```
from vpython import *
#GlowScript 2.7 VPython
## Written for
## MTH 212 Spring 2010 - Spring 2020
## Conversion from Visual Python to GlowScript Spring 2013
## Revised and updated to Glowscript 2.7 Fall 2020
## Fixed bugs, changed dependent differential equation variable
## from y to x and added a menu to set the near resonance
## forcing function's angle from 2.5 radians down to 2.01
radians
## Adam O. Hausknecht
## Department of Mathematics
## Umass Dartmouth


gZeroDamping = 0; gUnderDamped = 1; gCriticallyDamped = 2
gOverDamped = 3;  gResonance = 4;   gNearResonance = 5
gKind = gZeroDamping
gTitleStr = 'Zero-Damped'
gFinished = False
gSolutionStr = ' '
gAnimationRate = 10
gT = 0
gDt = 0.1
gDy = 0
gNRangle = 2.2
gNRangleStr = '2.2'

scene = canvas(title = "<h3>Hooke's Law Simulation V6</h3>")
scene.width = 500
scene.height = 440
scene.center = vector(0,80,0)
scene.background = vector(0.4, 0.4, 1)
scene.range = 60;
scene.userzoom = False;
scene.usespin = False;

## Create the scene objects
gMassRestY = 75
gAnchor =    box(pos = vector(0, 120,       -2), size =
vector(70, 30, 4), color = color.green)
```

```
40 gSpring =  helix(pos = vector(0, 104,       -2), axis = vector(
… 0, -1, 0), size = vector(10, 20,10),
41                  thickness= 0.5, color = color.red )
42 gMass =  pyramid(pos = vector(0, gMassRestY, 0), size =
… vector(10,20, 10),
43                  color = vector(.7, .7, .7), axis =
… vector(0,0,1))
44 gMass.rotate(angle = -pi/2.0, axis = vec(1,0,0), origin =
… gMass.pos )
45
46 gYlabel =   label(pos = vector(-48,55,0), text = '', height =
… 24, font = 'Verdana',  align = "left" )
47 gSolLabel = label(pos = vector(0,40,0), text = '', height = 24,
… font = 'Verdana',
48                   opacity = 1, background = color.white)
49 #box(pos = vector(0, 50,       -2), size = vector(100, 40, 1),
… color = color.black)
50
51 def setAnimationRate(menu):
52     global gAnimationRate
53     s = menu.selected
54     if s == 'Slow':
55         gAnimationRate = 5
56     elif s == 'Medium':
57         gAnimationRate = 10
58     elif s == 'Fast':
59         gAnimationRate = 20
60     else:
61         gAnimationRate = 30
62
63
64 def doSetKind(menu):
65     s = menu.selected
66     if s == 'Zero-Damping':
67         setKind(gZeroDamping)
68     elif s == 'Under-Damped':
69         setKind(gUnderDamped)
70     elif s == "Critically-Damped":
71         setKind(gCriticallyDamped)
72     elif s == "Over-Damped":
73         setKind(gOverDamped)
```

```
    elif s == "Resonance":
        setKind(gResonance)
    elif s == "Near-Resonance":
        setKind(gNearResonance)

def setKind(newKind):
    global gKind
    global gFuncts
    ## Need away to delete a series!
    if len(gFuncts) > 0:
        if gFuncts[gKind] != undefined:
            gFuncts[gKind].delete()

    gKind = newKind
    displayKind()
    reset()

def doUpdateNearResonance():
    global gKind
    global gFuncts

    gFuncts[gKind].delete()
    gKind = gNearResonance
    displayKind()
    reset()


def setNearResonance(menu):
    global gNRangle
    global gNRangleStr
    s = menu.selected
    if s == '2.5':
        gNRangle = 2.5
        gNRangleStr = '2.5'
    elif s == '2.4':
        gNRangle = 2.4
        gNRangleStr = '2.4'
    elif s == '2.3':
        gNRangle = 2.3
        gNRangleStr = '2.3'
    elif s == '2.2':
```

```
115         gNRangle = 2.2
116         gNRangleStr = '2.2'
117     elif s == '2.1':
118         gNRangle = 2.1
119         gNRangleStr = '2.1'
120     elif s == '2.05':
121         gNRangle = 2.05
122         gNRangleStr = '2.05'
123     else:
124         gNRangle = 2.01
125         gNRangleStr = '2.01'
126 
127     if gNearResonance == gKind:
128         doUpdateNearResonance()
129 
130 
131 
132 scene.append_to_title("<b>Change the model:</b>")
133 menu(pos=scene.title_anchor, choices=['Zero-Damping',
  … 'Under-Damped', 'Critically-Damped', 'Over-Damped', 'Resonance',
  … 'Near-Resonance'  ], bind=doSetKind )
134 
135 scene.append_to_title("<br><b>Near Resonance Angle: </b>");
136 menu(pos = scene.title_anchor, choices=['2.5', '2.4', '2.3',
  … '2.2', '2.1', '2.05', '2.01'], selected = '2.2', bind =
  … setNearResonance)
137 
138 scene.append_to_title("  <b>Animation Rate: </b>");
139 menu(pos = scene.title_anchor, choices=['Slow', 'Medium',
  … 'Fast', 'Very Fast'],  selected = 'Medium', bind =
  … setAnimationRate)
140 
141 theGraph = graph(width = 500, height = 200, xmin = 0, xmax =
  … 40*pi, ymin = -11, ymax = 11,
142                  foreground = color.black, background =
  … color.white)
143 
144 gTypeLabel = label( pos = vector(0,130,0), text= gTitleStr,
145                     color = color.black, opacity = 0,  box =
  … False, height = 20)
146 
```

```
147 gDiffEqLabel = label(pos = vector(0,120,0), text = "y'' + 4y =
… 0",
148                      color  = color.black, opacity = 0, box =
… False, height = 20)
149
150 gInitialConditionLabel = label(pos = vector(0,110,0), text =
… "y(0) = 4, y'(0) = 0",
151                      color = color.black, opacity = 0, box =
… False, height = 20)
152
153 # Create the function plots
154
155 gFuncts = [];
156 gFuncts.append( gcurve( color = color.orange,  dot = True, size
… = 4, dot_color = color.black) )
157 gFuncts.append( gcurve( color = color.green,   dot = True, size
… = 4, dot_color = color.black) )
158 gFuncts.append( gcurve( color = color.magenta, dot = True, size
… = 4, dot_color = color.black) )
159 gFuncts.append( gcurve( color = color.cyan,    dot = True, size
… = 4, dot_color = color.black) )
160 gFuncts.append( gcurve( color = color.red,     dot = True, size
… = 4, dot_color = color.black) )
161 gFuncts.append( gcurve( color = color.black,   dot = True, size
… = 4, dot_color = color.red )  )
162
163
164 ## Create the control call back functions
165 def displayKind():
166     gTitleStr = ""; gDiffEqStr = "";  gInitialCondStr =""
167
168     if (gKind == gZeroDamping):
169         gTitleStr = 'Zero-Damped'
170         gDiffEqStr = "x'' + 4x = 0"
171         gInitialCondStr = "x(0) = 8, x'(0) = 0"; ## => x =
… 8*cos(2*t)
172         gSolutionStr = "x = 8cos(2t)"
173     elif gKind == gUnderDamped:
174         gTitleStr = 'Under-Damped'
175         gDiffEqStr = "x'' + 0.2x' + 4.01x = 0"
176         gInitialCondStr = "x(0) = 8, x'(0) = -0.8"; ## => x =
```

```
176… 8*cos(2*t)*exp(-0.1*t)
177          gSolutionStr = "x = 8cos(2t)exp(-0.1t)"
178      elif gKind == gCriticallyDamped:
179          gTitleStr = 'Critically-Damped'
180          gDiffEqStr = "x'' + 0.2x' + 0.01x = 0"
181          gInitialCondStr = "x(0) = -7, x'(0) = 4.2"; ## => x =
…  3.5*(t-2)*exp(-0.1*t)
182          gSolutionStr = "x = 3.5(t-2)exp(-0.1t)"
183      elif gKind == gOverDamped:
184          gTitleStr = 'Over-Damped'
185          gDiffEqStr = "x'' + 0.3x' + 0.02x = 0"
186          gInitialCondStr = "x(0) = 8, x'(0) = -1.2"; ## => x =
…  4*(exp(-0.2*t) + exp(-0.1*t));
187          gSolutionStr = "y = 4( exp(-0.2t) + exp(-0.1t) )"
188      elif gKind == gResonance:
189          gTitleStr = 'Resonance'
190          gDiffEqStr = "x'' + 4x = 2cos(2t)"
191          gInitialCondStr = "x(0) = 0, x'(0) = 0"  ##  => x =
…  0.5t*sin(2*t);
192          gSolutionStr = "x = 0.5t sin(2t)"
193      else: ## gKind === gNearResonance
194          gTitleStr = 'Near-Resonance';
195          gDiffEqStr = "x'' + 4x = 2cos("+gNRangleStr+"t)"
196          gInitialCondStr = "x(0) = 0, x'(0) = 0"; ## => x =
…  2(cos(gNRangleStr*t) - cos(2*t))/(NRangleStr)**2);
197          gSolutionStr = "x=2( cos(2t) - cos("+gNRangleStr+"t)
…  )/("+gNRangleStr+"^2 - 4)"
198
199      gTypeLabel.text = gTitleStr
200      gDiffEqLabel.text = gDiffEqStr
201      gInitialConditionLabel.text = gInitialCondStr;
202      gSolLabel.text = gSolutionStr;
203      gSolLabel.color = gFuncts[gKind].color;
204
205
206
207 def reset():
208      global gT, gFinished
209      gT = 0.0
210      gFinished = False
211
```

```
212 def doStop():
213     global gFinished
214     gFinished = True
215 
216 
217 def  f(t, kind):  ## Compute the position of bottom of spring
218 
219     if kind == gZeroDamping:
220       return 8*cos(2*t)
221     elif kind == gUnderDamped:
222       return 8*cos(2*t)*exp(-0.1*t)
223     elif kind == gCriticallyDamped:
224       return 3.5*(t-2)*exp(-0.1*t);
225     elif kind == gOverDamped:
226       return 4*(exp(-0.2*t) + exp(-0.1*t))
227     elif kind == gResonance:
228       return 0.5*t*sin(2*t)
229     else:
230       temp = 2*( cos(gNRangle*t)-cos(2*t) )
231       return temp/(4-Math.pow(gNRangle,2) )
232 
233 
234 def roundTo4(value):
235     return "{:5.2f}".format(value)
236 
237 ## The animation loop
238 setKind(gZeroDamping);  ## Start with zero-damping
239 
240 
241 gDone = False;
242 while not gDone:
243     rate(gAnimationRate, wait)  ## Set animation rate
244     if not gFinished:
245         gT += gDt ## Increment time
246         dy = f(gT, gKind) ## Compute the position of the springs
…  bottom
247         #alert(dy)
248         gSpring.size = vector(20-dy,              10, 10) ##
…  Resize the spring
249         gMass.pos =    vector(    0, dy + gMassRestY,  0)  ##
…  Move the mass
```

```
250         ## Update the output label
251         dyStr = roundTo4(dy)
252         if dyStr.length == 6:
253             gYlabel.text = "t: "+ roundTo4(gT)+"  dx: "+dyStr +"
…   massX: "+ roundTo4(gMass.pos.y)
254         else:
255             gYlabel.text = "t: "+ roundTo4(gT)+"  dx: "+dyStr +"
…   massX: "+ roundTo4(gMass.pos.y)
256         ## Update the plot of x(t)
257         gFuncts[gKind].plot([gT, dy] )
258
259
260
```