USING OPEN SOURCE 2D AND 3D VISUALIZATION TOOLS TO MOTIVATE DIFFERENTIAL EQUATIONS

Adam O. Hausknecht
University of Massachusetts Dartmouth
Mathematics Department, 285 Old Westport Road, N. Dartmouth, MA 02747-2300
ahausknecht@umassd.edu

In recent years, I have changed my differential equations course to accommodate a broad spectrum of students with diverse math backgrounds, many of whom work twenty or more hours per week. I will first briefly describe my course and then present examples of how I use TEMATH, Visual Python, and videos to motivate and engage my students.

My MTH 212 Course

MTH 212 is a 3-credit differential equations course with Calculus I and II as its only prerequisites. However, about half of the students will have completed Calculus III. Note that linear algebra is *not* a prerequisite and there is no time limit on when the calculus courses were taken. In theory, there is a standard syllabus but in recent years not in practice. The reasons are many: First, about 10% to 40% of the students have D's and C–'s in their calculus courses while 10% to 40% of the students (largely from physics and electrical engineering) are extremely strong, and have solid A's and B's in their calculus courses. Thus, there is an extreme range in backgrounds and interests. Further, the size of the class can be anywhere from 30 to 60 students, and a section may or may not be held in a computer lab environment. In any case, I can't assume the students have a working knowledge of Maple or MATLAB nor can I assume that they have access to Maple or MATLAB outside the classroom. To further complicate matters, a grader may or may not be provided depending on the resources available in a given semester. Also, because this is a one-semester service course, many students won't purchase a text that costs more than $100. For this reason, I've stopped using my favorite text for the course by Paul Blanchard et al. (see [1]), and have switched to the text by James C. Robinson supplemented by the Schaum's Outline on Differential Equations (see [2] and [3]). Finally, because many of the students work 20+ hours per week or are taking overloads, it is becoming increasingly difficult to get them to do all the work necessary to learn the material! Consequently, I've reduced the number of topics to the following:

- Integration techniques and trivial ODE's
- Separable differential equations
- Euler and Runge-Kutta Methods
- Applications of first-order equations
- Applications of second-order linear equations
- Existence & uniqueness of solutions
- Autonomous differential equations
- First-order linear equations
- Second-order linear equations
- First-order 2D linear systems

In particular, unless I have an exceptionally strong class, I omit the Laplace Transform and infinite series solutions. To help engage my students in the classroom, I've reduced the amount of time I spend on formal lecturing and have increased my use of in-class worksheets. To motivate students' interest in the material, I have increased my use of software and video demonstrations of applications of differential equations. Examples are presented below.

Software Demonstrations

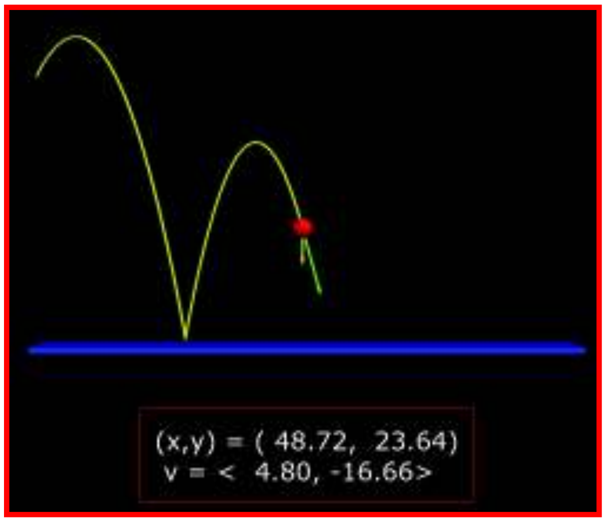


Figure 1: A bouncing ball simulation

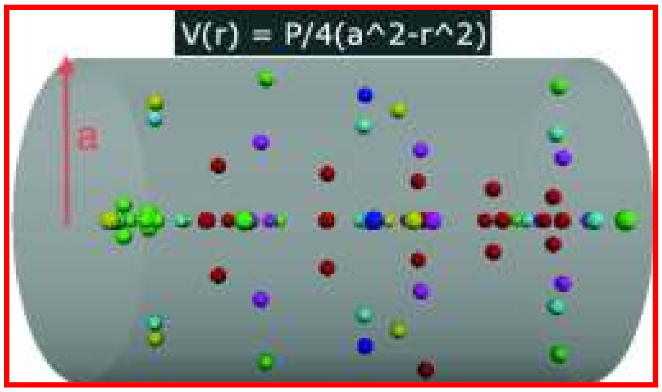


Figure 2: A Poiseuille Flow simulation

Example 1: A Visual Python simulation of a bouncing ball (see **Figure 1** above and [6]).

The model used is $d^2r/dt^2 = \langle 0, -g \rangle$ where $\mathbf{v}(0) = \langle v_{x,0}, v_{y,0} \rangle$, $t_{n,landing}$ is the time the ball lands for the n^{th}-time, and the velocity at the start t_n of the n^{th}-bounce is $\mathbf{v}_n(t_n) = bounceCoeff \langle v_{n-1,x}(t_{n-1,landing}), -v_{n-1,y}(t_{n-1,landing}) \rangle$ since $t_n = t_{n-1,landing}$. □

```
from visual import *
scene1 = display(title='Example Thrown Ball with Bounce',
         x=0, y=0, width=800, height=800, center=(50,20,4))
## The floor upon which the ball bounces
floor = box(pos = (50, 0, 0), length=100, height= 1, width=6,
            color=color.blue)
## The The Ball:
a0 = vector(0, -9.81, 0) ## initial acceleration
v0 = vector(6,   12, 0)  ## initial velocity
p0 = vector(0,   50, 0)  ## initial position
bounceCoeff = 0.8 ## Bounce coefficient; reduces v after each bounce
ball = sphere(pos=p0, color=color.red, radius = 2)
ball.velocity = v0
ball.trail = curve(color=color.yellow, radius = 0.3)
## Vectors used to represent the vector's a and v
## Note: They are scaled so that they are both visible
scaleFactor = 0.7
vArrow = arrow(pos = p0, axis=v0*scaleFactor, shaftwidth=0.5,
               color=color.green)
```

```
aArrow = arrow(pos = p0, axis=a0*scaleFactor, shaftwidth=0.5,
               color=color.yellow
## Output label
positionVelocityLabel = label(pos=(50,-20,0), text ="",
                    height=30,linecolor= color.red, border = 20)

def displayOutputStr(ball):
    formatStr = "(x,y) = (%6.2f, %6.2f) \n v = <%6.2f, %6.2f> "
    v = ball.velocity
    positionVelocityLabel.text = formatStr%(ball.x,ball.y,v.x,v.y)

## Initialize time variables
t = 0; dt = 0.1; totalTime = 0
finished = False
while not finished: ## Animation loop
    rate(10) ## Control animation rate
    displayOutputStr(ball) # Update the output
    # Compute the balls new position and velocity
    ball.pos = p0 + v0*t + a0*(t**2)/2.0
    ball.velocity = v0 + a0*t
    ball.trail.append(pos=ball.pos)
    # Update the velocity and acceleration arrows
    vArrow.pos = ball.pos
    vArrow.axis = ball.velocity*scaleFactor
    aArrow.pos = ball.pos
    # Increment time:
    t += dt         # Increment running time for current bounce
    totalTime += dt # Increment total running time for all bounces
    if ball.y < 2: # Ball is less than 2 m above the ground;
                   # Hence, the ball has hit the ground.
                   # Need to reset t, p0, and v0
        t = 0 # Reset running time for bounce back to zero
        p0.x = ball.x; p0.y = 2  # Reset the initial position of
                                 # the ball to the point of impact.
        # Reset the veloicty
        v0.x =  ball.velocity.x*bounceCoeff # Reduce Vx
        v0.y = -ball.velocity.y*bounceCoeff # Reverse and reduce Vy
    # Stop when ball reaches the end of the floor
    finished = (ball.x>=98)
```

Example 2: A Visual Python simulation of a Poiseuille Flow in a cylinder of radius a (see **Figure 2** above and page 34 of [2]).

The model used is $dv/dt = -Pr/2 + c/r$ where r is the distance from the center of the cylinder of a particle moving with velocity v, $P > 0$ is a constant and c is an arbitrary constant, $v(a) = 0$, $v(r) < \infty$, and $0 \le r \le a$.□

```
from visual import *
## Create the scene window and the pipe
scene2 = display(title='Poiseuille Flow', x=0, y=0,
       width=600, height=600, center=(15,0,0), background=(1,1,1))
a = 10; P = .1
pipe = cylinder(pos=(0,0,0), axis=(30,0,0), radius=10,
                color = color.gray(0.5), opacity=0.4)
useZParticles = True
pLabel = label(pos=(15,12,0), text='V(r) = P/4(a^2-r^2)',
               height=24, font='sans')
```

```
## Create the particles
particles = []
pColors = [color.red, color.green, color.blue,
           color.yellow, color.cyan, color.magenta]
for r in arange(-9.5, 9.5, 0.5):
    p = sphere(pos = (0, r, 0), radius = 0.5, color = color.red)
    p.startPos = vector(0, r, 0)
    V = P/4.0*(a**2-r**2)
    p.velocity = vector(V, 0, 0); p.t = 0; p.colorIndex = 0
    particles.append(p)
if useZParticles:
    for r in arange(-9.5, 9.5, 0.5):
        p = sphere(pos = (0, 0, r), radius = 0.5, color = color.red)
        p.startPos = vector(0, 0, r)
        V = P/4.0*(a**2-r**2)
        p.velocity = vector(V, 0, 0); p.t = 0; p.colorIndex = 0
        particles.append(p)
dt = 0.01
while True: ## The Animation Loop
    rate(100)
    for p in particles:
        p.t += dt ## Increment the partial's time
        if p.x < 30: ## Move the partial with the flow
            p.pos = p.startPos + p.velocity*p.t
        else: ## Move the particle to the start of the pipe
              ## and change its color - simulates many particles!
            p.pos = p.startPos; p.t = 0
            p.colorIndex = (p.colorIndex +1)%6
            p.color = pColors[p.colorIndex]
```

<u>*Example 3*</u>: Use TEMATH's first-order differential equation solver tool to demonstrate Euler's Method (see **Figure 3** below and [4]).

For the demo, I use the initial value problem $dy/dx = x - y^2$, $y(0) = 0$. This example is widely used because it is relatively easy to evaluate by hand and it is known that its solutions can't be expressed in terms of elementary functions (see [3]). I also repeat this example using a spreadsheet and assign the students a worksheet containing several more problems which they can either complete by hand or by using a spreadsheet during the class meeting. For extra credit, I have the students use a spreadsheet to implement the 4^{th} -Order Runge-Kutta Method and compare its solutions to those obtained by using Euler's Method. □

<u>*Example 4*</u>: Use TEMATH's first-order differential equation solver tool to show that $dy/dx = \sqrt[3]{y}$ does not have unique solutions (see **Figure 7** below and [4]).

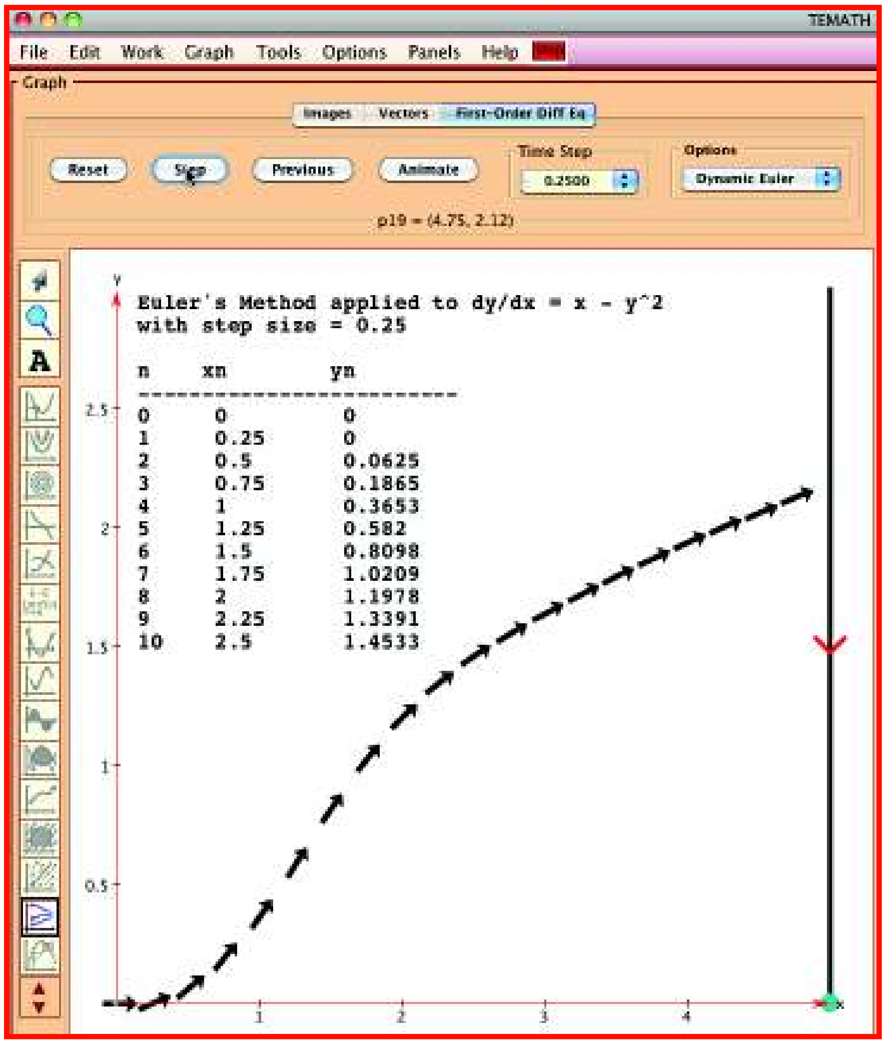
Given a constant *c*, the three functions

$$f(x) = 0\,,\ g(x) = \begin{cases} (2x/3 + c)^{3/2} & \text{if } x \ge -3c/2 \\ 0 & \text{otherwise} \end{cases}\ ,\ h(x) = \begin{cases} -(2x/3 + c)^{3/2} & \text{if } x \ge -3c/2 \\ 0 & \text{otherwise} \end{cases}$$

are all solutions of this differential equations that satisfy the same initial condition

$$f(-3c/2) = g(-3c/2) = h(-3c/2) = 0$$

Thus, $dy/dx = \sqrt[3]{y}$ fails to have a unique solution at an infinite number of points in its domain! □


TEMATH
File Edit Work Graph Tools Options Panels Help
Graph
Images Vectors First-Order Diff Eq
Reset Step Previous Animate

Time Step 0.2500
Options Dynamic Euler
p19 = (4.75, 2.12)

```
Euler's Method applied to dy/dx = x - y^2
with step size = 0.25

n    xn        yn
-------------------------
0    0         0
1    0.25      0
2    0.5       0.0625
3    0.75      0.1865
4    1         0.3653
5    1.25      0.582
6    1.5       0.8098
7    1.75      1.0209
8    2         1.1978
9    2.25      1.3391
10   2.5       1.4533
```

Figure 3: Using TEMATH to Demo Euler's Method

Example 5: A Visual Python simulation of Hooke's Law (see **Figure 4** below).

I use this simulation to help students relate the properties of the mass-spring system model and its parameters to its solutions. The simulation dynamically displays the motion of the mass and plots the position of the mass against time when the system is subject to *zero damping*, *critical damping*, *over-damping*, *under-damping*, *resonance,* and *near resonance.* In the case of resonance, the virtual spring is allowed to increase in length without bound, which of course can't happen, but does get the full attention of my students! □

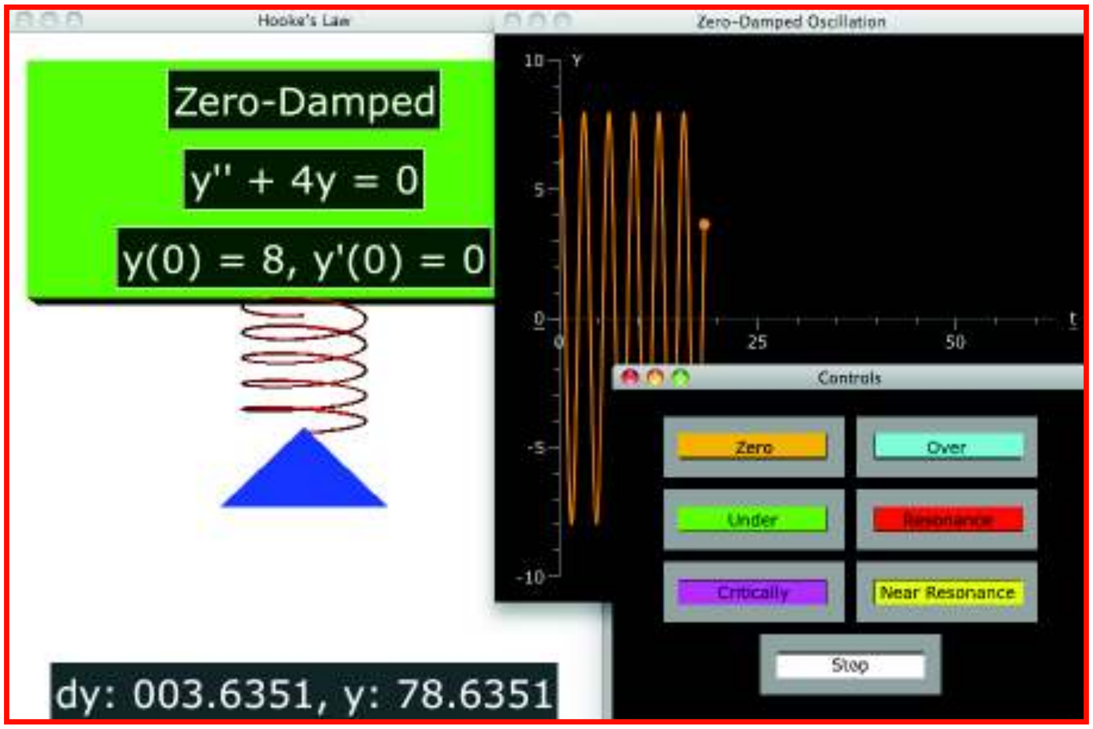


Figure 4: Hooke's Law Simulation

```
from visual import *
from visual.controls import *
from visual.graph import *
ZeroDamping = 0; UnderDamped = 1; CriticallyDamped = 2
OverDamped = 3; Resonance = 4; NearResonance = 5
kind = ZeroDamping
titleStr = 'Zero-Damped'; finished = False
scene = display(title="Hooke's Law", x=0, y=0,
    width=500, height=800, center=(0,80,0), background=(1,1,1))
# Create a control window
cWidth = 150; cHeight = 50
cWindow = controls(title = "Controls", x = 500, y = 450,
                   width = 400, height = 300, range = 200)

button( pos=(-80, 90), width=cWidth, height=cHeight, color =
      color.orange, text=”Zero”, action=lambda:setKind(ZeroDamping) )
button( pos=(-80, 30), width=cWidth, height=cHeight, color =
color.green, text="Under", action=lambda: setKind(UnderDamped) )
button( pos=(-80,-30), width=cWidth, height=cHeight,color =
color.magenta,text="Critically",action=lambda:setKind(CriticallyDamped))
button( pos=( 80, 90), width=cWidth, height=cHeight, color =
        color.cyan,text="Over", action=lambda: setKind(OverDamped) )
button( pos=( 80, 30), width=cWidth, height=cHeight, color =
      color.red, text="Resonance", action=lambda: setKind(Resonance) )
button( pos=( 80,-30), width=cWidth, height=cHeight, color =
color.yellow,text="Near Resonance",action=lambda:setKind(NearResonance))
button( pos=(0,-90), width=cWidth, height=cHeight, color = color.white,
        text="Stop", action=lambda: doStop() )
```

```
# Create a Graph window
graph1 = gdisplay(x=500, y=0,width=500, height=500,
  title = titleStr+' Oscillation', xtitle='t', ytitle='Y',
  xmin = 0., xmax=20*pi, ymin = -10, ymax = 10,
  foreground=color.white, background=(0,0,0))
typeLabel = label(pos=(0,130,0), text = titleStr,  height=30)
diffEqLabel = label(pos=(0,120,0), text= "y'' + 4y = 0",  height=30)
initialConditionLabel=label(pos=(0,110,0),text="y(0) = 4, y'(0) = 0",
                            height=30)

## Create the function plots
functs = []
functs.append(gcurve(color=color.orange, dot=True, size=10))
functs.append(gcurve(color=color.green,  dot=True, size=10))
functs.append(gcurve(color=color.magenta,dot=True,size=10))
functs.append(gcurve(color=color.cyan,   dot=True, size=10))
functs.append(gcurve(color=color.red,    dot=True, size=10))
functs.append(gcurve(color=color.yellow, dot=True, size=10))
for f in functs: f.gcurve.radius = 0.3

## Create the control call back functions
def displayKind():
    global kind, typeLabel, diffEqLabel, graph1, initialConditionLabel
    if kind == ZeroDamping:
        titleStr = 'Zero-Damped'; diffEqStr = "y'' + 4y = 0"
        initialCondStr = "y(0) = 8, y'(0) = 0"
        ## => y = 8*cos(2*t)
    elif kind == UnderDamped:
        titleStr = 'Under-Damped';
        diffEqStr = "y'' + 0.2y' + 4.01y = 0"
        initialCondStr = "y(0) = 8, y'(0) = -0.8"
        ## => y = 8*cos(2*t)*exp(-0.1*t)
    elif kind == CriticallyDamped:
        titleStr = 'Critically-Damped'
        diffEqStr = "y'' + 0.2y' + 0.01y = 0"
        initialCondStr = "y(0) = -7, y'(0) = 4.2"
        ## => y = 3.5*(t-2)*exp(-0.1*t)
    elif kind == OverDamped:
        titleStr = 'Over-Damped'; diffEqStr = "y'' + 0.3y' + 0.02y = 0"
        initialCondStr = "y(0) = 8, y'(0) = -1.2"
        ## => y = 4*(exp(-0.2*t) + exp(-0.1*t))
    elif kind == Resonance:
        titleStr = 'Resonance'; diffEqStr = "y'' + 4y = 2cos(2t)"
        initialCondStr = "y(0) = 0, y'(0) = 0"
        ## => y = 0.5t*sin(2*t)
    else:## kind == NearResonance:
         titleStr = 'Near Resonance';
         diffEqStr = "y'' + 4y = 2cos(2.1t)"
         initialCondStr = "y(0) = 0, y'(0) = 0"
         ## => y = 2/(cos(2.1*t) - cos(2*t))/(4-(2.1)**2)
    typeLabel.text = titleStr; diffEqLabel.text = diffEqStr
    initialConditionLabel.text = initialCondStr
    ## There is a bug in vPython that causes the program to sometimes
    ## exit when the next three lines are executed.
    graph1.display.visible = False
    graph1.display.title = titleStr+' Oscillation'
    graph1.display.visible = True
```

```
def setKind(newKind):
    global kind
    functs[kind].gcurve.pos = []
    functs[kind].dotobj.visible = False;
    kind = newKind
    functs[kind].dotobj.visible = True;
    displayKind(); reset()

def reset():
    global t, dt
    t = 0; dt = 0.1; finished = False

def doStop():
    global finished
    finished = True

def f(t, kind): ## Compute the position of bottom of spring
    if kind == ZeroDamping:
      return 8*cos(2*t)
    elif kind == UnderDamped:
      return 8*cos(2*t)*exp(-0.1*t)
    elif kind == CriticallyDamped:
      return 3.5*(t-2)*exp(-0.1*t)
    elif kind == OverDamped:
      return 4*(exp(-0.2*t) + exp(-0.1*t))
    elif kind == Resonance:
      return 0.5*t*sin(2*t)
    else:
      return 2*(cos(2.1*t)-cos(2*t))/(4-(2.1)**2)

## Create the scene objects
massRestY = 75
anchor = box(pos=(0,120,-2), length=70, height=30, width=4,
             color=color.green)
spring = helix(pos=(0,105,-2), axis=(0,-20, 0), radius=8,
               color = color.red)
mass = pyramid(pos=(0,massRestY,0), size=(10,20,10), color =
color.blue)
mass.rotate(angle=pi/2., axis=(0,0,1))
yLabel = label(pos=(0,55,0), text='', height=30)

## The animation loop
setKind(ZeroDamping) ## Start with zero-damping
cWindow.interact()
while not finished:
    rate(10)  ## Set animation rate
    cWindow.interact()  ## Give time to the controls
    t += dt ## Increment time
    dy = f(t, kind) ## Compute the position of the spring’s bottom
    spring.axis = (0, dy-20, 0) ## Resize the spring
    mass.y = dy + massRestY ## Move the mass
    ## Update the output label
    yLabel.text = "dy: %08.4f, y: %07.4f"%(dy, mass.y)
    ## Update the plot of y(t)
    functs[kind].plot(pos = (t,dy) )
```

Videos

Example 1: The Famous Tacoma Narrows Bridge Collapse (see **Figures 5a-5b** below).

Stock footage of the collapse with sound can be downloaded from http://www. archive. org/details/CEP176. The are also many versions at http://www. youtube.com. I show this video when discussing damping and resonance of mass-spring systems. This definitely receives the student's full attention! □

Figures 5a: Tacoma Narrows Bridge

Figure 5b: The Bridge flexing

Example 2: The Vibration Problem of the Millennium Bridge (see **Figure 6** below).

Unexpectedly its the opening day, the Millennium Bridge started vibrating when a large crowd began walking across it. The people in the crowd, in order to keep their balance, began walking in a manner that was more in-step with each other. This caused the magnitude of the vibrations to increase! Free videos of the opening day can be found at www.arup.com/MillenniumBridge/indepth/video.html.There are also many versions at www.youtube.com/. This example is particularly good because there is an additional video that documents how the ARUP engineers studied the problem. Using the results of their studies, the engineers modified the bridge by adding dampers which essentially eliminated the vibrations. There is a problem in Robinson's text (see pages 128-129 of [5]) that works through the ARUP engineers' remedy. □

Figure 6: The Millennium Bridge

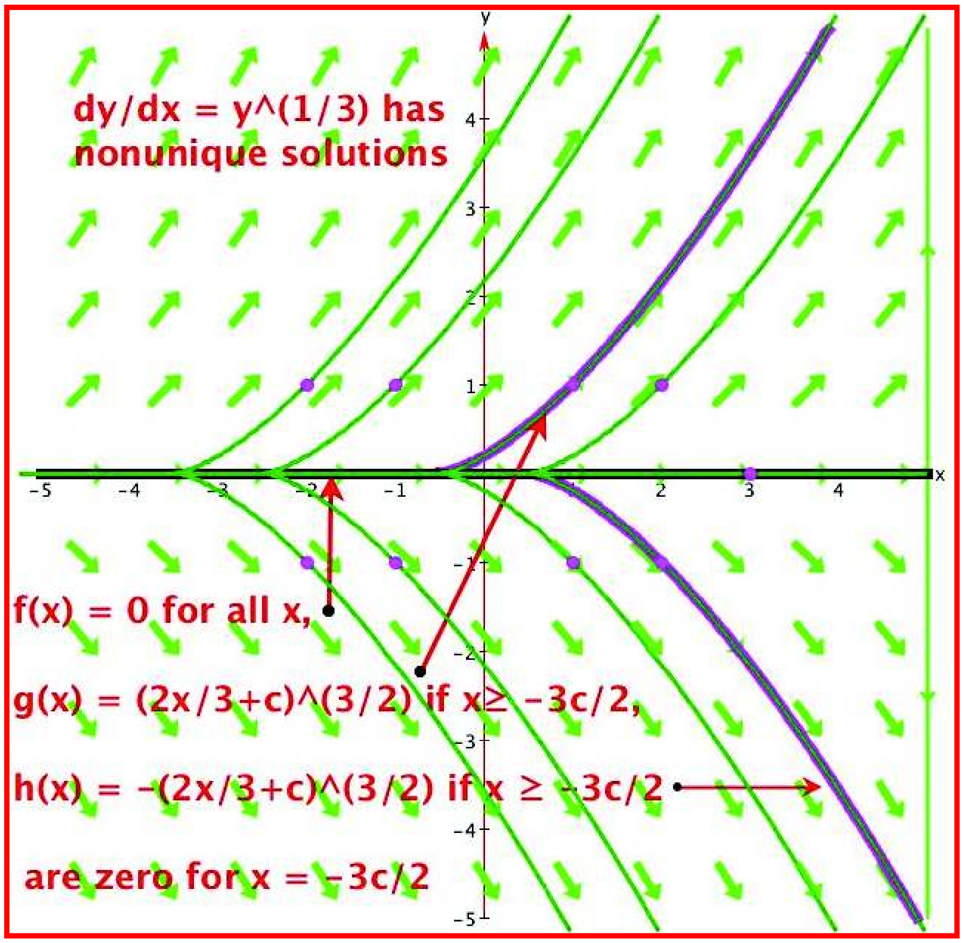


Figure 7: Using TEMATH to display non-unique solutions

Bibliography

[1] Ricard Bronson, Gabriel Costa, *Schaum's Outline of Differential Equations*, Third Edition, McGraw-Hill, 2006.

[2] Paul Blanchard, Robert L. Devaney, Glen R. Hall, Differential Equations, Forth Edition, Brooks/Cole, 2012.

[3] John H. Hubbard, Benjamin E. Lundell, *A First Look at Differential Algebra,* The American Mathematics Monthly*, Volume 118, NO. 3, March 2011,* 245-261.

[4] Robert Kowalczyk and Adam Hausknecht, *TEMATH - Tools for Exploring Mathematics Version 3.0b*, 2011, http://www2.umassd.edu/temath.

[5] James C. Robinson, *Ordinary Differential Equations*, Cambridge University Press, 2007.

[6] David Scherer, David Andersen, Ruth Chabay, Ari Heitner, Ian Peters, Bruce Sherwood, *Visual Python - 3D Programming for Ordinary Mortal*s, http://vpython.org/.