Bungee Jumping Drop

Shreya Chowdhary & Gail Romer

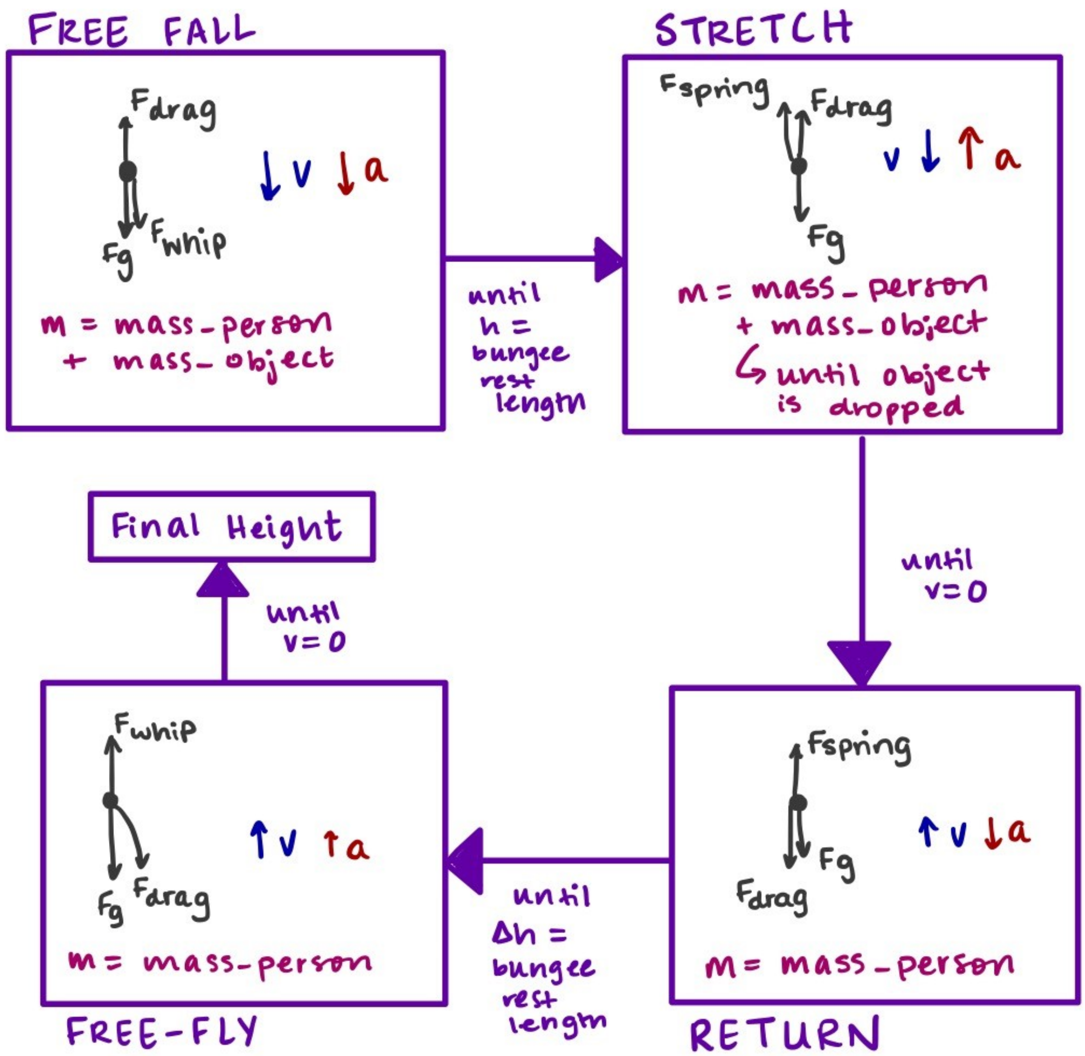
In [1]:
```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# 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 *
```

In [2]:
```
m = UNITS.meter
s = UNITS.second
kg = UNITS.kilogram
N = UNITS.newton;
```

If a bungee jumper falls while carrying an object, and drops at the lowest point in their jump, how much must the object weigh for the jumper to return to their initial starting height?

Model

Schematic Diagram



Differential Equations

Free Fall Stage

$$\frac{dh}{dt} = v$$
$$\frac{dv}{dt} = a$$

where

$$a = \frac{F_g + F_{whip} + F_{drag}}{m_{person} + m_{object}}$$

Stretch Stage

$$\frac{dh}{dt} = v$$
$$\frac{dv}{dt} = a$$

where

$$a = \frac{F_g + F_{drag} + F_{spring}}{m_{person} + m_{object}}$$

Return Stage

Same as the stretch stage, but with only the mass of the person.

Free Fly Stage

Same as the free fall stage, but with only the mass of the person.

and implementation in Python(to follow). (M) An explanation of the decisions you made in creating your model. (E.g., What assumptions are baked in? Why did you choose those? Are they good or bad assumptions?) These explanations should be thorough and belong in the essay portion of the text, not in code comments.

Assumptions

- First, we used Google to find the average weight for a human and assumed this would be a valid assumption based on the credibility of [the source (https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-12-439)](https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-12-439).
- To simplify calculations, we assumed a spherical shape for a human. It is difficult to determine the drag of a bungee jumper as humans are inconveniently-shaped and may fall in many different orientations. This simplification is based in reality, however, as many [videos of bungee jumpers (https://www.youtube.com/watch?v=g3pq1Pn_Mag)](https://www.youtube.com/watch?v=g3pq1Pn_Mag) show them curling into a ball as they fall.

Python Implementation

In [3]:
```
def make_system(params):
    """Makes a system based on params input, including init state
        Calculates necessary values based on parameters"""

    init = State(h = params.init_height, v = 0 * m/s)

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

    mu = params.mass_bungee/(params.mass_human + params.mass_object)

    system = System(params, init = init, area = area, mu = mu)

    return system
```

Force Functions

In [4]:
```
def drag_force(v, system):
    """Calculates drag force from system variables and a given velocity"""

    unpack(system)

    return -np.sign(v) * rho * v**2 * C_d * area / 2
```

In [5]:
```
def whip_force(system, state):
    """Calculates whip force from system and state variables"""

    unpack(system)

    h, v = state

    a = np.sign(v)*(mu * (v**2)/2)/(mu*(bungee_rest_length + h) + 2*bungee_rest_le

    return total_mass*a
```

In [6]:
```
def spring_force(h, system):
    """Calculates spring force from height difference and length of bungee cord"""

    unpack(system)

    return k*((init_height - h) - bungee_rest_length)
```

Slope Functions

In [7]:
```
def slope_stage_14(state, t, system): #Stages 1 & 4 involve the same forces
    """Slope function determines change in height and change in velocity due to fo
       Used in free-fall and free-fly stages"""

    unpack(system)
    h, v = state

    f_drag = drag_force(v, system)
    f_whip = whip_force(system, state)
    f_grav = (total_mass)*(-g)

    net_force = f_drag + f_whip + f_grav
#     Calculates acceleration from net force
    a = net_force/total_mass

    dhdt = v
    dvdt = a

    return dhdt, dvdt
```

In [8]:
```
def slope_stage_23(state, t, system):
    """Slope function determines change in height and change in velocity due to fo
       Used in stretch and return stages"""

    unpack(system)
    h, v = state

    f_drag = drag_force(v, system)
    f_spring = spring_force(h, system)
    f_grav = (total_mass)*(-g)

    net_force = f_drag + f_spring + f_grav
#     Calculates acceleration from net force
    a = net_force/total_mass

    dhdt = v
    dvdt = a

    return dhdt, dvdt
```

####Plot Functions

In [9]:
```
def plot_height(heights, stage):
    """Creates and labels plot of Height vs. Time"""
    plot(heights)
    title = "Height vs. Time for " + stage
    decorate(title = title, xlabel="Time (s)", ylabel="Height (m)")
    savefig("/graphs/height-" + stage +".png")
```

In [10]:
```
def plot_velocity(velocities, stage):
    """Creates and labels plot of Velocity vs. Time"""
    plot(velocities)
    title = "Velocity vs. Time for " + stage
    decorate(title = title, xlabel="Time (s)", ylabel="Velocity (m/s)")
    savefig("/graphs/velocity-" + stage + ".png")
```

Stage 1: Freefall

In [11]:
```
def free_fall_event(state, t, system):
    """Event function returns 0 when height difference is length of bungee cord"""

    unpack(system)

    h, v = state

    return bungee_rest_length - (init_height - h)
```

In [12]:
```
def free_fall(sys):
    """Finds Height and Velocity vs. Time for free fall stage, returns an updated

    unpack(sys)
    system = System(sys, total_mass = sys.mass_human + sys.mass_object)

#     Solves for height and velocity over time for free-fall stage
    free_fall_results, details = run_ode_solver(system, slope_stage_14, events = f

    h_final = get_last_value(free_fall_results.h) * m
    v_final = get_last_value(free_fall_results.v) * m/s

#     Plots data for free fall stage if p is true
    if p:
        plot_height(free_fall_results.h, "Free Fall Stage")
        plt.figure()
        plot_velocity(free_fall_results.v, "Free Fall Stage")


#     Creates new system and state from the end of free fall stage
    free_fall_state = State(h = h_final, v = v_final)
    t_final = get_last_label(free_fall_results) * s
    system = System(system, init = free_fall_state, t_0 = t_final)

    return system, free_fall_results
```

Stage 2: Stretch

In [13]:
```
def stretch_event(state, t, system):
    """Event function returns 0 when velocity is 0"""

    unpack(system)

    h, v = state

    return v
```

In [14]:
```
def stretch(sys):
    """Finds Height and Velocity vs. Time for stretch stage, returns an updated sy

    unpack(sys)
    system = System(sys, total_mass = sys.mass_human + sys.mass_object)

#     Solves for height and velocity over time for stretch stage
    stretch_results, details = run_ode_solver(sys, slope_stage_23, events = stretc


    h_final = get_last_value(stretch_results.h) * m
    v_final = get_last_value(stretch_results.v) * m/s

#     Plots data for stretch stage if p is true
    if p:
        plot_height(stretch_results.h, "Stretch Stage")
        plt.figure()
        plot_velocity(stretch_results.v, "Stretch Stage")

#     Creates new system and state from the end of stretch stage
    stretch_state = State(h = h_final, v = v_final)
    t_final = get_last_label(stretch_results) * s
    system = System(sys, init = stretch_state, t_0 = t_final)

    return system, stretch_results
```

Stage 3: Return

In [15]:
```
def return_event(state, t, system):
    """Event function returns 0 when height difference is length of bungee cord"""

    unpack(system)

    h, v = state

    return (init_height - h) - bungee_rest_length
```

In [16]:
```
def bungee_return(sys):
    """Finds Height and Velocity vs. Time for return stage, returns an updated sys

    unpack(sys)
    sys = System(sys,total_mass = mass_human)

#     Solves for height and velocity over time for return stage
    return_results, details = run_ode_solver(sys, slope_stage_23, events = return_

    h_final = get_last_value(return_results.h) * m
    v_final = get_last_value(return_results.v) * m/s

#     Plots data for return stage if p is true
    if p:
        plot_height(return_results.h, "Return Stage")
        plt.figure()
        plot_velocity(return_results.v, "Return Stage")

#     Creates new system and state from the end of return stage
    return_state = State(h = h_final, v = v_final)
    t_final = get_last_label(return_results) * s
    system = System(sys, init = return_state, t_0 = t_final)

    return system, return_results
```

Stage 4: Freefly

In [17]:
```
def free_fly_event(state,t,system):
    """Event function returns 0 when velocity is 0"""

    unpack(system)

    h, v = state

    return v
```

In [18]:
```
def free_fly(sys):
    """Finds Height and Velocity vs. Time for free-fly stage, returns an updated s

    unpack(sys)
    system = System(sys, total_mass = sys.mass_human + sys.mass_object)

#     Solves for height and velocity over time for free-fly stage
    free_fly_results, details = run_ode_solver(system, slope_stage_14, events = fr

    h_final = get_last_value(free_fly_results.h) * m
    v_final = get_last_value(free_fly_results.v) * m/s

#     Plots data for free-fly stage if p is true
    if p:
        plot_height(free_fly_results.h, "Free Fly Stage")
        plt.figure()
        plot_velocity(free_fly_results.v, "Free Fly Stage")

#     Creates new system and state from the end of free-fly stage
    free_fly_state = State(h = h_final, v = v_final)
    t_final = get_last_label(free_fly_results) * s
    system = System(system, init = free_fly_state, t_0 = t_final)

    return system, free_fly_results
```

Params, Error Functions, and FSolve

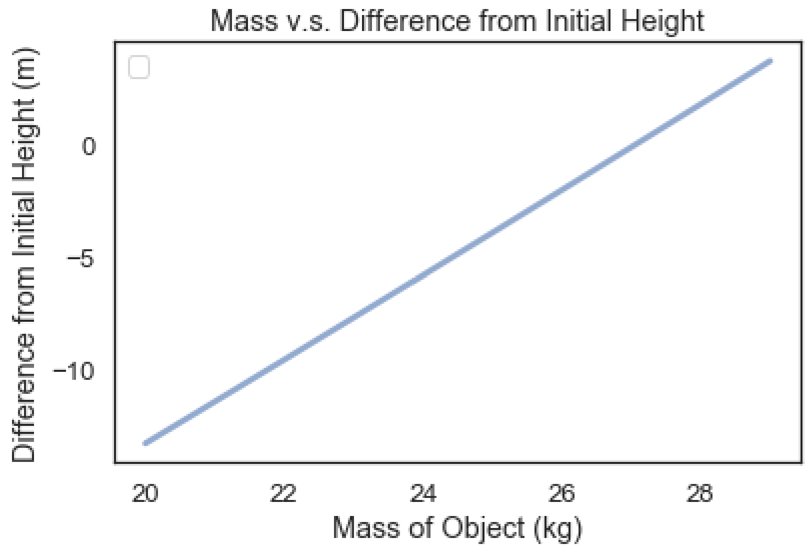
In [19]:
```
params = Params(init_height = 220 * m,
                g = 9.8 * m/s**2,
                mass_human = 62 * kg,
                mass_object = 10 * kg,
                diameter = 0.8128 * m,
                rho = 1.2 * kg/m**3,
                k = 40 * N/m,
                bungee_rest_length = 100 * m,
                t_0 = 0 *s,
                t_end = 2000 * s,
                dt = .01*s,
                C_d = 0.47,
                mass_bungee = 75 * kg,
                p=False);
```

In [20]:
```
def bungee_drop(mass, params):
    """Error function - returns 0 when final height equals initial height"""

    params = Params(params, mass_object = mass*kg)
    system = make_system(params)

#    Runs each stage, inputting system/state created from previous stage
    free_fall_system, free_fall_results= free_fall(system)
    stretch_system, stretch_results = stretch(free_fall_system)
    return_system, return_results = bungee_return(stretch_system)
    free_fly_system, free_fly_results= free_fly(return_system)

    return get_last_value(free_fly_results.h) * m - params.init_height
```

In [21]:
```
def sweep_mass(params):
    """Sweeps the mass of the object
       produces a plot of mass vs difference between final and initial height"""

#    Creates a Timeseries and array for sweeping
    mass_series = TimeSeries()
    mass_array = linrange(20,30,1)
    print(mass_array)

#    Sweeps mass and calculates difference between final and initial height
    for mass in mass_array:
        m = mass
        results = bungee_drop(m,params)
        mass_series[mass] = results
#    Plots swept data
    plot(mass_series)
    decorate(title = "Mass v.s. Difference from Initial Height",
            xlabel = "Mass of Object (kg)",
            ylabel = "Difference from Initial Height (m)")
    savefig("mass-sweep.png")
```

In [22]:
```
sweep_mass(params)
```

```
[20 21 22 23 24 25 26 27 28 29]
Saving figure to file mass-sweep.png
```



In [*]:
```
# Solves for mass that makes the difference between final and initial height equal
mass = fsolve(bungee_drop, 30, params)[0]
```

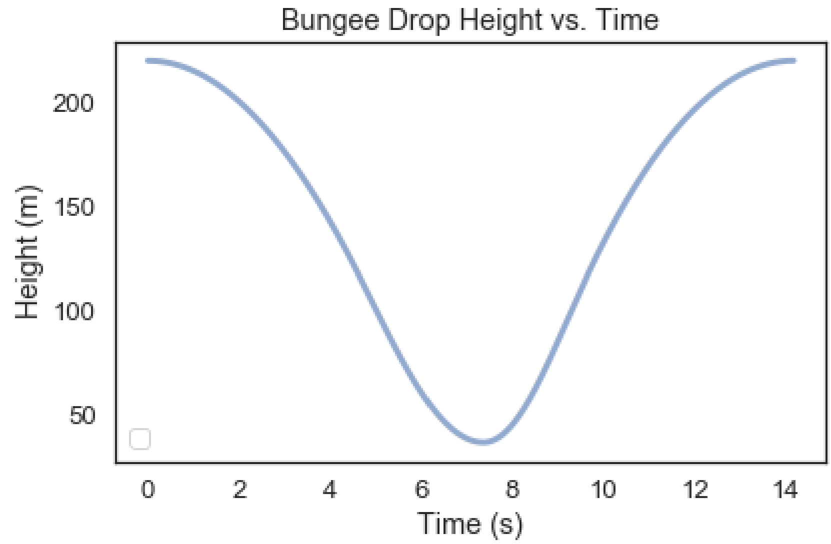
In [*]:
```
# Checks the mass produced by fsolve
bungee_drop(mass, params)
```

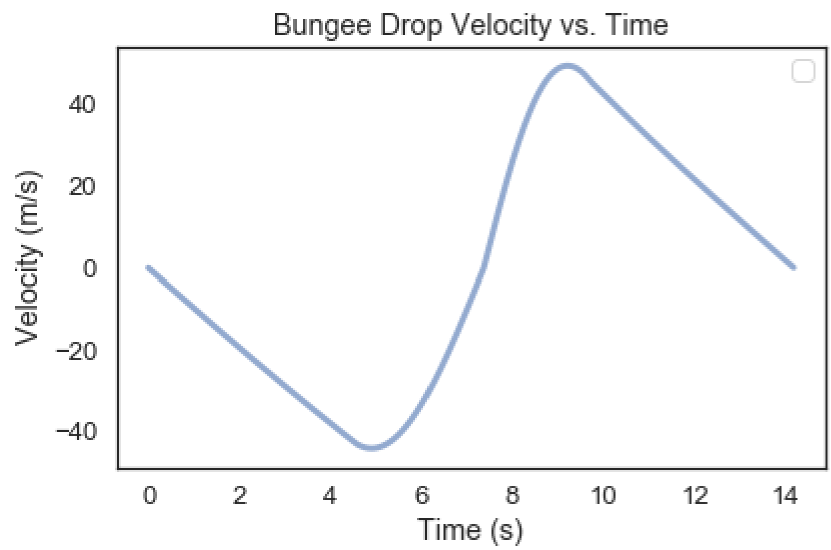
Final Plots

In [*]:
```
def plot_final(mass, params):
    """Runs, concatinates, and plots results from the four stages based on mass fo

    params = Params(params, mass_object = mass*kg)
    system = make_system(params)

#     Runs all stages
    free_fall_system, free_fall_results= free_fall(system)
    stretch_system, stretch_results = stretch(free_fall_system)
    return_system, return_results = bungee_return(stretch_system)
    free_fly_system, free_fly_results= free_fly(return_system)

#     Concatinates results into one array
    frames = [free_fall_results, stretch_results, return_results, free_fly_results
    result = pd.concat(frames)

#     Plots and labels figure of Height vs. Time
    plot(result.h)
    decorate(title = "Bungee Drop Height vs. Time", ylabel = "Height (m)", xlabel
    savefig("final_height.png")

#     Plots and labels New figure of Velocity vs. Time
    plt.figure()
    plot(result.v)
    decorate(title = "Bungee Drop Velocity vs. Time", ylabel = "Velocity (m/s)", x
    savefig("final_velocity.png")

#     Prints last value of height
    print(get_last_value(result.h))


plot_final(mass,params)
```

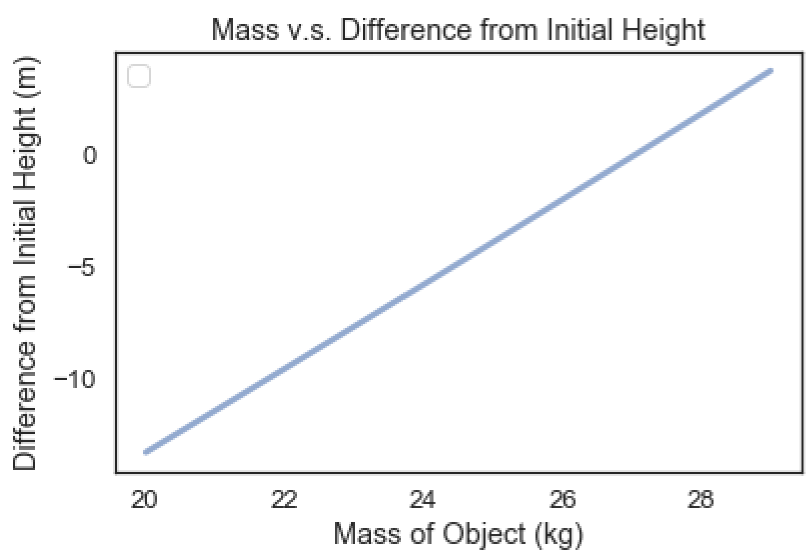
In [*]:
```
print(mass)
```

Results





According to our model, a bungee jumper of 62 kg should drop a mass of **27 kg** in order to return to their initial height.



The sweep of the mass v.s. the difference from the initial height illustrates how a mass of 27 kg causes the bungee jumper return to their final height.

Interpretation

We spent a lot of our time iterating on our model to get the stages to work correctly. We started by only accounting for the spring force and the force of gravity without including the change in mass. This allowed us to check that our stages and graphs worked out as they should as well as allowing us to fix bugs in the force functions. We then tested the stages in isolated batches before stringing them together to be sure each of them worked as well. Once the bugs were worked out, we added in the drag force, change in mass, and finally the whip force into the model. If we were to take this model further, we might consider making the model applicable to a larger variety of jumping styles by accounting for a change in angular position of the jumper, which would lead to a change in the drag coefficients based on their orientation. This would apply to scenarios where the jumper does not curl into a ball, and therefore doesn't stay in the exact same orientation in respect to the dirrection of their velocity. Though we could always take the model further, we believe our model is a fairly good representation of the bungee jumping system as it takes into account all of the forces on the bungee jumper. Our baseline assumptions that could cause discrepencies between our model and the real world system were that the jumper is a sphere and that the object the jumper carries doesn't change the drag coefficient.

According to https://www.bluebulbprojects.com/measureofthings/results.php?comp=weight&unit=kgms&amt=27&sort=pr&p=2 (https://www.bluebulbprojects.com/measureofthings/results.php?comp=weight&unit=kgms&amt=27&sort=pr&p=2) the 62kg bungee jumper should be carrying either 4 bowling balls, 5 gallons of paint, 6 cats, 9.5 bricks, 12 chihuahuas, 20 human brains, or 45 basketballs (some of which would be unreasonable because they would effect drag).