# Project 1

# ME-328-102: ME Analysis

# **Error Analysis for Explicit Euler Method for Parachutist Problem**

# Justin Dyer

University of South Alabama student, Author

# Landon Freeman

University of South Alabama student, Program Discussion

Index

* 1. Problem Description…………………………………………………………..……pg3
  2. Flowchart…………………………………………………………………………….….pg4
  3. Python Source Codes……………………………………………………..……pg5-8
  4. Results and Discussion……………………………………………………….pg9-12
  5. Conclusions…………………………………………………………………………..pg13

# 1-1 Problem Description:

The problem is derived from a parachutist free-falling with a parachute attached to him. The problem depends on the following: Mass(m), Gravity(g), Velocity(v), Time(t), and Drag Coefficient(c). To model this problem, we can use the following equation for a free-falling object (Eqn 1.1):

(Eqn 1.1)

According to the textbook, using calculus, one can obtain the Exact Solution, given as (Eqn 1.2):

(Eqn 1.2)

For our project, we are going to focus on Eqn 1.1 and use Eqn 1.2 to compare our results. To solve this problem, we will use python to compute the Explicit Euler’s Method to approximate a solution.

The given variables for the problem were given as shown:

* m = 68.1 kg
* c = 12.5 kg/s
* g = 9.8 m/s2

We are to compute the functions at the following time intervals:

* dt = 20.0s
* dt = 5.0s
* dt = 1.0s
* dt = 0.1s
* dt = 0.01s

# 1-2 Flowchart:

To begin this problem, we need to derive a flowchart to guide us while writing the program. We found that the following flowchart was best suited to guide us:

(Figure 1-1: Flowchart)

# 1-3 Python Source Codes:

For this project, we used the following Python source codes:

#Justin Dyer

#proj1.py

#J00655685

import math

import numpy as np

import matplotlib.pyplot as plt

#define the function of dv\_over\_dt

def dv\_over\_dt(g, c, m, v):

dfdt = g - c/m \*v

return dfdt

#define the function of v\_exact

def v\_exact\_func(g, c, m, t):

v = g\*m/c\*(1.0 - math.exp(-1.0\*c/m\*t))

return v

#define the error analysis function

def truePerRelError(t\_values, num\_values, exact\_values, ErrorCompareAtTime):

for i in range (0, len(t\_values)):

if t\_values[i] < ErrorCompareAtTime and t\_values[i+1] >= ErrorCompareAtTime:

num\_val\_compare = num\_values[i+1]

exact\_val\_compare = exact\_values[i+1]

if exact\_val\_compare != 0.0:

truePercentRelativeError = (exact\_val\_compare - num\_val\_compare)/exact\_val\_compare

else:

truePercentRelativeError = math.nan

return truePercentRelativeError, num\_val\_compare, exact\_val\_compare

#define the function to display key results on screen

def display\_values(t, v\_euler, v\_exact, truePercentRelativeError, percentRelativeError):

print('Display the results...')

length = len(t)

print('\_\_\_\_i\_\_\_\_ \_\_\_\_t\_\_\_\_\_ \_\_v\_euler\_\_ \_\_v\_exact\_\_ ')

for i in range(0, length):

print('{:10d}'.format(i), '{:10.6f}'.format(t[i]), '{10.6f}'.format(v\_euler[i]), '{:10.6f}'.format(v\_exact[i]) )

print('truePercentRelativeError = ', truePercentRelativeError)

print('percentRelativeError = ', percentRelativeError)

return 1

#define the function to plot and display/save the figure

def plotSaveResultsInOneFigure(t, v\_euler, v\_exact, dt, truePercentRelativeError, v\_euler\_compare, v\_exact\_compare, at\_t):

plt.plot(t, v\_euler, 'o-', t, v\_exact, '-x') #plot two curves for v\_euler and v\_exact

plt.legend(('v\_Explicit Euler Method', 'v\_Exact Solution'), loc = 0)

plottitle = 'Parachutist velocity vs time (dt = ' + str(dt) + 's)'

plt.title(plottitle)

textshown = '@t = ' + str(at\_t) + 's' + '\n True Percent Relative Error = ' + str(truePercentRelativeError\*100) + '%' \

+ '\n v\_Euler = ' + str(v\_euler\_compare) + ' m/s' \

+ '\n v\_Exact = ' + str(v\_exact\_compare) + ' m/s'

plt.text(10.0, 10.0, textshown)

plt.xlabel('t (s)') #label of x-axis

plt.ylabel('Velocity (m/s)')#label of y-axis

plt.grid(True) #add grid

plotFileName = 'plot\_dt' + str(dt) + '.jpg'

plt.savefig(plotFileName, format = 'jpg') # save plot in a jpg format file

plt.show() #show the plot on screen

return 1

#initial conditions and constants

g = 9.8

c = 12.5

m = 68.1

#create a dt input

dt = float(input('Input the time step. dt = '))

print('dt = ', dt, 'seconds')

num\_of\_points = int((100 - 100%dt)/dt) + 1

print('Number of data points = ', num\_of\_points)

t = [0.0]

v\_euler = [0.0]

v\_exact = [0.0]

ErrorCompareTime = 20.0

#create data for the plot

i = 0

while i < int(num\_of\_points - 1):

t.append(float(t[i]+dt))

v\_euler.append(float(v\_euler[i] + dv\_over\_dt(g, c, m, v\_euler[i])\*dt))

v\_exact.append(float(v\_exact\_func(g, c, m, t[i+1])))

i = i+1

#define the comparison

truePercentRelativeError, v\_euler\_compare, v\_exact\_compare = truePerRelError(t, v\_euler, v\_exact, ErrorCompareTime)

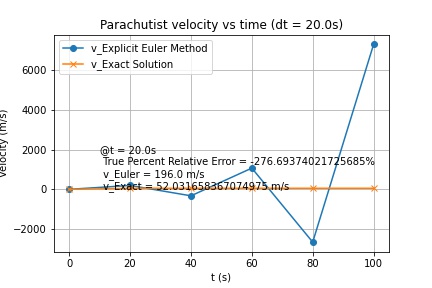
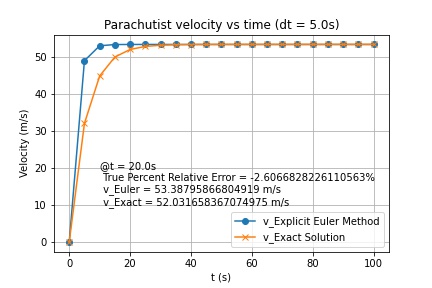
#plot ad save results

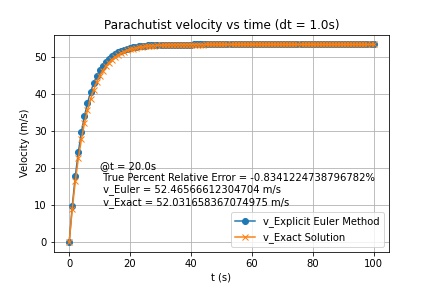
plotSaveResultsInOneFigure(t, v\_euler, v\_exact, dt, truePercentRelativeError, v\_euler\_compare, v\_exact\_compare, ErrorCompareTime)

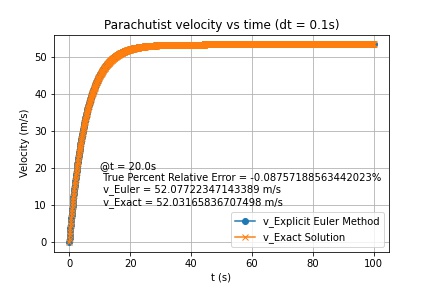
input('\nPress Return to exit')

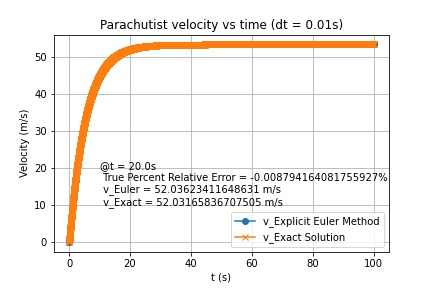
# 1-4 Results and Discussion:

The following plots were the results for running the program for the time intervals required:

(Figure 1.2: Resultant Graph of dt = 20.0s)(Figure 1.3: Resultant Graph of dt = 5.0s) 

(Figure 1.4: Resultant Graph of dt = 1.0s) 

(Figure 1.5: Resultant Graph of dt = 0.1s) 

(Figure 1.6: Resultant Graph of dt = 0.01s) 

Analyzing the plots, we can infer some crucial data from this project. As the time interval gets smaller, one can infer that the Euler Solution gets more accurate. The True Percent Relative Error approaches zero; therefore, our Euler’s approximation gets more precise to our Exact solution.

With the decreacing time interval associated with the increse of percision, one could say that the time interval and Euler’s method approximation are inversely proportional.

# 1-5 Conclusions:

We have learned to use the Explicit Euler Method for solving the ODE for the Parachutist problem in previous classes and we learned the approximated results from numerical methods. From our books, we found an exact solution for this problem.

For each time interval, with our knowledge of Python we wrote a code to plot a figure of velocity vs. time for both the exact solution and our Euler’s Method solution. We used these plots to compare the curve of Explicit Euler Solution and the curve of the Exact Solution.

We had five figures in total for this project once we completed the five different time intervals. The goal in this project was to have our approximation eventually be equal to zero. By presenting this in our graphs above, we were able to show this variance in accuracy as we moved from time interval to time interval.