Interpolation Exercise

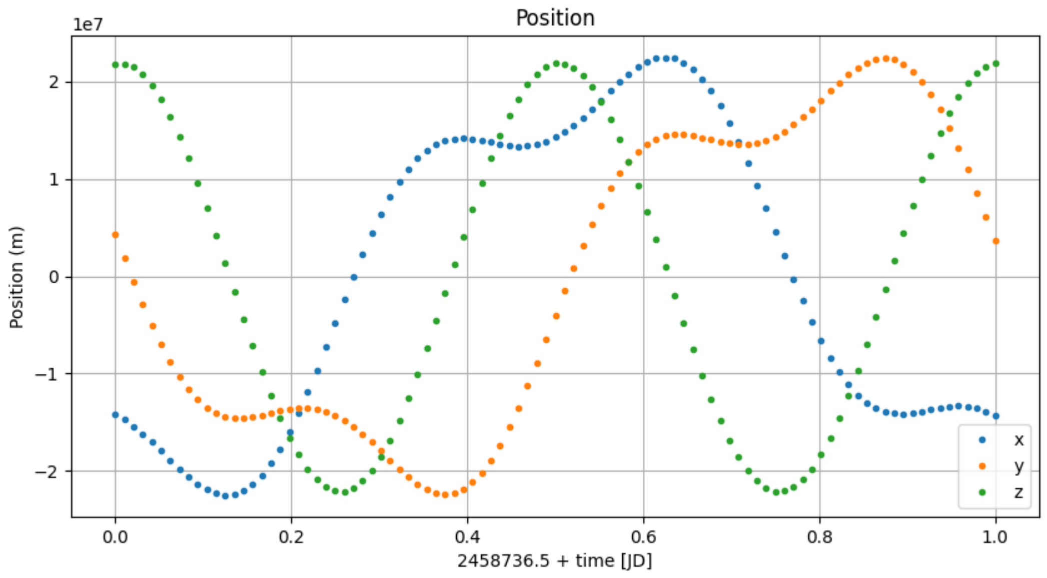
Marco Boscato: 2096921

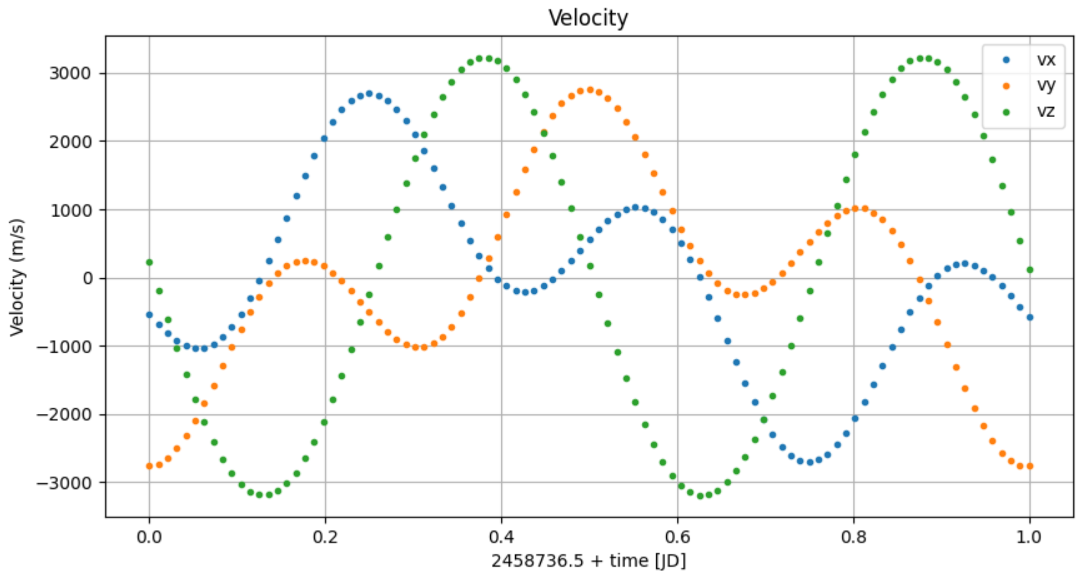
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib widget

# import the data
data = np.genfromtxt('data/G01_2070_2_900s_GMAT.txt')
print(data.shape)

# divide in time, position and velocity
time = data[:,1]
position = data[:,2:5]
velocity = data[:,5:8]

JD = 2458736.5
```

```
(97, 8)
```

The first thing we need to do is plot the position and velocity data over time (expressed in Julian Day fractions) to get an initial visualization of the data.

```
# plot position
plt.figure(figsize=(10,5))
plt.plot(time, position, '.', markersize=6)
plt.xlabel(f'{JD} + time [JD]')
plt.ylabel('Position (m)')
plt.legend(['x', 'y', 'z'])
plt.title('Position')
plt.grid()

# plot velocity
plt.figure(figsize=(10,5))
plt.plot(time, velocity, '.', markersize=6)
plt.xlabel(f'{JD} + time [JD]')
plt.ylabel('Velocity (m/s)')
plt.legend(['vx', 'vy', 'vz'])
plt.title('Velocity')
plt.grid()
```



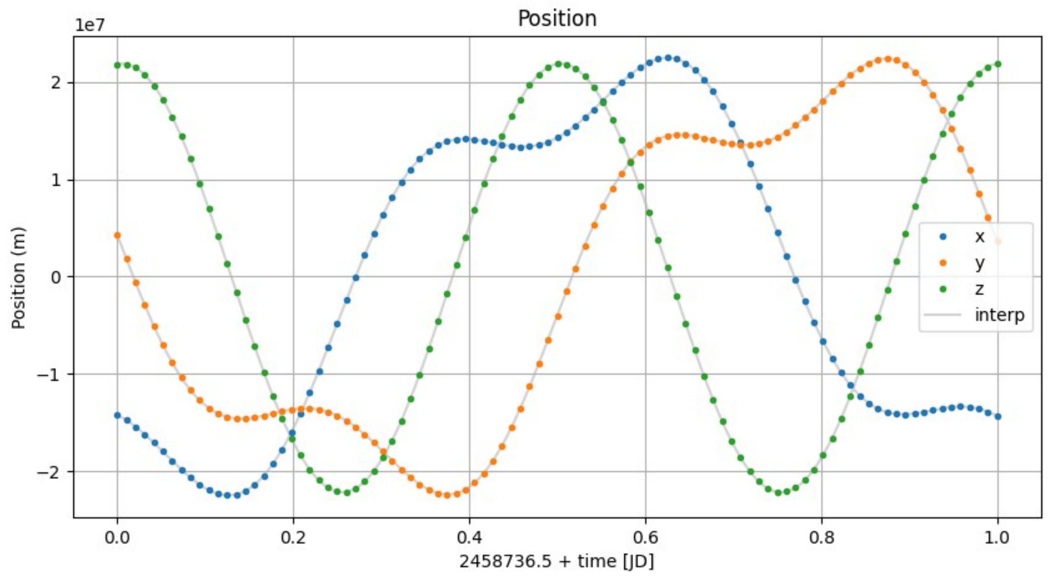


Interpolation

In this case, if we want to increase the accuracy of the data, we can use the Cubic Spline interpolation method, since we use the data points as the knots of the spline, where each segment of the spline curve is a cubic polynomial with smooth junctions. This results in less tendency to oscillate between data points. Between knots i and $i+1$ we use a cubic polynomial that spans the segment. The spline is a piecewise cubic curve, put together from the n cubics $f_{0,1}(x), f_{1,2}(x), \ldots, f_{n-1,n}(x)$, all of which have different coefficients.

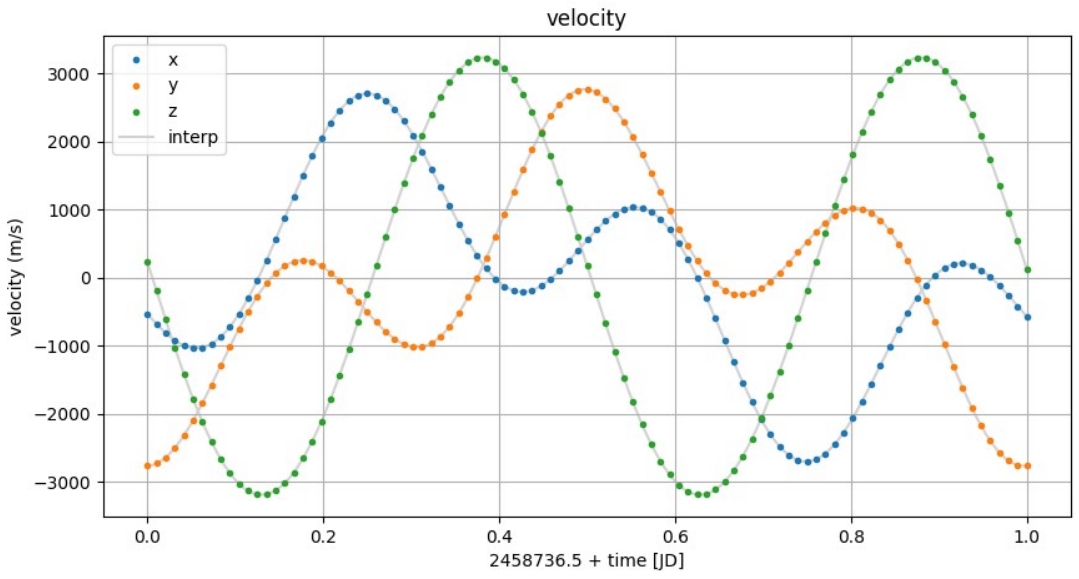
First we interpolate the position as a function of time, then we generate new position points by applying the interpolated function obtained before over an array of 1440 elements, corresponding to the time period of the orbit divided by $60\,s$ intervals.

```
# interpolation
from scipy.interpolate import CubicSpline
cs = CubicSpline(time, position)
time_new = np.linspace(time[0], time[-1], 1440) # 1440 points (1
minute intervals)
position_new = cs(time_new)

# plot the interpolated position
plt.figure(figsize=(10,5))
plt.plot(time, position, '.', markersize=6)
plt.plot(time_new, position_new, color='lightgrey',
label='interpolation', zorder=0)
plt.xlabel(f'{JD} + time [JD]')
plt.ylabel('Position (m)')
plt.legend(['x', 'y', 'z', 'interp'], loc='best')
plt.title('Position')
plt.grid()
```



We do the same process as above with the velocity data.

```
# interpolation
cs = CubicSpline(time, velocity)
velocity_new = cs(time_new)

# plot the interpolated position
plt.figure(figsize=(10,5))
plt.plot(time, velocity, '.', markersize=6)
plt.plot(time_new, velocity_new, color='lightgrey',
label='interpolation', zorder=0)
plt.xlabel(f'{JD} + time [JD]')
plt.ylabel('velocity (m/s)')
plt.legend(['x', 'y', 'z', 'interp'], loc='best')
plt.title('velocity')
plt.grid()
```



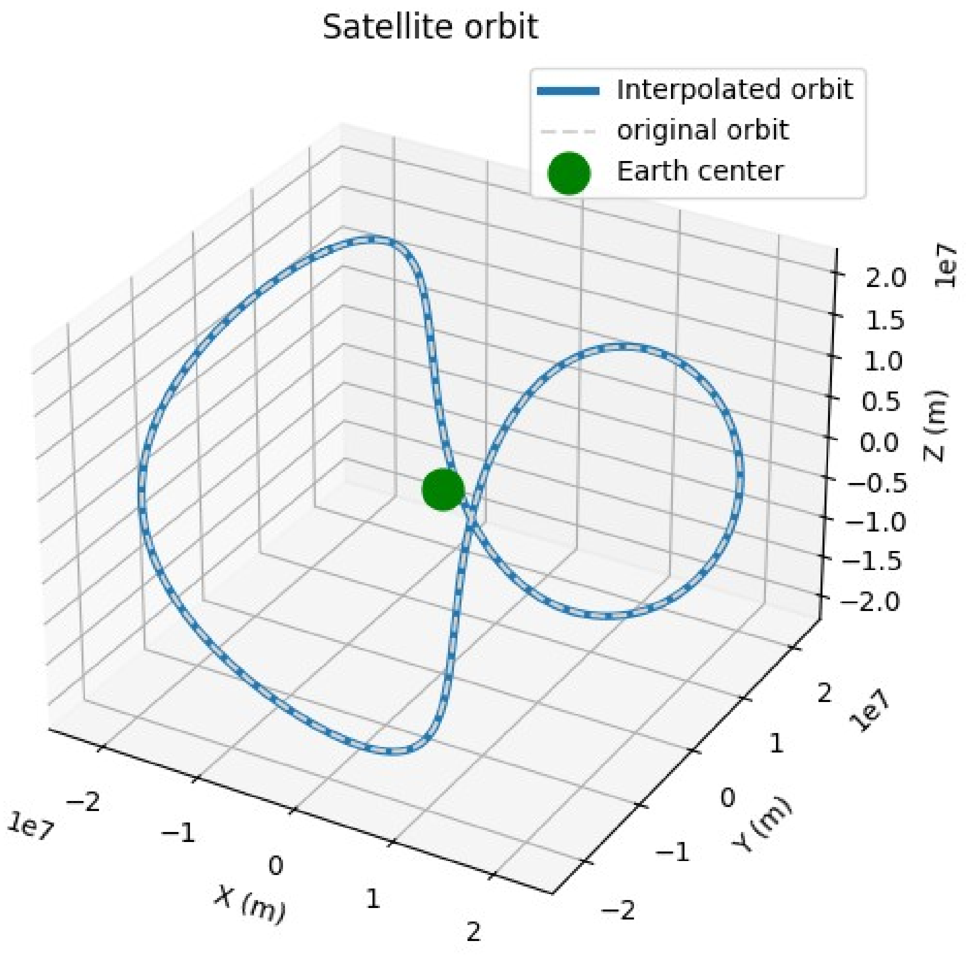
Finally, we create a new file containing all the new position and velocity points generated by the interpolation

```
# save the interpolated data

JD_array = np.full((1440,), JD)
#np.savetxt('data/G01_2070_2_900s_GMAT_interpolated.txt',
np.column_stack((JD_array, time_new, position_new, velocity_new)),
header='JD time(JD) x y z vx vy vz', fmt='%1.6f')
```

Plotting the orbit

To see the result, we visualize the original and the interpolated orbit of the satellite in a 3D plot to see if they combine or differ for some interpolation errors

```
# 3D plot
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot(position_new[:,0], position_new[:,1], position_new[:,2],
linewidth='3', label='Interpolated orbit')
ax.plot(position[:,0], position[:,1], position[:,2], label='original
orbit', linestyle='--', color='lightgrey')
ax.plot([0], [0], [0], 'go',  markersize=15, label='Earth center')
# plot the origin (earth center)
ax.set_xlabel('X (m)')
ax.set_ylabel('Y (m)')
ax.set_zlabel('Z (m)')
ax.set_title('Satellite orbit')
ax.legend()
plt.show()
```


Satellite orbit
Interpolated orbit
original orbit
Earth center
X (m)
Y (m)
Z (m)
1e7
1e7
1e7