J. Hedberg PHYS 35100 Fall 2023 HW1

Problem 1 → Derive 1d kinematics

start with definition of acceleration in terms of vel.

$$a = \frac{dv}{dt} \rightarrow \text{sep. variables}$$

$$dv = a\,dt \rightarrow \text{integrate}$$

$$\int_{v_0}^{v} v = a \int_{0}^{t} dt \qquad (a \text{ is constant})$$

$$v - v_0 = at$$

$$v = v_0 + at$$

Now use

$$\frac{dx}{dt} = v$$

$$\int dx = \int v\,dt$$

$$\int_{x_0}^{x} dx = \int_{0}^{t} (v_0 + at)\,dt$$

$$x - x_0 = v_0 t + \frac{1}{2} a t^2$$

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

Plot simple 1d kinematics

For CCNY PHYS 35100

Fall 2023

J. Hedberg

This notebook is found here:
https://colab.research.google.com/drive/1XYIAq_pELYUnjXVZKiKqOQcB-sc7FW7q?usp=sharing

Let's plot some PHYS 20700 level kinematics, namely the speed and position of an object undergoing constant acceleration, i.e. $\frac{dv}{dt} = \text{constant}$

Our first step is to import two libraries that will be useful: `numpy`, to handle variables in array formats and `mapplotlib` to the plotting.

```
import numpy as np
import matplotlib.pyplot as plt
```

Our experiment involves one constant value: the acceleration of our particle. We can call it a and give it a value of 5.

```
# Define a variable to store the value of  a

a = 5
```

Now, we create a list (or array) of time values. The function `np.linespace()` creates a linear spacing of values set by the 3 arguments in the parenthesis: begin, end, and how many. If you didn't know how the function works, you can always look it up in the documentation: Numpy Linspace

```
# make a list called time that ranges from 0 to 10 seconds.

time = np.linspace(0, 10, 10)
```

```
print(time)
```

```
[ 0.          1.11111111  2.22222222  3.33333333  4.44444444  5.55555556
  6.66666667  7.77777778  8.88888889 10.        ]
```

Next, we use our physics understanding to make a function for speed v, as a function of time. The following line will create a new list `speed` that is populated by the basic kinematic function that related speed, acceleration, and time.

(We'll start at rest for simplicity sake)

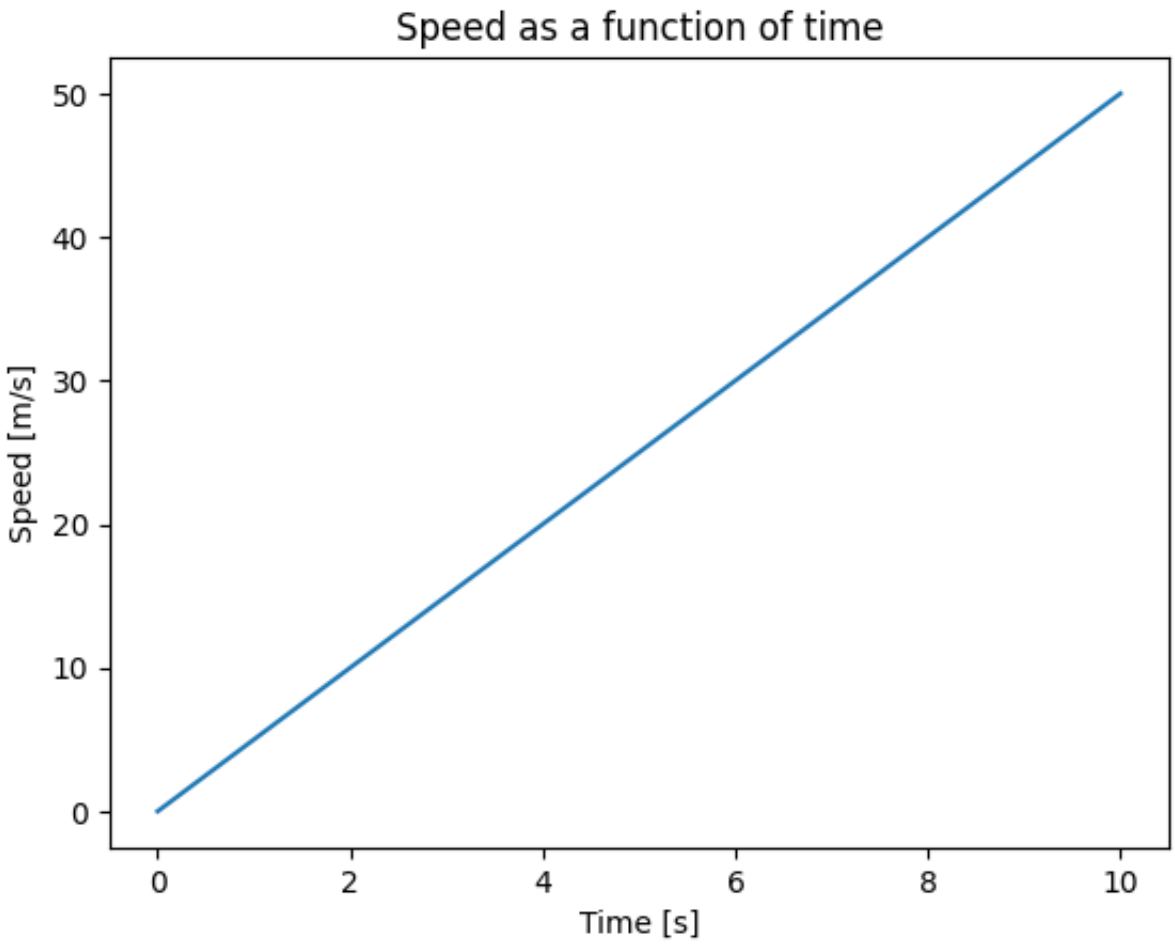
$$v = v_0 + at$$

```
# make a new list called speed

speed = a*time


print(speed)
```

```
[ 0.          5.55555556 11.11111111 16.66666667 22.22222222 27.77777778
 33.33333333 38.88888889 44.44444444 50.        ]
```

Now, we can plot speed as a function of time using our two lists of values that we just created. Make sure to label the axes (and include the units), and give the plot a title.

```
fig, ax = plt.subplots()

ax.plot(time, speed)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Speed [m/s]')
ax.set_title('Speed as a function of time')

plt.show()
```



We can also consider the position of the object. That will be given by:

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

(We can assume for simplicity that x_0 and v_0 are both zero.)

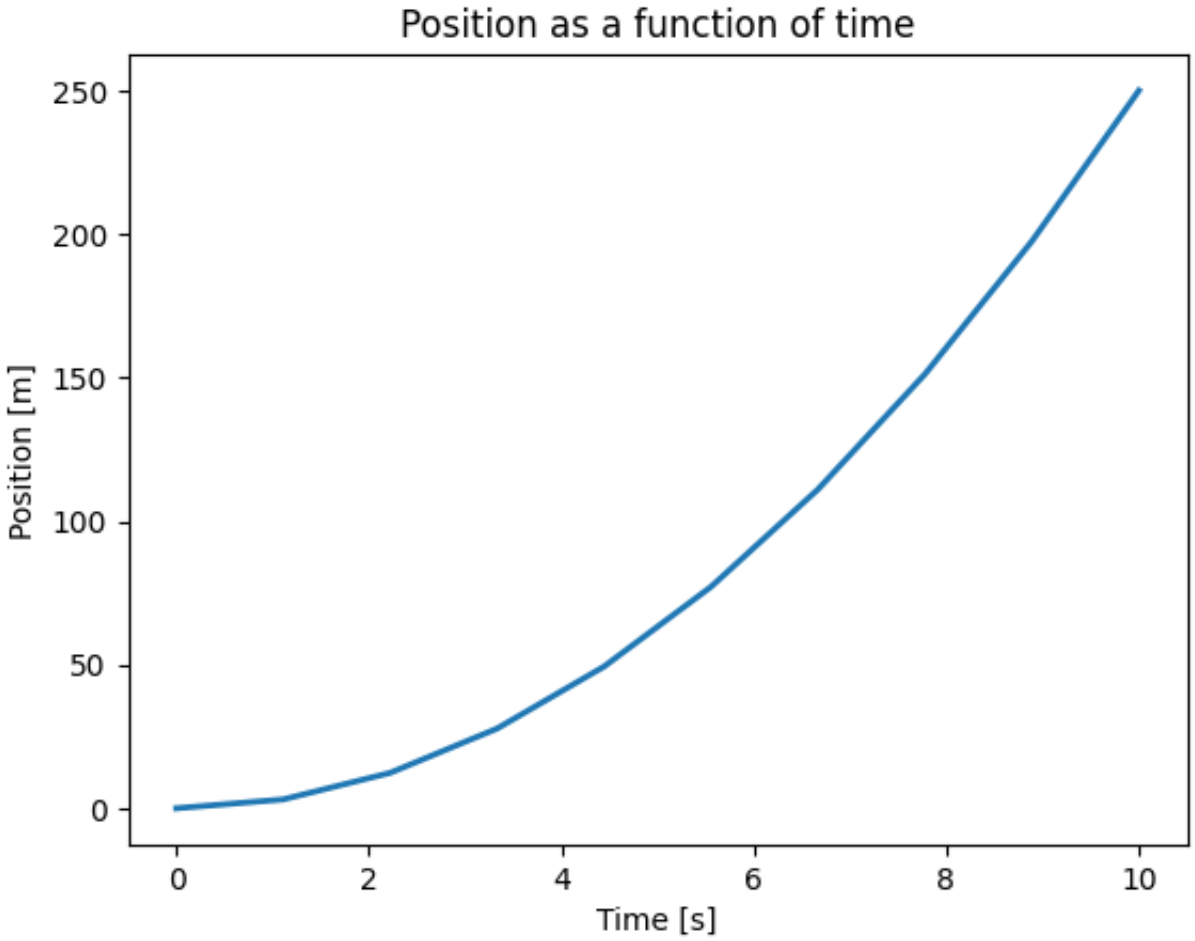
```
# create the position list based on the equation above
# we can use the np.square() function to ask for the square of each value in the ti
# rather than the square of the 1-d matrix.

position = 0.5*a*np.square(time)
```

```
print(position)
```

```
[  0.           3.08641975  12.34567901  27.77777778  49.38271605
  77.16049383 111.11111111 151.2345679  197.5308642  250.        ]
```

Do the same plot routine, but change the `speed` to `position`

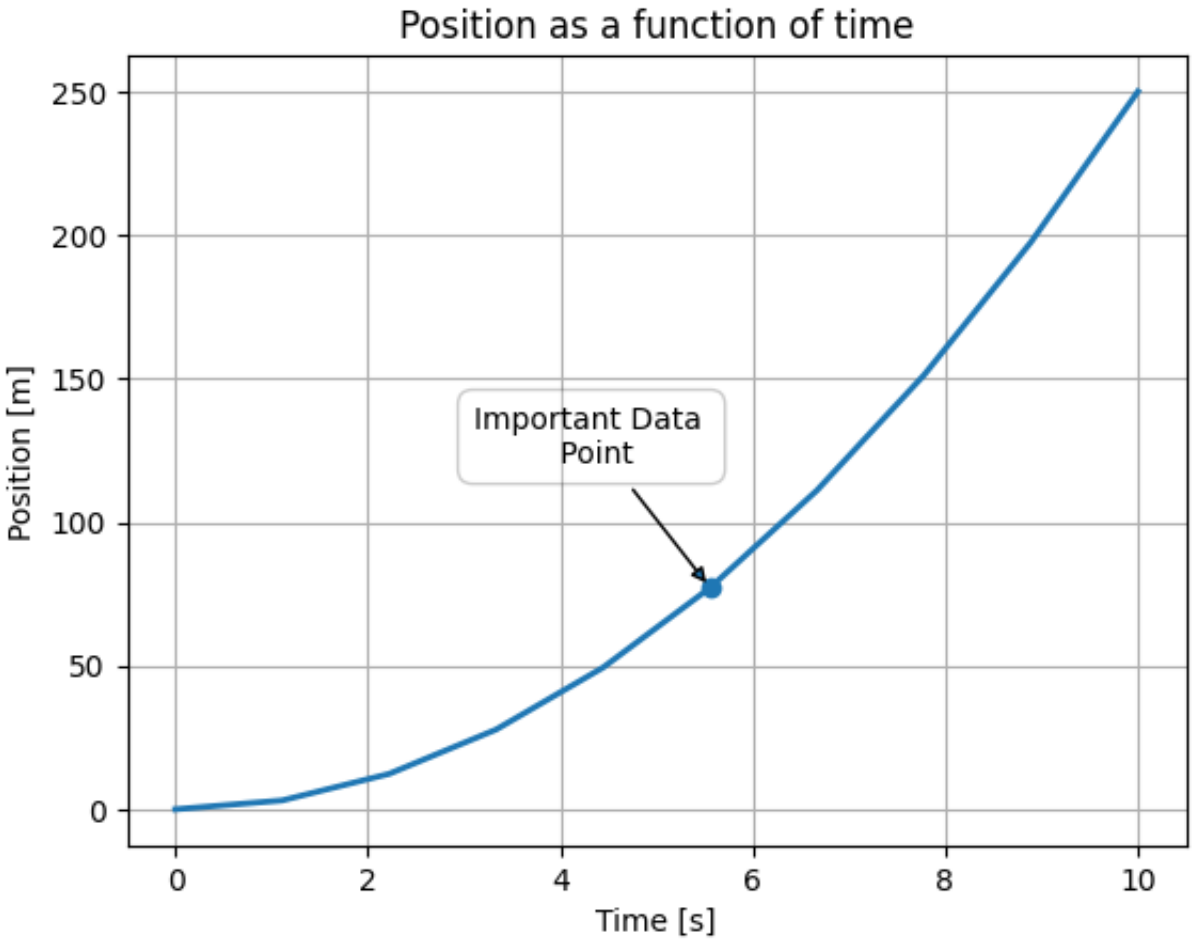
```
fig, ax = plt.subplots()

ax.plot(time, position, linewidth=2.0)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Position [m]')
ax.set_title('Position as a function of time')

plt.show()
```



We can annotate plots too, to help explain things. Here's an example where the point halfway through the motion is highlighted.

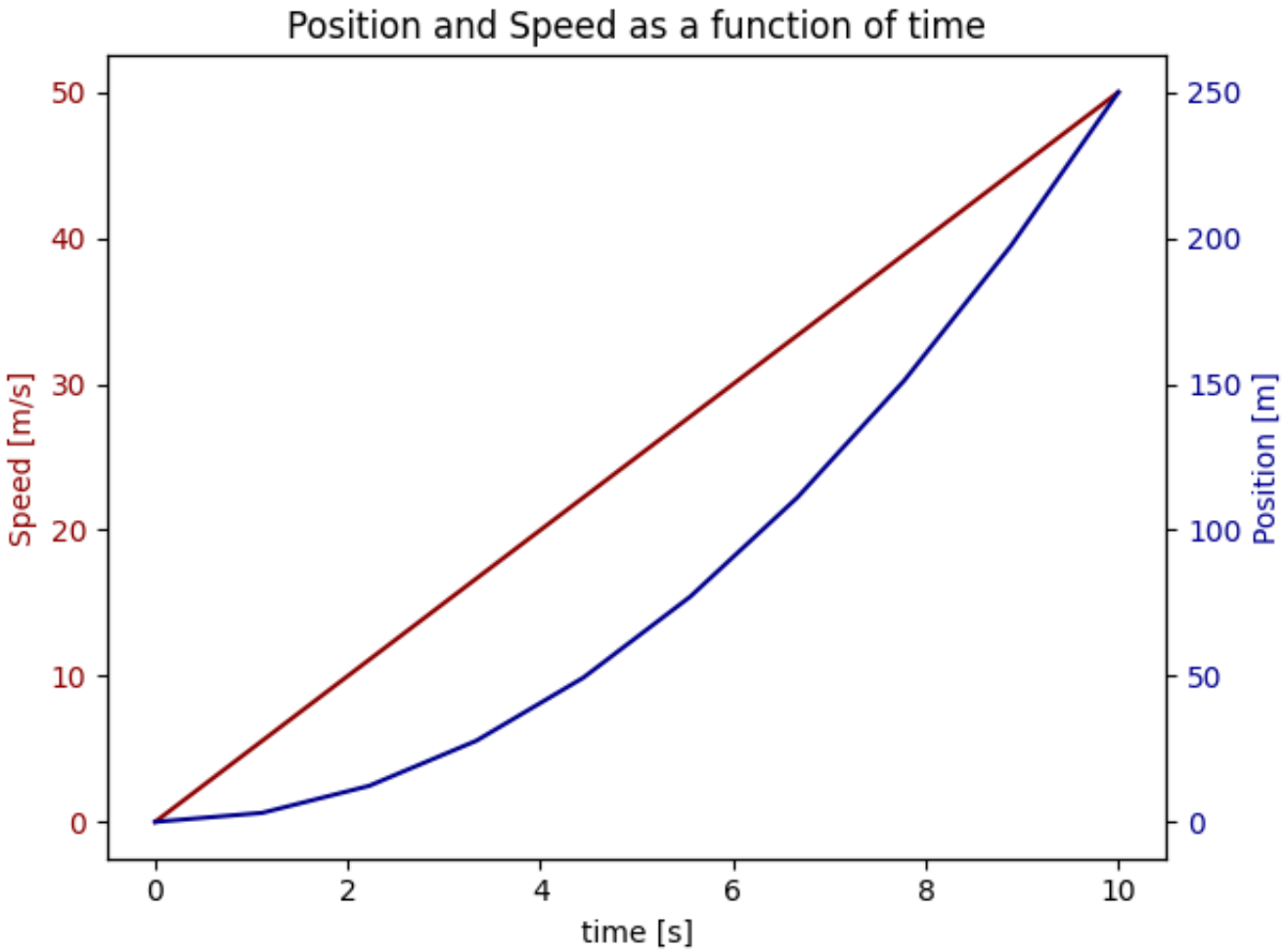
```
# Since our lists where 100 values long, we can just call our special point = 50.
```

```
# Since our lists where 100 values long, we can just call our special point = 50.

specialPoint = 5

fig, ax = plt.subplots()

ax.plot(time, position, linewidth=2.0)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Position [m]')
ax.set_title('Position as a function of time')
ax.grid()
# This line adds a circular 'o' marker at the specialPoint values for time and posi
ax.plot(time[specialPoint], position[specialPoint],'o', color = 'tab:blue')

# All of this is for the annotation.

plt.annotate("Important Data \n Point", # this is the text
                 xy=(time[5], position[5]), # these are the coordinates to positior
                 textcoords="offset points", # how to position the text
                 xytext=(-40,40), # distance from text to points (x,y)
                 ha='center',
                 va='bottom',
                 bbox=dict(boxstyle='round,pad=0.5', fc='white', alpha=0.2),
                 arrowprops=dict(arrowstyle = '-|>', connectionstyle='arc3, rad=0')

plt.show()
```



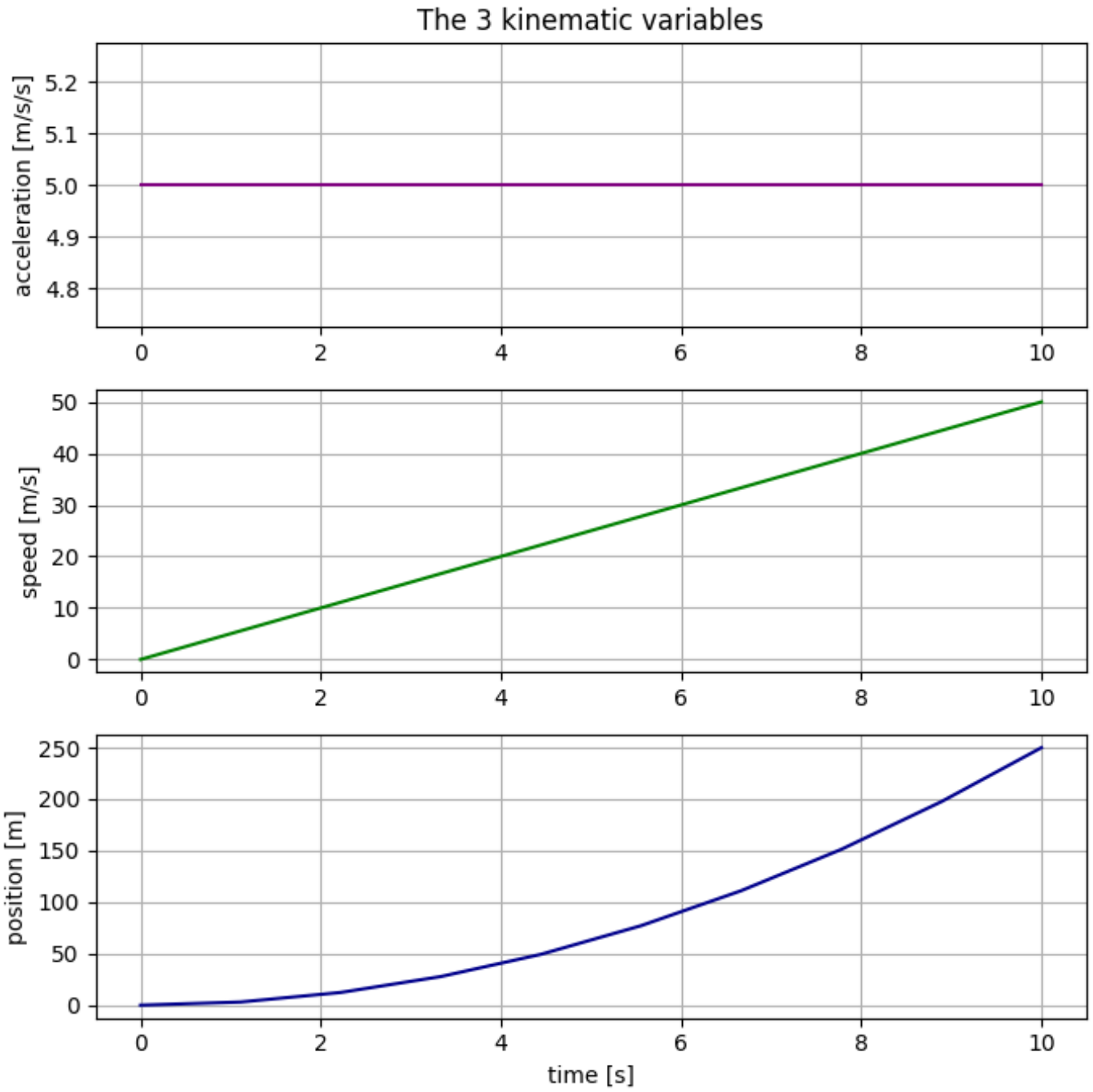
Now, let's plot both functions on the same graph. One will use the right axis, the other the left.

```
fig, ax1 = plt.subplots()

color = 'darkred'
ax1.set_xlabel('time [s]')
ax1.set_ylabel('Speed [m/s]', color=color)
ax1.plot(time, speed, color=color)
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_title('Position and Speed as a function of time')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'darkblue'
ax2.set_ylabel('Position [m]', color=color)  # we already handled the x-label with
ax2.plot(time, position, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



Or perhaps, it's better to separate the plots into different subplots. This example shows all the kinematic terms, a, v, and x separated into three subplots. Note how we made a new list called `acceleration` by filling an numpy array with the constant value a. The great thing about doing plots programaticcaly, rather than in excel, is that if we wanted to redo all this for free fall on the moon for example, where a is 1.6 m/s^2, we could just redifine that variable in the beginning, and run the whole notebook, and everything would be updated for that change.

```
# make a list of constant valued entries
acceleration = np.full((10, 1),a)

#plot all three kinematic variables
fig, axs = plt.subplots(3, 1, figsize=(7, 7))
axs[0].plot(time, acceleration, color = "purple")
axs[0].set_ylabel('acceleration [m/s/s]')
axs[0].grid(True)
axs[0].set_title('The 3 kinematic variables')

axs[1].plot(time, speed, color = "green" )
axs[1].set_ylabel('speed [m/s]')
axs[1].grid(True)

axs[2].plot(time, position, color = "darkblue")
axs[2].set_xlabel('time [s]')
axs[2].set_ylabel('position [m]')
axs[2].grid(True)

fig.tight_layout()
plt.show()
```



Great. This should be a useful starting point for plotting analytic functions. To test your understanding and further develop this skillset, try making the following modifications.

- Change the functions for speed and position to account of non-zero initial conditions (i.e. $v_0, x_0 \neq 0$)
- Change the coordinate system so that the acceleration is -9.8 m/s^2
- Make 3 plots like the very last example, but for a simple pendulum instead.

Colab paid products - Cancel contracts here