange, Thickness[0.015]}},
      DisplayFunction -> Identity];

starterplot = Graphics[{Red, PointSize[0.05],
         Point[starterpoint]}];
equilibriumplot = Graphics[{Red, PointSize[0.05],
         Point[{xeq[r], yeq[r]}]}];

starterpointlabel =
 Graphics[{Blue, Text["start", starterpoint, {0, 1.5}]}];

equiliblabel =
 Graphics[{Blue, Text["equilib", {xeq[r], yeq[r]}, {0, 1.5}]}];

Show[starterplot, trajectoryplot,
 equilibriumplot, starterpointlabel, equiliblabel ,
PlotRange -> All,
 Axes -> True, AxesLabel -> {"x", "y"}, AspectRatio -> 1,
  PlotLabel -> "Attractor", DisplayFunction -> $DisplayFunction];
```

```
2.5
```



Rerun a couple of times.

A genuine attractor.

See what happens for a sample negative r:

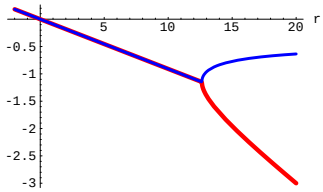
```
r = -0.3;

h = 2;
{xstarter, ystarter} = {Random[Real, {xeq[r] - h, xeq[r] + h}],
      Random[Real, {yeq[r] - h, yeq[r] + h}]};
endtime = 15;
starterpoint = {xstarter, ystarter};
```

Looks like a genuine repellor.

□T.3.a.iii)

Look at the plot of the real parts of the eigenvalues of the linearization matrix:

```
Show[eigenvalplot];
```

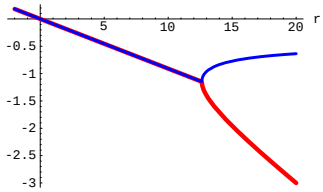


How do you interpret that fork?

□Answer:

Take another look:

```
Show[eigenvalplot];
```



This bifurcation happens for r somewhere between 12 and 13:

to see what the fork means, look at:

```
r = 12.62;
Eigenvalues[A[xeq[r], yeq[r]]]
```

```
{-1.14727 + 0.0369498 I, -1.14727 - 0.0369498 I}
```

```
r = 12.63;
Eigenvalues[A[xeq[r], yeq[r]]]
```

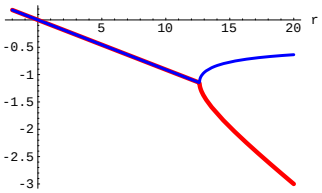
```
{-1.14818 + 0.00886072 I, -1.14818 - 0.00886072 I}
```

```
r = 12.64;
Eigenvalues[A[xeq[r], yeq[r]]]
```
```
{-1.18387, -1.11431}
```

The bifurcation point r between swirl and no swirl is between 12.63 and 12.64.

Take another look at the fork:

```
Show[eigenvalplot];
```



The handle of the fork on the left corresponds to (theoretical) swirl.

The right hand tines of the fork correspond to no swirl at all.

DE.08 Linearizations
Give it a Try!

G.1) Linearization plots

□G.1.a.i)

Here's a nonlinear system:

```
Clear[m, n, x, y, a, b, c, d, r, t]
m[x_, y_] = -0.9 Sin[x] + 1.5 y;
n[x_, y_] = -2.4 x + 1.2 y;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```
```
x'[t] == -0.9 Sin[x[t]] + 1.5 y[t]
y'[t] == -2.4 x[t] + 1.2 y[t]
```

You can see that {0, 0} is an equilbrium point for this nonlinear system:

```
{xeq, yeq} = {0, 0};
m[xeq, yeq] == 0
n[xeq, yeq] == 0
```
```
True
True
```

The linearization (Jacobian) matrix at {xeq, yeq} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[xeq, yeq]]
```
$$\begin{pmatrix} -0.9 & 1.5 \\ -2.4 & 1.2 \end{pmatrix}$$

The eigenvalues of A[xeq, yeq] are:

```
Eigenvalues[A[xeq, yeq]]
```
```
{0.15 + 1.58035 I, 0.15 - 1.58035 I}
```

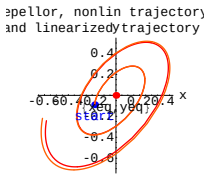
Say why this information signals that this equilibrium point is a repellor.

□G.1.a.ii)

Keep everything in part i) live and look at this plot:

```
endtime = 8;
h = 0.2;
s = Random[Real, {0, N[2 π]}];
{xstarter, ystarter} = {xeq, yeq} + h {Cos[s], Sin[s]};
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
nonlinsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
 {x[t] /. nonlinsolutions[[1]], y[t] /. nonlinsolutions[[1]]};
nonlineartrajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.01]}}, DisplayFunction → Identity];
```

```
Clear[x, y, linm, linn, t]
{linm[x_, y_], linn[x_, y_]} = A[xeq, yeq] . {x - xeq, y - yeq};
xlindiffeq = x'[t] == linm[x[t], y[t]];
ylindiffeq = y'[t] == linn[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
linsolutions =
 NDSolve[{xlindiffeq, ylindiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
{x[t_], y[t_]} = {x[t] /. linsolutions[[1]], y[t] /. linsolutions[[1]]};
lintrajectoryplot = ParametricPlot[{x[t], y[t]}, {t, 0, endtime},
  PlotStyle → {{Red, Thickness[0.01]}}, DisplayFunction → Identity];

starterplot = Graphics[{Blue, PointSize[0.05], Point[starterpoint]}];
starterpointlabel =
 Graphics[{Blue, Text["start", starterpoint, {0, 1.5}]}];
equilibriumplot = Graphics[{Red, PointSize[0.05], Point[{xeq, yeq}]}];
equiliblabel = Graphics[{Text["{xeq,yeq}", {xeq, yeq}, {0, 1.5}]}];

Show[starterplot, lintrajectoryplot, nonlineartrajectoryplot,
 starterpointlabel, equilibriumplot, equiliblabel, PlotRange → All,
 Axes → True, AxesLabel → {"x", "y"}, AspectRatio → 1, PlotLabel →
  "repellor, nonlin trajectory \n and linearized trajectory"];
```



Rerun a couple of times.

Each plot shows a trajectory of the non-linear system (orange) along with the linearized trajectory (red) starting at the same point.
Why is it guaranteed that the two trajectories share a lot of ink as they begin their trip?
How are Lyapunov's rules related to what you see?
Why is it guaranteed that the two trajectories eventually had to break apart?

□G.1.b.i)

Here's another nonlinear system:

```
Clear[m, n, x, y, a, b, c, d, r, t]
m[x_, y_] = -1.3 x + 0.5 y;
n[x_, y_] = 0.3 x - 0.8 Sin[1.5 y];
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```
```
x'[t] == -1.3 x[t] + 0.5 y[t]
y'[t] == -0.8 Sin[1.5 y[t]] + 0.3 x[t]
```

You can see that {0, 0} is an equilbrium point for this nonlinear system:

```
{xeq, yeq} = {0, 0};
m[xeq, yeq] == 0
n[xeq, yeq] == 0
```
```
True
True
```

The linearization matrix at {xeq, yeq} is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[xeq, yeq]]
```
$$\begin{pmatrix} -1.3 & 0.5 \\ 0.3 & -1.2 \end{pmatrix}$$

The eigenvalues of A[xeq, yeq] are:

```
Eigenvalues[A[xeq, yeq]]
```
```
{-1.64051, -0.859488}
```

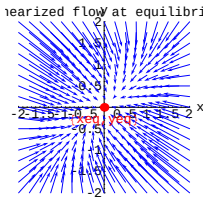
Say why this information signals that this equilibrium point is an attractor.

□G.1.b.ii)

Keep everything in part i) live and look at this flow plot of the linearized system at {0, 0}:

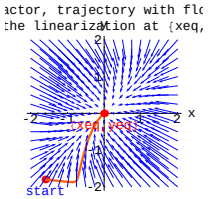
```
h = 2;
{xlow, xhigh} = {xeq - h, xeq + h};
{ylow, yhigh} = {yeq - h, yeq + h};
jump = (xhigh - xlow)/16;
```

```
Clear[linearizedField]
linearizedField[x_, y_] = A[xeq, yeq].{x - xeq, y - yeq};
linearizedflowplot = Table[Arrow[linearizedField[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → 0.25, HeadSize → 0.12],
  {x, xlow, xhigh, jump}, {y, ylow, yhigh, jump}];
equilibriumplot = Graphics[{Red, PointSize[0.05], Point[{xeq, yeq}]}];
equiliblabel =
 Graphics[{Red, Text["{xeq,yeq}", {xeq, yeq}, {0, 1.5}]}];
linearizedflows = Show[linearizedflowplot, equilibriumplot,
  equiliblabel, PlotRange → {{xeq - h, xeq + h}, {yeq - h, yeq + h}},
  Axes → True, AxesLabel → {"x", "y"},
  PlotLabel → "Linearized flow at equilibrium"];
```



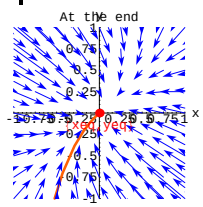
Now look at this:

```
endtime = 16;
{xstarter, ystarter} = {-1.6, -1.8};
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.05], Point[starterpoint]}];
starterpointlabel =
 Graphics[{Blue, Text["start", starterpoint, {0, 1.5}]}];
scaler = 0.75 h;
setup = Show[starterplot, trajectoryplot,
  linearizedflows, starterpointlabel, PlotRange → All, Axes → True,
  AxesLabel → {"x", "y"}, AspectRatio → 1, PlotLabel → "Attractor,
     trajectory with flow \n of the linearization at {xeq,yeq}",
  DisplayFunction → $DisplayFunction];
```



This plot shows the linearized flow together with a trajectory in the nonlinear system.
Look at the final stages of the trajectory as closing in on the attractor equilibrium point at {0, 0}:

```
s = 1.0;
Show[setup, PlotRange → {{xeq - s, xeq + s}, {yeq - s, yeq + s}},
 PlotLabel → "At the end"];
```

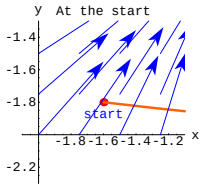


At the end, the trajectory is in good harmony with the linearized flow.
How are Lyapunov's rules related to what you see?
How do you account for this?

□G.1.b.iii)

Stay with the same setup as in part ii).
Look at the linearized flow together with a trajectory in the nonlinear system at the very beginning of the trip:

```
s = 0.5;
Show[setup,
 PlotRange → {{xstarter - s, xstarter + s}, {ystarter - s, ystarter + s}},
 PlotLabel → "At the start"];
```

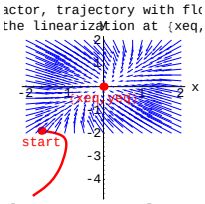


At the beginning, the trajectory is running against the grain of the linearized flow.
How do you account for this?

□G.1.b.iv)

Now look at this new trajectory shown with the flow of the linearization:

```
endtime = 16;
{xstarter, ystarter} = {-1.6, -1.9};
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot = ParametricPlot[{x[t], y[t]}, {t, 0, endtime},
  PlotStyle → {{Red, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.05], Point[starterpoint]}];
starterpointlabel =
 Graphics[{Red, Text["start", starterpoint, {0, 1.5}]}];
scaler = 0.75 h;
Show[starterplot, trajectoryplot, linearizedflows, starterpointlabel,
 PlotRange → All, Axes → True, AxesLabel → {"x", "y"},
 AspectRatio → 1, PlotLabel → "Attractor, trajectory
    with flow \n of the linearization at {xeq,yeq}",
 DisplayFunction → $DisplayFunction];
```



This trajectory doesn't give a damn about the linearized flow . Give you opinion about why that's not too unsettling.

□G.1.c)

Here's another nonlinear system:

```
Clear[m, n, x, y, a, b, c, d, r, t]
m[x_, y_] = 0.7 x - 2.8 x y + 10;
n[x_, y_] = -0.44 y + 0.8 y x - 0.9;
DiffEqsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DiffEqsystem]]
```

$$x'[t] == 10 + 0.7\,x[t] - 2.8\,x[t]\,y[t]$$
$$y'[t] == -0.9 - 0.44\,y[t] + 0.8\,x[t]\,y[t]$$

The equilibrium points are:

```
equisols = ColumnForm[Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}]]
```

$$\{x \to -10.0199,\ y \to -0.106435\}$$
$$\{x \to 0.784156,\ y \to 4.80449\}$$

Concentrate on the second equilbrium point:

```
{xeq, yeq} = {0.784156, 4.80449}
```

$$\{0.784156,\ 4.80449\}$$

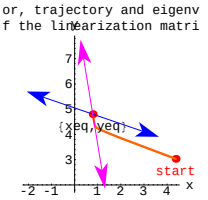
Calculate the eigenvalues of the linearization matrix at {xeq, yeq}:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
Chop[Eigenvalues[A[xeq, yeq]]]
```

$$\{-12.0637,\ -0.501525\}$$

This equilibrium point is a very strong attractor.
Now look at this plot:

```
s = Random[Real, {0, N[2 π]}];
h = 4;
endtime = 16;
{xstarter, ystarter} = {xeq, yeq} + h {Cos[s], Sin[s]};
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
approxsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.05], Point[starterpoint]}];
equilibriumplot = Graphics[{Red, PointSize[0.05], Point[{xeq, yeq}]}];
equiliblabel = Graphics[{Text["{xeq,yeq}", {xeq, yeq}, {0, 1.5}]}];
starterpointlabel =
 Graphics[{Red, Text["start", starterpoint, {0, 1.5}]}];
scaler = 0.75 h;
Clear[eigenvector]
{eigenvector[1], eigenvector[2]} = Eigenvectors[A[xeq, yeq]];
eigenplot = {Arrow[eigenvector[1], Tail → {xeq, yeq},
   VectorColor → Blue, ScaleFactor → 0.75 h, HeadSize → 1],
  Arrow[-eigenvector[1], Tail → {xeq, yeq},
   VectorColor → Blue, ScaleFactor → 0.75 h, HeadSize → 1],
  Arrow[eigenvector[2], Tail → {xeq, yeq},
   VectorColor → Magenta, ScaleFactor → 0.75 h, HeadSize → 1],
  Arrow[-eigenvector[2], Tail → {xeq, yeq},
   VectorColor → Magenta, ScaleFactor → 0.75 h, HeadSize → 1]};
setup = Show[starterplot, trajectoryplot,
  equilibriumplot, equiliblabel, starterpointlabel,
  eigenplot, PlotRange → All, Axes → True, AxesLabel → {"x", "y"},
  AspectRatio → 1, PlotLabel → "Attractor, trajectory
     and eigenvectors \n      of the linearization matrix",
  DisplayFunction → $DisplayFunction];
```



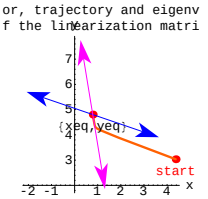
Rerun several times.

Each plot depicts a trajectory in the orginal nonlinear system headed toward the attractor at {xeq, yeq} shown with scaled eigenvectors of the linearization (Jacobian) matrix at {xeq, yeq}.
Notice that the trjectories start out running nearly parallel to the blue eigenvectors and then shift over to try to merge with the line through the equilbrium point running in the direction of the magenta eigenvectors.
How do you account for this?

◻G.1.c.ii)

Look at the last plot:

```
Show[setup];
```



Here is a calculation of the eigenvaluess of the linearization matrix at {xeq, yeq}:

```
Eigenvalues[A[xeq, yeq]]
```

```
{-12.0637, -0.501525}
```

Look at the plot and then say which of the eigenvalues corresponds to the magenta eigenvectors.

G.2) Lyapunov's Rules

In this problem, you are going to have the experience of trying out Lyapunov's rules for yourself.

◻G.2.a.i)

Here's a nonlinear system:

```
Clear[m, n, x, y, t]
Clear[m, n]
m[x_, y_] = x^2 - 4 x + 2 y + 2 x y - y^2;
n[x_, y_] = -x^2 - 4 y - 2 x + 2 x y + y^2;

DEsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DEsystem]]
```

$$x'[t] == -4\,x[t] + x[t]^2 + 2\,y[t] + 2\,x[t]\,y[t] - y[t]^2$$
$$y'[t] == -2\,x[t] - x[t]^2 - 4\,y[t] + 2\,x[t]\,y[t] + y[t]^2$$

The linearization (Jacobian) matrix A[x, y] at {x, y} for this system is:

Some folks call this matrix by the name "Jacobian."

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} -4 + 2\,x + 2\,y & 2 + 2\,x - 2\,y \\ -2 - 2\,x + 2\,y & -4 + 2\,x + 2\,y \end{pmatrix}$$

Here come the equilibrium points:

```
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
Solve[Field[x, y] == {0, 0}]
```

$$\left\{\{x \to 0,\ y \to 0\},\ \left\{x \to \frac{1}{2} - \frac{3\,I}{2},\ y \to \frac{3}{2} + \frac{I}{2}\right\},\ \left\{x \to \frac{1}{2} + \frac{3\,I}{2},\ y \to \frac{3}{2} - \frac{I}{2}\right\},\ \{x \to 1,\ y \to 3\}\right\}$$

The equilibrium points involving $I = \sqrt{-1}$ don't count.

Throw away the imaginary stuff and go with these two equilibrium points:

```
Clear[xeq, yeq]
{xeq[1], yeq[1]} = {0, 0}
{xeq[2], yeq[2]} = {1, 3}
```

```
{0, 0}
{1, 3}
```

The eigenvalues of the linearization matrix at {xeq[1], yeq[1]} are:

```
Eigenvalues[A[xeq[1], yeq[1]]]
```

```
{-4 - 2 I, -4 + 2 I}
```

The eigenvalues of the linearization matrix at {xeq[2], yeq[2]} are:

```
Eigenvalues[A[xeq[2], yeq[2]]]
```

```
{4 - 2 I, 4 + 2 I}
```

Use Lyapunov's rules to determine which equilibrium point is a repeller and which is an attractor.

◻G.2.a.ii)

Run some experiments in effort to come up with opinions on the following question:
When you start a trajectory near the repeller, does it go to the attractor?

◻G.2.b.i)

Here's a new system:

```
Clear[m, n, x, y, t]
Clear[m, n]
m[x_, y_] = -y + x^2 + y^2;
n[x_, y_] = x - x^2 - y^2;

DEsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DEsystem]]
```

$$x'[t] == x[t]^2 - y[t] + y[t]^2$$
$$y'[t] == x[t] - x[t]^2 - y[t]^2$$

Here come the equilibrium points:

```
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
N[Solve[Field[x, y] == {0, 0}]]
```

```
{{x → 0, y → 0}, {x → 0.5, y → 0.5}}
```

This gives you two equilibrium points:

```
Clear[xeq, yeq]
{xeq[1], yeq[1]} = {0, 0}
{xeq[2], yeq[2]} = {0.5, 0.5}
```

```
{0, 0}
{0.5, 0.5}
```

The linearization matrix (Jacobian) is:

```
Clear[A, gradm, gradn]
gradm[x_, y_] = {∂x m[x, y], ∂y m[x, y]};
gradn[x_, y_] = {∂x n[x, y], ∂y n[x, y]};
A[x_, y_] = {gradm[x, y], gradn[x, y]};
MatrixForm[A[x, y]]
```

$$\begin{pmatrix} 2x & -1+2y \\ 1-2x & -2y \end{pmatrix}$$

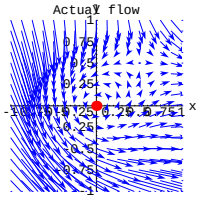
The eigenvalues of the linearization matrix at {0, 0} are:

```
Eigenvalues[A[0, 0]]
```

```
{-I, I}
```

The linearization predicts that trajectories starting near the equilibrium point at {0, 0} cycle around and around and around the equilibrium point on closed curves.
Now look at the actual flow near the equilibrium point at {0, 0}:

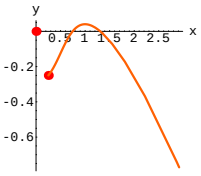
```
h = 1;
flowplot = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → 0.3, HeadSize → 0.1],
  {x, xeq[1] - h, xeq[1] + h, h/8}, {y, yeq[1] - h, yeq[1] + h, h/8}];
equilibplot =
 Graphics[{Red, PointSize[0.06], Point[{xeq[1], yeq[1]}]}];
actualflow = Show[flowplot, equilibplot, Axes → True,
  AxesOrigin → {xeq[1], yeq[1]}, AxesLabel → {"x", "y"},
  PlotRange → {{xeq[1] - h, xeq[1] + h}, {yeq[1] - h, yeq[1] + h}},
  PlotLabel → "Actual flow"];
```



Take a look at a trajectory starting near the equilibrium point:

```
{a, b} = {0.25, -0.25};
starterpoint = {a, b};
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t]];
equationy = y'[t] == n[x[t], y[t]];
```

```
starterx = x[0] == a;
startery = y[0] == b;
endtime = 2.2;
approxsolutions = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.06], Point[starterpoint]}];
Show[starterplot, equilibplot, trajectoryplot, PlotRange → All,
 Axes → True, AxesLabel → {"x", "y"}, AspectRatio → 1,
 DisplayFunction → $DisplayFunction];
```



The linearization at the equilibrium point {0, 0} dropped the strong hint that this trajectory would cycle around and around {0, 0}. Does it? Why are you not surprised?

□G.2.b.ii)

Explain the wisdom behind:

□No information rule:

If the equilibrium point is a neither a repeller nor an attractor nor a saddle, then you are walking on thin ice if you trust the information you get from the linearization at the equilibrium point.

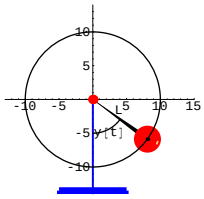
G.3) Energy and the undamped pendulum oscillator

Before you starton this one, it would be a very good idea to become familiar with the Basic problem on the pendulum oscillator.

□G.3.a.i)

Here's another look at everyone's favorite:

```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
Clear[y, t, s]
L = 10;
y[t] = 0.3 π;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
all = Show[pendulum[L, y[t]], angle, labels, bobpath, Axes → True,
  AxesOrigin → {0, 0}, DisplayFunction → $DisplayFunction];
```



Thanks to Ben Halperin for the pendulum graphics.
In the grapics for this part, the sample value $L = 10$ is used.

The position of the center of the bob at time t is given by:

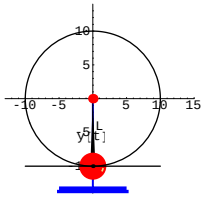
```
Clear[position, L, M, g, y, t];
position[t_] = L {Sin[y[t]], -Cos[y[t]]}
```

```
{L Sin[y[t]], -L Cos[y[t]]}
```

When y[t] = 0, the pendulum looks like this:

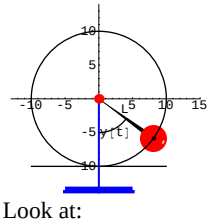
```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
Clear[y, t, s]
L = 10;
y[t] = 0;
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
zeropotentialenergyline = Graphics[Line[{{-L, -L}, {L, -L}}]];
```

```
all = Show[pendulum[L, y[t]], labels, bobpath,
  zeropotentialenergyline, Axes → True, AxesOrigin → {0, 0},
  DisplayFunction → $DisplayFunction];
```



This is the position of zero potential energy.
See the pendulum in a different position:

```
bobpath = ParametricPlot[
  10 {Sin[s], -Cos[s]}, {s, -π, π}, DisplayFunction → Identity];
Clear[y, t, s]
L = 10;
y[t] = 0.3 π;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]}];
zeropotentialenergyline = Graphics[Line[{{-L, -L}, {L, -L}}]];
all = Show[pendulum[L, y[t]], angle, labels, bobpath,
  zeropotentialenergyline, Axes → True, AxesOrigin → {0, 0},
  DisplayFunction → $DisplayFunction];
```



Look at:

```
Clear[L, y, t]
position[t]
```

```
{L Sin[y[t]], -L Cos[y[t]]}
```

For a given y[t], the center of the bob is
L – L Cos[y[t]]
units above the zeropotential energy line.
So, as the physics folks say:

```
Clear[potentialenergy, M, g, L]
potentialenergy[t_] = M g (L - L Cos[y[t]])
```

$g\,M\,(L - L\,\text{Cos}[y[t]])$

The kinetic energy $\frac{m v^2}{2}$ at the center of the bob at time t is:

```
Clear[kineticenergy]
kineticenergy[t_] = 1/2 Simplify[M position'[t] . position'[t]]
```

$\frac{1}{2} L^2 M y'[t]^2$

The total energy is:

```
Clear[totalenergy]
totalenergy[t_] = kineticenergy[t] + potentialenergy[t]
```

$g\,M\,(L - L\,\text{Cos}[y[t]]) + \frac{1}{2} L^2 M y'[t]^2$

One physical principle is that the the total energy of the undamped pendulum remains constant as time varies.
This is the same as saying that
totalenergy′[t] = 0
for all times t.
Execute this:

```
Solve[totalenergy'[t] == 0, y''[t]]
```

$\{\{y''[t] \to -\frac{g\,\text{Sin}[y[t]]}{L}\}\}$

Compare your result with the undamped pendulum oscillator differential equation as propounded in the Basics:

```
pendulumoscillator = y''[t] + g Sin[y[t]]/L == 0
```

$\frac{g\,\text{Sin}[y[t]]}{L} + y''[t] == 0$

Write up your conclusion.

□G.3.a.ii)

Look at this:

```
Clear[potentialenergy, M, g, L]
potentialenergy[t_] = M g (L - L Cos[y[t]])
```

$g\,M\,(L - L\,\text{Cos}[y[t]])$

```
Clear[kineticenergy]
kineticenergy[t_] = 1/2 Simplify[M position'[t] . position'[t]]
```

$\frac{1}{2} L^2 M y'[t]^2$

```
Clear[totalenergy]
totalenergy[t_] = kineticenergy[t] + potentialenergy[t]
```

$g\,M\,(L - L\,\text{Cos}[y[t]]) + \frac{1}{2} L^2 M y'[t]^2$

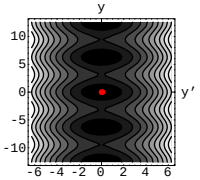
The total energy at a time t at which y[t] = x and y' [t] = y is:

```
Clear[emergy, x, y]
energy[x_, y_] = totalenergy[t] /. {y[t] → y, y'[t] → x}
```

$\frac{1}{2} L^2 M x^2 + g\,M\,(L - L\,\text{Cos}[y])$

Now look at this contour plot of energy[x,y];

```
L = 5;
g = 9.8;
M = 1;
equilibriumplot = Graphics[{PointSize[0.04], Red, Point[{0, 0}]}];
levelcurves = ContourPlot[Evaluate[energy[x, y]], {x, -2 π, 2 π},
  {y, -4 π, 4 π}, PlotPoints → 50, ContourSmoothing → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
setup = Show[levelcurves, equilibriumplot, Axes → True,
  AxesLabel → {"y'", "y"}, DisplayFunction → $DisplayFunction];
```



On each plotted curve, energy[x,y] stays constant.
The lighter shading indicates larger values of energy[x,y].

Throw in a trajectory in the system by converting the undamped pendulum oscillator to a nonlinear system:

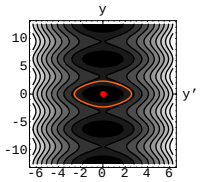
```
Clear[x, y, t]
pendulumoscillator = (y''[t] + (g / L) Sin[y[t]] == 0);
Clear[x]
ColumnForm[Thread[{new = (y'[t] == x[t]),
pendulumoscillator /. {y'[t] -> x[t], y''[t] -> x'[t]}}]]
```

$y'[t] == x[t]$
$1.96\,\text{Sin}[y[t]] + x'[t] == 0$

```
Clear[m, n, x, y, t]
m[x_, y_] = -(g / L) Sin[y];
n[x_, y_] = x;
pendulumsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[pendulumsystem]]
```

$x'[t] == -1.96\,\text{Sin}[y[t]]$
$y'[t] == x[t]$

```
endtime = 7;
{xstarter, ystarter} = {1.6, 1.6};
starterpoint = {xstarter, ystarter};

Clear[x, y];
xdiffeq = (x'[t] == m[x[t], y[t]]);
ydiffeq = (y'[t] == n[x[t], y[t]]);

xstartereqn = (x[0] == xstarter);
ystartereqn = (y[0] == ystarter);

nonlinsolutions = NDSolve[{xdiffeq, ydiffeq,
  xstartereqn, ystartereqn}, {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} = {x[t] /. nonlinsolutions[[1]],
        y[t] /. nonlinsolutions[[1]]};

trajectoryplot = ParametricPlot[{x[t], y[t]}, {t, 0, endtime},
   PlotStyle ->
   {{CadmiumOrange, Thickness[0.01]}}, DisplayFunction -> Identity];

Show[setup, trajectoryplot, DisplayFunction -> $DisplayFunction];
```



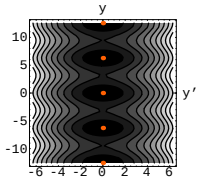
On each plotted curve, energy[x, y] stays constant.
The lighter shading indicates larger values of energy[x, y].

Explain how you know that all the curves (not just the plotted trajectory) shown in the plot above are all trajectories in this system.

□G.3.a.iii)

Take another look at the plot of some level curves of energy[x, y] as above:

```
equilbriumplots = Table[Graphics[
   {PointSize[0.03], CadmiumOrange, Point[{0, k 2 π}]}], {k, -2, 2}];
newsetup = Show[levelcurves, equilbriumplots, Axes → True,
  AxesLabel → {"y'", "y"}, DisplayFunction → $DisplayFunction];
```

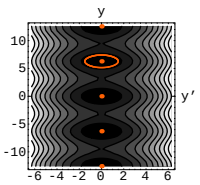


Discuss what other information you get from this plot.
What makes the pendulum operate with high energy? Low energy?

There are some additional equilibrium points other than {0, 0}.
Where are they and what do they mean?
To get a handle on these questions, you might want to play with trajectories such as:

```
endtime = 13;
{xstarter, ystarter} = {1.5, 2 π};
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x'[t] == m[x[t], y[t]];
ydiffeq = y'[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
nonlinsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
```

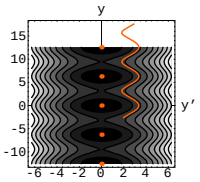
```
 {x[t] /. nonlinsolutions[[1]], y[t] /. nonlinsolutions[[1]]};
trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.01]}}, DisplayFunction → Identity];
(Show[newsetup, trajectoryplot, DisplayFunction → $DisplayFunction];)
```



Or:

```
endtime = 8;
{xstarter, ystarter} = {1.98, -2.6}
starterpoint = {xstarter, ystarter};
Clear[x, y];
xdiffeq = x′[t] == m[x[t], y[t]];
ydiffeq = y′[t] == n[x[t], y[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
nonlinsolutions =
 NDSolve[{xdiffeq, ydiffeq, xstartereqn, ystartereqn},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[x, y, t];
{x[t_], y[t_]} =
 {x[t] /. nonlinsolutions[[1]], y[t] /. nonlinsolutions[[1]]};
trajectoryplot =
 ParametricPlot[{x[t], y[t]}, {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.01]}}, DisplayFunction → Identity];
Show[newsetup, trajectoryplot, DisplayFunction → $DisplayFunction];
```

{1.98, -2.6}



This is an ordinary damped oscillator.
Go ahead and assume that M, g, L and b are positive.
Use your knowledge of ordinary oscillators to predict how the actual pendulum oscillator reacts when you:
→ Make M (mass) bigger and bigger.
→ Make L (length) bigger and bigger.
→ Make g (gravity) bigger and bigger.
→ Make b (friction) bigger and bigger.

□G.4.b.i) Flows and trajectories for the damped pendulum oscillator

The undamped pendulum oscillator is modeled by this second order differential equation:

```
Clear[t, L, g, b, M, y]
dampedpendosc = y″[t] + b y′[t]/M + g Sin[y[t]]/L == 0
```

$$\frac{g \sin[y[t]]}{L} + \frac{b\, y'[t]}{M} + y''[t] == 0$$

You make this second order differential equation into a system of two first order differential equations by putting

x[t] = y′[t]

and then replacing y′[t] by x[t] and replacing y″[t] by x′[t]:

```
Clear[x]
ColumnForm[Thread[{new = (y'[t] == x[t]),
dampedpendosc /. {y'[t] -> x[t], y''[t] -> x'[t]}}] ]
```

y′[t] == x[t]

$\frac{g \sin[y[t]]}{L} + \frac{b\, x[t]}{M} + x'[t] == 0$

Clean this up:

```
Clear[m, n, x, y, t]
m[x_, y_] = -g Sin[y]/L - b x/M;
n[x_, y_] = x;
dampedpendulumsystem =
 ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[dampedpendulumsystem]]
```

$x'[t] == -\frac{g \sin[y[t]]}{L} - \frac{b\, x[t]}{M}$

y′[t] == x[t]

Feast your eyes on the flow in the sample case

L = 5, g = 9.8, b = 0.2, and M = 1:

G.4) Playing with the damped pendulum oscillator and its linearization

□G.4.a.i)

Advanced references give this as a model of the damped pendulum oscillator:

```
Clear[t, L, g, b, M, y]
pendulumoscillator = y″[t] + b y′[t]/M + g Sin[y[t]]/L == 0
```

$$\frac{g \sin[y[t]]}{L} + \frac{b\, y'[t]}{M} + y''[t] == 0$$

Yet most elementary math and physics texts give this as a model of the damped pendulum oscillator:

```
simplependulumoscillator = y″[t] + b y′[t]/M + g y[t]/L == 0
```

$$\frac{g\, y[t]}{L} + \frac{b\, y'[t]}{M} + y''[t] == 0$$

Adapt the linearization of the undamped pendulum oscillator from the Basics in effort to linearize the damped pendulum oscillator at its equilibrium point at {0, 0} in effort to explain where the simplified version comes from.

□G.4.a.ii) Linear predictions

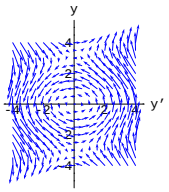
Because linear {differential] equations are easy to solve and study,
the theory of linear oscillation is the most highly developed area of
mechanics.
In many non-linear problems, linearization produces a
satisfactory approximate solution. Even when this is not the case,
the study of the linear part of a problem is often a first step,
to be followed by the study of the relation between motions in a
nonlinear system and its linear model.
-------World DiffEq Master V. I. Arnold
writing in his advanced book Mathematical Methods of Classical Mechanics
(Springer-Verlag, New York, 1989)

Look again at the simple damped pendulum oscillator:

```
simplependulumoscillator = y″[t] + b y′[t]/M + g y[t]/L == 0
```

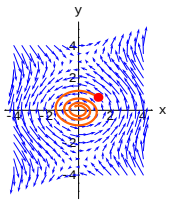
$$\frac{g\, y[t]}{L} + \frac{b\, y'[t]}{M} + y''[t] == 0$$

```
b = 0.2;
g = 9.8;
L = 5.0;
M = 1;
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
flowplot = Table[Arrow[Field[x, y], Tail → {x, y}, VectorColor → Blue,
   ScaleFactor → 0.3, HeadSize → 0.3], {x, -4, 4, 0.5}, {y, -4, 4, 0.5}];
pendulumflow = Show[flowplot, Axes → True, AxesLabel → {"y'", "y"}];
```



Throw in a trajectory:

```
{startingvelocity, startingangle} = {1.2, 0.8};
starterpoint = {startingvelocity, startingangle};

Clear[x, y, t]
equationx = (x'[t] == m[x[t], y[t]]);
equationy = (y'[t] == n[x[t], y[t]]);
starterx = (x[0] == startingvelocity);
startery = (y[0] == startingangle);
endtime = 15;

approxsolutions = NDSolve[{equationx, equationy,
  starterx, startery}, {x[t], y[t]}, {t, 0, endtime}];

Clear[trajectory]
trajectory[t_] = {x[t] /. approxsolutions[[1]],
        y[t] /. approxsolutions[[1]]};

trajectoryplot = ParametricPlot[trajectory[t], {t, 0, endtime},
   PlotStyle -> {{CadmiumOrange, Thickness[0.015]}},
       DisplayFunction -> Identity];

experiment1 = Show[flowplot, trajectoryplot,
    PlotRange -> All, Axes -> True, AxesLabel -> {"x", "y"},
    DisplayFunction -> $DisplayFunction,
  Epilog -> {Red, PointSize[0.06],
          Point[starterpoint]}];
```

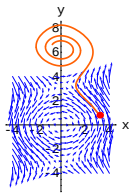


What does this plot signify?
Why do you think that happened?

□G.4.b.ii)

Throw in another trajectory corresponding to a bigger starting velocity::

```
{startingvelocity, startingangle} = {3.2, 0.8};
starterpoint = {startingvelocity, startingangle};
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t]];
equationy = y'[t] == n[x[t], y[t]];
starterx = x[0] == startingvelocity;
startery = y[0] == startingangle;
endtime = 15;
approxsolutions = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
experiment1 =
 Show[flowplot, trajectoryplot, PlotRange → All, Axes → True,
  AxesLabel → {"x", "y"}, DisplayFunction → $DisplayFunction,
  Epilog → {Red, PointSize[0.06], Point[starterpoint]}];
```

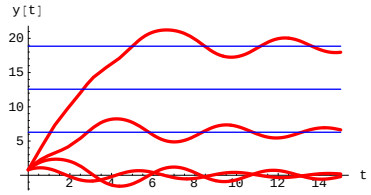


What does this plot signify?
Why do you think that happened?

□G.4.b.iii)

Stay with the same setup as in the part immediately above:

```
{b, L, g, M} = {0.2, 5.0, 9.8, 1.0};
Clear[y, yslow, yfaster, yevenfaster, yreallyfast]
dampedpendosc = y''[t] + b y'[t]/M + g Sin[y[t]]/L == 0;
endtime = 15;
slowstart = NDSolve[
  {dampedpendosc, y[0] == 0.8, y'[0] == 1}, y[t], {t, 0, endtime}];
yslow[t_] = y[t] /. slowstart[[1]];
fasterstart = NDSolve[
  {dampedpendosc, y[0] == 0.8, y'[0] == 2.55}, y[t], {t, 0, endtime}];
yfaster[t_] = y[t] /. fasterstart[[1]];
evenfasterstart = NDSolve[
  {dampedpendosc, y[0] == 0.8, y'[0] == 3.2}, y[t], {t, 0, endtime}];
yevenfaster[t_] = y[t] /. evenfasterstart[[1]];
reallyfaststart = NDSolve[
  {dampedpendosc, y[0] == 0.8, y'[0] == 6.0}, y[t], {t, 0, endtime}];
yreallyfast[t_] = y[t] /. reallyfaststart[[1]];
Plot[{yslow[t], yfaster[t], yevenfaster[t], yreallyfast[t]},
 {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.01]}},
 AxesLabel → {"t", "y[t]"}, PlotRange → All, AspectRatio → 1/2,
 Epilog → {{Blue, Line[{{0, 2 π}, {endtime, 2 π}}]},
   {Blue, Line[{{0, 4 π}, {endtime, 4 π}}]},
   {Blue, Line[{{0, 6 π}, {endtime, 6 π}}]}}];
```



Two of these looks something like ordinary damped linear oscillators, and the other two don't look like ordinary damped linear oscillators at all.
What do these plots mean?

□G.4.c.i)

Go to the sample case of L = 5.0, g = 9.8, b = 0.2, and M = 1 studied above.
The corresponding nonlinear damped pendulum oscillator diffeq is:

```
{b, L, g, M} = {0.2, 5.0, 9.8, 1.0};
Clear[t, y]
dampedpendosc = y''[t] + b y'[t]/M + g Sin[y[t]]/L == 0
```

```
1.96 Sin[y[t]] + 0.2 y'[t] + y''[t] == 0
```

Its linearized version is:

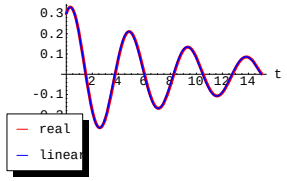
```
linearversion = y''[t] + b y'[t]/M + g y[t]/L == 0
```

```
1.96 y[t] + 0.2 y'[t] + y''[t] == 0
```

Now look at these plots of the actual damped pendulum oscillator and its linearized version:
The first plot goes with starter data
{y[0], y′[0]} = {0.3, 0.2}:

```
{starterangle, startervel} = {0.3, 0.2};
endtime = 15;
Clear[t, y, yreal, ylinear];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
simplething = NDSolve[{linearversion,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
ylinear[t_] = y[t] /. simplething[[1]];
```

```
Plot[{yreal[t], ylinear[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
 PlotLegend → {"real", "linear"}, LegendSize → 0.6];
```



The formula for the solution of the linearized version of this damped pendulum oscillator is:

```
This baroque code is needed to get around a major bug in Mathematica 3.0.
Clear[bb, MM, gg, LL, ss, vv, linearformula]
clearlinversion = y''[t] + bb y'[t]/MM + gg y[t]/LL == 0;
linearsol =
 DSolve[{clearlinversion, y[0] == ss, y'[0] == vv}, y[t], t] /.
  {ss → starterangle, vv → startervel, bb → b, MM → M, gg → g, LL → L};
linearformula[t_] = Chop[ComplexExpand[y[t] /. linearsol[[1]]]]
```

```
0.3 E^(-0.1 t) Cos[1.39642 t] + 0.164706 E^(-0.1 t) Sin[1.39642 t]
```

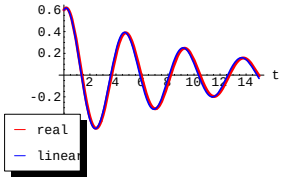
Do you trust this formula to model the actual damped pendulum oscillator?
Why or why not?

□G.4.c.ii)

The next plot raises the starter data from
{y[0], y′[0]} = {0.3, 0.2} to {y[0], y′[0]} = {0.6, 0.2}:

```
{starterangle, startervel} = {0.6, 0.2};
endtime = 15;
Clear[t, y, yreal, ylinear];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
simplething = NDSolve[{linearversion,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
ylinear[t_] = y[t] /. simplething[[1]];
```

```
Plot[{yreal[t], ylinear[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
 PlotLegend → {"real", "linear"}, LegendSize → 0.6];
```



The formula for the solution of the linearized version of this damped pendulum oscillator is:

```
Clear[bb, MM, gg, LL, ss, vv, linearformula]
clearlinversion = y''[t] + bb y'[t]/MM + gg y[t]/LL == 0;
linearsol =
 DSolve[{clearlinversion, y[0] == ss, y'[0] == vv}, y[t], t] /.
  {ss → starterangle, vv → startervel, bb → b, MM → M, gg → g, LL → L};
linearformula[t_] = Chop[ComplexExpand[y[t] /. linearsol[[1]]]]
```

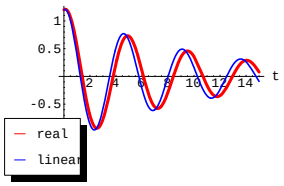
$0.6\,E^{-0.1\,t}\,\text{Cos}[1.39642\,t] + 0.18619\,E^{-0.1\,t}\,\text{Sin}[1.39642\,t]$

Do you trust this formula to model the actual damped pendulum oscillator?
Why or why not?

□G.4.c.iii)

The next plot raises the starter data from
$\{y[0], y'[0]\} = \{0.6, 0.2\}$ to $\{y[0], y'[0]\} = \{1.2, 0.2\}$:

```
{starterangle, startervel} = {1.2, 0.2};
endtime = 15;
Clear[t, y, yreal, ylinear];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
simplething = NDSolve[{linearversion,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
ylinear[t_] = y[t] /. simplething[[1]];
Plot[{yreal[t], ylinear[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
 PlotLegend → {"real", "linear"}, LegendSize → 0.6];
```



The formula for the solution of the linearized version of this damped pendulum oscillator is:

```
Clear[bb, MM, gg, LL, ss, vv, linearformula]
clearlinversion = y''[t] + bb y'[t]/MM + gg y[t]/LL == 0;
linearsol =
 DSolve[{clearlinversion, y[0] == ss, y'[0] == vv}, y[t], t] /.
  {ss → starterangle, vv → startervel, bb → b, MM → M, gg → g, LL → L};
linearformula[t_] = Chop[ComplexExpand[y[t] /. linearsol[[1]]]]
```

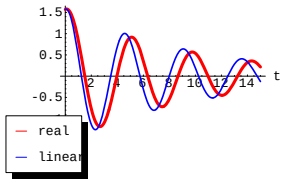
$1.2\,E^{-0.1\,t}\,\text{Cos}[1.39642\,t] + 0.229157\,E^{-0.1\,t}\,\text{Sin}[1.39642\,t]$

Do you trust this formula to model the actual damped pendulum oscillator?
Why or why not?

□G.4.c.iv)

The next plot raises the starter data from
$\{y[0], y'[0]\} = \{1.2, 0.2\}$ to $\{y[0], y'[0]\} = \{\frac{\pi}{2}, 0.2\}$:

```
{starterangle, startervel} = {π/2, 0.2};
endtime = 15;
Clear[t, y, yreal, ylinear];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
simplething = NDSolve[{linearversion,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
ylinear[t_] = y[t] /. simplething[[1]];
Plot[{yreal[t], ylinear[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
 PlotLegend → {"real", "linear"}, LegendSize → 0.6];
```



The formula for the solution of the linearized version of this damped pendulum oscillator is:

```
Clear[bb, MM, gg, LL, ss, vv, linearformula]
clearlinversion = y''[t] + bb y'[t]/MM + gg y[t]/LL == 0;
linearsol =
 DSolve[{clearlinversion, y[0] == ss, y'[0] == vv}, y[t], t] /.
  {ss → starterangle, vv → startervel, bb → b, MM → M, gg → g, LL → L};
linearformula[t_] = Chop[ComplexExpand[y[t] /. linearsol[[1]]]]
```

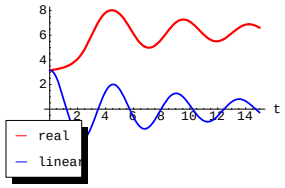
$1.5708\,E^{-0.1\,t}\,\text{Cos}[1.39642\,t] + 0.25571\,E^{-0.1\,t}\,\text{Sin}[1.39642\,t]$

Do you trust this formula to model the actual damped pendulum oscillator?

□G.4.c.v)

The next plot raises the starter data from
$\{y[0], y'[0]\} = \{\frac{\pi}{2}, 0.2\}$ to $\{y[0], y'[0]\} = \{\pi, 0.2\}$:

```
{starterangle, startervel} = {π, 0.2};
endtime = 15;
Clear[t, y, yreal, ylinear];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
simplething = NDSolve[{linearversion,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
ylinear[t_] = y[t] /. simplething[[1]];
Plot[{yreal[t], ylinear[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.01], Red}, {Thickness[0.008], Blue}},
 AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
 PlotLegend → {"real", "linear"}, LegendSize → 0.6];
```



The formula for the solution of the linearized version of this damped pendulum oscillator is:

```
Clear[bb, MM, gg, LL, ss, vv, linearformula]
clearlinversion = y''[t] + bb y'[t]/MM + gg y[t]/LL == 0;
linearsol =
 DSolve[{clearlinversion, y[0] == ss, y'[0] == vv}, y[t], t] /.
  {ss → starterangle, vv → startervel, bb → b, MM → M, gg → g, LL → L};
linearformula[t_] = Chop[ComplexExpand[y[t] /. linearsol[[1]]]]
```

$3.14159\,E^{-0.1\,t}\,\text{Cos}[1.39642\,t] + 0.368197\,E^{-0.1\,t}\,\text{Sin}[1.39642\,t]$

Do you trust this formula to model the actual damped pendulum oscillator?

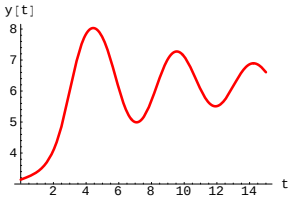
□G.4.c.vi)

Even though in all the above plots the starter velocity was kept the same, the quality of the formulas deteriorated as the starter angle was raised. Why is this no surprise?

□G.4.c.vii)

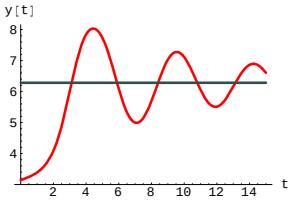
Look again at part of that last plot of the nonlinear pendulum oscillator:

```
last =
 Plot[yreal[t], {t, 0, endtime}, PlotStyle → {{Thickness[0.01], Red}},
  AxesLabel → {"t", "y[t]"}, AspectRatio → 1/GoldenRatio];
```



Throw in the horizontal line y[t] = 2 π:

```
extraline = Graphics[
  {DarkSlateGray, Thickness[0.01], Line[{{0, 2 π}, {endtime, 2 π}}]}];
Show[last, extraline];
```



Explain what information is conveyed by the plot.
Did the pendulum go over the top? If so, then what happened after it did? Does it just go around, around and around?
In your opinion, why is this damped pendulum oscillator so interested in the line

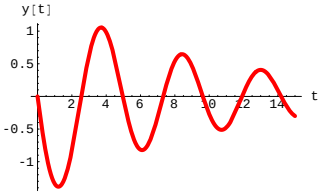
y[t] = 2 π ?

□G.d.i)

Check out what happens when you start the pendulum above at starting angle y[0] = 0, but you give it a whack to the left by starting with y′[0] = −2:

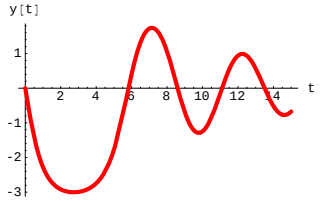
```
{starterangle, startervel} = {0, -2};
{b, L, g, M} = {0.2, 5.0, 9.8, 1.0};
Clear[t, y]
dampedpendosc = y''[t] + b y'[t] / M + g Sin[y[t]] / L == 0
Clear[t, y, yreal, ysimple];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
Plot[yreal[t], {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}}, PlotRange → All,
 AxesLabel → {"t", "y[t]"}, AspectRatio → 1 / GoldenRatio];
```

$$1.96 \sin[y[t]] + 0.2\, y'[t] + y''[t] == 0$$



Check out what happens when you start the pendulum above at starting angle y[0] = 0, but you give it a heck of a whack to the left by starting with y′[0] = −3:

```
{starterangle, startervel} = {0, -3.2};
Clear[t, y, yreal, ysimple];
realthing = NDSolve[{dampedpendosc,
   y[0] == starterangle, y'[0] == startervel}, y[t], {t, 0, endtime}];
yreal[t_] = y[t] /. realthing[[1]];
Plot[yreal[t], {t, 0, endtime},
 PlotStyle → {{Thickness[0.015], Red}}, PlotRange → All,
 AxesLabel → {"t", "y[t]"}, AspectRatio → 1 / GoldenRatio];
```



Can you give this damped pendulum a starting whack so hard the damped pendulum oscillators goes over the top?
If you can, then show off your ability with a decisive plot.

G.5) Frequency of the undamped pendulum

□G.5.a)

Here are two undamped pendulums.

```
Clear[y, t]
y[t] = -π/6;
Show[GraphicsArray[{pendulum[10, y[t]], pendulum[5, y[t]]}]];
```

Given starter data on y[0] and y′[0], the motion of the shorter pendulum is modeled by:

```
g = 9.8;
L = 5;
Clear[y, t]
shortpendulumoscillator = y''[t] + g Sin[y[t]] / L == 0
```

$$1.96 \sin[y[t]] + y''[t] == 0$$

Given starter data on y[0] and y′[0], the motion of the longer pendulum is modeled by:

```
L = 10;
longpendulumoscillator = y''[t] + g Sin[y[t]] / L == 0
```

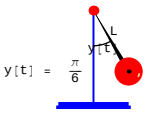
$$0.98 \sin[y[t]] + y''[t] == 0$$

If you go with the same starting data on y[0] and y′[0], then which of the two pendulums has the higher frequency (= shorter period)?
Show off your answer with convincing plots or movies.

□G.5.b)

Here's a single undamped pendulum:

```
Clear[y, t, s]
L = 10;
y[t] = π/6;
Clear[s]
angle = ParametricPlot[
  0.5 L {Sin[s], -Cos[s]}, {s, 0, y[t]}, DisplayFunction → Identity];
labels = Graphics[{Text["y[t]", 1/10 5.5 L {Sin[y[t]/2], -Cos[y[t]/2]}],
   Text["L", 0.4 L {Sin[y[t]], -Cos[y[t]]} + {Cos[y[t]], Sin[y[t]]}]};
Show[calibratedpendulum[L, y[t]], angle, labels,
 DisplayFunction → $DisplayFunction];
```



Given starter data on y[0] and y′[0], the motion of this pendulum is modeled by:

```
g = 9.8;
Clear[y, t]
pendulumoscillator = y''[t] + g Sin[y[t]] / L == 0
```

$$0.98 \sin[y[t]] + y''[t] == 0$$

If you go with a small starting angle y[0] and y′[0] = 0, then do you expect the pendulum to wiggle back and forth faster or slower than if you start the pendulum off with a larger y[0] and y′[0] = 0?
Back up your answer with convincing plots or movies.

G.6) The Van der Pol oscillator

□G.6.a.i)

You can get an undamped ordinary oscillator this way:

```
Clear[y, t, b]
y''[t] + b y'[t] + y[t] == 0 /. b → 0
```

$$y[t] + y''[t] == 0$$

You can get a damped oscillator by making b positive and small:

```
Clear[y, t, b]
y''[t] + b y'[t] + y[t] == 0 /. b → 0.3
```

```
y[t] + 0.3 y'[t] + y''[t] == 0
```

You can get a negatively damped (self-excited) oscillator by making b negative and small:

```
Clear[y, t, b]
y''[t] + b y'[t] + y[t] == 0 /. b → -0.23
```

```
y[t] - 0.23 y'[t] + y''[t] == 0
```

Here's how you get a van der Pol oscillator:

```
Clear[t, y]
vanderPoloscillator = y''[t] + (y[t]^2 - 1) y'[t] + y[t] == 0
```

```
y[t] + (-1 + y[t]^2) y'[t] + y''[t] == 0
```

To make the van der Pol oscillator into a system of two first order differential equations, you put

$$x[t] = y'[t]$$

and then replace y′[t] by x[t] and replace y″[t] by x′[t]:

```
Clear[x]
ColumnForm[Thread[{new = (y'[t] == x[t]),
vanderPoloscillator /. {y'[t] -> x[t], y''[t] -> x'[t]}}] ]
```

```
y'[t] == x[t]
y[t] + x[t] (-1 + y[t]^2) + x'[t] == 0
```

```
Clear[m, n, x, y, t]
m[x_, y_] = -(y^2 - 1) x - y;
n[x_, y_] = x;
vanderPolsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[vanderPolsystem]]
```

```
x'[t] == -y[t] + x[t] (1 - y[t]^2)
y'[t] == x[t]
```

The equilibrium points come from:

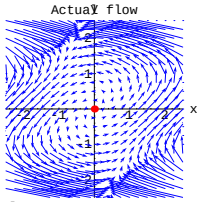
```
Solve[{m[x, y] == 0, n[x, y] == 0}, {x, y}]
```

```
{{x → 0, y → 0}}
```

The friendly point {0, 0} is an equilibrium point for the van der Pol oscillator. Here comes the a plot of the van der Pol flow near the equilibrium point:

```
{xequilib, yequilib} = {0, 0};
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]};
```

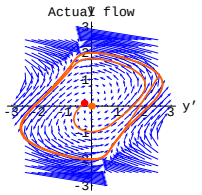
```
h = 2.5;
flowplot = Table[Arrow[Field[x, y], Tail → {x, y},
   VectorColor → Blue, ScaleFactor → 0.2, HeadSize → 0.15],
  {x, xequilib - h, xequilib + h, h/10},
  {y, yequilib - h, yequilib + h, h/10}];
actualflow =
 Show[flowplot, Axes → True, AxesOrigin → {xequilib, yequilib},
  AxesLabel → {"x", "y"}, PlotRange →
   {{xequilib - h, xequilib + h}, {yequilib - h, yequilib + h}},
  Epilog → {PointSize[0.04], Red, Point[{xequilib, yequilib}]},
  PlotLabel → "Actual flow"];
```



Throw in a trajectory starting at a random point near the equilibrium point:

```
h = 0.3;
s = Random[Real, {0, 2 π}];
{a, b} = {xequilib, yequilib} + h {Cos[s], Sin[s]}
starterpoint = {a, b};
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t]];
equationy = y'[t] == n[x[t], y[t]];
starterx = x[0] == a;
startery = y[0] == b;
endtime = 24.0;
ndssol = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] = {x[t] /. ndssol[[1]], y[t] /. ndssol[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.01]}}, DisplayFunction → Identity];
Show[actualflow, trajectoryplot, PlotRange → All, Axes → True,
 AxesLabel → {"y'", "y"}, DisplayFunction → $DisplayFunction,
 Epilog → {{PointSize[0.04], Red, Point[starterpoint]},
   {PointSize[0.04], CadmiumOrange, Point[{xequilib, yequilib}]}}];
```

```
{-0.278058, 0.112621}
```



Rerun a couple of times.

Initially, the trajectory is propelled away from the equilibrium point. Run a few more if you like.

The overwhelming visual evidence is that the equilibriium point {0, 0} is a repellor for the van der Pol oscillator system.

Confirm this visual evidence by calculating the eigenvalues of the linearization matrix at {0, 0}.

□G.6.a.ii)

Show a plot of the linearized field with the linearized field vectors plotted with their tails at the same points the actual field vectors above are plotted.

→ Comment on the quality of the approximation of the actual flow by the flow of the linearization near the equilibrium point.

→ Why did the linearization have absolutely no chance of predicting the limit cycle which is so vivid in trajectories in the van der Pol flow?

→ Some of the formulas coming from the linearization of the pendulum oscillator can be used to predict long term behavior of trajectories for the pendulum oscillator.

In contrast, only clueless, inexperienced persons would ever use formulas coming from the linearization of the van der Pol oscillator to try to predict long-term behavior of the van der Pol oscillator. Why?

G.7) Gradient systems and Hamiltonian systems: Max-Min and level curves

□G.7.a.i) Gradient systems

Given a function f[x, y], you get the gradient of f[x, y] by going with

$$\text{gradf}[x, y] = \{\partial_x f[x, y], \partial_y f[x, y]\}.$$

This is the same as saying

$$\nabla f[x, y] = \{\partial f[x, y]/\partial x, \partial f[x, y]/\partial y\}.$$

If you're running this lesson from some versions of Windows, the funny characters above are partial derivative symbols.

To get the gradient system coming from f[x, y], you put

$$\{x'[t], y'[t]\} = \text{gradf}[x[t], y[t]],$$

which is the same as putting

$$\{x'[t], y'[t]\} = \nabla f[x[t], y[t]].$$

Here's a the gradient system coming from

$$f[x, y] = x\,\text{Cos}[y] + y\,\text{Sin}[x]:$$

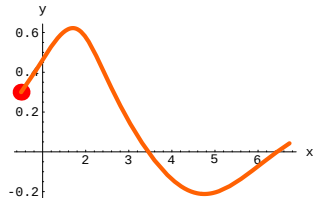
```
Clear[m, n, x, y, t, f, gradf]
f[x_, y_] = x Cos[y] + y Sin[x];
```

```
gradf[x_, y_] = {∂x f[x, y], ∂y f[x, y]};
{m[x_, y_], n[x_, y_]} = gradf[x, y];
gradsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[gradsystem]]
```

$x'[t] == \text{Cos}[y[t]] + \text{Cos}[x[t]]\, y[t]$
$y'[t] == \text{Sin}[x[t]] - \text{Sin}[y[t]]\, x[t]$

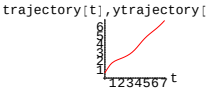
Here's a trajectory in this gradient field of f[x, y]:

```
{a, b} = {0.5, 0.3};
starterpoint = {a, b};
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t]];
equationy = y'[t] == n[x[t], y[t]];
starterx = x[0] == a;
startery = y[0] == b;
endtime = 7;
approxsolutions = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] =
 {x[t] /. approxsolutions[[1]], y[t] /. approxsolutions[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.06], Point[starterpoint]}];
Show[starterplot, trajectoryplot, PlotRange → All, Axes → True,
 AxesLabel → {"x", "y"}, DisplayFunction → $DisplayFunction,
 AspectRatio → 1/GoldenRatio];
```



Agree that {xtrajectory[t], ytrajectory[t]} specifies the point on this trajectory at time t.
Here's what happens to f[xtrajectory[t], ytrajectory[t]] as t advances from 0:

```
Clear[xtrajectory, ytrajectory];
{xtrajectory[t_], ytrajectory[t_]} =
 {x[t], y[t]} /. approxsolutions[[1]];
Plot[f[xtrajectory[t], ytrajectory[t]],
 {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.015]}},
 AxesLabel → {"t", "f[xtrajectory[t],ytrajectory[t]]"}];
```



Going up. Use your knowledge of the fact that when you plot gradf[x, y] with tail at {x, y}, the direction of gradf[x, y] is the direction of greatest initial increase in f[x, y] to explain why this curve had no choice but to go up.
Explain this bold statement:
No matter what function f[x, y] you go with, if
{xtrajectory[t], ytrajectory[t]}
parameterizes a trajectory in the gradient field of f[x, y], then you are guaranteed that f[xtrajectory[t], ytrajectory[t]] goes up as t goes up.

□G.7.a.ii)

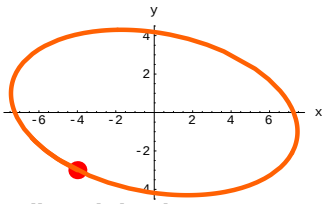
Here's a system:

```
Clear[m, n, x, y, t]
m[x_, y_] = 0.2 x + 1.5 y;
n[x_, y_] = -0.5 x - 0.2 y;
DEsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[DEsystem]]
```

$x'[t] == 0.2\, x[t] + 1.5\, y[t]$
$y'[t] == -0.5\, x[t] - 0.2\, y[t]$

And one of its trajectories:

```
{a, b} = {-4, -3};
starterpoint = {a, b};
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t]];
equationy = y'[t] == n[x[t], y[t]];
starterx = x[0] == a;
startery = y[0] == b;
endtime = 8;
ndssol = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] = {x[t] /. ndssol[[1]], y[t] /. ndssol[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.06], Point[starterpoint]}];
Show[starterplot, trajectoryplot, PlotRange → All, Axes → True,
 AxesLabel → {"x", "y"}, DisplayFunction → $DisplayFunction];
```



An elliptical closed curve.
How does the shape of this trajectory signal that the given system is not a gradient system?

□**Tip:**

Remember from part i) above:

No matter what function f[x, y] you go with, if

{xtrajectory[t], ytrajectory[t]}

parameterizes a trajectory in the gradient field of f[x, y], then you are guaranteed that

f[xtrajectory[t], ytrajectory[t]]

goes up as t goes up.

□G.7.b.i)

Here's the gradient of
$f[x, y] = x^3 + E^{3y} - 3\, x\, E^y$:

```
Clear[f, gradf, x, y]
f[x_, y_] = x^3 + E^(3 y) - 3 x E^y;
gradf[x_, y_] = {∂x f[x, y], ∂y f[x, y]}
```

$\{-3\, E^y + 3\, x^2,\ 3\, E^{3y} - 3\, E^y\, x\}$

Here's the corresponding gradient system:

```
Clear[m, n, x, y, t]
{m[x_, y_], n[x_, y_]} = gradf[x, y];
Gradientsystem = ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[Gradientsystem]]
```

$x'[t] == -3\, E^{y[t]} + 3\, x[t]^2$
$y'[t] == 3\, E^{3\, y[t]} - 3\, E^{y[t]}\, x[t]$

The field for this system is:

```
Clear[Field]
Field[x_, y_] = {m[x, y], n[x, y]}
```

$\{-3\, E^y + 3\, x^2,\ 3\, E^{3y} - 3\, E^y\, x\}$

This is the same as gradf[x, y].

```
gradf[x, y]
```

$\{-3\, E^y + 3\, x^2,\ 3\, E^{3y} - 3\, E^y\, x\}$

You go after the the equillibrium points by finding where
Field[x, y] = {0, 0}:

```
Chop[N[Solve[Field[x, y] == {0, 0}]]]
```

$\{\{y \to -0.5\, \text{Log}[\frac{1}{x}],\ \text{Log}[\frac{1}{x^2}] \to 0.5\, \text{Log}[\frac{1}{x}]\}\}$

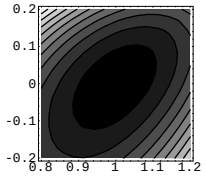
Remembering that Log[1] = 0, you can see that the equilibrium point is:

```
{xequilib, yequilib} = {1, 0}
```

{1, 0}

Look at the level curves of f[x, y] in the vicinity of the equilibrium point:

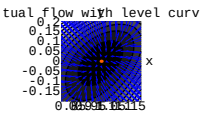
```
h = 0.2;
levelcurves = ContourPlot[Evaluate[f[x, y]],
  {x, xequilib - h, xequilib + h}, {y, yequilib - h, yequilib + h},
  ContourSmoothing → Automatic, AxesLabel → {"x", "y"}];
```



The lighter shading indicates larger values of f[x, y].

Throw in the flow of this gradient field:

```
flowplot = Table[Arrow[Field[x, y], Tail → {x, y},
    VectorColor → Blue, ScaleFactor → 0.1, HeadSize → 0.02],
   {x, xequilib - h, xequilib + h, h/8},
   {y, yequilib - h, yequilib + h, h/8}];
equilibplot = Graphics[
   {PointSize[0.04], CadmiumOrange, Point[{xequilib, yequilib}]}];
Show[levelcurves, flowplot, equilibplot, Axes → True,
 AxesOrigin → {xequilib, yequilib}, AxesLabel → {"x", "y"},
 PlotRange → {{xequilib - h, xequilib + h}, {yequilib - h, yequilib + h}},
 PlotLabel → "Actual flow with level curves"];
```



As you can see, the equilibrium point is a repellor for this gradient system.

And as you can see, the equilibrium point is probably a local minimizer in the sense that

f[xequilib, yequilib] < f[x, y]

for all other {x, y} near the equilibrium point.

Explain why you are certain that the equilibrium point is a local minimizer of f[x, y].

Go after candidates for maximizers and minimizers:

```
Chop[N[Solve[gradf[x, y] == {0, 0}]]]
```

```
{{x → -0.707107, y → -0.707107}, {x → 0.707107, y → 0.707107}}
```

This gives you two candidates. Calculate the linearization matrices for these gradient systems at each of these two points and do an eigenvalue analysis in effort to find whether any or all of these points are in fact local maxmizers or minimizers. Report your results.

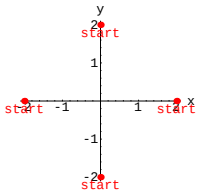
□G.7.c) Fishing for local maximizers

Just as you are working on gradients and max-min, an old retired prof (sort of a geezer) sits down at a computer next to you and says, "I don't use Solve commands or anything like that when I am looking for local max's in a certain region. Instead I just go fishing in the gradient flow. For instance if I am trying to find local max's near {0, 0} of

$$f[x, y] = \frac{E^{-0.3\,(x-0.4)^2}\,\mathrm{Sin}[x\,y+0.4]}{1+y^2},$$

I throw four corks in like this:

```
Clear[xstarter, ystarter, k]
h = 2;
{xstarter[1], ystarter[1]} = {h, 0};
{xstarter[2], ystarter[2]} = {0, h};
{xstarter[3], ystarter[3]} = {-h, 0};
{xstarter[4], ystarter[4]} = {0, -h};
starterplots = Table[Graphics[{Red, PointSize[0.04],
    Point[{xstarter[k], ystarter[k]}]}], {k, 1, 4}];
starterlabels = Table[Graphics[{Red,
    Text["start", {xstarter[k], ystarter[k]}, {0, 1}]}], {k, 1, 4}];
Show[starterplots, starterlabels, PlotRange → All, Axes → True,
 AxesLabel → {"x", "y"}];
```



"Then I enter the function and plot the trajectories in the gradient flow that start at the plotted points and then I see where they go:"

□G.7.b.ii)

Given a function f[x, y], call a point {xmin, ymin} a local minimizer of f[x, y] if

f[x, y] > f[xmin, ymin]

for all other {x, y} near {xmin, ymin}.

And call a point {xmax, ymax} a local maximizer of f[x, y] if

f[x, y] < f[xmax, ymax].

Explain why you don't need to look at any linearizations to know that local minimizers of f[x, y] are repellers and the local maximizers are attractors for the gradient system

{x′[t], y′[t]} = gradf[x[t], y[t]]

□G.7.b.iii)

On the other hand, if you go with a fixed function f[x, y], you look for candidates for local maximizers and minimizers by setting

gradf[x, y] = {0, 0}

and trying to solve for {x, y}.

Once you have a candidate $\{x_0, y_0\}$, how do you know that $\{x_0, y_0\}$ is guaranteed to be an equilibrium point of the gradient system

{x′[t], y′[t]} = gradf[x[t], y[t]]?

If you linearize this gradient system at the same point $\{x_0, y_0\}$ and your eigenvalue analysis reveals that $\{x_0, y_0\}$ is an attractor, then what's your reaction? If your eigenvalue analysis reveals that $\{x_0, y_0\}$ is an repellor, what's your reaction?

□G.7.b.iv)

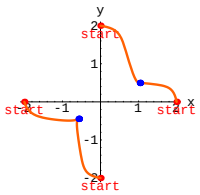
Now you have the opportunity to practice what you preach.

Go with this function:

```
Clear[f, gradf, x, y]
f[x_, y_] = (x + y)/(1 + x^2 + y^2);
gradf[x_, y_] = {∂x f[x, y], ∂y f[x, y]}
```

$$\left\{-\frac{2\,x\,(x+y)}{(1+x^2+y^2)^2}+\frac{1}{1+x^2+y^2},\ -\frac{2\,y\,(x+y)}{(1+x^2+y^2)^2}+\frac{1}{1+x^2+y^2}\right\}$$

```
Clear[f, gradf, m, n, x, y, t]
f[x_, y_] = E^(-0.3 (x-0.4)^2) Sin[x y + 0.4]/(1 + y^2);
gradf[x_, y_] = {∂x f[x, y], ∂y f[x, y]};
{m[x_, y_], n[x_, y_]} = gradf[x, y];
equationx = x′[t] == m[x[t], y[t]];
equationy = y′[t] == n[x[t], y[t]];
endtime = 25;
ndssol1 = NDSolve[{equationx, equationy, x[0] == h, y[0] == 0},
  {x[t], y[t]}, {t, 0, endtime}];
ndssol2 = NDSolve[{equationx, equationy, x[0] == 0, y[0] == h},
  {x[t], y[t]}, {t, 0, endtime}];
ndssol3 = NDSolve[{equationx, equationy, x[0] == -h, y[0] == 0},
  {x[t], y[t]}, {t, 0, endtime}];
ndssol4 = NDSolve[{equationx, equationy, x[0] == 0, y[0] == -h},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory1, trajectory2, trajectory3, trajectory4]
trajectory1[t_] = {x[t], y[t]} /. ndssol1[[1]];
trajectory2[t_] = {x[t], y[t]} /. ndssol2[[1]];
trajectory3[t_] = {x[t], y[t]} /. ndssol3[[1]]; trajectory4[t_] =
 {x[t], y[t]} /. ndssol4[[1]]; trajectoryplots = ParametricPlot[
  {trajectory1[t], trajectory2[t], trajectory3[t], trajectory4[t]},
  {t, 0, endtime}, PlotStyle → {{CadmiumOrange, Thickness[0.015]}},
  DisplayFunction → Identity]; endplots =
 {Graphics[{Blue, PointSize[0.04], Point[trajectory1[endtime]]}],
  Graphics[{Blue, PointSize[0.04], Point[trajectory2[endtime]]}],
  Graphics[{Blue, PointSize[0.04], Point[trajectory3[endtime]]}],
  Graphics[{Blue, PointSize[0.04], Point[trajectory4[endtime]]}]};
Show[starterplots, starterlabels, trajectoryplots,
 endplots, PlotRange → All, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```



The old guy goes on to say,"Those blue dots at trajectory1[endtime], trajectory2[endtime], trajectory3[endtime] and trajectory4[endtime] are the points at which the trajectories stalled. So, goldurn it, I'm pretty well convinced that that these are local maximizers.

Is this guy still with it?
Does his idea have any merit?.
If so, explain why.

◻G.7.d.i) Hamiltonian systems

Given a function f[x, y], you get the gradient system coming from f[x, y] by going with

m[x, y] = ∂_x f[x, y] (which is the same as ∂f[x, y]/∂x)

n[x, y] = ∂_y f[x, y] (which is the same as ∂f[x, y]/∂y)

```
If you're running this lesson from Windows, the funny characters above
                are partial derivative symbols.
```

and putting

{x′[t], y′[t]} = {m[x[t], y[t]], n[x[t], y[t]]}.

Given a function f[x, y], you get the Hamiltonian system coming from f[x, y] by going with

m[x, y] = $-\partial_y$ f[x, y] (which is the same as $-\partial$f[x, y]/∂y)

n[x, y] = ∂_x f[x, y] (which is the same as ∂f[x, y]/∂x)

and putting

{x′[t], y′[t]} = {m[x[t], y[t]], n[x[t], y[t]]}.

Here's the Hamiltonian system coming from

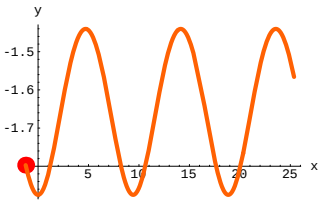
f[x, y] = $y^2 - y \sin[\frac{x}{3}]^2$:

```
Clear[m, n, x, y, t, f]
f[x_, y_] = y^2 - y Sin[x/3]^2;

{m[x_, y_], n[x_, y_]} = {-∂yf[x, y], ∂xf[x, y]};
hamiltoniansystem =
 ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[hamiltoniansystem]]
```

```
x′[t] == Sin[x[t]/3]^2 - 2 y[t]
y′[t] == -(2/3) Cos[x[t]/3] Sin[x[t]/3] y[t]
```

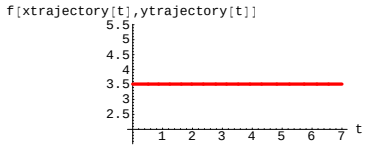
Here's a random trajectory in this Hamiltonian flow:

```
{a, b} = {Random[Real, {-2, 2}], Random[Real, {-2, 2}]};
starterpoint = {a, b};
Clear[x, y, t]
equationx = x′[t] == m[x[t], y[t]];
equationy = y′[t] == n[x[t], y[t]];
starterx = x[0] == a;
startery = y[0] == b;
endtime = 7;
ndssol = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] = {x[t] /. ndssol[[1]], y[t] /. ndssol[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.06], Point[starterpoint]}];
Show[starterplot, trajectoryplot, PlotRange → All, Axes → True,
 AxesLabel → {"x", "y"}, DisplayFunction → $DisplayFunction,
 AspectRatio → 1/GoldenRatio];
```



Agree that {xtrajectory[t], ytrajectory[t]} specifies the point on this trajectory at time t.
Here's what happens to f[xtrajectory[t], ytrajectory[t]]:

```
Clear[xtrajectory, ytrajectory];
{xtrajectory[t_], ytrajectory[t_]} = {x[t], y[t]} /. ndssol[[1]];
Plot[f[xtrajectory[t], ytrajectory[t]],
 {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.015]}},
 AxesLabel → {"t", "f[xtrajectory[t],ytrajectory[t]]"},
 PlotRange → {f[a, b] - 2, f[a, b] + 2}];
```



Got any idea why that happened?

◻G.7.d.ii)

Here's the Hamiltonian system coming from

f[x, y] = $x^2 - 5 \sin[x] y + 4 y^2$:

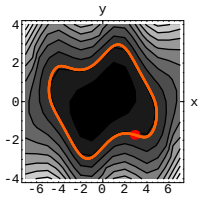
```
Clear[f, gradf, m, n, x, y]
f[x_, y_] = x^2 - 5 Sin[x] y + 4 y^2;
{m[x_, y_], n[x_, y_]} = {-∂yf[x, y], ∂xf[x, y]};
hamiltoniansystem =
 ({x'[t], y'[t]} == {m[x[t], y[t]], n[x[t], y[t]]});
ColumnForm[Thread[hamiltoniansystem]]
```

```
x′[t] == 5 Sin[x[t]] - 8 y[t]
y′[t] == 2 x[t] - 5 Cos[x[t]] y[t]
```

Here's a trajectory for this system shown with a contour plot of some of the level curves of f[x, y]:

```
{a, b} = {Random[Real, {-5, 5}], Random[Real, {-3, 3}]};
starterpoint = {a, b};
Clear[x, y, t]
equationx = x′[t] == m[x[t], y[t]];
equationy = y′[t] == n[x[t], y[t]];
starterx = x[0] == a;
startery = y[0] == b;
endtime = 5;
ndssol = NDSolve[{equationx, equationy, starterx, startery},
  {x[t], y[t]}, {t, 0, endtime}];
Clear[trajectory]
trajectory[t_] = {x[t] /. ndssol[[1]], y[t] /. ndssol[[1]]};
trajectoryplot =
 ParametricPlot[trajectory[t], {t, 0, endtime}, PlotStyle →
   {{CadmiumOrange, Thickness[0.015]}}, DisplayFunction → Identity];
starterplot = Graphics[{Red, PointSize[0.06], Point[starterpoint]}];
levelcurves = ContourPlot[Evaluate[f[x, y]], {x, -7, 7}, {y, -4, 4},
  ContourSmoothing → Automatic, DisplayFunction → Identity];
```

```
Show[levelcurves, starterplot, trajectoryplot,
 PlotRange → All, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```



Rerun several times

Give your opinion about why each plot turns out the way it does.

◻G.7.d.iii)

Go with:

```
Clear[f, x, y]
f[x_, y_] = -2.3 Sin[y] + 0.9 Cos[3.0 x]
```

```
0.9 Cos[3. x] - 2.3 Sin[y]
```

When you set

f[x, y] = f[0, 0]

and vary x, then y also has to vary to maintain the equality

f[x, y] = f[0, 0].

So, when you set

f[x, y] = f[0, 0],

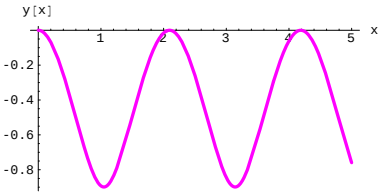
you are implicitly defining y in terms of of x.
Mathematica can give you the formula for this function:

```
sols = Solve[f[x, y] == f[0, 0], y];
y[x_] = N[y /. sols[[1]], 6]
```

```
1. ArcSin[0.0434783 (-9. + 9. Cos[3. x])]
```

And you can plot at least part of y[x] as a function of x:

```
ParametricPlot[{x, y[x]}, {x, 0, 5},
 PlotStyle → {{Magenta, Thickness[0.01]}}, AxesLabel → {"x", "y[x]"},
 AspectRatio → 1/2];
```

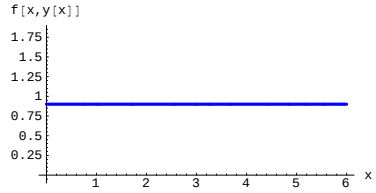


At every point {x, y} on this curve, you are guaranteed that
f[x, y] = f[0, 0]:

```
x = Random[Real, {0, 5}];
f[x, y[x]] == f[0, 0]
```

```
True
```

```
Plot[f[x, y[x]], {x, 0, 6}, PlotStyle → {{Blue, Thickness[0.01]}},
 AxesLabel → {"x", "f[x,y[x]]"}, AspectRatio → 1/2,
 PlotRange → {f[0, 0] - 1, f[0, 0] + 1}];
```



Now go with:

```
Clear[f, x, y]
f[x_, y_] = -2.3 Sin[y] + E^Cos[y] Cos[3.0 x]
```

```
E^Cos[y] Cos[3. x] - 2.3 Sin[y]
```

When you set
f[x, y] = f[0, 0],
you are implicitly defining y in terms of x. But when you go to *Mathematica* for a formula for y[x], you get:

```
sols = Solve[f[x, y] == f[0, 0], y]
```

```
Solve[E^Cos[y] Cos[3. x] - 2.3 Sin[y] == E, y]
```

No dice. No formula is available.
In spite of this apparent setback, you can use a trajectory plot in an appropriate Hamiltonian system to come up with a nice plot of y in terms of x through the implicit definition
f[x, y] = f[0, 0].
Do it.

G.8) Chaos: Lorenz's chaotic 3D oscillator

```
This problem is a memory hog.
If you want to work on this one, it might be a good idea
to copy and paste it into a new notebook.
In fact it might be a good idea to use a very fast computer with lots of
memory
```

From E. Atlee Jackson's book "Perspectives of Nonlinear Dynamics" (Cambridge, 1991):
"In 1963, E. N. Lorenz published a study of a highly simplified model of a particular hydrodynamic flow [involving long-range weather forecasting] problem, which has become a classic in the area of nonlinear dynamics. The importance of this model is . . . that it illustrates how a simple model can produce very rich and varied forms of dynamics."
Folks like to call this "rich and varied behavior" by the name "chaos."

DiffEq&*Mathematica* invites you to experience chaos first hand.

□G.8.a.i)

Lorenz's chaotic diffeq system is an innocent looking 3 D nonlinear system.
Here it is:

```
Clear[m, n, p, x, y, z, t]
s = 10.0;
b = 2.6667;
r = 28.0;
m[x_, y_, z_] = -s (x - y);
n[x_, y_, z_] = r x - y - x z;
p[x_, y_, z_] = -b z + x y;
Lorenzsystem =
({x'[t], y'[t], z'[t]} ==
{m[x[t], y[t], z[t]], n[x[t], y[t], z[t]], p[x[t], y[t], z[t]]});
ColumnForm[Thread[Lorenzsystem]]
```

```
x'[t] == -10. (x[t] - y[t])
y'[t] == 28. x[t] - y[t] - x[t] z[t]
z'[t] == x[t] y[t] - 2.6667 z[t]
```

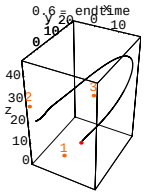
The equilibrium points are:

```
equisols =
 Solve[{m[x, y, z] == 0, n[x, y, z] == 0, p[x, y, z] == 0}, {x, y, z}];
Clear[xeq, yeq, zeq]
{xeq[1], yeq[1], zeq[1]} = {x, y, z} /. equisols[[1]]
{xeq[2], yeq[2], zeq[2]} = {x, y, z} /. equisols[[2]]
{xeq[3], yeq[3], zeq[3]} = {x, y, z} /. equisols[[3]]
```

```
{0., 0., 0.}
{-8.48533, -8.48533, 27.}
{8.48533, 8.48533, 27.}
```

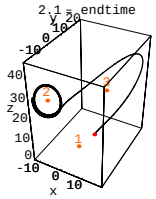
See a trajectory starting at {5, 5, 5} together with the three equilbrium points:

```
{xstarter, ystarter, zstarter} = {5, 5, 5};
endtime = 0.6;
Clear[x, y, z, k, t]
equationx = (x'[t] == m[x[t], y[t], z[t]]);
equationy = (y'[t] == n[x[t], y[t], z[t]]);
equationz = (z'[t] == p[x[t], y[t], z[t]]);
xstartereqn = (x[0] == xstarter);
ystartereqn = (y[0] == ystarter);
zstartereqn = (z[0] == zstarter);


nonlinsols = NDSolve[{equationx, equationy, equationz,
  xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime},
  MaxSteps -> 2000];

trajectoryplot =
  ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
       {t, 0, endtime},
  PlotPoints -> 500, DisplayFunction -> Identity];
starterplot = Graphics3D[
  {Red, PointSize[0.03], Point[{xstarter, ystarter, zstarter}]}];
equilibplots =
Table[Graphics3D[{CadmiumOrange, PointSize[0.03],
    Point[{xeq[k], yeq[k], zeq[k]}]}], {k, 1, 3}];
h = 4;
equilabels = {Graphics3D[
   {CadmiumOrange, Text["1", {xeq[1], yeq[1], zeq[1] + h}]}],
      Graphics3D[
   {CadmiumOrange, Text["2", {xeq[2], yeq[2], zeq[2] + h}]}],
      Graphics3D[
   {CadmiumOrange, Text["3", {xeq[3], yeq[3], zeq[3] + h}]}]};


Show[trajectoryplot, starterplot, equilibplots,
 equilabels, Axes -> True, AxesLabel -> {"x", "y", "z"},
  PlotRange -> All, PlotLabel -> endtime "= endtime",
 DisplayFunction -> $DisplayFunction];
```



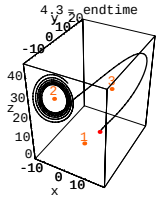
So far so good.
See more of the trajectory:

```
endtime = 2.1;

nonlinsols = NDSolve[{equationx, equationy, equationz,
  xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime},
  MaxSteps -> 2000];

trajectoryplot =
  ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
       {t, 0, endtime},
  PlotPoints -> 500, DisplayFunction -> Identity];

Show[trajectoryplot, starterplot, equilibplots,
 equilabels, Axes -> True, AxesLabel -> {"x", "y", "z"},
  PlotRange -> All, PlotLabel -> endtime "= endtime",
 DisplayFunction -> $DisplayFunction];
```



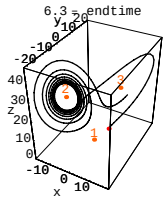
Uh-oh.
See more:

```
endtime = 4.3;

nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 2000];
trajectoryplot = ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
  {t, 0, endtime}, PlotPoints → 500, DisplayFunction → Identity];
Show[trajectoryplot, starterplot, equilibplots, equilabels,
 Axes → True, AxesLabel → {"x", "y", "z"}, PlotRange → All,
 PlotLabel → endtime "= endtime", DisplayFunction → $DisplayFunction];
```



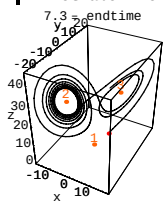
Seems to be settling into a predictable pattern.
See more:

```
endtime = 6.3;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 2000];
trajectoryplot = ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
  {t, 0, endtime}, PlotPoints → 500, DisplayFunction → Identity];
Show[trajectoryplot, starterplot, equilibplots, equilabels,
 Axes → True, AxesLabel → {"x", "y", "z"}, PlotRange → All,
 PlotLabel → endtime "= endtime", DisplayFunction → $DisplayFunction];
```



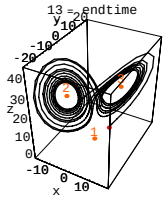
Whoops. Someting attracted its interest.
See more:

```
endtime = 7.3;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 2000];
trajectoryplot = ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
  {t, 0, endtime}, PlotPoints → 500, DisplayFunction → Identity];
Show[trajectoryplot, starterplot, equilibplots, equilabels,
 Axes → True, AxesLabel → {"x", "y", "z"}, PlotRange → All,
 PlotLabel → endtime "= endtime", DisplayFunction → $DisplayFunction];
```



Maybe it's settling into another predictable pattern.
See a lot more:

```
endtime = 13;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 2000];
trajectoryplot = ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
  {t, 0, endtime}, PlotPoints → 500, DisplayFunction → Identity];
last = Show[trajectoryplot, starterplot, equilibplots,
  equilabels, Axes → True, AxesLabel → {"x", "y", "z"},
  PlotRange → All, PlotLabel → endtime "= endtime",
  DisplayFunction → $DisplayFunction];
```



Describe what you see.

□G.8.a.ii)

The linearization (Jacobian) matrix for the Lorenz system at a point {x, y, z} is:

```
Clear[A, gradm, gradn, gradp, x, y, z]
gradm[x_, y_, z_] = {∂x m[x, y, z], ∂y m[x, y, z], ∂z m[x, y, z]};
gradn[x_, y_, z_] = {∂x n[x, y, z], ∂y n[x, y, z], ∂z n[x, y, z]};
gradp[x_, y_, z_] = {∂x p[x, y, z], ∂y p[x, y, z], ∂z p[x, y, z]};
A[x_, y_, z_] = {gradm[x, y, z], gradn[x, y, z], gradp[x, y, z]};
MatrixForm[A[x, y, z]]
```

$$\begin{pmatrix} -10. & 10. & 0 \\ 28. - z & -1 & -x \\ y & x & -2.6667 \end{pmatrix}$$

The equilibrium points are:

```
ColumnForm[Table[{xeq[k], yeq[k], zeq[k]}, {k, 1, 3}]]
```

{0., 0., 0.}
{-8.48533, -8.48533, 27.}
{8.48533, 8.48533, 27.}

The corresponding eigenvalues of the linearization matrix at each of the three equilibrium points are:

```
ColumnForm[Table[Eigenvalues[A[xeq[k], yeq[k], zeq[k]]], {k, 1, 3}]]
```

{-22.8277, 11.8277, -2.6667}
{-13.8546, 0.0939504 + 10.1946 I, 0.0939504 - 10.1946 I}
{-13.8546, 0.0939504 + 10.1946 I, 0.0939504 - 10.1946 I}

Does this information begin to give you even the slightest clue about why the trajectory plots out they way it does?
If so, then what is the clue?

□G.8.a.iii)

To add to the investigation, look at this:

```
N[Eigenvalues[A[xeq[2], yeq[2], zeq[2]]], 3]
Chop[N[Eigenvectors[A[xeq[2], yeq[2], zeq[2]]], 3]]
```

{-13.9, 0.094 + 10.2 I, 0.094 - 10.2 I}
{{0.856, -0.33, 0.399}, {-0.266 - 0.295 I, 0.0321 - 0.569 I, 0.719},
 {-0.266 + 0.295 I, 0.0321 + 0.569 I, 0.719}}

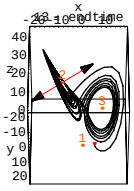
At {xeq[2], yeq[2], zeq[2]}, you spot a big time sucking eigenvector:

```
sucker2 = {0.856, -0.33, 0.399}
```

{0.856, -0.33, 0.399}

See the last trajectory and this sucking eigenvector from a viewpoint perpendicular to this sucking eigenvector:

```
sucker2plot = {Arrow[sucker2, Tail → {xeq[2], yeq[2], zeq[2]},
    VectorColor → Red, ScaleFactor → 15, HeadSize → 5],
  Arrow[-sucker2, Tail → {xeq[2], yeq[2], zeq[2]},
    VectorColor → Red, ScaleFactor → 15, HeadSize → 5]};
Show[last, sucker2plot, ViewPoint → 30 {1, 0, 0} × sucker2];
```



Now move to {xeq[3], yeq[3], zeq[3]}

```
N[Eigenvalues[A[xeq[3], yeq[3], zeq[3]]], 3]

Chop[N[Eigenvectors[A[xeq[3], yeq[3], zeq[3]]], 3]]
```

{-13.9, 0.094 + 10.2 I, 0.094 - 10.2 I}
{{0.856, -0.33, -0.399}, {0.266 + 0.295 I, -0.0321 + 0.569 I, 0.719},
 {0.266 - 0.295 I, -0.0321 - 0.569 I, 0.719}}

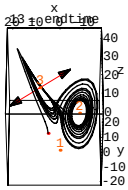
At {xeq[3], yeq[3], zeq[3]}, you spot another big time sucking eigenvector:

```
sucker3 = {0.856, -0.33, -0.399}
```

{0.856, -0.33, -0.399}

See the last trajectory and this sucking eigenvector from a viewpoint perpendicular to this sucking eigenvector:

```
sucker3plot = {Arrow[sucker3, Tail → {xeq[3], yeq[3], zeq[3]},
    VectorColor → Red, ScaleFactor → 15, HeadSize → 5],
   Arrow[-sucker3, Tail → {xeq[3], yeq[3], zeq[3]},
    VectorColor → Red, ScaleFactor → 15, HeadSize → 5]};
Show[last, sucker3plot, ViewPoint → 30 {1, 0, 0} × sucker3];
```



What two planes try to grab the interest of the trajectory? Got any idea why?

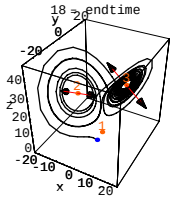
□G.8.a.iv)

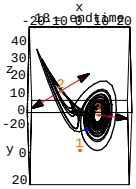
It might be a good idea to quit
Mathematica here and get a refreshed copy before going on.

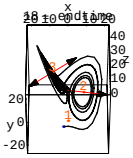
Here's what happens when you start at a random starting point:

```
endtime = 18;
{xstarter, ystarter, zstarter} = {Random[Real, {-10, 10}],
  Random[Real, {-10, 10}], Random[Real, {-10, 10}]};
Clear[m, n, p, x, y, z, t]
s = 10.0;
b = 2.6667;
r = 28.0;
m[x_, y_, z_] = -s (x - y);
n[x_, y_, z_] = r x - y - x z;
p[x_, y_, z_] = -b z + x y;
Clear[x, y, z, k, t]
equationx = x'[t] == m[x[t], y[t], z[t]];
equationy = y'[t] == n[x[t], y[t], z[t]];
equationz = z'[t] == p[x[t], y[t], z[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
zstartereqn = z[0] == zstarter;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 2000];
```

```
trajectoryplot = ParametricPlot3D[{x[t], y[t], z[t]} /. nonlinsols[[1]],
  {t, 0, endtime}, PlotPoints → 700, DisplayFunction → Identity];
starterplot = Graphics3D[
  {Blue, PointSize[0.03], Point[{xstarter, ystarter, zstarter}]}];
equisols =
 Solve[{m[x, y, z] == 0, n[x, y, z] == 0, p[x, y, z] == 0}, {x, y, z}];
Clear[xeq, yeq, zeq]
{xeq[1], yeq[1], zeq[1]} = {x, y, z} /. equisols[[1]];
{xeq[2], yeq[2], zeq[2]} = {x, y, z} /. equisols[[2]];
{xeq[3], yeq[3], zeq[3]} = {x, y, z} /. equisols[[3]];
equilibplots = Table[Graphics3D[{CadmiumOrange,
    PointSize[0.03], Point[{xeq[k], yeq[k], zeq[k]}]}], {k, 1, 3}];
h = 4;
equilabels = {Graphics3D[
   {CadmiumOrange, Text["1", {xeq[1], yeq[1], zeq[1] + h}]}],
  Graphics3D[{CadmiumOrange,
    Text["2", {xeq[2], yeq[2], zeq[2] + h}]}], Graphics3D[
   {CadmiumOrange, Text["3", {xeq[3], yeq[3], zeq[3] + h}]}]};
sucker2 = {0.856, -0.33, 0.399};
sucker3 = {0.856, -0.33, -0.399};
suckerplots = {Arrow[sucker2, Tail → {xeq[2], yeq[2], zeq[2]},
   VectorColor → Red, ScaleFactor → 15, HeadSize → 5],
  Arrow[-sucker2, Tail → {xeq[2], yeq[2], zeq[2]},
   VectorColor → Red, ScaleFactor → 15, HeadSize → 5],
  Arrow[sucker3, Tail → {xeq[3], yeq[3], zeq[3]},
   VectorColor → Red, ScaleFactor → 15, HeadSize → 5],
  Arrow[-sucker3, Tail → {xeq[3], yeq[3], zeq[3]},
   VectorColor → Red, ScaleFactor → 15, HeadSize → 5]};
outcome = Show[trajectoryplot, starterplot, equilibplots, equilabels,
  suckerplots, Axes → True, AxesLabel → {"x", "y", "z"},
  PlotRange → All, PlotLabel → endtime "= endtime",
  DisplayFunction → $DisplayFunction];
Show[outcome, ViewPoint → 30 {1, 0, 0} × sucker2];
Show[outcome, ViewPoint → 30 {1, 0, 0} × sucker3];
```







Rerun several times and report anything you observe.

□G.8.b.i)

Stay with the same chaotic system:

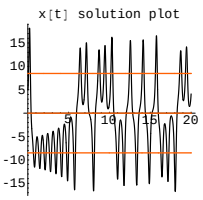
```
s = 10.0;
b = 2.6667;
r = 28.0;
Clear[m, n, x, y, t]
m[x_, y_, z_] = -s (x - y);
n[x_, y_, z_] = r x - y - x z;
p[x_, y_, z_] = -b z + x y;

Lorenzsystem =
({x'[t], y'[t], z'[t]} ==
 {m[x[t], y[t], z[t]], n[x[t], y[t], z[t]], p[x[t], y[t], z[t]]});

ColumnForm[Thread[Lorenzsystem]]
```

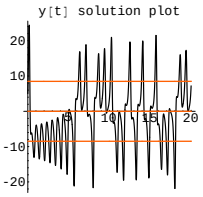
```
x'[t] == -10. (x[t] - y[t])
y'[t] == 28. x[t] - y[t] - x[t] z[t]
z'[t] == x[t] y[t] - 2.6667 z[t]
```

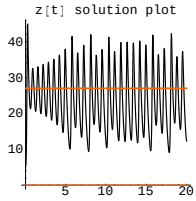
Look at the individual solution plots for t running from 0 to 20 with starter point {5, 5, 5}:

```
endtime = 20;
{xstarter, ystarter, zstarter} = {5, 5, 5};
```

```
Clear[x, y, t]
equationx = x'[t] == m[x[t], y[t], z[t]];
equationy = y'[t] == n[x[t], y[t], z[t]];
equationz = z'[t] == p[x[t], y[t], z[t]];
xstartereqn = x[0] == xstarter;
ystartereqn = y[0] == ystarter;
zstartereqn = z[0] == zstarter;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, xstartereqn, ystartereqn, zstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 3000];
xsolplot = Plot[{x[t] /. nonlinsols[[1]], xeq[1], xeq[2], xeq[3]},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.007]}, {CadmiumOrange},
    {CadmiumOrange}, {CadmiumOrange}}, PlotPoints → 500,
  AspectRatio → 1, PlotLabel → "x[t] solution plot"];
ysolplot = Plot[{y[t] /. nonlinsols[[1]], yeq[1], yeq[2], yeq[3]},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.007]}, {CadmiumOrange},
    {CadmiumOrange}, {CadmiumOrange}}, PlotPoints → 500,
  AspectRatio → 1, PlotLabel → "y[t] solution plot"];
zsolplot = Plot[{z[t] /. nonlinsols[[1]], zeq[1], zeq[2], zeq[3]},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.007]}, {CadmiumOrange},
    {CadmiumOrange}, {CadmiumOrange}}, PlotPoints → 500,
  AspectRatio → 1, PlotLabel → "z[t] solution plot"];
```







The horizontal lines correspond to the equilibrium values.

Describe what you see.
The result will be your own description of chaos.

□G.8.b.ii)

Now build in small random errors in the starter data:

```
endtime = 20;
h = 0.2;
{newxstarter, newystarter, newzstarter} = {5, 5, 5} +
  {Random[Real, {-h, h}], Random[Real, {-h, h}], Random[Real, {-h, h}]}
```

```
{4.80826, 5.18991, 5.06207}
```

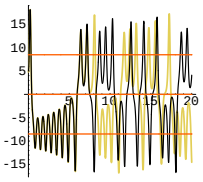
Compare:

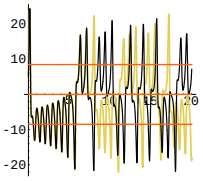
```
{xstarter, ystarter, zstarter}
```

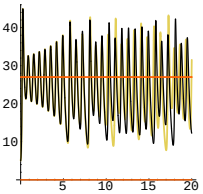
```
{5, 5, 5}
```

The two starter points are very close.
See the individual solution plots corresponding to the new starter points shown with the individual solution plots corresponding to the old starter points:

```
Clear[x, y, z, t]
equationx = x'[t] == m[x[t], y[t], z[t]];
equationy = y'[t] == n[x[t], y[t], z[t]];
equationz = z'[t] == p[x[t], y[t], z[t]];
newxstartereqn = x[0] == newxstarter;
newystartereqn = y[0] == newystarter;
newzstartereqn = z[0] == newzstarter;
nonlinsols = NDSolve[{equationx, equationy,
   equationz, newxstartereqn, newystartereqn, newzstartereqn},
  {x[t], y[t], z[t]}, {t, 0, endtime}, MaxSteps → 3000];
newxsolplot = Plot[x[t] /. nonlinsols[[1]],
  {t, 0, endtime}, PlotStyle → {{Banana, Thickness[0.01]}},
  PlotPoints → 500, DisplayFunction → Identity];
newysolplot = Plot[y[t] /. nonlinsols[[1]],
  {t, 0, endtime}, PlotStyle → {{Banana, Thickness[0.01]}},
  PlotPoints → 500, DisplayFunction → Identity];
newzsolplot = Plot[z[t] /. nonlinsols[[1]],
  {t, 0, endtime}, PlotStyle → {{Banana, Thickness[0.01]}},
  PlotPoints → 500, DisplayFunction → Identity];
Show[newxsolplot, xsolplot,
 AspectRatio → 1, DisplayFunction → $DisplayFunction];
Show[newysolplot, ysolplot,
 AspectRatio → 1, DisplayFunction → $DisplayFunction];
Show[newzsolplot, zsolplot, AspectRatio → 1,
 DisplayFunction → $DisplayFunction];
```







What you see are the individual solution plots corresponding to the new starter points shown with the individual solution plots corresponding to the old starter points.

Describe what you see - including a few words on the issue of extreme sensitivity to errors in starter data.