Runge-Kutta: Jessica Murphy

13/03/2023

Homework 19

This python programme involves an Initial Value Problem where it uses a numerical technique to model the rate at which the volume of water in a tank changes - it is called the **Runge-Kutta** technique and is implemented through **third-order Runge-Kutta technique**.

Write a Python notebook which uses the third order Runge-Kutta technique to estimate how the volume of water changes over a 10 second period. Use a step size of Δt = 2 seconds. Here a = 0.1 Litre s^{-2} and b = 0.5 s^{-1}

The differential equation which describes how the volume of water in the tank changes as time goes by is;

$$\frac{dV}{dt} = at - bV$$

It is an increasing differential equation that describes how water flows into the large tank at a rate that increases with time.

It depends on two terms: a positive term describing how V increases as t increases, and a negative term describing how V itself decreases by flowing out of a hole in the base of the tank. The rate at which the volume of water in a tank changes with respect to time is proportional to these two terms, where a and b are constants of proportionality.

This code calculates the accuracy of the Runge-Kutta method for each of the three step sizes (2, 0.2, and 0.02), and then plots a log10-log10 graph of accuracy versus step size. Finally, the code uses np.polyfit to verify the order of the Runge-Kutta method.

The output of the code should include the log-log plot of accuracy versus step size, as well as the order of the Runge-Kutta method, which should be close to 3. A discussion of the analysis of the slope and graph is made at the end of the notebook.

1. Define a function for the differential equation.
   - set the initial conditions of a and b
2. Import python library NumPy
3. Set the number of steps required
4. Set up NumPy arrays to hold the time (***t***) and volume (***V***) and initialise.
5. Use a for loop to step along the t-axis. Use the Runge-Kutta technique to obtain formulae for k1, k2 and k3 to simulate more accurate physical conditions.
6. Print the estimated value of V after 10 seconds
7. In a new cell, import Matplotlib
8. Assign arrays to hold the solutions of V(10) with time steps 2, 0.2, and 0.02
9. Use the analytical solution to calculate the error
10. To get a linearly space graph, plot on a log-log graph
11. Plot the accuracy versus step size on a graph
12. Verify the order of the Runge-Kutta method by printing the slope of the graph
13. Graph a line of best fit along the points
14. Draw conclusions from the graph and slope, and discuss.

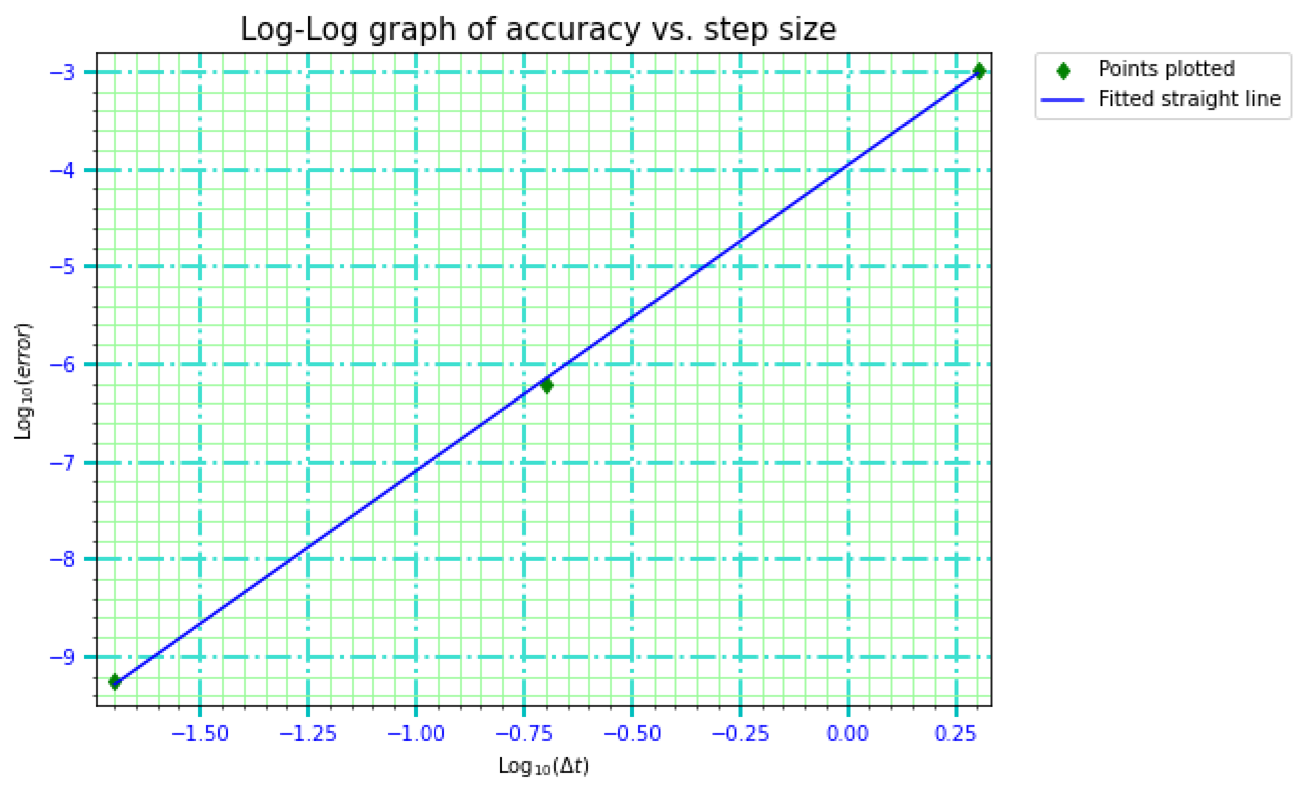
In [1]:
```python
# solve the diffrential equation dV/dt = at - bV
# This is the rate at which the volume of water in the tank changes
# initial conditions a = 0.1 litre per second^2, b = 0.5 per second
# use third order Runge Kutta technique to determine V(10)

# define the differential equation, a function while setting the conditions
def slope(t, v):
    # set the initial conditions of a and b
    a = 0.1 # Rate of water flow into the tank in units (Litre s^-2)
    b = 0.5 # Rate of water flow out of the tank in units (s^-1)
    m = (a*t) - (b*v)
    return m

# import python library numpy
import numpy as np

# determine the number of steps required
t_max = 10 # iterate up to a maximum of t_max in seconds
delta_t = 0.02 # set the time step (seconds)
N = int(t_max / delta_t) # calculate the number of time jumps

# set up NumPy arrays to store the x and y values and initialise, explicitly stating variables T and V start at 0
# need one more element in each array than the number of jumps, to include the zeroth element
T = np.zeros(N + 1)
V = np.zeros(N + 1)
T[0] = 0 # start at t = 0
V[0] = 0 # The tank is initially empty, volume = 0

# use a for loop to step along the t-axis and implement the third-order Runge-Kutta technique
for i in range(N):

    #calculate three temporary intermediate values k1, k2, and k3.
    # use slope at start to estimate the change in V over the interval
    k1 = slope(T[i], V[i]) * delta_t

    # use slope at centre of interval to estimate the change in V over the interval
    k2 = slope(T[i] + 0.5*delta_t, V[i] + 0.5*k1) * delta_t

    # estimate of the slope of the volume function at the end of the current time interval, based on the current volume (V[i]) and the two previous slope estimates (k1 and k2).
    k3 = slope(T[i] + delta_t, V[i] - k1 + 2*k2) * delta_t # more optimal, and k1 and k2 errors 'cancel' eachother out

    V[i+1] = V[i] + ((1/6) * k1) + ((2/3) * k2) + ((1/6) * k3) # estimate V at the end of the interval, k2 value seen to dominate equation
    T[i+1] = T[i] + delta_t # calculate t at the end of the interval

# print the estimated value of V when t = 10 s
print('The estimated value of V after 10 seconds is {}'.format(V[N]))
```

```
The estimated value of V after 10 seconds is 1.6026951782336107
```

Δ t (s)	Volume (V)
2	1.6016460905349794
0.2	1.6026945705253637
0.02	1.6026951782336107

In [2]:
```python
# import matplotlib to plot graph
import matplotlib.pyplot as plt
%matplotlib inline

# use results from solutions
del_t = np.array([2, 0.2, 0.02])
v_10 = np.array([1.6016460905349794, 1.6026945705253637, 1.6026951782336107])

# use analytical solution to calculate error
v_analytic = 1.6026951787996095
error = abs(v_10 - v_analytic)

# plot on a log-log graph and fit a straight line
log_delta_t = np.log10(del_t)
log_error = np.log10(error)

# Plot the accuracy versus step size on a graph
plt.figure(figsize=(8,6)) # increase the figure size
plt.plot(log_delta_t, log_error, 'gd', label='Points plotted')
fontsize = 15 # stating fontsizes wanted for title
axessize = 12
plt.title('Log-Log graph of accuracy vs. step size ', fontdict={'fontsize': fontsize}) # title graph
plt.xlabel('Log$_{10}(\Delta t)$')
plt.ylabel('Log$_{10}(error)$',)
plt.rc('axes', titlesize=axessize) # increase axes titles
plt.xlim(-1.74, .33) # maximise area of graph
plt.ylim(-9.5, -2.8)

# customise grid
plt.tick_params(axis='both', direction='out', length=6, width=2, labelcolor='b', colors='c', grid_color='deeppink', grid_alpha=1) # make axes ticks red and writing blue
plt.grid(visible=True, color='turquoise',linestyle='-.', linewidth=2) # major grid lines
plt.minorticks_on()
plt.grid(visible=True, which='minor', color='palegreen', alpha=0.8, ls='-', lw=1) # minor grid lines

# Verify the order of the Runge-Kutta method
[m,c] = np.polyfit(log_delta_t, log_error, 1)
print('The slope of the graph = {0:5.4}'.format(m))

# fit straight line
log_delta_t_101 = np.linspace(log_delta_t[0], log_delta_t[-1], 101)
log_error_101 = m * log_delta_t_101 + c # y = mx +c
plt.plot(log_delta_t_101, log_error_101, 'b-', label='Fitted straight line')
plt.legend(bbox_to_anchor = (1.05, 1), loc = 'upper left', borderaxespad=0) # adding legend
plt.show()
```

```
The slope of the graph = 3.134
```



Discussion and conclusion drawn

Here, we see a straight line on the log-log graph. The slope is equal to **3.134** which is approximately 3, and this represents the power to which the step size (Δt) must be raised in order to obtain a proportional decrease in the error (i.e., the rate of convergence of the method)

Since the slope is positive, it indicates that the error decreases as the step size decreases, which is expected behavior for a well-behaved numerical method. If the slope is negative, it indicates that the error increases as the step size decreases, which would be a sign of instability or poor accuracy.

The value of the estimated V(10) is very close to the analytical solution, and the slope obtained from the np.polyfit function applied to the plot provides an estimate of the order of accuracy. Essentially, **the plot indicates that the third-order Runge-Kutta technique has an order of accuracy close to 3**. This is consistent with the theoretical expectation.