

2D to 3D Isotropic Oscillators with Drag - Shuttlecock Services

Using the ODEs from JMA Danby:

$$\frac{d^2x}{dt^2} = -k_D * v * v_x \downarrow$$

$$\frac{d^2x}{dt^2} = -k_D * \sqrt{\dot{x}^2 + \dot{z}^2} * \dot{x}$$

$$\frac{d^2z}{dt^2} = -k_D * v * v_z - g \downarrow$$

$$\frac{d^2z}{dt^2} = -k_D * \sqrt{\dot{x}^2 + \dot{z}^2} * \dot{z} - g$$

Given $v = \sqrt{v_x^2 + v_z^2}$

$$v_{\text{term_shuttlecock}} = \sqrt{mg/k} \approx 22.3 \text{ ft/sec}$$

and $k_D = 0.0643 \text{ ft}^{-1}$

Everything is measured in imperial units, as my primary source, JMA Danby, uses them for recorded values.

Use Euler method to implement in a "for" loop in Python

Import Libraries

```
In [1]: # prepares notebook for "inline" graphing, and imports libraries that we'll use later.
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from numpy import * # imports all of numpy without "np" nickname
```

Start Code Exploration - Test Euler Method in 2D

Initialize Variables and Init. Arrays; Input to Array

```
In [2]: # Constants and Variables
m = (0.18 / 16) # lbs (since everything is in feet in J.M.A. Danby; 0.18 oz divided by
t_0 = 0
x_0 = 0
z_0 = (1.15 * 3.281) # ft (1.15 m times 3.281 ft/m) - https://rb.gy/lwa9pw
# v = 50 ft/s, and if v = sqrt(v_x**2 + v_z**2), we'll assume a serving angle of 20 de
```

```

v_x_0 = 50 * np.cos(np.pi * 85 / 180)
v_z_0 = 50 * np.sin(np.pi * 85 / 180)
k_D = 0.00643 # ft^-1
g = 32.17 # ft/s^2

# Start time variations of Position and Velocity
delta_t = 0.0001
Numerical_pos_x = np.array([])
Numerical_pos_z = np.array([])
Numerical_v_x = np.array([])
Numerical_v_z = np.array([])
Numerical_speed = np.array([])
time = np.array([])
t_min = 0
t_max = 1.5 # Analytical Guess from Changing t_max and viewing graphs
n = round((t_max-t_min)/ delta_t)

# Put initialized vars into first rows of arrays
Numerical_pos_x = np.append(Numerical_pos_x, x_0)
Numerical_pos_z = np.append(Numerical_pos_z, z_0)
Numerical_v_x = np.append(Numerical_v_x, v_x_0)
Numerical_v_z = np.append(Numerical_v_z, v_z_0)
time = np.append(time, t_0)

x = Numerical_pos_x[0] # initaial x value
z = Numerical_pos_z[0] # initial y value
x_dot = Numerical_v_x[0] # initial v_x value
z_dot = Numerical_v_z[0] # initial v_y value
t = time[0] # initial t value

```

Initial Euler ODE Calculations & Plots (Check if Code Works -> More Realistic)

```

In [3]: for i in range(1, n): #For all the time steps
    x_ddot = -k_D * sqrt(x_dot**2 + z_dot**2) * x_dot
    z_ddot = -k_D * sqrt(x_dot**2 + z_dot**2) * z_dot - g

    x_dot += x_ddot*delta_t #Euler method equation
    z_dot += z_ddot*delta_t

    x += x_dot*delta_t
    z += z_dot*delta_t

    t += delta_t #Update t to a new value based on Δt

    Numerical_pos_x = np.append(Numerical_pos_x, x)
    Numerical_pos_z = np.append(Numerical_pos_z, z) #Put your newest y value at the end
    Numerical_v_x = np.append(Numerical_v_x, x_dot)
    Numerical_v_z = np.append(Numerical_v_z, z_dot)

    time = np.append(time, t) #Put your newest t value at the end of your table of t

# PLOT!
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2,sharex=True)
fig.suptitle('Positions & Velocities, both x & z', fontsize=14)

```

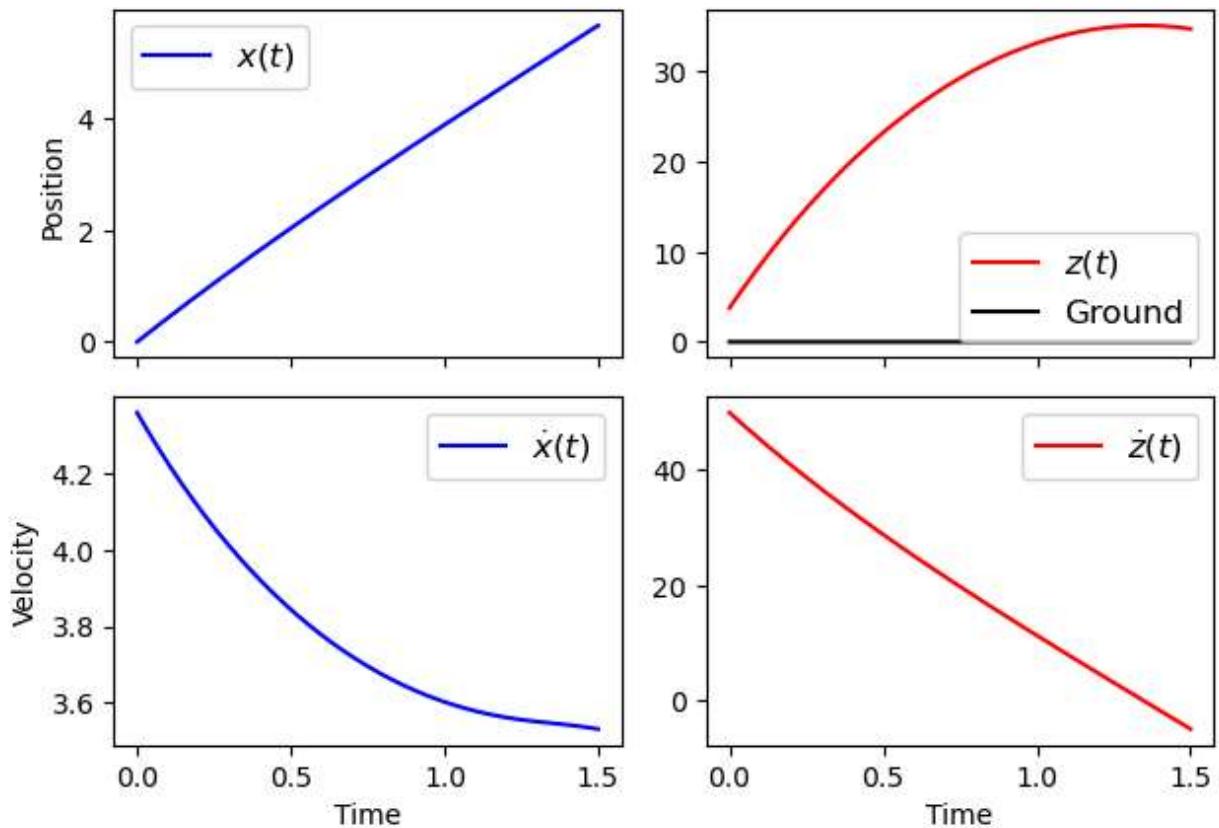
```
x_position = ax1.plot(time, Numerical_pos_x, 'b', label=r'$x(t)$')
z_position = ax2.plot(time, Numerical_pos_z, 'r', label=r'$z(t)$')
ground_line = ax2.plot(time, np.linspace(0,0,n), 'black', label=r'Ground')

x_velocity = ax3.plot(time, Numerical_v_x, 'b', label=r'$\dot{x}(t)$')
z_velocity = ax4.plot(time, Numerical_v_z, 'r', label=r'$\dot{z}(t)$')

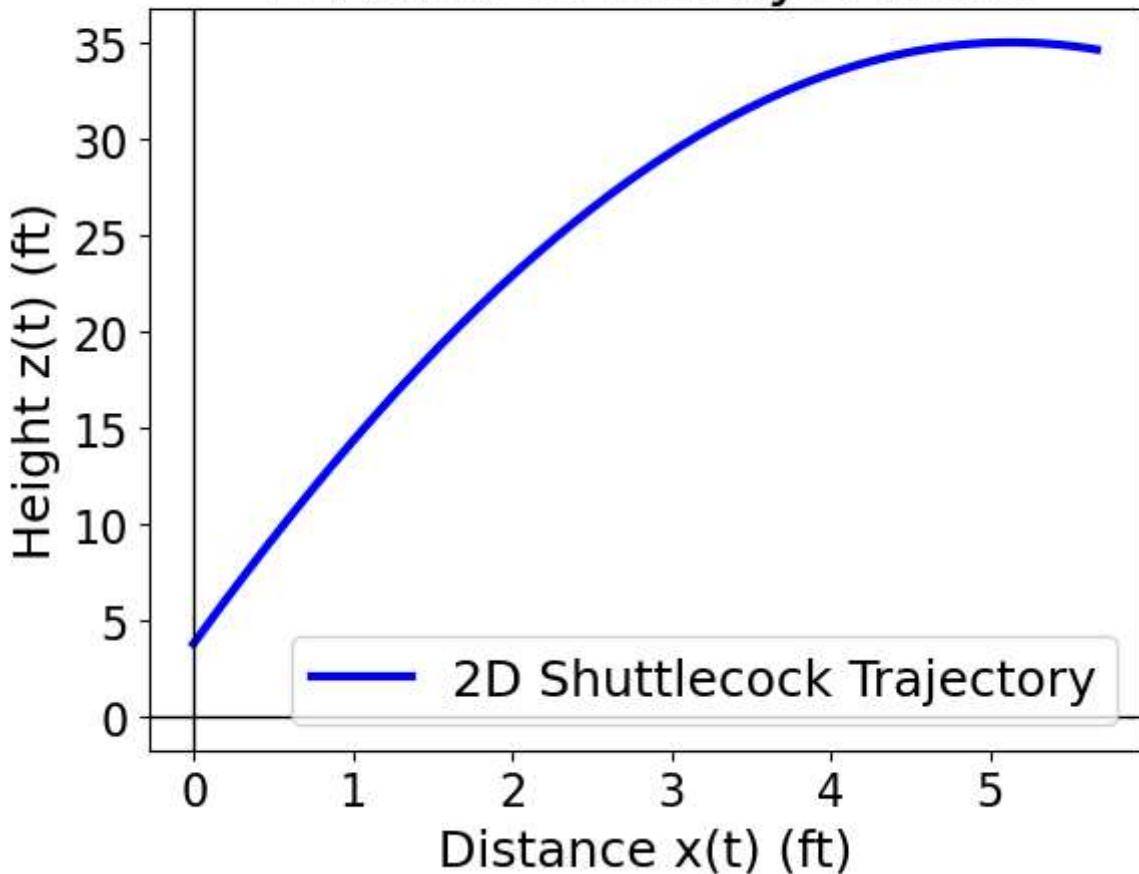
ax1.set_ylabel('Position')
ax3.set_ylabel('Velocity')
ax3.set_xlabel('Time')
ax4.set_xlabel('Time')

ax1.legend(loc='best', fontsize = 12)
ax2.legend(loc='best', fontsize = 12)
ax3.legend(loc='best', fontsize = 12)
ax4.legend(loc='best', fontsize = 12)
fig.tight_layout()
plt.show()

z_vs_x = plt.plot(Numerical_pos_x, Numerical_pos_z, 'b', label="2D Shuttlecock Traject
plt.title("Height vs Distance\nA Mimic of Danby's Work", fontsize=22)
plt.axhline(0, color='black', lw=1)
plt.axvline(0, color='black', lw=1)
plt.xlabel('Distance x(t) (ft)', fontsize=18)
plt.xticks(fontsize=16)
plt.ylabel('Height z(t) (ft)', fontsize=18)
plt.yticks(fontsize=16)
plt.legend(loc='best', fontsize=18)
plt.show()
```

Positions & Velocities, both x & z 

Height vs Distance A Mimic of Danby's Work



Add Y-Component (Test 3D Models), Model Best Services From Right Court, Initial Velocity of 10 m/s

ϕ is xz angle, θ is yz angle

3D Variables and Arrays; Input to Array

```
$ddot{x} = -k_D * sqrt{dot{x}^2 + dot{y}^2 + dot{z}^2} * dot{x}
$ddot{y} = -k_D * sqrt{dot{x}^2 + dot{y}^2 + dot{z}^2} * dot{y}
$ddot{z} = -k_D * sqrt{dot{x}^2 + dot{y}^2 + dot{z}^2} * dot{z} - g
```

Prepping 3 Dimensions

```
In [4]: # Constants and Variables
m = (0.18 / 16) # lbs (since everything is in feet in J.M.A. Danby; 0.18 oz divided by
t_0 = 0
x_0 = 8.5 # ft (if back court line is 0.0 ft, then service line is 13.5 ft, given the
y_0 = 13.5 # ft (if far-left court line is 0.0 ft, then divider is 8.5 ft, given the
z_0 = (1.15 * 3.281) # ft (1.15 m times 3.281 ft/m) - https://rb.gy/lwa9pw
```

```
# v = 10 m/s, so (82*3.281) ft/s, and if v = sqrt(v_x**2 + v_y**2 + v_z**2)
v_0 = 10 * 3.281 # ft/s
theta = np.pi * 77 / 180
phi = np.pi * 8 / 180
v_x_0 = v_0 * np.sin(phi)
v_y_0 = v_0 * np.cos(theta) * np.cos(phi)
v_z_0 = v_0 * np.sin(theta) * np.cos(phi)
k_D = 0.00643 # ft^-1
g = 32.17 # ft/s^2

# Start time variations of Position and Velocity
delta_t = 0.001
Numerical_pos_x = np.array([])
Numerical_pos_y = np.array([])
Numerical_pos_z = np.array([])
Numerical_v_x = np.array([])
Numerical_v_y = np.array([])
Numerical_v_z = np.array([])
Numerical_speed = np.array([])
time = np.array([])
t_min = 0
t_max = 3 # Analytical Guess from Changing t_max and viewing graphs
n = round((t_max-t_min)/ delta_t)

# Put initialized vars into first rows of arrays
Numerical_pos_x = np.append(Numerical_pos_x, x_0)
Numerical_pos_y = np.append(Numerical_pos_y, y_0)
Numerical_pos_z = np.append(Numerical_pos_z, z_0)
Numerical_v_x = np.append(Numerical_v_x, v_x_0)
Numerical_v_y = np.append(Numerical_v_y, v_y_0)
Numerical_v_z = np.append(Numerical_v_z, v_z_0)
time = np.append(time, t_0)

x = Numerical_pos_x[0] # initaial x value
y = Numerical_pos_y[0] # initaial y value
z = Numerical_pos_z[0] # initial z value
x_dot = Numerical_v_x[0] # initial v_x value
y_dot = Numerical_v_y[0] # initial v_y value
z_dot = Numerical_v_z[0] # initial v_z value
t = time[0] # initial t value
```

In [5]:

```
# Euler iterations
for i in range(1, n): #For all the time steps
    x_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * x_dot
    y_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * y_dot
    z_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * z_dot - g

    x_dot += x_ddot*delta_t #Euler method equation
    y_dot += y_ddot*delta_t
    z_dot += z_ddot*delta_t

    x -= x_dot*delta_t # Starting from midline, serving to the Left (-x direction)
    y += y_dot*delta_t
    z += z_dot*delta_t

    t += delta_t #Update t to a new value based on Δt

    Numerical_pos_x = np.append(Numerical_pos_x, x) #Put your newest x value at the end
```

```
Numerical_pos_y = np.append(Numerical_pos_y, y)
Numerical_pos_z = np.append(Numerical_pos_z, z)
Numerical_v_x = np.append(Numerical_v_x, x_dot)
Numerical_v_y = np.append(Numerical_v_y, y_dot)
Numerical_v_z = np.append(Numerical_v_z, z_dot)

time = np.append(time, t) #Put your newest t value at the end of your table of t
```

2D Plots of a 3D System!

```
In [6]: fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(2,3,sharex=True)
fig.suptitle('Positions & Velocities, x, y, & z', fontsize=14)

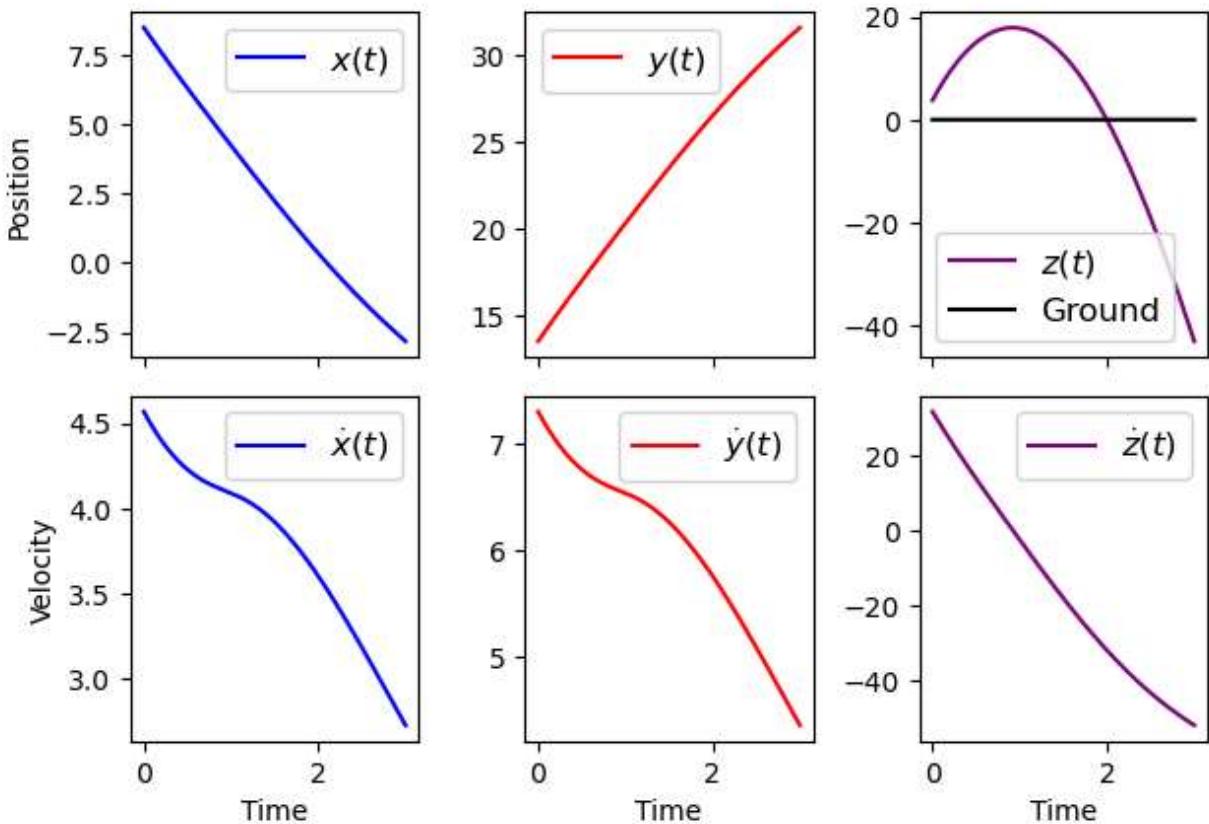
x_position = ax1.plot(time, Numerical_pos_x, 'b', label=r'$x(t)$')
y_position = ax2.plot(time, Numerical_pos_y, 'r', label=r'$y(t)$')
z_position = ax3.plot(time, Numerical_pos_z, 'purple', label=r'$z(t)$')
ground_line = ax3.plot(time, np.linspace(0,0,n), 'black', label=r'Ground')

x_velocity = ax4.plot(time, Numerical_v_x, 'b', label=r'$\dot{x}(t)$')
y_velocity = ax5.plot(time, Numerical_v_y, 'r', label=r'$\dot{y}(t)$')
z_velocity = ax6.plot(time, Numerical_v_z, 'purple', label=r'$\dot{z}(t)$')

ax1.set_ylabel('Position')
ax4.set_ylabel('Velocity')
ax4.set_xlabel('Time')
ax5.set_xlabel('Time')
ax6.set_xlabel('Time')

ax1.legend(loc='best', fontsize = 12)
ax2.legend(loc='best', fontsize = 12)
ax3.legend(loc='best', fontsize = 12)
ax4.legend(loc='best', fontsize = 12)
ax5.legend(loc='best', fontsize = 12)
ax6.legend(loc='best', fontsize = 12)
fig.tight_layout()
plt.show()
```

Positions & Velocities, x, y, & z



Solving for 1 Ideal Service - 3D Plot

```
In [7]: ax = plt.axes(projection='3d')
ax.set_title("3-Dimensional Shuttlecock Shot")
# ax.set_ahline(0, color='black', lw=1)
# ax.set_axvline(0, color='black', lw=1)
ax.set_xlabel('x(t) (ft)')
ax.set_ylabel('y(t) (ft)')
ax.set_zlabel('z(t) (ft)')

for hits_ground in range(len(time)):
    if Numerical_pos_z[hits_ground] <= 0:
        # print(hits_ground)
        z_vs_xy = ax.plot(Numerical_pos_x[:hits_ground], Numerical_pos_y[:hits_ground],
                           start_point = ax.plot(x_0, y_0, z_0, 'blue', marker='x', label=r'Serve Start')
                           start_point_ground = ax.plot(x_0, y_0, 0, 'blue', marker='x')
                           start_drop_down = ax.plot(linspace(x_0, x_0, n), linspace(y_0, y_0, n), linspace(z_0, 0, n))
                           end_point = ax.plot(Numerical_pos_x[hits_ground], Numerical_pos_y[hits_ground], 'blue', marker='x')

    # Plotting Court
    court_back = ax.plot(linspace(0,17,n), linspace(0,0,n), linspace(0,0,n), 'black')
    court_left = ax.plot(linspace(0,0,n), linspace(0,40,n), linspace(0,0,n), 'black')
    court_up = ax.plot(linspace(0,17,n), linspace(40,40,n), linspace(0,0,n), 'black')
    court_right = ax.plot(linspace(17,17,n), linspace(0,40,n), linspace(0,0,n), 'black')
    long_mid = ax.plot(linspace(8.5,8.5,n), linspace(0,40,n), linspace(0,0,n), 'black')
    service_line_1 = ax.plot(linspace(0,17,n), linspace(13.5,13.5,n), linspace(0,0,n), 'black')
    service_line_2 = ax.plot(linspace(0,17,n), linspace(25.5,25.5,n), linspace(0,0,n), 'black')
```

```

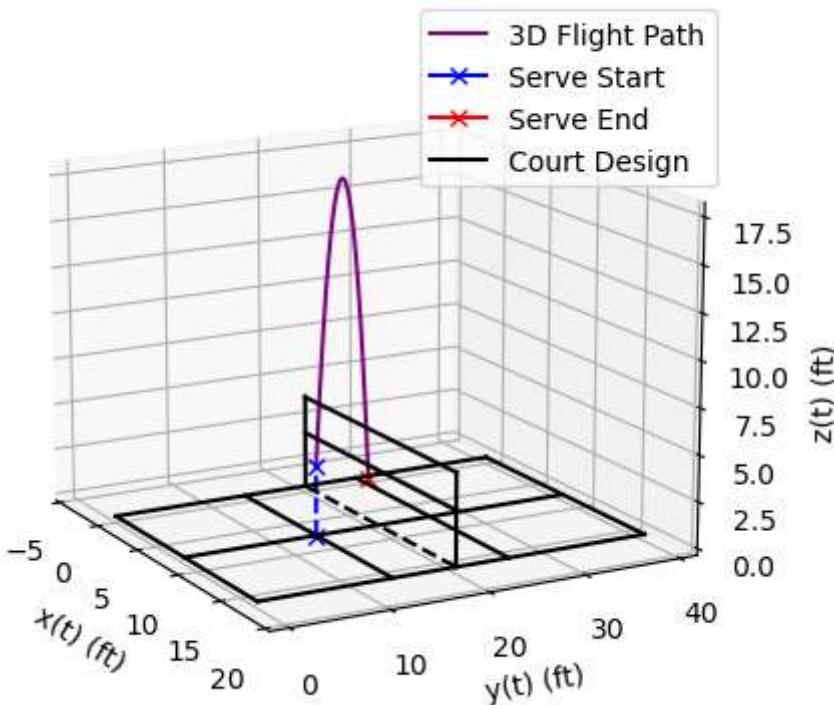
net_left = ax.plot(linspace(0,0,n), linspace(20,20,n), linspace(0,5,n), 'black')
net_right = ax.plot(linspace(17,17,n), linspace(20,20,n), linspace(0,5,n), 'black')
net_bottom = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(3,3,n), 'black')
net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(5,5,n), 'black')
net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(0,0,n), 'black')

# Viewing Angle
# for azimuth in range(0,30,15):
#     print(azimuth)
ax.view_init(azim=29, elev=15)
ax.set_xlim3d(-5, 20)
ax.set_ylim3d(-1, 41)
# ax.set_zlim3d(0, 6)

ax.legend(loc='best', fontsize=10)
ax.figsize=(100,100,100)
plt.show()
break

```

3-Dimensional Shuttlecock Shot



Function for 3D Modeling (Indiv. Graphs)!

```

In [8]: # Given initial v is 10 m/s or ~32.81 ft/s
def Service_Deg(theta, phi, title, azm, elev):
    # Constants and Variables
    m = (0.18 / 16) # lbs (since everything is in feet in J.M.A. Danby; 0.18 oz divide
    t_0 = 0
    x_0 = 8.5 # ft (if back court line is 0.0 ft, then service line is 13.5 ft, given
    y_0 = 13.5 # ft (if far-left court line is 0.0 ft, then divider is 8.5 ft, given t
    z_0 = (1.15 * 3.281) # ft (1.15 m times 3.281 ft/m) - https://rb.gy/lwa9pw
    # v = 10 m/s, so (82*3.281) ft/s, and if v = sqrt(v_x**2 + v_y**2 + v_z**2)
    v_0 = 10 * 3.281 # ft/s
    theta = np.pi * theta / 180
    phi = np.pi * phi / 180

```

```

v_x_0 = v_0 * np.sin(phi)
v_y_0 = v_0 * np.cos(theta) * np.cos(phi)
v_z_0 = v_0 * np.sin(theta) * np.cos(phi)
k_D = 0.00643 # ft^-1
g = 32.17 # ft/s^2

# Start time variations of Position and Velocity
delta_t = 0.001
Numerical_pos_x = np.array([])
Numerical_pos_y = np.array([])
Numerical_pos_z = np.array([])
Numerical_v_x = np.array([])
Numerical_v_y = np.array([])
Numerical_v_z = np.array([])
Numerical_speed = np.array([])
time = np.array([])
t_min = 0
t_max = 3 # Analytical Guess from Changing t_max and viewing graphs
n = round((t_max-t_min)/ delta_t)

# Put initialized vars into first rows of arrays
Numerical_pos_x = np.append(Numerical_pos_x, x_0)
Numerical_pos_y = np.append(Numerical_pos_y, y_0)
Numerical_pos_z = np.append(Numerical_pos_z, z_0)
Numerical_v_x = np.append(Numerical_v_x, v_x_0)
Numerical_v_y = np.append(Numerical_v_y, v_y_0)
Numerical_v_z = np.append(Numerical_v_z, v_z_0)
time = np.append(time, t_0)

x = Numerical_pos_x[0] # initaial x value
y = Numerical_pos_y[0] # initaial y value
z = Numerical_pos_z[0] # initial z value
x_dot = Numerical_v_x[0] # initial v_x value
y_dot = Numerical_v_y[0] # initial v_y value
z_dot = Numerical_v_z[0] # initial v_z value
t = time[0] # initial t value

for i in range(1, n): #For all the time steps
    x_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * x_dot
    y_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * y_dot
    z_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * z_dot - g

    x_dot += x_ddot*delta_t #Euler method equation
    y_dot += y_ddot*delta_t
    z_dot += z_ddot*delta_t

    x -= x_dot*delta_t # Starting from midline, serving to the left (-x direction)
    y += y_dot*delta_t
    z += z_dot*delta_t

    t += delta_t #Update t to a new value based on Δt

    Numerical_pos_x = np.append(Numerical_pos_x, x) #Put your newest x value at the
    Numerical_pos_y = np.append(Numerical_pos_y, y)
    Numerical_pos_z = np.append(Numerical_pos_z, z)
    Numerical_v_x = np.append(Numerical_v_x, x_dot)
    Numerical_v_y = np.append(Numerical_v_y, y_dot)
    Numerical_v_z = np.append(Numerical_v_z, z_dot)

```

```

time = np.append(time, t) #Put your newest t value at the end of your table c

ax = plt.axes(projection='3d')
ax.set_title(TITLE)
# ax.set_axhline(0, color='black', lw=1)
# ax.set_axvline(0, color='black', lw=1)
ax.set_xlabel('x(t) (ft)')
ax.set_ylabel('y(t) (ft)')
ax.set_zlabel('z(t) (ft)')

for hits_ground in range(len(time)):
    if Numerical_pos_z[hits_ground] <= 0:
        # print(hits_ground)
        Flight_Path = ax.plot(Numerical_pos_x[:hits_ground], Numerical_pos_y[:hits_ground], Numerical_pos_z[:hits_ground])

        start_point = ax.plot(x_0, y_0, z_0, 'blue', marker='x', label=r'Serve Start')
        start_point_ground = ax.plot(x_0, y_0, 0, 'blue', marker='x')
        start_drop_down = ax.plot(linspace(x_0, x_0, n), linspace(y_0, y_0, n), linspace(z_0, 0, n))
        end_point = ax.plot(Numerical_pos_x[hits_ground], Numerical_pos_y[hits_ground], Numerical_pos_z[hits_ground])

    # Plotting Court
    court_back = ax.plot(linspace(0,17,n), linspace(0,0,n), linspace(0,0,n), 'black')
    court_left = ax.plot(linspace(0,0,n), linspace(0,40,n), linspace(0,0,n), 'black')
    court_up = ax.plot(linspace(0,17,n), linspace(40,40,n), linspace(0,0,n), 'black')
    court_right = ax.plot(linspace(17,17,n), linspace(0,40,n), linspace(0,0,n), 'black')
    long_mid = ax.plot(linspace(8.5,8.5,n), linspace(0,40,n), linspace(0,0,n), 'black')
    service_line_1 = ax.plot(linspace(0,17,n), linspace(13.5,13.5,n), linspace(0,0,n), 'black')
    service_line_2 = ax.plot(linspace(0,17,n), linspace(25.5,25.5,n), linspace(0,0,n), 'black')
    net_left = ax.plot(linspace(0,0,n), linspace(20,20,n), linspace(0,5,n), 'black')
    net_right = ax.plot(linspace(17,17,n), linspace(20,20,n), linspace(0,5,n), 'black')
    net_bottom = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(3,3,n), 'black')
    net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(5,5,n), 'black')
    net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(0,0,n), 'black')

    # Viewing Angle
    # for azmith in range(0,30,15):
        # print(azmith)
        ax.view_init(azim=AZM, elev=ELEV)
        ax.set_xlim3d(-1, 18)
        ax.set_ylim3d(-1, 41)
        # ax.set_zlim3d(0, 6)

        ax.legend(loc='best', fontsize=10)
        ax(figsize=(100,100,100))
        plt.show()
    return Flight_Path
break

```

Individual Graphs!!!

In [9]:

```

# Rights and Lefts are from Server's Perspective
theta_f_1 = 78.07
phi_f_1 = 8.32
title_f_1 = f"Short Service - Far Left"

# ideal_title= f"Ideal Services"

```

```

theta_f_r = 78.35
phi_f_r = 0.01
title_f_r = f"Short Service - Far Right"

theta_b_l = 23.72
phi_b_l = 16.4
title_b_l = f"Long Service - Far Left"

theta_b_r = 19.96
phi_b_r = 0.01
title_b_r = f"Long Service - Far Right"

# Different Views
azm1 = -45
azm2 = -150
elev1 = 15
elev2 = 45

Front_Left = Service_Deg(theta_f_l, phi_f_l, title_f_l, azm1, elev1)
# Front_Left2 = Service_Deg(theta_f_l, phi_f_l, title_f_l, azm2, elev2)

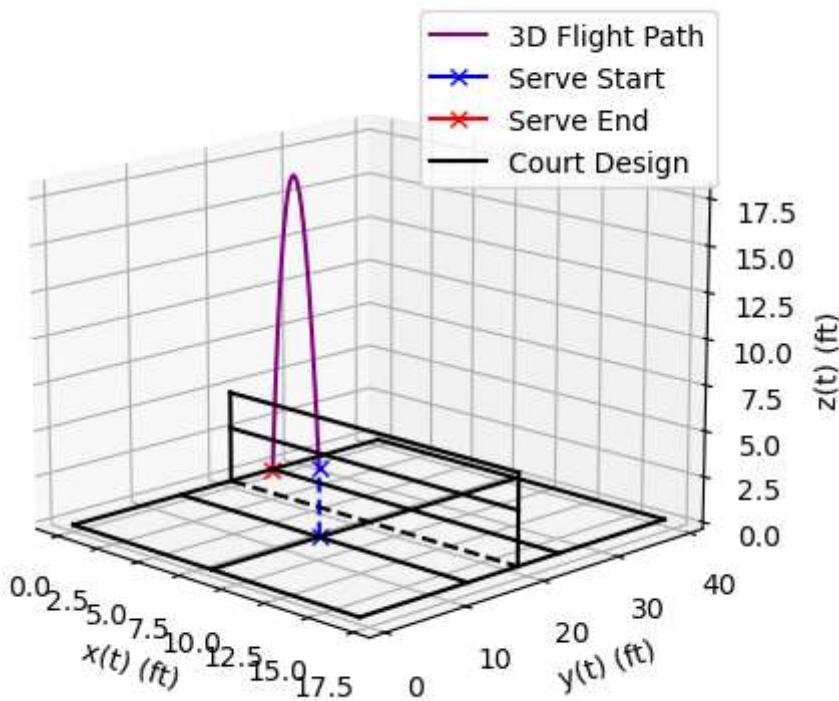
Front_Right = Service_Deg(theta_f_r, phi_f_r, title_f_r, azm1, elev1)
# Front_Right2 = Service_Deg(theta_f_r, phi_f_r, title_f_r, azm2, elev2)

Back_Left = Service_Deg(theta_b_l, phi_b_l, title_b_l, azm1, elev1)
# Back_Left2 = Service_Deg(theta_b_l, phi_b_l, title_b_l, azm2, elev2)

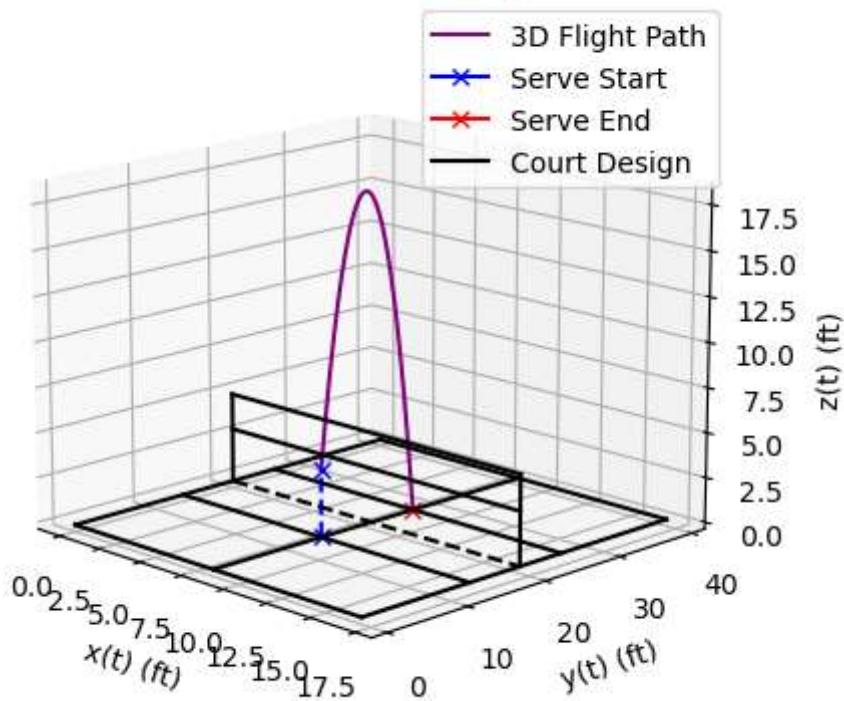
Back_Right = Service_Deg(theta_b_r, phi_b_r, title_b_r, azm1, elev1)
# Back_Right2 = Service_Deg(theta_b_r, phi_b_r, title_b_r, azm2, elev2)

```

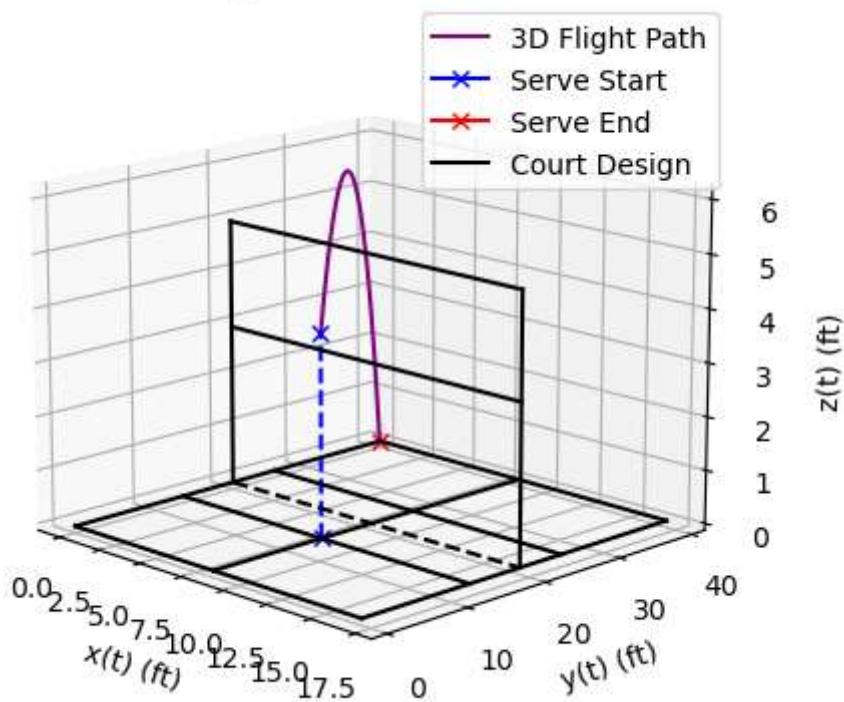
Short Service - Far Left



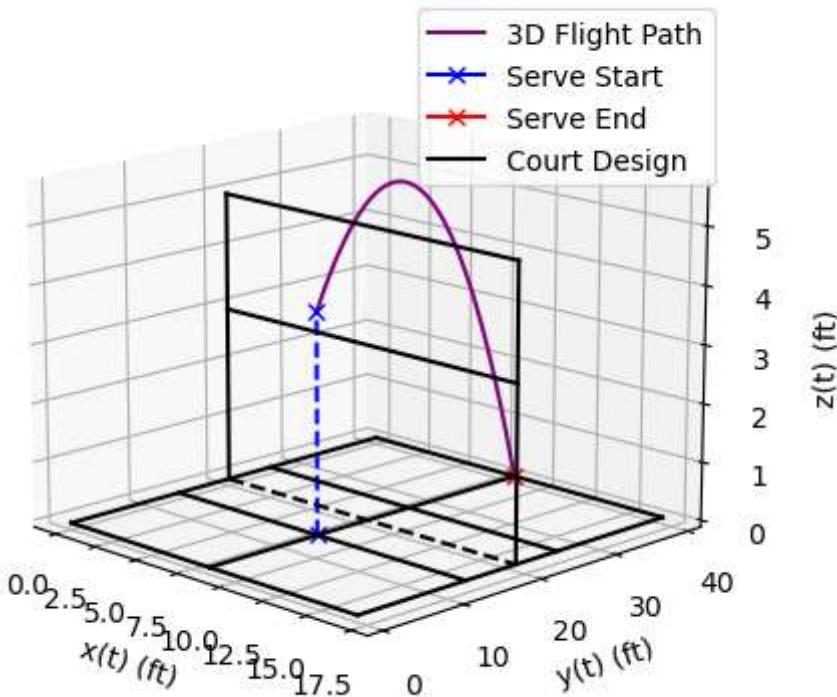
Short Service - Far Right



Long Service - Far Left



Long Service - Far Right



Code for Unified Graph

Function

```
In [10]: def Service_Deg(THETA, PHI):
    # Constants and Variables
    m = (0.18 / 16) # lbs (since everything is in feet in J.M.A. Danby; 0.18 oz divide
    t_0 = 0
    x_0 = 8.5 # ft (if back court line is 0.0 ft, then service line is 13.5 ft, given
    y_0 = 13.5 # ft (if far-left court line is 0.0 ft, then divider is 8.5 ft, given t
    z_0 = (1.15 * 3.281) # ft (1.15 m times 3.281 ft/m) - https://rb.gy/lwa9pw
    # v = 10 m/s, so (82*3.281) ft/s, and if v = sqrt(v_x**2 + v_y**2 + v_z**2)
    v_0 = 10 * 3.281 # ft/s
    theta = np.pi * THETA / 180
    phi = np.pi * PHI / 180
    v_x_0 = v_0 * np.sin(phi)
    v_y_0 = v_0 * np.cos(theta) * np.cos(phi)
    v_z_0 = v_0 * np.sin(theta) * np.cos(phi)
    k_D = 0.00643 # ft^-1
    g = 32.17 # ft/s^2

    # Start time variations of Position and Velocity
    delta_t = 0.001
    Numerical_pos_x = np.array([])
    Numerical_pos_y = np.array([])
    Numerical_pos_z = np.array([])
    Numerical_v_x = np.array([])
    Numerical_v_y = np.array([])
    Numerical_v_z = np.array([])
    Numerical_speed = np.array([])
    time = np.array([])
```

```

t_min = 0
t_max = 3 # Analytical Guess from Changing t_max and viewing graphs
n = round((t_max-t_min)/ delta_t)

# Put initialized vars into first rows of arrays
Numerical_pos_x = np.append(Numerical_pos_x, x_0)
Numerical_pos_y = np.append(Numerical_pos_y, y_0)
Numerical_pos_z = np.append(Numerical_pos_z, z_0)
Numerical_v_x = np.append(Numerical_v_x, v_x_0)
Numerical_v_y = np.append(Numerical_v_y, v_y_0)
Numerical_v_z = np.append(Numerical_v_z, v_z_0)
time = np.append(time, t_0)

x = Numerical_pos_x[0] # initaial x value
y = Numerical_pos_y[0] # initaial y value
z = Numerical_pos_z[0] # initial z value
x_dot = Numerical_v_x[0] # initial v_x value
y_dot = Numerical_v_y[0] # initial v_y value
z_dot = Numerical_v_z[0] # initial v_z value
t = time[0] # initial t value

for i in range(1, n): #For all the time steps
    x_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * x_dot
    y_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * y_dot
    z_ddot = -k_D * sqrt(x_dot**2 + y_dot**2 + z_dot**2) * z_dot - g

    x_dot += x_ddot*delta_t #Euler method equation
    y_dot += y_ddot*delta_t
    z_dot += z_ddot*delta_t

    x -= x_dot*delta_t # Starting from midline, serving to the left (-x direction)
    y += y_dot*delta_t
    z += z_dot*delta_t

    t += delta_t #Update t to a new value based on Δt

    Numerical_pos_x = np.append(Numerical_pos_x, x) #Put your newest x value at the
    Numerical_pos_y = np.append(Numerical_pos_y, y)
    Numerical_pos_z = np.append(Numerical_pos_z, z)
    Numerical_v_x = np.append(Numerical_v_x, x_dot)
    Numerical_v_y = np.append(Numerical_v_y, y_dot)
    Numerical_v_z = np.append(Numerical_v_z, z_dot)

    time = np.append(time, t) #Put your newest t value at the end of your table of values

zero = 0
for hits_ground in range(len(time)):
    if Numerical_pos_z[hits_ground] <= 0:
        zero = hits_ground
        break

return Numerical_pos_x, Numerical_pos_y, Numerical_pos_z, zero

```

Ideal Service Angles

Found via Guess & Check

```
In [11]: theta_f_l = 78.07
phi_f_l = 8.32

theta_f_r = 78.35
phi_f_r = 0.01

theta_b_l = 23.72
phi_b_l = 16.4

theta_b_r = 19.96
phi_b_r = 0.01

n = round(3 / 0.001) # Bringing out of function for graphing the court over an Linspace
```

Ideal Graph!

I used Spyder for the images on the poster presentation because of its easy graph-manipulation

```
In [12]: ax = plt.axes(projection='3d')
ax.set_title('Ideal Shuttlecock Services', fontsize=20)
ax.set_xlabel('x(t) (ft)', fontsize=16)
ax.set_ylabel('y(t) (ft)', fontsize=16)
ax.set_zlabel('z(t) (ft)', fontsize=16)

# Starting Point
x_0 = 8.5
y_0 = 13.5
z_0 = 1.15 * 3.281
start_point = ax.plot(x_0, y_0, z_0, 'blue', marker='x', label='Serve Start', linewidth=3)
start_point_ground = ax.plot(x_0, y_0, 0, 'blue', marker='x', linewidth=3)
start_drop_down = ax.plot(linspace(x_0, x_0, n), linspace(y_0, y_0, n), linspace(0, z_0, n), 'red', linewidth=3)

# Plot trajectories for each type of serve
front_left = Service_Deg(theta_f_l, phi_f_l)
front_left_x = front_left[0]
front_left_y = front_left[1]
front_left_z = front_left[2]
zero_fl = front_left[3]
end_point = ax.plot(front_left_x[:zero_fl], front_left_y[:zero_fl], 0, 'red', marker='x')
ax.plot(front_left_x[:zero_fl], front_left_y[:zero_fl], front_left_z[:zero_fl], 'teal', linewidth=3)

front_right = Service_Deg(theta_f_r, phi_f_r)
front_right_x = front_right[0]
front_right_y = front_right[1]
front_right_z = front_right[2]
zero_fr = front_right[3]
ax.plot(front_right_x[:zero_fr], front_right_y[:zero_fr], front_right_z[:zero_fr], 'green', linewidth=3)
end_point = ax.plot(front_right_x[zero_fr], front_right_y[zero_fr], 0, 'red', marker='x')

back_left = Service_Deg(theta_b_l, phi_b_l)
back_left_x = back_left[0]
back_left_y = back_left[1]
back_left_z = back_left[2]
zero_bl = back_left[3]
ax.plot(back_left_x[:zero_bl], back_left_y[:zero_bl], back_left_z[:zero_bl], color='sienna', linewidth=3)
end_point = ax.plot(back_left_x[zero_bl], back_left_y[zero_bl], 0, 'red', marker='x',
```

```
back_right = Service_Deg(theta_b_r, phi_b_r)
back_right_x = back_right[0]
back_right_y = back_right[1]
back_right_z = back_right[2]
zero_br = back_right[3]
ax.plot(back_right_x[:zero_br], back_right_y[:zero_br], back_right_z[:zero_br], 'purple')
end_point = ax.plot(back_right_x[zero_br], back_right_y[zero_br], 0, 'red', marker='x')

# Court Code
court_back = ax.plot(linspace(0,17,n), linspace(0,0,n), linspace(0,0,n), 'black', line
#, label=r'Court Design'
court_left = ax.plot(linspace(0,0,n), linspace(0,40,n), linspace(0,0,n), 'black', line
court_up = ax.plot(linspace(0,17,n), linspace(40,40,n), linspace(0,0,n), 'black', line
court_right = ax.plot(linspace(17,17,n), linspace(0,40,n), linspace(0,0,n), 'black', l
long_mid = ax.plot(linspace(8.5,8.5,n), linspace(0,40,n), linspace(0,0,n), 'black', li
service_line_1 = ax.plot(linspace(0,17,n), linspace(13.5,13.5,n), linspace(0,0,n), 'bl
service_line_2 = ax.plot(linspace(0,17,n), linspace(25.5,25.5,n), linspace(0,0,n), 'bl
net_left = ax.plot(linspace(0,0,n), linspace(20,20,n), linspace(0,5,n), 'black', linew
net_right = ax.plot(linspace(17,17,n), linspace(20,20,n), linspace(0,5,n), 'black', li
net_bottom = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(3,3,n), 'black', li
net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(5,5,n), 'black', linew
net_top = ax.plot(linspace(0,17,n), linspace(20,20,n), linspace(0,0,n), 'black', lines

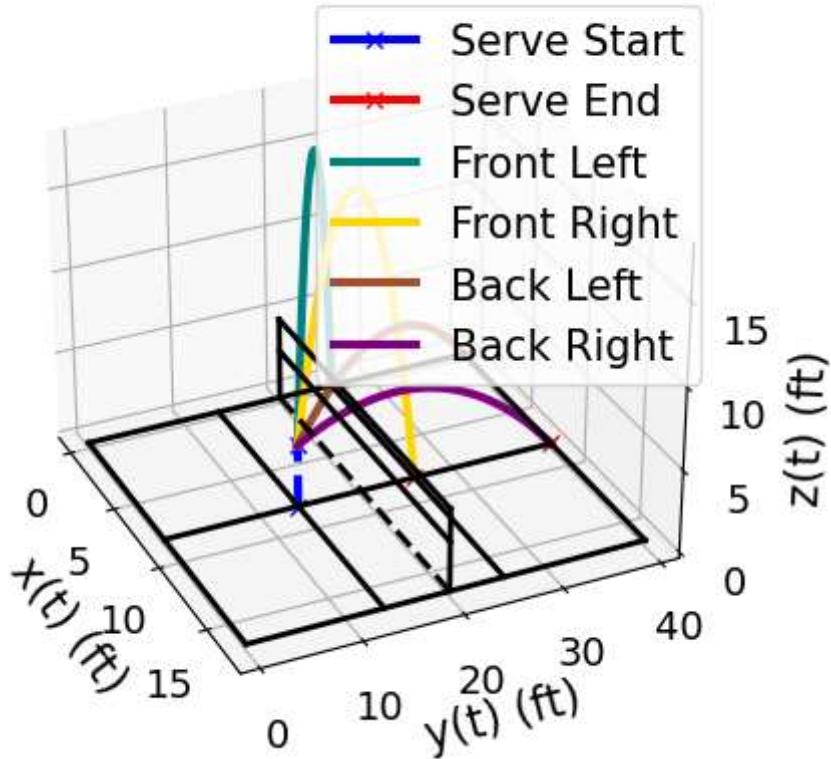
ax.tick_params(labelsize=14)

ax.view_init(azim=-25, elev=30)
ax.set_xlim3d(-1, 18)
ax.set_ylim3d(-1, 41)
ax.set_zlim3d(0, 18)

ax.legend(loc='best', fontsize=15)
# ax.figsize=(100,100,100)
```

Out[12]: <matplotlib.legend.Legend at 0x1fdccb45550>

Ideal Shuttlecock Services



Initial Velocity Graph (For 1st Optimal Service)

```
In [13]: # Initial Launch Conditions
x_0 = 8.5 # ft (if back court line is 0.0 ft, then service line is 13.5 ft, given the
y_0 = 13.5 # ft (if far-left court line is 0.0 ft, then divider is 8.5 ft, given the w
z_0 = (1.15 * 3.281) # ft (1.15 m times 3.281 ft/m) - https://rb.gy/Lwa9pw
v_0 = 10 * 3.281 # ft/s
theta = np.pi * 78.07 / 180
phi = np.pi * 8.32 / 180
v_x_0 = v_0 * np.sin(phi)
v_y_0 = v_0 * np.cos(theta) * np.cos(phi)
v_z_0 = v_0 * np.sin(theta) * np.cos(phi)

# Set up graph and plot
v_ax = plt.axes(projection='3d')
v_ax.set_title('Initial Shuttlecock Service Velocity', fontsize=20)
v_ax.set_xlabel('x(t) (ft)', fontsize=16)
v_ax.set_ylabel('y(t) (ft)', fontsize=16)
v_ax.set_zlabel('z(t) (ft)', fontsize=16)

# Init. Vel
v_ax.plot(linspace(x_0, -v_x_0+x_0, 100), linspace(y_0, v_y_0+y_0, 100), linspace(z_0,
    'purple', label=r'$\mathbf{v}_o$', linewidth=3)
v_ax.plot(-v_x_0+x_0, v_y_0+y_0, (v_z_0+z_0), 'purple', marker='<')

# Init. Vel in xy plane
v_ax.plot(linspace(x_0, -v_x_0+x_0, 100), linspace(y_0, v_y_0+y_0, 100), linspace(z_0,
    'red', label=r'$\mathbf{v}_{xy}$ Projection', linestyle='--', linewidth=3)
v_ax.plot(-v_x_0+x_0, v_y_0+y_0, z_0, 'red', marker='v')
```

```

# Init. Vel in xz plane
v_ax.plot(linspace(x_0, -v_x_0+x_0, 100), linspace(y_0, y_0, 100), linspace(z_0, v_z_0, 100),
           'blue', label=r'$\mathbf{v_{xz}}$ Projection', linestyle='--', linewidth=3)
v_ax.plot(-v_x_0+x_0, y_0, v_z_0+z_0, 'blue', marker='^')

# Theta angle from xy up to Init. Vel -----
v_ax.text((x_0*.75), (y_0*1.2), (z_0*2), r'$\theta$', color='orange', fontsize=16)

# radius = v_θ
# x_y_plane = linspace(v_θ, 0, 100)
# xy_to_z = sqrt(radius**2 - x_y_plane**2)
# circ = np.array([])
# circ_count = 0
# for i in range(100):
#     if xy_to_z[i] <= v_θ:
#         circ = np.append(circ, xy_to_z[i]/2)
#     if xy_to_z[i] == v_θ:
#         circ_count = i-10
# v_ax.plot(linspace(v_x_θ/1.5, v_x_θ/3, circ_count), linspace(v_y_θ/1.5, v_y_θ/3, circ_count))

# Theta angle from xz over to Init. Vel -----
v_ax.text((x_0*.7), (y_0*1.1), (z_0*4), r'$\phi$', color='green', fontsize=16)

# radius = v_θ
# x_z_plane = linspace(v_θ, 0, 100)
# xz_to_y = sqrt(radius**2 - x_z_plane**2)
# circ = np.array([])
# circ_count = 0
# for i in range(100):
#     if xz_to_y[i] <= v_θ:
#         circ = np.append(circ, xz_to_y[i]/8)
#     if xz_to_y[i] == v_θ:
#         circ_count = i
# v_ax.plot(linspace(v_x_θ/2, v_x_θ/6, circ_count), circ[:circ_count], linspace(v_z_θ/2, v_z_θ/6, circ_count))

# Plot xyz axis at origin
v_ax.plot(linspace(x_0, -v_x_0+x_0, 100), linspace(y_0, y_0, 100), linspace(0, 0, 100))
v_ax.plot(linspace(x_0, x_0, 100), linspace(y_0, v_y_0+y_0, 100), linspace(0, 0, 100))
v_ax.plot(linspace(x_0, x_0, 100), linspace(y_0, y_0, 100), linspace(0, v_z_0, 100), 'red')
v_ax.tick_params(labelsize=14)

v_ax.view_init(azim=5, elev=15)
v_ax.legend(loc='best', fontsize=15)
plt.show()

```

Initial Shuttlecock Service Velocity

