Runge-Kutta

Kaylin Shanahan 2023

This is a python programme which uses the third order Runge-Kutta technique to estimate how the volume of water changes in a tank over a period of time.

The water flows into a large tank at a rate that increases with time. The water also flows out through a hole in the base at a rate which is proportional to the volume of water in the tank. The rate at which the volume of water in the tank changes is described by:

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

Where $a = 0.1\ Litre\ s^2$ and $b = 0.5\ s^{-1}$.

Initially, the tank is empty. The time period for which the volume of water changes will be 10 seconds, and the step size for change in time will be 2 seconds.

The third order Runge-Kutta, as the first and second orders, suppose that, in general, that the slope is a function of both x and y:

$$\frac{dy}{dx} = f'(x, y)$$

There are three temporary intermediate values k_1, k_2 and k_3:

1. $k_1 = f(x_i, y_i)$
2. $k_2 = f(x_i + \frac{h}{2}, y_i + k_1\frac{h}{2})$
3. $k_3 = f(x_i + \frac{h}{2}, y_i + (-k_1 + 2k_2)h)$

Then, the third order Runge-Kutta can be expressed as:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 4k_2 + k_3)h$$

By then applying this method to our given simple equation, we can estimate how the volume of water changes in a tank over a period of 10 seconds.

In [90]:

```python
# Solve the differential equation dV/dt = at - bV
# Intial conditons: a = 0.1, b = 0.5, V = 0
# Use 3rd order Runge Kutta tecnique to determine y(10)

# Import numpy for use in mathematics
import numpy as np

# Calculate the slope of the function
def slope(t, V):
    m = a*t - b*V
    return m

# Set the initial conditions
t_init = 0    # Initial time elapsed in seconds
V_init = 0    # Initial volume of water in tank in litres
a = 0.1       # Constant a in units of litres s^-1
b = 0.5       # Constant b in units of s^-1

# Determine the number of steps required
t_max = 10    # Iterate up to a maximum time of t_max (seconds)
delta_t = 2   # Set the time step (seconds)
N = int((t_max - t_init)/delta_t)   # Calculate number of jumps

# Set up NumPy arrays to hold the x and y values and initialise
t = np.zeros(N + 1)
V = np.zeros(N + 1)
t[0] = t_init
V[0] = V_init

# Use a for loop to step along the x_axis
for i in range(N):

    # Use the slope at start to estimate change in y over interval
    k1 = slope(t[i], V[i]) * delta_t

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

    # Use slope at end of interval to estimate change in y at end of interval
    k3 = slope(t[i] + delta_t, V[i] - k1 + 2 * k2) * delta_t

    V[i+1] = V[i] + k2
    t[i+1] = t[i] + delta_t
```

```
# Print the x and y values
for i in range(N+1):
    print("i ={0:3}, t ={1:6.3}, V ={2:12.10}".format(i, t[i], V[i]))
```

```
i =  0, t =   0.0, V =          0.0
i =  1, t =   2.0, V =          0.2
i =  2, t =   4.0, V =          0.5
i =  3, t =   6.0, V =         0.85
i =  4, t =   8.0, V =        1.225
i =  5, t =  10.0, V =       1.6125
```

In [79]:

```
print("The volume of water remaining in the tank after 10 seconds is: {0:3} litres".format(V[5]))
```

```
The volume of water remaining in the tank after 10 seconds is: 1.6125 seconds
```

- The programme outputs a final estimate for how the volume of water change over a 10 second period, using the initial step size of 2 seconds, of 1.6125 Litres.

Table of Δt and V Values

Δt	V
2	1.6125
0.2	1.602719499
0.02	1.602695405

- As evidenced by the table above, with a large step size a reasonable answer can be achieved, but to achieve a more accurate answer the step size must be increased

Accuracy and Order of Runge-Kutta Technique:

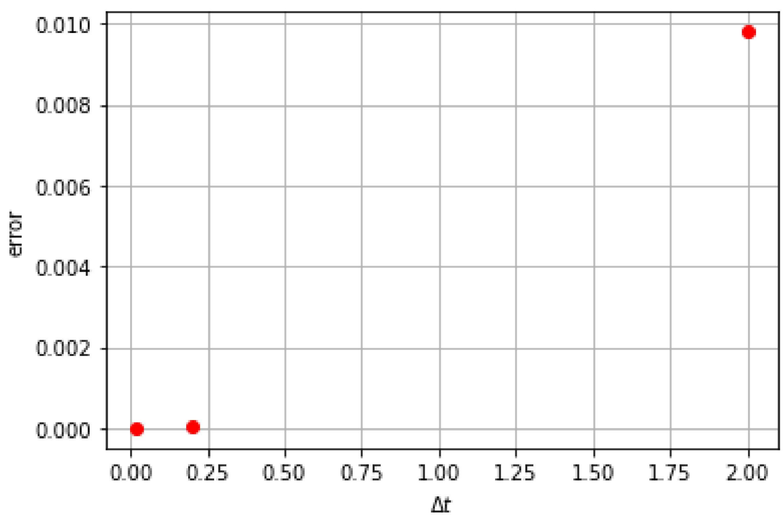
In [88]:

```
# Check the "order" of Runge-Kutta technique
import numpy as np
import matplotlib.pyplot as plt

# Use results from solution to dV/dt = at - bV, V(0) = 0
delta_t = np.array([2, 0.2, 0.02])
V_10 = np.array([1.6125, 1.602719499, 1.602695405])

# Use analytical solution to calculate Euler error
V_analytic = 1.6026951787996095
error = abs(V_10 - V_analytic)

plt.plot(delta_t, error, "ro")
plt.xlabel("$\Delta t$")
plt.ylabel("error")
plt.grid()
plt.show()
```



- The relationship appears to be an exponential curve, so it is third order.

- This can also be verified by plotting the data on a log-log graph
- The slope of a log-log graph will be the power in the relationship between the error and the step size; the order of the Runge-Kutta technique used

In [85]:

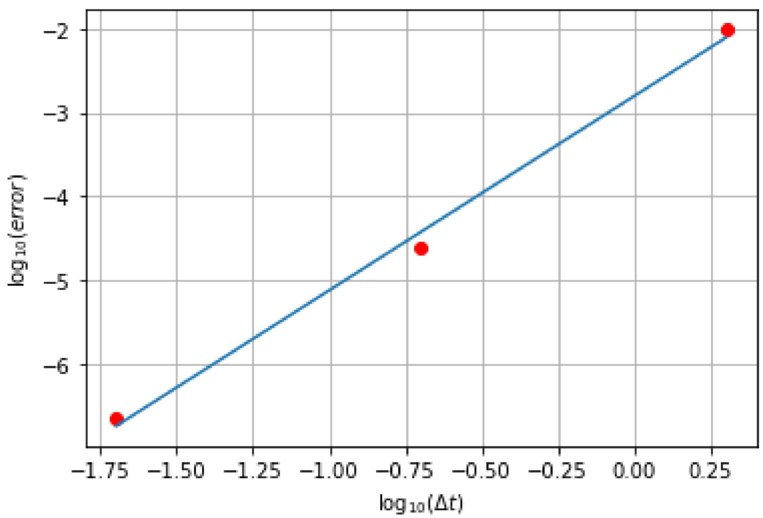
```
# Plot on a log-log graph and fit a straight line
log_delta_t = np.log10(delta_t)
log_error = np.log10(error)

plt.plot(log_delta_t, log_error, "ro")
plt.xlabel("log$_{10}(\Delta t)$")
plt.ylabel("log$_{10}(error)$")
plt.grid()

[m,c] = np.polyfit(log_delta_t, log_error, 1)
print("slope = {0:6.4}".format(m))

log_delta_t_101 = np.linspace(log_delta_t[0], log_delta_t[-1], 101)
log_error_101 = m * log_delta_t_101 + c
plt.plot(log_delta_t_101, log_error_101)
plt.show()
```

```
slope =  2.318
```



- By plotting the difference between the Runge-Kutta estimate of volume of water (V) and the exact analytical solution, as a function of Δt, on a log-log graph we can confirm that the technique is third order

Conclusion:

- With the above information, I can verify that this runge kutta technique is of the third order.