PHAS0029-C6-assignment

February 27, 2025

1 PHAS0029 Session 6: ODEs, part 1 - assignment

Updated: 18/02/2025

This notebook contains your tasks for Session 5. Please make sure you go through the theory notebook before attempting the tasks here. Remember to use text cells to describe your reasoning and results, and comments to annotate the code. You can cut and paste code, equations and images from the theory notebook if you want, as long as your notebook makes it clear where you have pasted material and where it came from. All headings have been removed from this notebook to give you a chance to structure it yourself. This should be a good preparation for your final assignment where you will be creating a notebook from scratch.

[3]:
```
%matplotlib inline

import numpy as np

import matplotlib.pyplot as plt
```

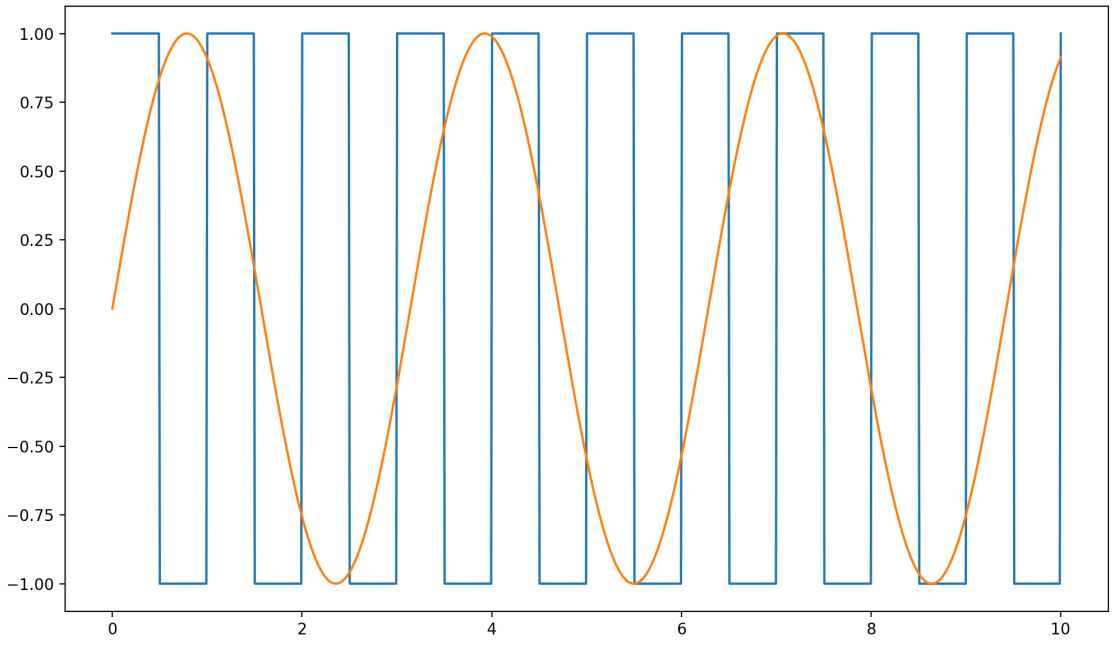
1.0.1 Task 1

Write two Python functions called Vinsq and Vinsin to represent mathematical function for the square and sine wave inputs respectively, as defined in the theory notebook.
In the next (added) cell, use your function Vinsq to plot a square wave in the range $0 \leq t \leq 10\mu$s. Make sure you include enough time points to ensure the signal is properly square. Hint 1: you may find the function np.floor useful here. Hint 2: A good way of determining whether an integer n is odd or even is to use a construct like $(-1)^n$

[57]:
```
def Vinsin(x):
    """Returns sine wave"""
    return np.sin(2*x)

def Vinsq(x):
    """Returns square wave"""
    curve = 2 * x
    two_t = np.floor(curve)   #makes square wave amplitude between -1 and 1
    plot = (-1)**two_t

    return plot
```

[58]:
```
#plotting a square wave and a sine wave
x = np.linspace(0,10,1000)


plt.figure()
plt.plot(x,Vinsq(x), label = "Square wave") #sqaure wave

plt.plot(x,Vinsin(x), label = "Sine wave") #sine wave
plt.show()
```

[58]:



[59]:
```
%run -i c5checkpoint1-patch.py
```

```
************
 Well done! All test passed. You can move to the next part of the task.
************
```

[59]:
```
<Figure size 1200x700 with 0 Axes>
```

1.0.2 Task 2

Write a function called <tt>f</tt> to represent the RHS of the differential equation. Your fun

NB. Although you should generally avoid it, this time your function should use global variables for resistance and capacitance as a single variable RC (time constant)- make sure you explain that in your docstring. This is to make sure you appreciate the difference and can use both local and global variables in the future.

[86]:
```
def f(v_in,v_out,t):
    """differential euqation RHS. We are using global
    varibales this time to deine RC"""

    RHS = (v_in(t) - v_out)/RC #from diff equation
    return RHS
```

[87]:
```
%run -i c5checkpoint3-patch.py
```

```
************
 Well done! All test passed. You can move to the next part of the task.
************
```

1.0.3 Task 3

Write a function called RungeKutta that performs fourth-order Runge-Kutta function. Make sure your function takes 5 arguments: input function, RHS function from the differential equation, time step, initial position and time data.

[105]:
```
def RungeKutta(v_in,f,h,x,data):
    """performs 4th order Runge Kutta method"""

    xpoints = []
    for t in data:
        xpoints.append(x)

        k1 = h * f(v_in,x,t)
        k2 = h*f(v_in,x+0.5*k1, t+0.5*h)
        k3 = h*f(v_in,x+0.5*k2, t+0.5*h)
        k4 = h*f(v_in,x+k3, t+h)          #taylor expansion to 4th order
        x = x + (k1 + 2*k2 + 2*k3 + k4)/6 # add up all taylor expansions

    return xpoints
```

[106]:
```
%run -i c5checkpoint2-patch.py # NB. this checkpoint may take a few seconds to␣
↪run
```

```
************
 Well done! All test passed. You can move to the next part of the task.
************
```

1.0.4 Task 4

Define the start and end points, initial conditions, create an array of time values and solve the equation for the square wave.

Plot the input square wave and the output of this (on the same plot) for the range $0 \leq t \leq 10$ μs when the time constant $RC = 1\mu$s. Repeat this for time constants $RC = 0.1$ and 0.01μs (i.e. three plots, each with an input and output line). Use the initial condition $V_{\text{out}} = 0$V.

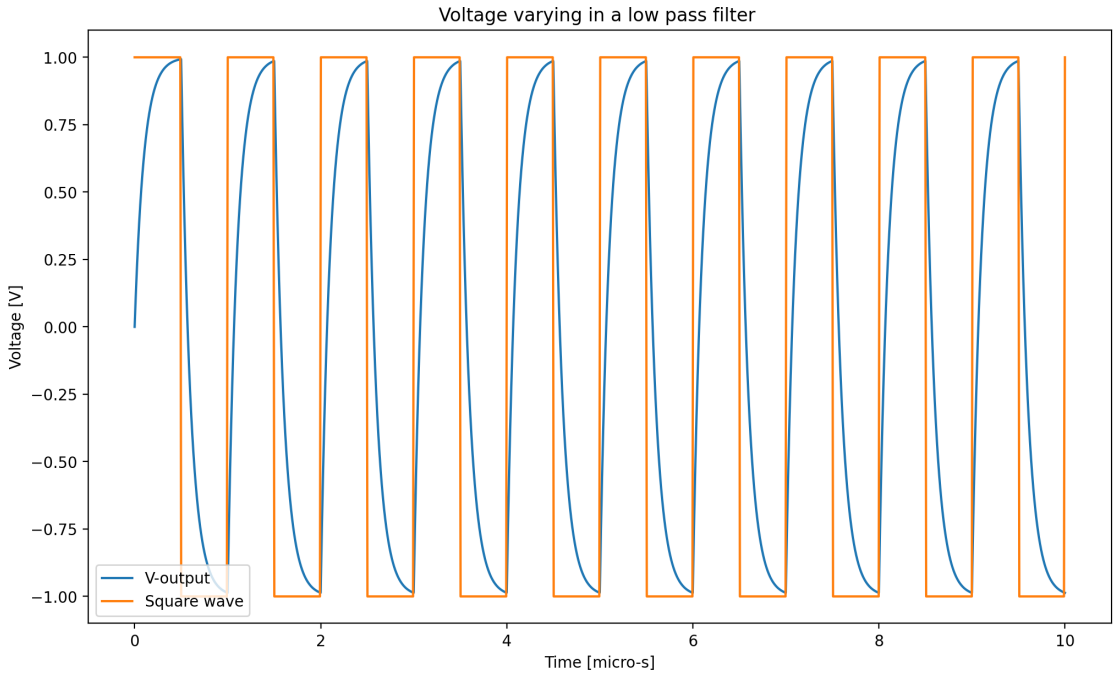
Your code should be properly annotated and you should not redefine any functions from above.

In a text cell, describe what you see and what you think the circuit is doing. (Hint: Why do you think it is called a "low-pass filter"?)

[167]:
```
#plotting square wave and v_out- the solved diff equation
RC = 0.1
t = np.linspace(0,10,10000)
y = RungeKutta(Vinsq,f,1/1000,0,t) #step should be the same as the t values in␣
 ↪linspace
plt.figure()
plt.plot(t,y,label = 'V-output')
plt.plot(x,Vinsq(x), label = "Square wave")
plt.xlabel('Time [micro-s]')
plt.ylabel('Voltage [V]')
plt.legend(loc="lower left",prop={'size':10})
plt.title("Voltage varying in a low pass filter ")
```

[167]:
```
Text(0.5, 1.0, 'Voltage varying in a low pass filter ')
```

[167]:

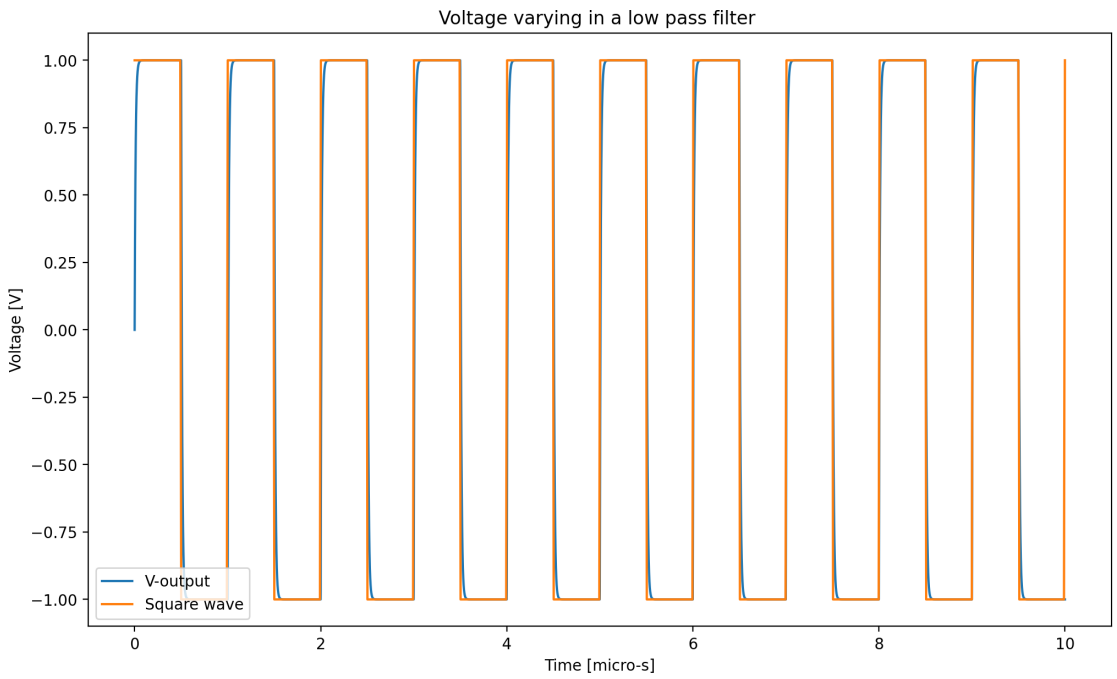


[168]:
```
#same thing as previous cell, just diffeent RC value
RC = 0.01
t = np.linspace(0,10,10000)
y = RungeKutta(Vinsq,f,1/1000,0,t)
plt.figure()
plt.plot(t,y,label = 'V-output')
```

```
plt.plot(x,Vinsq(x), label = "Square wave")
plt.xlabel('Time [micro-s]')
plt.ylabel('Voltage [V]')
plt.legend(loc="lower left",prop={'size':10})
plt.title("Voltage varying in a low pass filter ")
```

[168]: Text(0.5, 1.0, 'Voltage varying in a low pass filter ')

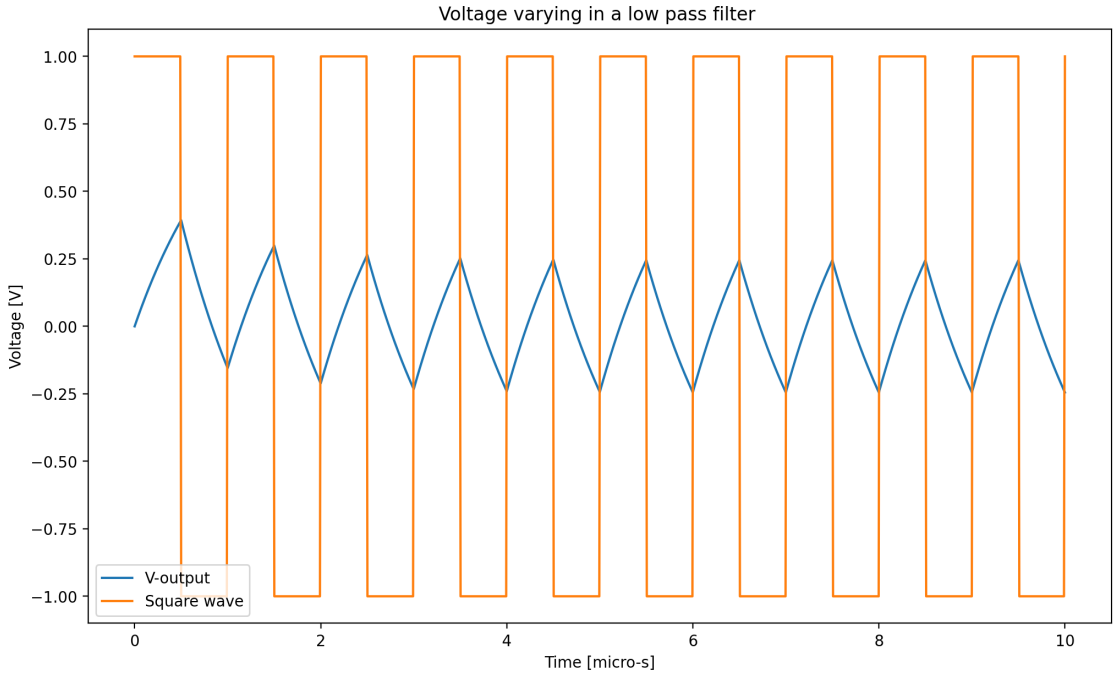
[168]:



[169]:
```
#same thing as previous cell
RC = 1
t = np.linspace(0,10,10000)
y = RungeKutta(Vinsq,f,1/1000,0,t)
plt.figure()
plt.plot(t,y,label = 'V-output')
plt.plot(x,Vinsq(x), label = "Square wave")
plt.xlabel('Time [micro-s]')
plt.ylabel('Voltage [V]')
plt.legend(loc="lower left",prop={'size':10})
plt.title("Voltage varying in a low pass filter ")
```

[169]: Text(0.5, 1.0, 'Voltage varying in a low pass filter ')

[169]:



What seems to be happening is you have an output signal, which is between -1V and 1V, but if the output voltage gets too big, then it gets cut off. When RC = 1, the input signal does not reach the 1,-1 boundary, but for all 3 graphs the amplitude shifts after every half wavelengths. The lower the value of RC, the closer you get to the original square wave. It is called a low pass filter as it does not allow voltage to be larger than a certain value.

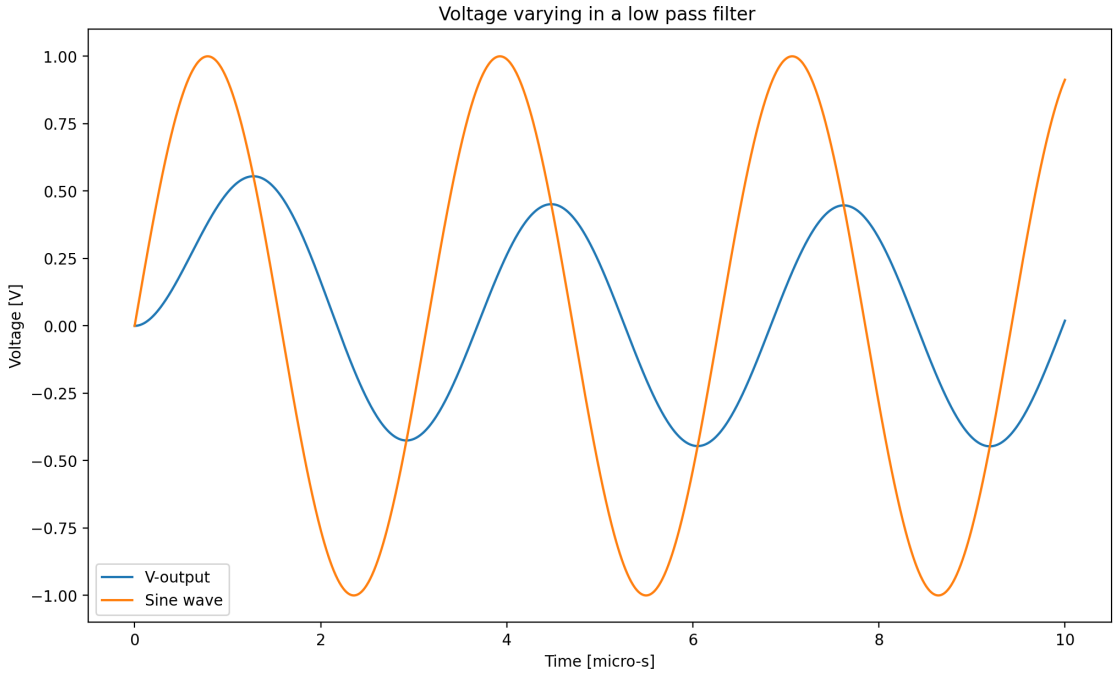
1.0.5 Task 5

Repeat the task (solve and plots) for a sine wave: $V_{\text{in}} = \sin(2t)$. What effect does the filter have now? NB. You should not redefine any functions from above.

[170]:
```
#same thing as before for different v input value

RC = 1
t = np.linspace(0,10,1000)

y = RungeKutta(Vinsin,f,1/100,0,t)
plt.figure()
plt.plot(t,y, label = 'V-output')
plt.plot(t,Vinsin(t), label = "Sine wave")
plt.xlabel('Time [micro-s]')
plt.ylabel('Voltage [V]')
plt.legend(loc="lower left",prop={'size':10})
plt.title("Voltage varying in a low pass filter ")
```

[170]:
```
Text(0.5, 1.0, 'Voltage varying in a low pass filter ')
```

[170]:


Voltage varying in a low pass filter
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Voltage [V]
V-output
Sine wave
0
2
4
6
8
10
Time [micro-s]

[0]: