# r2knowle_a3q2

March 1, 2023

## 1 A3-Q2: Golf Driving Range

[290]:
```
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
```

[322]:
```
# Supplied functions
def Ground(d):
    '''
     h = Ground(d)

     Returns the height (in metres) of the ground at a horizontal distance
     d (metres) from the origin.
    '''
    return 2.*(np.cos(d/4.)-np.sin(d/11.)-1)

def GroundSlope(d):
    '''
     h = GroundSlope(d)

     Returns the slope of the ground at a horizontal distance
     d (metres) from the origin.
    '''
    return 2.*(-1./4*np.sin(d/4) - 1./11*np.cos(d/11.))
```

### 1.1 (a) MyOde

[659]:
```
def MyOde(f, tspan, y0, h, event=(lambda t,y:1)):
    '''
     t,y = MyOde(f, tspan, y0, h, event=[])

     Numerically solves the initial value problem

        dy(t)/dt = f(t,y)
            y(0) = y0

     using the Modified Euler time-stepping method.
```

```
    Input
      f        a Python dynamics function with calling sequence
                  dydt = f(t, y)
      tspan    2-tuple giving the start and end times, [start, end]
      y0       initial state of the system (as a 1D vector)
      h        the time step to use (this is not adaptive time stepping)
      events   an event function with calling sequence
                  val = events(t, y)
               The computation stops as soon as a negative value is
               returned by the event function.

    Output
      t        1D vector holding time stamps
      y        an array that holds one state vector per row (corresponding
               to the time stamps)

      Notes:
          - t and y have the same number of rows.

          - The first element of t should be tspan[0], and the first
            row of y should be the initial state, y0.

          - If the computation was stopped by the triggering of an event,
            then the last row of t and y should correspond to the
            time that linear interpolation indicates for the zero-crossing
            of the event-function.
   '''

   # Initialize output arrays, tlst and ylst
   t = tspan[0]
   y = deepcopy(y0)

   tlst = []
   ylst = []
   notfirst = False;

   numBounces = 0;

   while (t < tspan[1] and numBounces < 3):
       prevz = y;
       prevt = t;
       normalEulers = y + h*f(t,y);
       newTime = t+h;

       RumeEulers = y + (h/2) * (f(t,y) + f(newTime, normalEulers))
```

```
        if (notfirst and (event(newTime, RumeEulers) < 0)):

            zz = prevz
            tt = prevt;

            while (event(tt, zz) > 0):
                prevt = tt;
                prevz = zz;
                tt += h/40
                zz = zz + h/40 * f(tt, zz)


            velocity = np.array([RumeEulers[2], RumeEulers[3]])

            slope = GroundSlope(zz[0])
            normSlope = -1/GroundSlope(zz[0])

            length = np.sqrt(1+slope**2)
            normLength = np.sqrt(1+normSlope**2)

            rep = 1/length
            normrep = 1/normLength


            u = np.array([rep, slope*rep])
            U = np.array([normrep, normSlope*normrep ])

            new = 0.85*(np.dot(velocity,u) *u - np.dot(velocity, U) *U)



            new = np.array([prevz[0], prevz[1], new[0], new[1]])


            y = new
            t = prevt

            tlst.append(t)
            ylst.append(y)

            numBounces += 1;



        else:
            tlst.append(newTime)
            ylst.append(RumeEulers)
```

```
            y = RumeEulers;
            t = newTime;

        notfirst = True;




    return np.array(tlst), np.array(ylst)
```

## 1.2 (b) Dynamics Function: projectile

[660]:
```
def projectile(t, z):

    '''
    z[0] = x
    z[1] = y
    z[2] = x'
    z[3] = y'
    '''

    K = 0.3
    g = 9.81;

    return np.array([z[2], z[3], -K * z[2], -g -K* z[3]])
```

## 1.3 (c) Events Function: projectile_events

[661]:
```
def projectile_events(t, z):

    val = 1

    if (Ground(z[0]) > z[1]):
        val = -1;

    return val
```

## 1.4 (d) Three flights

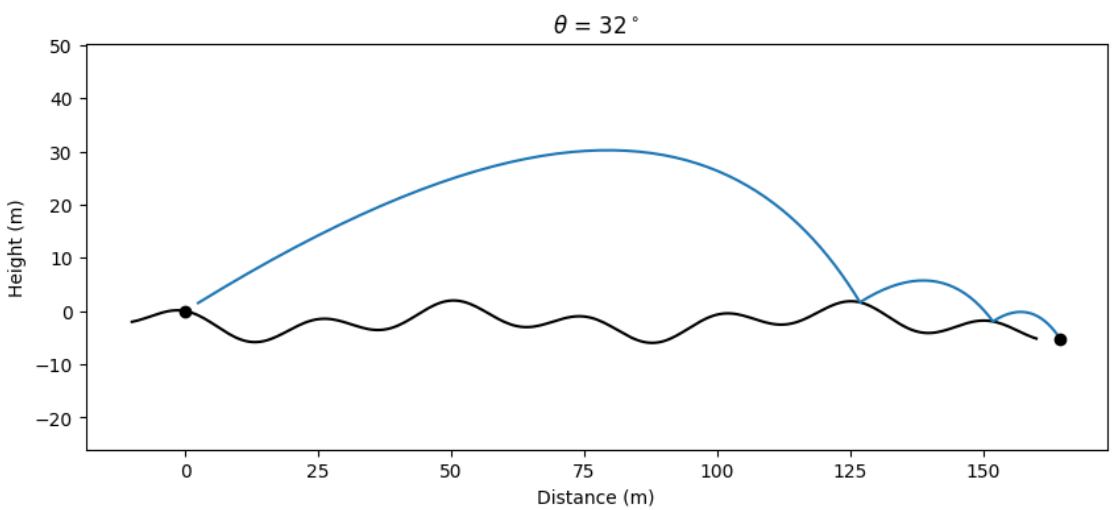
[710]:
```
theta = 32
S = 58
tspan = [0, 30]
h = 0.05
theta_rad = theta/180.*np.pi
yStart = np.array([0, 0, S*np.cos(theta_rad), S*np.sin(theta_rad)])
```

```
t,y = MyOde(projectile, tspan, yStart, h, projectile_events)
```

[711]:
```
# Plot the ground
x = np.linspace(-10, 160, 300)
hills = Ground(x)
plt.figure(figsize=[10,4])
plt.plot(x,hills, 'k')
plt.axis('equal')

plt.plot(y[:,0], y[:,1])  # Plot ball trajectory
plt.plot([0],[0], 'ko')   # Plot initial ball position
plt.plot(y[-1,0], y[-1,1], 'ko')  # Plot final ball position

plt.title(r'$\theta$ = '+str(theta)+'$^\circ$');
plt.xlabel('Distance (m)')
plt.ylabel('Height (m)');
```




[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: