Appendix A: Figures

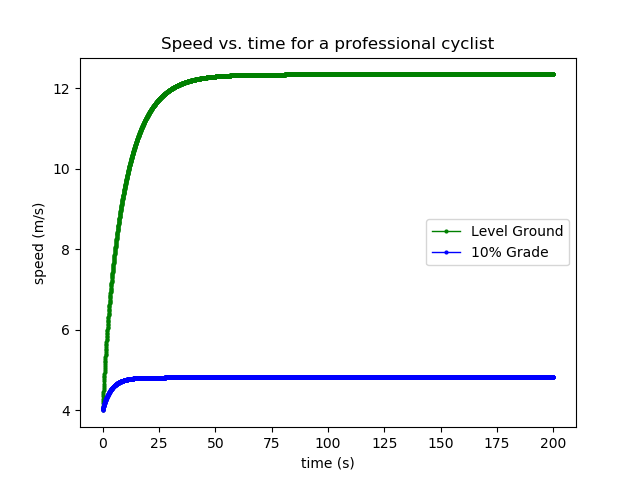


Figure 1: A graph of the speed of a professional cyclist, comparing the same initial conditions. One trial took place on level ground, the other on a 10% gradient – corresponding to an angle of about 5 or 6 degrees. The cyclist was assumed to generate about 400W of power, and the initial speed was assumed to be 4 m/s.

Writing the code for this was a pain in the butt. The concept of gravity was fairly simple to implement, once I’d worked out the theory behind it and double checked that Numerical Python supported the trigonometry I was going to use. I was immediately suspicious when the graph spat out showed the cyclist rolling down the hill order of magnitudes faster than the speed of light. Having seen this problem before in the lab, I modified my data types from plain integers to floating point values, solving the problem.

The next problem was that the graph was now similar to a reciprocal function. I was initially confused at this but realized that the angle I’d been using - 30 degrees - was roughly five or six times higher than the angle that the outline wanted me to test. When I fixed this mistake, the program righted itself immediately. The final task came when I tried to plot both values on the same axes, but ended up with two copies of the same data on the same axes - a pesky problem I stumbled over until I spotted that I had forgotten to change the angle of the slope when moving on to the second data set. After that tiny oversight was fixed, the program performed perfectly.

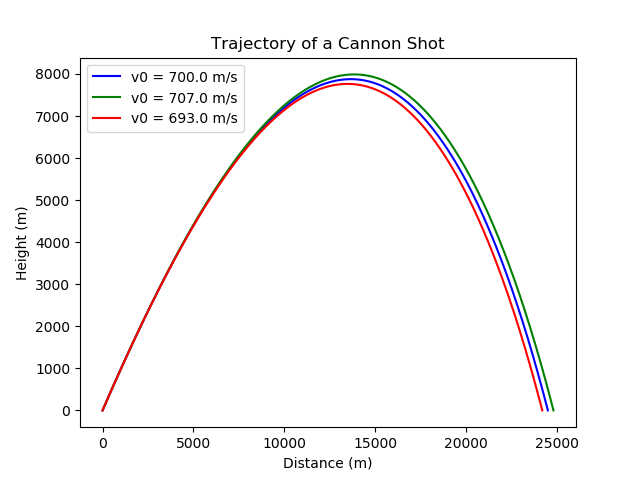


Figure 2: A graph of the trajectory of a cannon shot, showing simulations at varying muzzle velocities. These projections determine the shell’s path using the Euler method, iterating until the shell hits the ground. In addition, they calculate the force of drag on the shot, and use the adiabatic approximation to account for the changing density of the atmosphere with altitude.

These figures were created using initial conditions with a temperature and pressure of 300K at sea level, using a launch angle of 45 degrees and a time interval of 0.0012s between calculations. It is none too apparent from the graph itself, but each simulation’s shell makes landfall about two to three hundred metres apart – in a context like this, that tiny, <1% difference in range means the difference between a direct hit or a wide miss.

Appendix B: Bicyclist Source Code

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sun Jan 26 09:41:04 2020

This is a skeleton code to solve the bike racing problem described in Sec. 2.1 of Giordano and Nakanishi.

@author: atkinson

Adapted for Assignment 2 on Thursday, Feb. 6, 2020 by David Stothers

"""

import numpy as np

from matplotlib import pyplot as plt

# Define important variables

m = 80 # Mass of the rider [kg]

rho = 1.29 # density of air [kg/m^3]

t\_init = 0.0 # starting time of the simulation [s]

t\_final = 200.0 # end time of the simulation [s]

N = 4001 # Number of time steps

t\_list = np.linspace(t\_init, t\_final, N) # times at which we calculate v

dt = t\_list[1] - t\_list[0] # time step

g = -9.81 #gravitational acceleration

# Values below obtained from G&N

P=400

C=1

A=0.33

dragconst = -(rho/2.0)\*C\*A

theta = 0.0 #the angle in degrees of the relevant slope

v=np.arange(N+1,dtype=float)

v[0]=4.0

F=np.arange(N,dtype=float)

Fdrag=np.arange(N,dtype=float)

Fg = float(float(m\*g)\*np.sin(float((theta\*np.pi)/180.0))) #the force on the motion due to gravity

# Main loop

# This loop has problems that you will fix in the lab

#for i, t in enumerate(t\_list):

for i in range(N):

print(i, t\_list[i])

Fdrag[i] = dragconst\*(v[i]\*\*2) # calculate drag

#Edited to account for gravity

F[i] = (P/v[i])+Fdrag[i]+Fg #calculate net force

v[i+1] = v[i]+((F[i]/m))\*dt #calculate next interval's velocity

theta = float(np.arctan(0.1)) #the angle in radians of the relevant slope

Fg = float(float(m\*g)\*np.sin(theta)) #the force on the motion due to gravity

v2=np.arange(N+1,dtype=float)

v2[0]=4.0

F=np.arange(N,dtype=float)

Fdrag=np.arange(N,dtype=float)

for i in range(N):

print(i, t\_list[i])

Fdrag[i] = dragconst\*(v2[i]\*\*2) # calculate drag

#Edited to account for gravity

F[i] = (P/v2[i])+Fdrag[i]+Fg #calculate net force

v2[i+1] = v2[i]+((F[i]/m))\*dt #calculate next interval's velocity

# Output results

plt.figure(0)

plt.clf()

plt.xlabel('time (s)')

plt.ylabel('speed (m/s)')

plt.title('Speed vs. time for a professional cyclist')

v=v[:-1]

v2=v2[:-1]

plt.plot(t\_list,v,'go-', linewidth=1, markersize=2,label="Level Ground")

plt.plot(t\_list,v2,'bo-', linewidth=1, markersize=2,label="10% Grade")

plt.legend()

plt.show()

Appendix C: Cannon Shell Source Code

# Import useful libraries

import numpy as np

from matplotlib import pyplot as plt

"""

Docstring goes here:

This code is intended to calculate the trajectory of a cannon shell,

using gravity, drag, and changes in drag with altitude.

it is intended to generate trajectory data for Question 2 of the second

assignment, and is built off of the skeleton code given in Lab 5.

@author: davidstothers

"""

def trajectory(v0, theta, x0, y0, T0):

"""

The docstring for the function should tell you what the function does,

what the input variables mean, and what the output variables mean.

This function calculates the movement of a projectile through air, given

the initial position, height, speed, launch angle, and ambient temperature.

Input:

v0 - the magnitude of the initial velocity

theta - the launch angle in radians, taken to be 0 level to the ground

and increasing upwards

x0 - the initial x position, taken to be positive pointing forwards

y0 - the initial y position, taken to be positive pointing up

rho0 - the fluid density at sea level, in kilopascals.

T0 - the temperature at sea level, in Kelvin.

Output:

x - an array of the different x values found.

y - an array of the different y values found.

vx - an array of the different vx values found.

vy - an array of the different vy values found.

t - an array of the different t values found.

x\_range - the maximum distance the shell flies.

"""

# estimated maximum time of flight for the shell, plus a little extra

tmax\_est = 180

# Number of time steps

N = 150001

t = np.linspace(0, tmax\_est, N, dtype=np.float)

dt = t[1] - t[0]

rho0 = 101.32 # Atmospheric pressure at sea level, in kilopascals.

alpha = 2.5

a = 6.5\*10\*\*-3

B2\_over\_m = 4e-5 # prefactor for the drag force in m^{-1}

g = 9.80 # gravitational acceleration in m/s^2

# define the position and velocity arrays and set their initial values

x = np.arange(0,N,dtype=float)

y=np.arange(0,N,dtype=float)

vx=np.arange(0,N,dtype=float)

vy=np.arange(0,N,dtype=float)

v = np.arange(0,N,dtype=float)

x[0]=x0

y[0]=y0

vx[0]=v0\*np.cos(theta)

vy[0]=v0\*np.sin(theta)

v[0]=v0

anety = np.arange(0,N,dtype=float) # Drag taken to be positive opposing motion

adragy = np.arange(0,N,dtype=float)

adragx = np.arange(0,N,dtype=float)

adragy[0] = B2\_over\_m \* vy[0]\*v[0]

# iterate over times

has\_landed = False

for i0 in np.arange(N-1):

# calculate the position and velocity versus time here

v[i0]=np.sqrt((vx[i0]\*\*2) + (vy[i0]\*\*2))

rho = rho0\*(1-(a\*y[i0]/T0))\*\*alpha

#Using the adiabatic approximation

adragy[i0] = B2\_over\_m\*(rho/rho0)\*vy[i0]\*v[i0]

adragx[i0] = B2\_over\_m\*(rho/rho0)\*vx[i0]\*v[i0]

anety[i0]=g+adragy[i0]

vx[i0+1]=vx[i0]-(adragx[i0]\*dt)

vy[i0+1]=vy[i0]-(anety[i0]\*dt)

x[i0+1]=x[i0]+(vx[i0]\*dt)

y[i0+1]=y[i0]+(vy[i0]\*dt)

if y[i0] < 0.0:

has\_landed = True

break

if not(has\_landed):

print("Error: simulation is not long enough")

# calculate the range of the shell using interpolation

# y = y1 + ((x - x1) / (x2 - x1)) \* (y2 - y1)

# ((y - y1)/(y2-y1))\*(x2-x1) + x1 = x

x\_range = x[i0-1] + ((0-y[i0-1])/(y[i0]-y[i0-1]))\*(x[i0]-x[i0-1])

# Now return only the portion of the arrays we care about.

return x[:i0], y[:i0], vx[:i0], vy[:i0], t[:i0], x\_range

# Now start the main part of the program.

# First define constants:

v0 = 700.0 # initial speed in m/s

#Taking the temperature of a warm, sunny day.

T0 = 300 # Temperature at sea level, in kelvin.

# No air resistance

theta1 = np.pi/4

x1, y1, vx1, vy1, t1, x1\_range = trajectory(v0, theta1, 0., 0., T0)

plt.figure(0)

plt.clf()

plt.xlabel("Distance (m)")

plt.ylabel("Height (m)")

plt.title("Trajectory of a Cannon Shot")

plt.plot(x1, y1, 'b', label='v0 = '+str(v0)+' m/s')

print('range in metres at v0 = '+str(v0)+' m/s: ', x1\_range)

v0=707.0

x2, y2, vx2, vy2, t2, x2\_range = trajectory(v0, theta1, 0., 0., T0)

plt.plot(x2, y2, 'g', label='v0 = '+str(v0)+' m/s')

print('range in metres at v0 = '+str(v0)+' m/s: ', x2\_range)

v0=693.0

x3, y3, vx3, vy3, t3, x3\_range = trajectory(v0, theta1, 0., 0., T0)

plt.plot(x3, y3, 'r', label='v0 = '+str(v0)+' m/s')

print('range in metres at v0 = '+str(v0)+' m/s: ', x3\_range)

plt.legend()

#print('range at theta = pi/4: ', x1\_range)

plt.show()