```python
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  7 22:43:06 2019
HW6b
@author: Ryan Dalby
"""
import numpy as np
import matplotlib.pyplot as plt

def trapezoidalRuleIntegration(func, xi, xf, numSegments):
    """
    Given a function f(x) and starting integration value (xi) and ending integration value (xf)
    along with the number of segments to apply the trapezoidal rule will take the definite integral between the
    """

    h = (xf-xi)/numSegments
    x = xi #current x

    runningSum = func(x) #first segment
    x += h
    #now loop over middle segments
    for i in range(numSegments-1):
        runningSum += 2 * func(x)
        x += h
    runningSum += func(x) #final segment

    I = (h/2.0) * runningSum #integral value
    return I

def simpsonsOneThirdRuleIntegration(func, xi, xf, numSegments):
    """
    Given a function f(x) and starting integration value (xi) and ending integration value (xf)
    along with the number of segments (must be even number) to apply simpson's 1/3 rule to take
    the definite integral between the designated values.
    """
    if(np.mod(numSegments, 2) == 1): #if total number of segments is odd then can't use method
        raise Exception("Number of segments must be even")

    h = (xf-xi)/numSegments
    x = xi #current x

    runningSum = func(x) #first segment
    x += h
    #now loop over middle segments
    for i in range(numSegments-1):
        if (np.mod((i+1), 2)) == 1 : #Segment is odd
            runningSum += 4 * func(x)
        else: #segment is even
            runningSum += 2 * func(x)
        x += h
    runningSum += func(x) #final segment
    I = (h/3.0) * runningSum #integral value
    return I


g = 9.81 #m/s^2
m = 68.1 #kg
cd = 0.25 #kg/m
vFunc = lambda t: np.sqrt((g*m)/cd) * np.tanh(np.sqrt((g * cd)/m) * t)

#Part 1
print("Part 1:")
print("Using the Trapezoidal Rule Integration with 10 segments the distance fallen after 10s = {:.5f}m".format(
```

```python
print("Using the Simpson's 1/3 Rule with 10 segments distance the fallen after 10s = {:.5f}m".format(simpsonsO
print()

#Part 2
print("Part 2:")

iterVals = np.arange(100, 110000, 1000)
trapError = []
simpError = []
trueVal = 334.178167 #analytical solution
for i in iterVals: #get true error for each segment size
    trapError.append(np.abs(trueVal - trapezoidalRuleIntegration(vFunc,0,10, i)))
    simpError.append(np.abs(trueVal - simpsonsOneThirdRuleIntegration(vFunc,0,10, i)))

trapMinIndex = iterVals[np.argmin(trapError)]
trapMin = np.min(trapError)
simpMinIndex = iterVals[np.argmin(simpError)]
simpMin = np.min(simpError)
print("The optimum number of segments using the Trapezodial Rule for this problem is {}".format(trapMinIndex))
print("The optimum number of segments using Simpson's 1/3 Rule for this problem is {}".format(simpMinIndex))

fig, ax1 = plt.subplots(figsize = (10,5))
ax1.plot(iterVals, trapError, label = "Trapezodial Rule")
ax1.plot(iterVals, simpError, label = "Simpson's 1/3 Rule")
ax1.scatter(trapMinIndex, trapMin)
ax1.scatter(simpMinIndex, simpMin)
ax1.set_title("True error versus number of segments used")
ax1.set_xlabel("Number of segments")
ax1.set_ylabel("True Error (m)")
ax1.legend()
```