Modeling and Simulation in Python

Chapter 23

Lab Author @ Joseph Simone

```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import functions from the modsim.py module
from modsim import *
```

Code from the previous chapter

```
m = UNITS.meter
s = UNITS.second
kg = UNITS.kilogram
degree = UNITS.degree
```

degree

```
t_end = 20 * s
dt = t_end / 100

params = Params(x = 0 * m,
                y = 1 * m,
                g = 9.8 * m/s**2,
                mass = 145e-3 * kg,
                diameter = 73e-3 * m,
                rho = 1.2 * kg/m**3,
                C_d = 0.3,
                angle = 45 * degree,
                velocity = 40 * m / s,
                t_end=t_end,
                dt=dt)
```

values

x

0 meter

y

1 meter

g

9.8 meter / second ** 2

mass

0.145 kilogram

diameter

0.073 meter

rho

1.2 kilogram / meter ** 3

C_d

0.3

angle

45 degree

velocity

40.0 meter / second

t_end

20 second

dt

0.2 second

```
def make_system(params):
    """Make a system object.

    params: Params object with angle, velocity, x, y,
                diameter, duration, g, mass, rho, and C_d

    returns: System object
    """
    angle, velocity = params.angle, params.velocity

    # convert angle to degrees
    theta = np.deg2rad(angle)

    # compute x and y components of velocity
    vx, vy = pol2cart(theta, velocity)

    # make the initial state
    R = Vector(params.x, params.y)
    V = Vector(vx, vy)
    init = State(R=R, V=V)

    # compute area from diameter
    diameter = params.diameter
    area = np.pi * (diameter/2)**2

    return System(params, init=init, area=area)
```

```
def drag_force(V, system):
    """Computes drag force in the opposite direction of `V`.

    V: velocity Vector
    system: System object with rho, C_d, area

    returns: Vector drag force
    """
    rho, C_d, area = system.rho, system.C_d, system.area

    mag = rho * V.mag**2 * C_d * area / 2
    direction = -V.hat()
    f_drag = direction * mag
    return f_drag
```

```
def slope_func(state, t, system):
    """Computes derivatives of the state variables.

    state: State (x, y, x velocity, y velocity)
    t: time
    system: System object with g, rho, C_d, area, mass

    returns: sequence (vx, vy, ax, ay)
    """
    R, V = state
    mass, g = system.mass, system.g

    a_drag = drag_force(V, system) / mass
    a_grav = Vector(0, -g)

    A = a_grav + a_drag

    return V, A
```

```
def event_func(state, t, system):
    """Stop when the y coordinate is 0.

    state: State object
    t: time
    system: System object

    returns: y coordinate
    """
    R, V = state
    return R.y
```

Optimal launch angle

To find the launch angle that maximizes distance from home plate, we need a function that takes launch angle and returns range.

```
def range_func(angle, params):
    """Computes range for a given launch angle.

    angle: launch angle in degrees
    params: Params object

    returns: distance in meters
    """
    params = Params(params, angle=angle)
    system = make_system(params)
    results, details = run_ode_solver(system, slope_func, events=event_func)
    x_dist = get_last_value(results.R).x
    print(angle, x_dist)
    return x_dist
```

Let's test `range_func`.

```
range_func(45, params)
```

```
45 102.67621633303261 meter
```

$$102.67621633303261 \ meter$$

And sweep through a range of angles.

```
angles = linspace(20, 80, 21)
sweep = SweepSeries()

for angle in angles:
    x_dist = range_func(angle, params)
    sweep[angle] = x_dist
```

Plotting the `Sweep` object, it looks like the peak is between 40 and 45 degrees.

```
plot(sweep, color='C2')
decorate(xlabel='Launch angle (degree)',
         ylabel='Range (m)',
         title='Range as a function of launch angle',
         legend=False)

savefig('figs/chap23-fig01.pdf')
```

We can use `maximize` to search for the peak efficiently.

```
bounds = [0, 90] * degree
res = maximize(range_func, bounds, params)
```

`res` is an `ModSimSeries` object with detailed results:

```
res
```

`x` is the optimal angle and `fun` the optional range.

```
optimal_angle = res.x
```

```
max_x_dist = res.fun
```

Under the hood

Read the source code for `maximize` and `minimize_scalar`, below.

Add a print statement to `range_func` that prints `angle`. Then run `maximize` again so you can see how many times it calls `range_func` and what the arguments are.

```
source_code(maximize)
```

```
source_code(minimize_scalar)
```

The Manny Ramirez problem

Finally, let's solve the Manny Ramirez problem:

What is the minimum effort required to hit a home run in Fenway Park?

Fenway Park is a baseball stadium in Boston, Massachusetts. One of its most famous features is the "Green Monster", which is a wall in left field that is unusually close to home plate, only 310 feet along the left field line. To compensate for the short distance, the wall is unusually high, at 37 feet.

Although the problem asks for a minimum, it is not an optimization problem. Rather, we want to solve for the initial velocity that just barely gets the ball to the top of the wall, given that it is launched at the optimal angle.

And we have to be careful about what we mean by "optimal". For this problem, we don't want the longest range, we want the maximum height at the point where it reaches the wall.

If you are ready to solve the problem on your own, go ahead. Otherwise I will walk you through the process with an outline and some starter code.

As a first step, write a function called `height_func` that takes a launch angle and a params as parameters, simulates the flights of a baseball, and returns the height of the baseball when it reaches a point 94.5 meters (310 feet) from home plate.

```
def event_func(state, t, system):
    """Stop when the y coordinate is 0.

    state: State object
    t: time
    system: System object

    returns: y coordinate - Green Monster
    """
    R, V = state
    return R.x - system.green_monster
```

Always test the slope function with the initial conditions.

```
system = make_system(params)
system.set(green_monster = 94.5 * m)
event_func(system.init, 0, system)
```

```
def height_func(angle, params):

    params = Params(params, angle=angle)
    system = make_system(params)
    system.set(green_monster = 94.5 * m)

    results, details = run_ode_solver(system, slope_func, events=event_func)
    height = get_last_value(results.R).y

    return height
```

Test your function with a launch angle of 45 degrees:

```
height_func(45 * degree, params)
```

10.953200948514532 *meter*

Now use `maximize` to find the optimal angle. Is it higher or lower than the angle that maximizes range?

```
b = [0, 90] * degree
r = maximize(height_func, b, params)
```

values

success

True

x

44.25082945182002 degree

fun

10.967930469934105 meter

```
oa = r.x
```

44.25082945182002 *degree*

```
oh = r.fun
```

10.967930469934105 *meter*

With initial velocity 40 m/s and an optimal launch angle, the ball clears the Green Monster with a little room to spare.

Which means we can get over the wall with a lower initial velocity.

Finding the minimum velocity

Even though we are finding the "minimum" velocity, we are not really solving a minimization problem. Rather, we want to find the velocity that makes the height at the wall exactly 11 m, given given that it's launched at the optimal angle. And that's a job for `root_bisect`.

Write an error function that takes a velocity and a `Params` object as parameters. It should use `maximize` to find the highest possible height of the ball at the wall, for the given velocity. Then it should return the difference between that optimal height and 11 meters.

```
# Solution goes here
```

Test your error function before you call `root_bisect`.

Then use `root_bisect` to find the answer to the problem, the minimum velocity that gets the ball out of the park.

```
# Solution goes here
```

```
# Solution goes here
```

And just to check, run `error_func` with the value you found.

```
# Solution goes here
```