

Math 2280–1

Wednesday March 2, 2011

4.1, 4.3: First order systems of Differential Equations

Why you should expect existence and uniqueness for the IVP,
and how to approximate solutions numerically.

First: finish Tuesday's notes.

Example: Consider the initial value problem related to page 4 of Monday's notes, which we finished discussing yesterday

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} y \\ -1.01 x - 0.2 y \end{bmatrix}$$
$$\begin{bmatrix} x(0) \\ y(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Remember, this was a first order system we got by converting a second order DE for a certain damped spring oscillation problem,

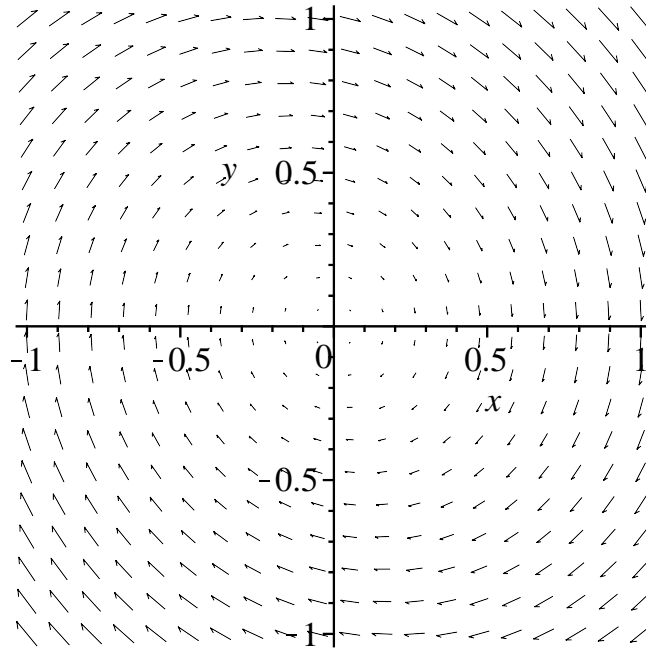
$$x''(t) + 0.2 x'(t) + 1.01 x(t) = 0$$
$$x(0) = 1$$
$$x'(0) = 0$$

so we can quickly deduce the solution to the system of first order DE's from the solution and its derivative, to the second order differential equation IVP. The system IVP solution is

$$\text{ans} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} e^{-0.1t} (\cos(t) + 0.1 \sin(t)) \\ -1.01 e^{-0.1t} \sin(t) \end{bmatrix}$$

But let's pretend we don't know the solution, and try to understand why we would expect one, and why we would expect it to be unique, just based on the geometric meaning of this IVP: If we plot the solution curve $[x(t), y(t)]$ in the plane, it passes through $[1, 0]$ at $t = 0$, and then moves so that its tangent vector $[x'(t), y'(t)]$ agrees with the DE vector field at the point $[x(t), y(t)]$, namely with $[y - 1.01 \cdot x, -0.2 \cdot y]$. So if we create a picture of the vector field we can visualize where the solution trajectory must start and then go:

```
> restart;with(DEtools):with(plots):with(LinearAlgebra):
> fieldplot([y,-1.01*x-.2*y],x=-1..1,y=-1..1);
```



We can use Euler's method to get an approximate numerical solution! In other words, approximate using the tangent line approximation, and for the tangent vector use what the differential equation says it is (those "equals" symbols are really "approximately equals" signs):

$$\begin{bmatrix} x(t + \Delta t) \\ y(t + \Delta t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} + \Delta t \begin{bmatrix} \frac{d}{dt} x(t) \\ \frac{d}{dt} y(t) \end{bmatrix}$$

$$\begin{bmatrix} x(t + \Delta t) \\ y(t + \Delta t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} + \Delta t \begin{bmatrix} y(t) \\ -1.01 x(t) - 0.2 y(t) \end{bmatrix}.$$

With the specified initial conditions we start at the point $[1, 0]$, and then follow the tangent, to get a curve which looks a lot like a spiral. If we want to get the appropriate "for loop" to get an Euler (or improved Euler or Runge–Kutta) numerical approximation for a first order system of DE's, we can just transpose Euler (or improved Euler or Runge–Kutta) for first order equations into code first order systems . . . all the original justifications make just as much sense in vector form!!! (Or, if we really need to do numerical work we can use one of Maple's or Matlab's numerical packages, which are usually some sort of Runge–Kutta scheme.)

```
> restart :
> with(LinearAlgebra) : with(plots) : with(DEtools) :
> f1:=(x,y,t)->y:
  f2:=(x,y,t)->-1.01*x-.2*y:
    #our right hand side F(X,t)=[f1(X,t),f2(X,t)]
    #in our first order system of DE's dX/dt = F(t,X).
> n:=100:
  #number of time steps
t0:=0: #initial time
tn:=evalf(2*Pi):
  #final time choice
h:=(tn-t0)/n:
  #time step
xs:=Vector(n+1):
ys:=Vector(n+1):  #vectors to hold Euler approximates
xs[1]:=1:
ys[1]:=0:        #initial conditions
```

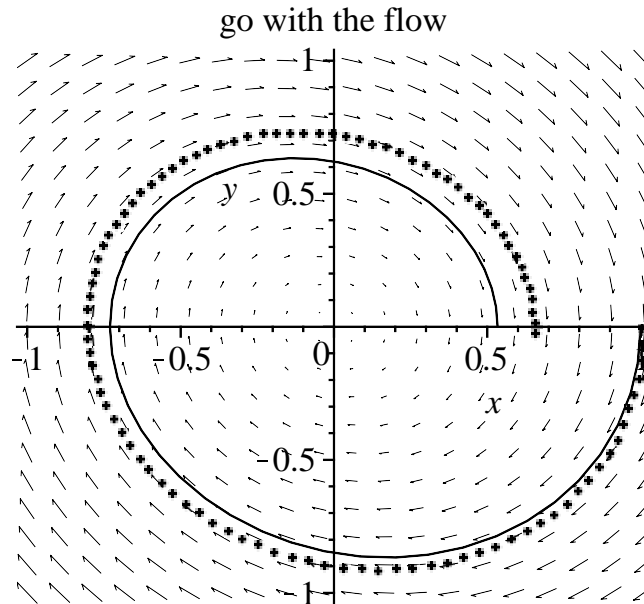
Euler loop

```
> #Euler loop:
  for i from 1 to n do
    X:=xs[i]:
    Y:=ys[i]:
    t:=t0+(i-1)*h:
    k1:=f1(X,Y,t):
    k2:=f2(X,Y,t):
    #k1 and k2 are velocity components
    xs[i+1]:=X+h*k1:
    ys[i+1]:=Y+h*k2:
    #update xs and ys with Euler!
  end do:
```

```

> tangefield:=fieldplot([y,-1.01*x-.2*y],x=-1..1.01,y=-1..1.):
alpha:=arctan(.1):
C:=1/cos(alpha):
Eulerapprox:=pointplot({seq([xs[i],ys[i]],i=1..n+1)}):
exactsol:=plot([C*exp(-.1*s)*cos(s-alpha),
C*exp(-.1*s)*(-.1*cos(s-alpha)-sin(s-alpha)),
s=0..2*Pi],color=black): #This is the actual
#solution to the IVP
display({tangefield,Eulerapprox,exactsol},title="go with the flow");

```



Any first order system is analogous to the above example, and that's why we expect existence and uniqueness in general. As with first order DE's, we'll get better approximate solutions using more sophisticated algorithms, like improved Euler and Runge Kutta. For example, here's some improved Euler code, first for the first order scalar ODE, then converted to the system above.

Original improved Euler loop, downloaded and stolen from the file *numerical1.mw*, in our class lectures and homework directories.

```

> for i from 1 to n do
    k1:=f(x,y):           #left-hand slope
    k2:=f(x+h,y+h*k1):    #approximation to right-hand slope
    k:=(k1+k2)/2:         #approximation to average slope
    y:= y+h*k:            #improved Euler update
    x:= x+h:              #update x
    print(x,y,exp(x));
end do:

```

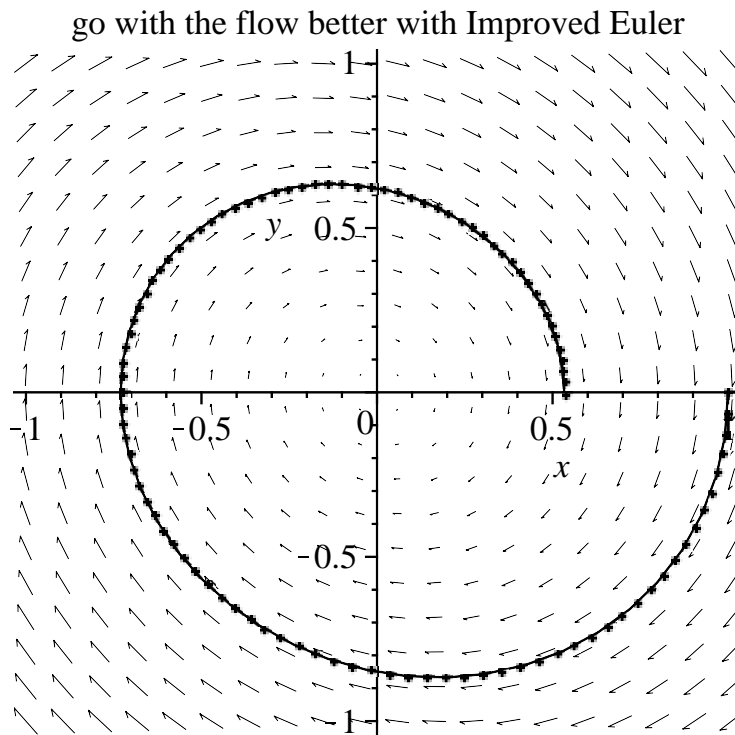
We now vectorize everything, and store the [x,y] values in a vector so we can make a picture like we did for unimproved Euler. (Before executing the code one the next page I re-entered all the initialization definitions from the previous page.)

Improved Euler loop for first order system of two differential equations, following a re-initialization

```
> n := 100 :
    #number of time steps
t0 := 0 : #initial time
tn := evalf(2 * Pi) :
    #final time choice
h := (tn - t0) / n :
    #time step
xs := Vector(n + 1) :
ys := Vector(n + 1) :    #vectors to hold Euler approximates
xs[1] := 1 :
ys[1] := 0 :            #initial conditions

> #Improved Euler loop:
for i from 1 to n do
    X:=xs[i]:
    Y:=ys[i]:
    t:=t0+(i-1)*h:
    k11:=f1(X,Y,t):
    k12:=f2(X,Y,t):
        #k11 and k12 are left-hand tangent estimates
    k21:=f1(X+h*k11,Y+h*k12,t):
    k22:=f2(X+h*k11,Y+h*k12,t):
        #k21 and k22 are right-hand tangent estimates
    xs[i+1]:=X+h*(k11+k21)/2:
    ys[i+1]:=Y+h*(k12+k22)/2:
        #update xs and ys with improved Euler!
od:

> tangeffield:=fieldplot([y,-1.01*x-.2*y],x=-1..1.01,y=-1..1.):
alpha:=arctan(.1):
C:=1/cos(alpha):
Eulerapprox:=pointplot({seq([xs[i],ys[i]],i=1..n+1)}):
exactsol:=plot([C*exp(-.1*s)*cos(s-alpha),
C*exp(-.1*s)*(-.1*cos(s-alpha)-sin(s-alpha)),
s=0..2*Pi],color=black): #This is the actual
    #solution to the IVP, more complicated than
    #the formula in yesterday's notes, but correct
    #for this IVP, I hope.
display({tangeffield,Eulerapprox,exactsol},title="go with the flow better with
Improved Euler");
```



Remarks:

(1) Any single n th order differential equation is equivalent to a related first order system of n first order differential equations, as we've been discussing. You get this system in the same way as we did in the worked out example. Then the existence and uniqueness theorem for first order systems, which makes clear geometric sense, implies the existence and uniqueness theorem that we quoted a month ago for n th order (linear) differential equations (which wasn't as intuitively believable).

(2) If you want to solve an n th order differential equation numerically you first convert it to a first order system, and then use Euler or Runge–Kutta, to solve it! (Or you use a package that does something like that while you aren't looking.)

Other neat examples:

The pictures we drew above are called "phase portraits", like the 1–d phase diagrams we drew for autonomous single first order equations in chapter 2. (To get pictures analogous to the 2–dimensional (t, x) slope field pictures you would need to introduce a t – axis, so you'd be drawing a 3–dimensional (t, x, y) sketch.) If you remember that for the mechanical examples the vertical y –direction corresponds to velocity and the horizontal x –direction corresponds to position, then the phase portrait is a beautiful way to exhibit position and velocity simultaneously (although time is not explicitly shown).

Here are some other mechanical systems converted into first order systems:

Undamped harmonic oscillator:

Let $m = k = 1$ and $c = 0$ in the mass–spring differential equation. Thus, for displacement $x(t)$ from equilibrium,

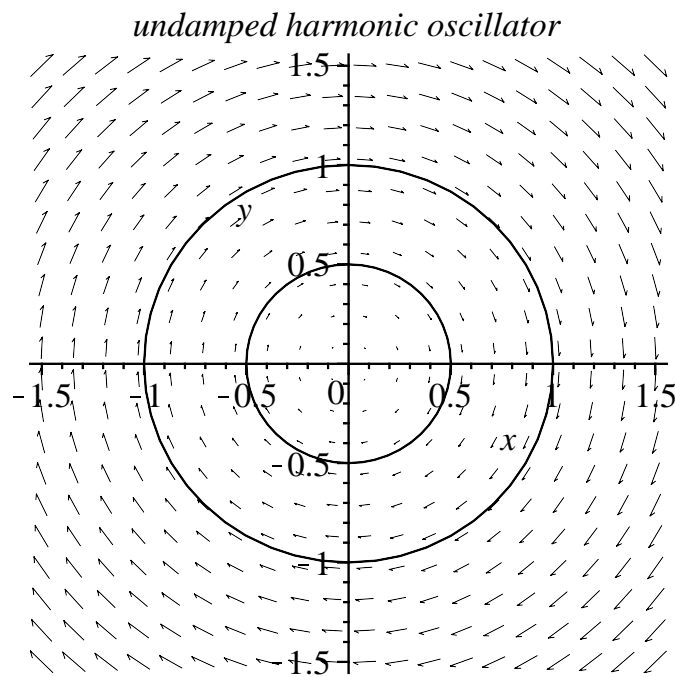
$$\frac{d^2}{dt^2} x(t) + x(t) = 0.$$

Converting this to a first order system as before, i.e. introducing the function $y(t)$ as a proxy for $x'(t)$, yields

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} y \\ -x \end{bmatrix}$$

Can you explain the picture on the next page? Can you explain why the solutions curves $[x(t), y(t)]$ traverse circles, directly from the DE? How is this related to the total energy (KE+PE) of the undamped oscillator being constant?

```
> restart:
with(plots):
plot1:=fieldplot([y,-x],x=-1.5..1.5,y=-1.5..1.5):
plot2:=plot([cos(t),-sin(t),t=0..2*Pi],
[.5*cos(t),-.5*sin(t),t=0..2*Pi],color=black):
display({plot1,plot2},title='undamped harmonic oscillator');
```



Nonlinear pendulum: Recall that the non-linear pendulum equation was

$$L \left(\frac{d^2}{dt^2} \theta(t) \right) + g \sin(\theta(t)) = 0$$

and that we derived this equation by setting the derivative of the total energy equal to zero, where this sum of kinetic and potential energy was given by

$$E = \frac{1}{2} m L^2 \left(\frac{d}{dt} \theta(t) \right)^2 + m g L (1 - \cos(\theta(t))).$$

If we take $L = g$, and write

$$\begin{aligned} x &= \theta(t) \\ y &= \frac{d}{dt} \theta(t) \end{aligned}$$

we get the equivalent first order system

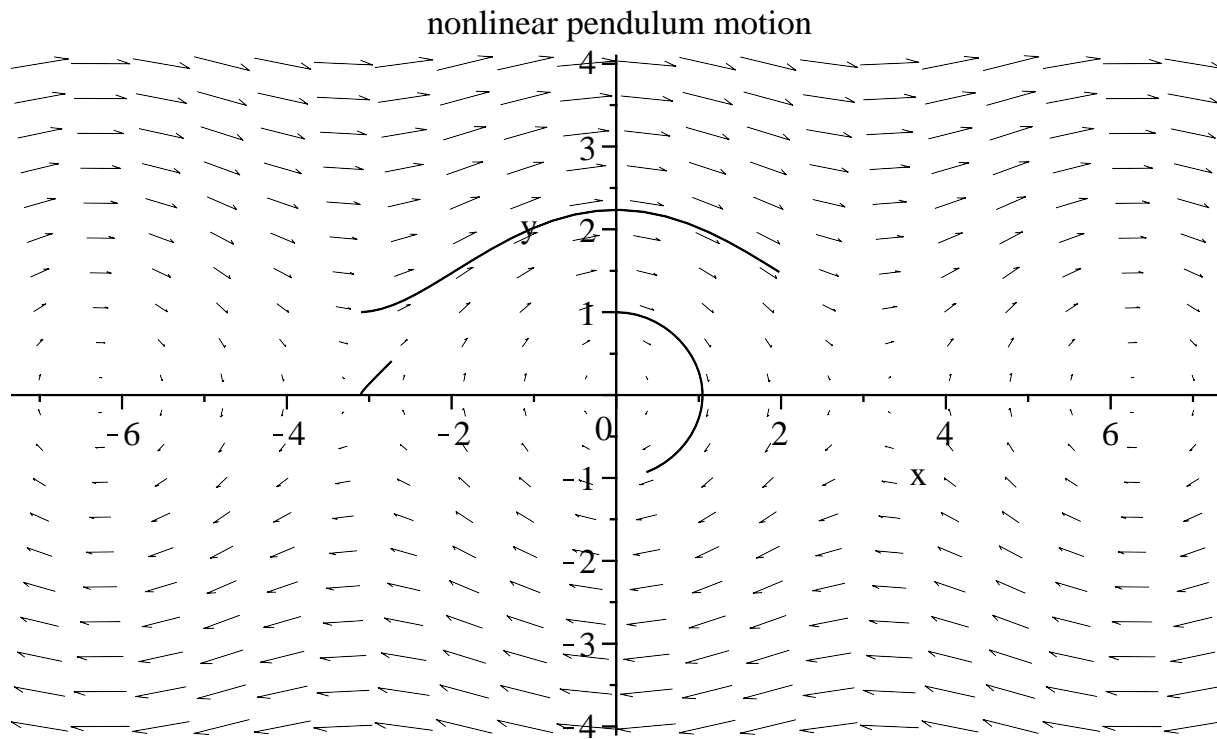
$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} y \\ -\sin(x) \end{bmatrix}.$$

What are the equilibrium solutions of this autonomous system?

Which ones do you think are "stable"?

Interpret this picture:

```
> velocfield:=fieldplot([y,-sin(x)],x=-7..7,y=-4..4):
pendeqtn1:={diff(x(t),t)=y(t),diff(y(t),t)=-sin(x(t)),
  x(0)=0,y(0)=1}:
traj1:=odeplot(dsolve(pendeqtn1,[x(t),y(t)], type=numeric),[x(t),y(t)],0..3,
  numpoints=25,color=black):
pendeqtn2:={diff(x(t),t)=y(t),diff(y(t),t)=-sin(x(t)),
  x(0)=-3.1,y(0)=0}:
traj2:=odeplot(dsolve(pendeqtn2,[x(t),y(t)],type=numeric),
  [x(t),y(t)],0..3,numpoints=25,color=black):
pendeqtn3:={diff(x(t),t)=y(t),diff(y(t),t)=-sin(x(t)),
  x(0)=-3.1,y(0)=1}:
traj3:=odeplot(dsolve(pendeqtn3,[x(t),y(t)], type=numeric),[x(t),y(t)],
  0..3,numpoints=25,color=black):
display({velocfield,traj1,traj2,traj3},
  title="nonlinear pendulum motion");
```

>

Notice that if you magnify the picture near the origin, it starts looking like the undamped harmonic oscillator of the previous example. If we added appropriate damping to our pendulum model, then the magnified picture near the origin could conceivably look like our original example of the damped harmonic oscillator. These are geometric illustrations of the idea of linearization! (You could also linearize near the unstable equilibria.)