## נספח ג – תדפיס של התוכנה rocket\_project.py

# coding: utf-8

import numpy as np

#

def set\_initial\_values(x, z, velocity, theta\_velocity\_degree, fuel\_mass,

parameters):

"""

function for initalizing all the variables of the simulations

x, z: the initial position of the rocket in meters

velocity: the magnitude of the rocket velocity at t=0 in m/s

theta\_velocity\_degree: the degree between the rocket velocity

and the x axis

fuel\_mass: fuel mass at t=0 in kg

parameters: dict returned from set\_parameters function

returns: a dictionary of the rocket values at t=0

"""

values = {

"x": x,

"z": z,

"vx": velocity\*np.cos(np.radians(theta\_velocity\_degree)),

"vz": velocity\*np.sin(np.radians(theta\_velocity\_degree)),

"mass": fuel\_mass+parameters["m0"]}

print(values)

return values

def set\_parameters(theta\_rocket\_degree, gas\_velocity, m0, Cd, Cl, Sd, Sl,

dmdt):

"""

function for initalizing all the constant parameters of the simulations

theta\_rocket\_degree: the degree between the rocket and the x axis.

This value impact the direction of thrust

gas\_velocity: the magnitude of the rocket gas velocity in m/s

m0: self mass of the rocket in kg. this is the mass after all fuel is gone

Cd, Cl: drag and lift coefficients, respectivly

Sd, Sl: drag and lift surface size in m^2

dmdt : mass loss coefficient (referd to as alpha in the work)

returns: a dictionary of the rocket constant parameters

"""

parameters = {

"theta\_rocket": np.radians(theta\_rocket\_degree),

"gas\_velocity": gas\_velocity,

"m0": m0,

"Cd": Cd,

"Cl": Cl,

"Sd": Sd,

"Sl": Sl,

"dmdt": dmdt}

print(parameters)

return parameters

def Rho(z):

"""

air density at altitude of z meters based on

https://en.wikipedia.org/wiki/Density\_of\_air#Altitude

z: altitude in meters

returns: air density in kg/m^3

"""

p0 = 101.325e3 # sea level standard atmospheric pressure, Pa

T0 = 288.15 # sea level standard temperature, K

g = 9.80665 # earth-surface gravitational acceleration, m/s²

L = 0.0065 # temperature lapse rate, K/m

R = 8.31447 # ideal (universal) gas constant, J/(mol·K)

M = 0.0289644 # molar mass of dry air, kg/mol

T = T0-L\*z

if T < 0:

return(0)

p = p0 \* (T/T0)\*\*(g\*M/(R\*L))

rho = p\*M/(R\*T)

return(rho)

def Gravity(z, me\_kg=5.972e24, re\_m=6371e3):

"""

Gravity at altitude z meters above sea level

z: altitude in meters

returns: gravity aceleration in m/sec^2

"""

from scipy.constants import G

return(G\*me\_kg/(re\_m+z)\*\*2)

def Drag(values, parameters):

"""

calculate the drag force = 0.5\*Cd\*Sd\*(v^2)

values and parameters - the dictionaries of the current rocket state

returns drag force in N

"""

return(0.5 \* parameters["Cd"] \* Rho(values["z"]) \* parameters["Sd"] \*

(values["vx"]\*\*2 + values["vz"]\*\*2))

def Lift(values, parameters):

"""

calculate the lift force = 0.5\*Cl\*Sl\*rho\*(v^2)

values and parameters - the dictionaries of the current rocket state

returns lift force in N

"""

return(0.5 \* parameters["Cl"] \* Rho(values["z"]) \* parameters["Sl"] \*

(values["vx"]\*\*2 + values["vz"]\*\*2))

def Thrust(t, values, parameters):

"""

calculate the thrus force = alpha \* (v\_gas)

values and parameters - the dictionaries of the current rocket state

returns thrust force in N

"""

return dm\_dt(t, values) \* parameters["gas\_velocity"]

def theta\_velocity(vx, vz):

# returns the velocity degree using its x and z elements

return(np.arctan2(vz, vx))

# % EQUATIONS

def dm\_dt(t, values, parameters):

# equation for the mass change

# dm/dt = alpha (if there is fuel to burn)

# dm/dt = 0 (if there is no more fuel to burn)

m = values["mass"]

if m > parameters["m0"]: # rocket mass contains fuel

return(-parameters["dmdt"])

else: # no change in mass if we are at the rocket self mass

return(0)

def dvz\_dt(t, values, parameters):

# acellaration in z direction - see the related work for more details

# returns the rhs of the equation dvz/dt = az

z = values["z"]

vx = values["vx"]

vz = values["vz"]

m = values["mass"]

theta\_rocket = parameters["theta\_rocket"]

gas\_velocity = parameters["gas\_velocity"]

# the froces in Z direction

F\_thrust\_z = \

-dm\_dt(t, values, parameters) \* gas\_velocity \* np.sin(theta\_rocket)

F\_drag\_z = Drag(values, parameters)\*np.sin(theta\_velocity(-vx, -vz))

# F\_lift is perpendicular to v

F\_lift\_z = -Lift(values, parameters)\*np.cos(theta\_velocity(-vx, -vz))

return ((F\_thrust\_z + F\_drag\_z + F\_lift\_z)/m-Gravity(z))

def dvx\_dt(t, values, parameters):

# acellaration in x direction - see the related work for more details

# returns the rhs of the equation dvx/dt = ax

vx = values["vx"]

vz = values["vz"]

m = values["mass"]

theta\_rocket = parameters["theta\_rocket"]

gas\_velocity = parameters["gas\_velocity"]

# the froces in X direction

F\_thrust\_x = \

-dm\_dt(t, values, parameters) \* gas\_velocity \* np.cos(theta\_rocket)

# drag is opposite to velocity

F\_drag\_x = Drag(values, parameters) \* np.cos(theta\_velocity(-vx, -vz))

# F lift is perpendicular to v

F\_lift\_x = Lift(values, parameters) \* np.sin(theta\_velocity(-vx, -vz))

return ((F\_thrust\_x + F\_drag\_x + F\_lift\_x)/m)

def dz\_dt(t, values, parameters):

# an euqation for the position using the relation dz/dt = vz

# returns the rhs of the equation dz/dt = vz

vz = values["vz"]

return(vz)

def dx\_dt(t, values, parameters):

# an euqation for the position using the relation dx/dt = vx

# returns the rhs of the equation dx/dt = vx

vx = values["vx"]

return(vx)

# % Euler Cromer Solver

# equation to solve. see Euler\_Cromer\_Step function for more details

rhs\_dict = {"mass": dm\_dt, "x": dx\_dt, "z": dz\_dt, "vx": dvx\_dt, "vz": dvz\_dt}

def Euler\_Cromer\_Step(rhs\_dict, values, parameters, t0, dt=1e-3):

new\_values = values.copy()

"""

rhs\_dict holds the equations to solve in the form {variable: eqaution}

the update step according to euler is

new\_value = old\_value + dt \* (equation evaluated at this time)

since we update each field in a seperate step (sequential update)

we get the euler-cromer method. for example:

key1 is update

key2 is updated using the values of the updated key1

which i exactly euler-cromer method.

(in regular euler key2 uses old key1)

"""

for key in rhs\_dict.keys():

new\_values[key] += rhs\_dict[key](t0, values, parameters) \* dt

return new\_values

def Euler\_Cromer(values, parameters, t\_init, t\_final, dt=1e-3,

rhs\_dict=rhs\_dict):

import pandas as pd # for returning a dataframe object

from tqdm import tqdm # for displaying a progress bar during simulation

t\_values = np.arange(t\_init, t\_final, dt) # time vector to simulate

# we collect the results at each time step into the list results

results = []

results.append(values.copy())

for t in tqdm(t\_values[1:]):

# run the euler-cromer step onec to go from t to t+dt

results.append(Euler\_Cromer\_Step(rhs\_dict, values=results[-1],

parameters=parameters, t0=t, dt=dt))

if results[-1]["z"] < 0: # exit condition if rocket under sea level

results.pop()

break

# return a DataFrame object for better reading of the results

df = pd.DataFrame(results, index=t\_values[:len(results)])

df.index.name = "time"

return df

RocketSimulation (c) by Roi Dvir

RocketSimulation is licensed under a Creative Commons Attribution NonCommercial-NoDerivatives 4.0 International License.

You should have received a copy of the license along with this work. If not, see <http://creativecommons.org/licenses/by-nc-nd/4.0/>.