Fall 2021

APPLIED PHYSICS - LAB

Lab-5

- Solutions to Lab-4
- Iterative motion

VPython documentation
VPython vector operations
Vpython basic mathematics

Solution to Lab-4:

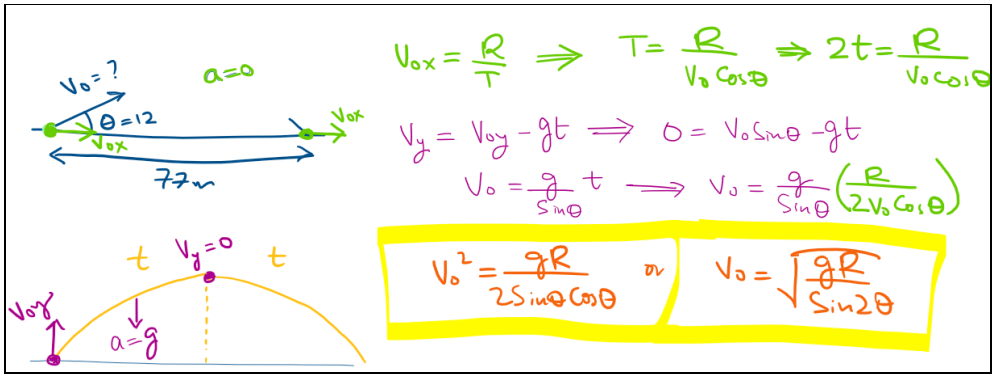
The following is one simple sample solution, but the resulting values should be the same.

```
GlowScript 3.1 VPython

print("TASK1")
x1 = vector(0,0,0)
x2 = vector(200,0,0)
del_x = x2-x1

d1 = mag(x1)
d2 = mag(x2)
del_d = d2-d1

t1, t2 = 0, 102.96
del_t = t2-t1

vel_avg = del_x / del_t
sp = del_d / del_t

print("The average velocity = ",vel_avg,"m/s")
print("The speed = ",sp,"m/s")

print("==================================================")
print("TASK2")

pool_length = 50    # length of the pool
swim_f = vector(+pool_length,0,0)
swim_b = vector(-pool_length,0,0)

# lap 1
lap = 1

x1 = vector(0,0,0)
x2 = swim_f
del_x = x2-x1

d1, d2 = 0, lap*pool_length
del_d = d2-d1
```

```
t1, t2 = 0, 24.31
del_t = t2-t1

vel_avg = del_x / del_t
sp = del_d / del_t

print("The average velocity = ",vel_avg,"m/s")
print("The speed = ",sp,"m/s")


# lap 2
lap = 2

x1 = vector(0,0,0)
x2 = swim_f + swim_b
del_x = x2-x1

d1, d2 = 0, lap*pool_length
del_d = d2-d1

t1, t2 = 0, 50.29
del_t = t2-t1

vel_avg = del_x / del_t
sp = del_d / del_t

print("The average velocity = ",vel_avg,"m/s")
print("The speed = ",sp,"m/s")



# lap 3
lap = 3

x1 = vector(0,0,0)
x2 = swim_f + swim_b + swim_f
del_x = x2-x1

d1, d2 = 0, lap*pool_length
del_d = d2-d1

t1, t2 = 0, 76.84
del_t = t2-t1

vel_avg = del_x / del_t
sp = del_d / del_t

print("The average velocity = ",vel_avg,"m/s")
print("The speed = ",sp,"m/s")


# lap 4
# you should be able to write this one now
```

```
TASK1
The average velocity =  < 1.9425, 0, 0 > m/s
The speed =  1.9425 m/s
=====================================================
TASK1
The average velocity =  < 2.05677, 0, 0 > m/s
The speed =  2.05677 m/s
The average velocity =  < 0, 0, 0 > m/s
The speed =  1.98847 m/s
The average velocity =  < 0.650703, 0, 0 > m/s
The speed =  1.95211 m/s
The average velocity =  < 0, 0, 0 > m/s
The speed =  1.9425 m/s
```

Problem1




```
g, R, ang = 9.8, 77, radians(12)
V_in = sqrt(g*R/(2*sin(ang)*cos(ang)))
print("the initial speed should be = ", V_in, "m/s")
```

```
the initial speed should be =  43.0727 m/s
```

Problem 2

```
GlowScript 3.1 VPython

g, R, v0 = 9.8, 45.7, 460

# x2 = x1 + v0*cos(ang)*t
# 45.7 = 0 + v0*cos(ang)*t
# t= x2/(v0*cos(ang))

# y2 = y1 + v0*sin(ang)*t - 0.5*g*t*t     notice the -ve sign for the g.
# 0 = 0 + v0*sin(ang)*t - 0.5*g*t*t
# v0*sin(ang) = 0.5*g*t
# v0*sin(ang) = 0.5*g*[x2/(v0*cos(ang))]
# v0*sin(ang)*v0*cos(ang) = 0.5*g*x2
# v0*v0*2*sin(ang)*cos(ang) = g*x2
# sin(2*ang) = g*x2/(v0*v0)
# ang = 0.5(asin(g*x2/(v0*v0)))
```

```
ang = 0.5*(asin(g*R/(v0*v0)))

print("The angle that the rifle should make from the horizontal is ",
ang,"rad")
print("or ",degrees(ang)," degrees")
print("in other words")
above = R*tan(ang)      # point above the target
print("The rifle should point ",above," meter above the target \n for the
bullet to strike the target.")
```

```
The angle that the rifle should make from the horizontal is  1.05827e-3 rad
or  0.0606345  degrees
in other words
The rifle should point  0.048363  meter above the target
 for the bullet to strike the target.
```

Iterative Motion:

To begin animating objects we first remind ourselves how objects move. By definition, the motion requires change of position with respect to time. In the computational sense we view the position as a vector and hence a motion would mean changing this position vector gradually. In reality we see things move smoothly and continuously however, in every understanding of our's there must be small discrete steps that make up a smooth continuous motion.

$X_{new} = X_{old} + v\Delta t$

Similarly, the computer can only process specified values at a time and therefore repetition of the position vector in such a manner (iteration) is what results in motion.

Let's first create a scene with an object:

```
1  GlowScript 3.1 VPython
2
3  from vpython import *
4
5  x_ax = curve(pos=[vector(-15,0,0),vector(15,0,0)], color=color.red)
6  y_ax = curve(pos=[vector(0,-15,0),vector(0,15,0)], color=color.blue)
7  z_ax = curve(pos=[vector(0,0,-15),vector(0,0,15)], color=color.green)
8
9  ball = sphere(pos=vector(1,1,1), radius=0.3, color=color.cyan)
```

Now let's say that the object moved 3 units towards positive x-axis in 1sec. For this purpose, if we iterate the problem by hand using time step of 0.2 sec, in one dimension, it would look something like this;

$X_{new} = X_{old} + v\Delta t$

$X_{new} = 1 + 3_{units/sec} * 0.2 = 1.6$ units from the origin

$X_{new} = 1.6 + 3_{units/sec} * 0.2 = 2.2$ units from the origin

$X_{new} = 2.2 + 3$ units/sec $* 0.2 = 2.8$ units from the origin

$X_{new} = 2.8 + 3$ units/sec $* 0.2 = 3.2$ units from the origin

TASK ↦ Check for yourself if the above iteration outputs correspond to the same velocity we started with i.e. $v=+3$ units/sec.

We implement this iterative calculation using a `while` loop. We expect to see the iterative update in position of the ball.

```
11  # we use the velocity
12  v = vector(3,0,0)
13  # we define time step for now
14  dt = 0.2
15
16  while True:
17      rate(2)
18      ball.pos = ball.pos + v*dt
```

Great! Let's now control the duration of animation. With `True` the animation runs endlessly inside `while` loop but we can set a finite numerical condition for it to stop at time or our liking. Also, we'll be adding a print command to print the position of the ball at each step.

```
11  # we use the velocity
12  v = vector(3,0,0)
13  # we define time step for now
14  dt = 0.2
15  t = 0
16
17  while t<2:
18      rate(2)
19      ball.pos = ball.pos + v*dt
20      t = t + dt
21      print(ball.pos)
22
```

One thing you could not have missed is the `rate()` function. This is an option inside the `while` loop that allows us to control how fast the animation/simulation should proceed, basically a playback speed. You should keep it to some usual values. **But** keep in mind it has *nothing to do with dt*. The `dt` is what determines how far the next step in motion will be.

> **TASK** ↦ Reduce the time step *dt* and observe where the motion appears approximately smooth. You can increase the upper limit for `t` with the `while` condition.

Observe that the motion is still the same only the step size has reduced.

> **TASK** ↦ Select a velocity vector in arbitrary direction and observe the motion. *{to help trace the motion use* `make_trail=True` *in ball's parameters.}*

We also have arrows to represent motion, remember??? So make a pointer object right after the sphere object above, like this;

```
pointer = arrow(pos=vector(0,0,0), axis=ball.pos,
shaftwidth=0.2, opacity=0.8)
```

Now ask yourself, Should you run the program now??? What do you expect to happen???

> **TASK** ↦ Edit the `while` loop code so that this pointer becomes a position vector for the ball.