

Untitled-1

```
1  # %%
2  import os
3  import glob
4  import requests
5  import zipfile
6
7  import numpy as np
8  import pandas as pd
9  from tqdm import tqdm
10 from matplotlib import pyplot as plt
11 import numpy as np
12 import pandas as pd
13 import matplotlib.animation
14 import datetime
15 import math
16
17
18
19
20 # %%
21 def download_and_extract_zip(url):
22     # Create a folder named .temp if it doesn't exist
23     temp_folder = "temp"
24     if not os.path.exists(temp_folder):
25         os.makedirs(temp_folder)
26
27     try:
28         # Send a GET request to the URL to download the zip file
29         response = requests.get(url)
30         response.raise_for_status()
31
32         # Extract the filename from the URL
33         filename = os.path.join(temp_folder, os.path.basename(url))
34
35         # Save the downloaded zip file to the temporary folder
36         with open(filename, "wb") as file:
37             file.write(response.content)
38
39         # Extract the contents of the zip file to the temporary folder
40         with zipfile.ZipFile(filename, "r") as zip_ref:
41             zip_ref.extractall(temp_folder)
42
43         os.remove(filename)
44
45     except Exception as e:
46         print(str(e))
47
48 # Prepare the download links and download all files
```

```
49 url_list_cams = \  
50     [f"https://ceres.ta3.sk/iaumdcdb/dataDBs/video_offline/iaumdcCAMsv3_201{n}.csv.zip"  
51     for n in range(7)]  
52  
53 url_list_sonotaco = \  
54  
55     [f"https://ceres.ta3.sk/iaumdcdb/dataDBs/video_offline/iaumdcSNMv3_S{str(n).zfill(2)}.csv.zip"  
56     for n in np.arange(8, 23, 1)]  
57  
58 url_list = url_list_cams + url_list_sonotaco  
59  
60 for k in tqdm(url_list):  
61     download_and_extract_zip(k)  
62  
63 # %%  
64 # Get all csv files  
65 meteor_data_filepaths = glob.glob("temp/*.csv")  
66  
67 # %%  
68 # Read in all files and combine them in one single dataframe  
69 def combine_csv_files(filepaths):  
70     # Initialize an empty DataFrame to store the combined data  
71     combined_df = pd.DataFrame()  
72  
73     for filepath in filepaths:  
74         try:  
75             # Read each CSV file and append it to the combined DataFrame  
76             df = pd.read_csv(filepath, delimiter=";")  
77             combined_df = pd.concat([combined_df, df], ignore_index=True)  
78         except Exception as e:  
79             print(f"Error reading file {filepath}: {str(e)}")  
80  
81     return combined_df  
82  
83 df = combine_csv_files(meteor_data_filepaths)  
84  
85 # %%  
86 # Danger zone: we want to check which columns contain NaN values. Now since Jupyter-Lab limits  
87 the  
88 # output print, we are going to ... remove this limit. But note: if you want to display now a  
89 huge  
90 # dataframe you are going to have a bad time  
91 from IPython.display import display  
92 print(f"Initial number of max columns: {pd.options.display.max_columns}")  
93 print(f"Initial number of max rows: {pd.options.display.max_rows}")  
94  
95 pd.set_option('display.max_columns', None)  
96 pd.set_option('display.max_rows', None)  
97  
98 # %%  
99 # Check if any values in each column are NaN
```

```

97 any_nans = df.isna().any()
98 all_nans = df.isna().all()
99
100 # Create a new DataFrame to store the results
101 nan_df = pd.DataFrame({'Any NaN values': any_nans, 'All NaN values': all_nans})
102
103 print(nan_df)
104
105 # Link ma gara check ganea if nan value wrong ayo vanea:
106 https://ceres.ta3.sk/iaumdcdb/public/docs/parametersdescription.txt
107
108 # %%
109 df
110
111 # %%
112 # List of columns to drop
113 columns_to_drop = ["IID", "DB", "IC", "Ano", "delta_Dayy", "LS", "delta_LS",
114                   "HB", "delta_HB", "HM", "delta_HM", "HE", "delta_HE",
115                   "delta_RA", "delta_DECL", "Vi", "delta_Vi", "delta_Vg",
116                   "delta_Vh", "delta_cZ", "delta_mv", "Qm", "Qa", "cZ",
117                   "delta_q", "delta_e", "delta_a1", "delta_a", "delta_Qa",
118                   "delta_i", "delta_arg", "delta_nod", "delta_pi", "sh",
119                   "Mas", "delta_Mas", "lgM", "delta_lgM", "cor", "crh",
120                   "mr", "delta_mr", "Hrf", "delta_Hrf", "LpA", "delta_LpA", "dur"]
121
122 # Drop only existing columns
123 df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)
124
125 # %%
126 # The longitude of perihelion (here: pi), contains NaN values ... around 50 % are not available.
127 # But
128 # instead of dropping all rows are create 2 separate dataframes for the future. We still can
129 # use
130 # some parameters
131 df_orbit_compl = df[~df["pi"].isna()].copy()
132 df_orbit_error = df[df["pi"].isna()].copy()
133
134 # %%
135 # Print the resulting number of rows
136 print(len(df_orbit_compl))
137 print(len(df_orbit_error))
138
139 # %%
140 # Cross check if we still have NaN values in our "healthy" dataframe
141 print(any(df_orbit_compl.isna().all()))
142
143 # %%
144 # Create a folder for the data
145 folder = "meteor_data"
146 if not os.path.exists(folder):

```

```
145     os.makedirs(folder)
146
147 # Store the dataframes
148 df_orbit_compl.to_csv("meteor_data/meteor_compl.csv")
149 df_orbit_error.to_csv("meteor_data/meteor_error.csv")
150
151 # %%
152 # Reminder how the data looks like
153 df_orbit_compl
154
155 # %%
156 #We merge both dataframes and extract the columns of interest:
157 #     - RA: Right ascension of the radiant in degrees
158 #     - DECL: Declination of the radiant in degrees
159 #     - Vg: Geo-Centric velocity in km/s
160 #     - Yr, Mn, Dayy: Year, month and day+fraction of a day
161 df_radiants = pd.concat([df_orbit_compl[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]],
162                          df_orbit_error[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]]])
163
164 # %%
165 # Checking the ranges of the RA and DECL values (we need to convert it for the matplotlib
166 # plotting
167 # function
168 print(f"Minimum RA value: {df_radiants.RA.min()}")
169 print(f"Maximum RA value: {df_radiants.RA.max()}")
170 print(f"Minimum DECL value: {df_radiants.DECL.min()}")
171 print(f"Maximum DECL value: {df_radiants.DECL.max()}")
172
173 # %%
174 # Convert to radians
175 df_radiants.loc[:, "RA_rad"] = np.radians(df_radiants["RA"])
176 df_radiants.loc[:, "DECL_rad"] = np.radians(df_radiants["DECL"])
177
178 # %%
179 # Add a column for the plot
180 df_radiants.loc[:, 'RA_rad4plot'] = \
181     df_radiants['RA_rad'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)
182
183 # %%
184 # Add some styles
185 plt.style.use('dark_background')
186 plt.figure(figsize=(12, 8))
187 plt.subplot(projection="aitoff")
188
189 # Plot the radiants
190 plt.scatter(df_radiants['RA_rad4plot'], \
191            df_radiants['DECL_rad'], color='white', linestyle='None', \
192            alpha=.01, s=1)
193
194 # Convert the longitude values finally in right ascension hours
```

```
194 plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
195             labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
196                    '22 h', '20 h', '18 h', '16 h', '14 h'])
197
198 # Plot the labels
199 plt.xlabel('Right ascension in hours')
200 plt.ylabel('Declination in deg.')
201
202 # Add a grid
203 plt.grid(True)
204
205 # %%
206 # Read in the meteor data
207 df_orbit_compl = pd.read_csv("meteor_data/meteor_compl.csv",
208                             index_col=0)
209 df_orbit_error = pd.read_csv("meteor_data/meteor_error.csv",
210                              index_col=0)
211
212 # %%
213 # Add some styles
214 plt.style.use('dark_background')
215 fig = plt.figure(figsize=(12, 8))
216 plt.subplot(projection="aitoff")
217
218 # Add a color for the velocity values
219 cm = plt.colormaps.get_cmap('jet')
220
221 # Plot the radiant
222 cr = plt.scatter(df_radiants['RA_rad4plot'], \
223                 df_radiants['DECL_rad'], linestyle='None', \
224                 alpha=.01, s=1, c=df_radiants["Vg"].values, cmap=cm)
225
226 # Create a colormap
227 sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
228                                                         max(df_radiants["Vg"].values)))
229
230
231 # Convert the longitude values finally in right ascension hours
232 plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
233             labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
234                    '22 h', '20 h', '18 h', '16 h', '14 h'])
235
236 # Plot the labels
237 plt.xlabel('Right ascension in hours')
238 plt.ylabel('Declination in deg.')
239
240 # Add a grid
241 plt.grid(True)
242
243 # Add the colorbar
```

```
244 ax = plt.gca()
245 color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
246 color_bar.set_alpha(1)
247 color_bar.set_label('Geocentric entry velocity in km/s')
248
249 # %%
250 # The Perseids's are active between July and September with a peak around the 12th of August.
    Let's
251 # filter the data +/- 7 days around the 12th to see, where the Perseids come from.
252 # Spoiler: the appear to come from ... Perseus ...
253
254 # First we add a datetime object column:
255 df_radiants.loc[:, "datetime"] = \
256     df_radiants.apply(lambda x: datetime.datetime(year=int(x["Yr"]),
257                                                     month=int(x["Mn"]),
258                                                     day=math.floor(x["Dayy"])),
259                       axis=1)
260
261 # %%
262 # And let's take a look:
263 df_radiants
264
265
266 # %%
267 # Now let's filter for the Perseids and let's check the plot again:
268 df_radiants_perseids_peak = \
269 df_radiants[(df_radiants["datetime"].dt.month == 8) &
270             ((df_radiants["datetime"].dt.day >= 5) & (df_radiants["datetime"].dt.day <=
19))].copy()
271
272 # %%
273 df_radiants_perseids_peak
274
275 # %%
276 # Add some styles
277 plt.style.use('dark_background')
278 fig = plt.figure(figsize=(12, 8))
279 plt.subplot(projection="aitoff")
280
281 # Add a color for the velocity values
282 cm = plt.colormaps.get_cmap('jet')
283
284 # Plot the radiants
285 cr = plt.scatter(df_radiants_perseids_peak['RA_rad4plot'], \
286                 df_radiants_perseids_peak['DECL_rad'], linestyle='None', \
287                 alpha=.01, s=1, c=df_radiants_perseids_peak["Vg"].values, cmap=cm)
288
289 # Create a colormap
290 sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants_perseids_
_peak["Vg"].values),
```

```

291                                     max(df_radiants_perseids↵
    _peak["Vg"].values)))
292
293
294 # Convert the longitude values finally in right ascension hours
295 plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
296           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
297                 '22 h', '20 h', '18 h', '16 h', '14 h'])
298
299 # Plot the labels
300 plt.xlabel('Right ascension in hours')
301 plt.ylabel('Declination in deg.')
302
303 # Add a grid
304 plt.grid(True)
305
306 # Add the colorbar
307 ax = plt.gca()
308 color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
309 color_bar.set_alpha(1)
310 color_bar.set_label('Geocentric entry velocity in km/s')
311
312 plt.title("Radiants around the Perseids' peak")
313
314 # %%
315 # Read in the meteor data
316 df_orbit_compl = pd.read_csv("meteor_data/meteor_compl.csv",
317                             index_col=0)
318 df_orbit_error = pd.read_csv("meteor_data/meteor_error.csv",
319                              index_col=0)
320
321 df_radiants = pd.concat([df_orbit_compl[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]],
322                          df_orbit_error[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]]])
323
324 # Now filter
325 df_radiants = df_radiants.loc[(df_radiants["Vg"]>=11) & (df_radiants["Vg"]<=73)].copy()
326
327 # Convert to radians
328 df_radiants.loc[:, "RA_rad"] = np.radians(df_radiants["RA"])
329 df_radiants.loc[:, "DECL_rad"] = np.radians(df_radiants["DECL"])
330
331 # Add a column for the plot
332 df_radiants.loc[:, 'RA_rad4plot'] = \
333     df_radiants['RA_rad'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)
334
335 # First we add a datetime object column:
336 df_radiants.loc[:, "datetime"] = \
337     df_radiants.apply(lambda x: datetime.datetime(year=int(x["Yr"]),
338                                                    month=int(x["Mn"]),
339                                                    day=math.floor(x["Dayy"])),

```

```
340 axis=1)
341
342 # Add now the day's fraction
343 df_radiants.loc[:, "datetime"] = \
344     df_radiants.apply(lambda x: x["datetime"] + datetime.timedelta(days=x["Dayy"]%1), axis=1)
345
346 # %%
347 # Add a Day of Year (DOY) column, based on the datetime column
348 df_radiants.loc[:, "doy"] = df_radiants.datetime.dt.day_of_year.copy()
349
350 # %%
351 # Add some styles
352 plt.style.use('dark_background')
353 fig = plt.figure(figsize=(12, 8))
354
355 # Add aitoff projection
356 plt.subplot(projection="aitoff")
357
358 # Get axes
359 ax = plt.gca()
360
361 # Add a color for the velocity values
362 cm = plt.colormaps.get_cmap('jet')
363
364 filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==0].copy()
365
366 # Plot the radiants
367 cr = plt.scatter(filtered_df_radiants['RA_rad4plot'], \
368                 filtered_df_radiants['DECL_rad'], linestyle='None', \
369                 s=1, c=filtered_df_radiants["Vg"], cmap=cm)
370
371 # Create a colormap
372 sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
373                                                         max(df_radiants["Vg"].values)))
374
375 # A function that creates a sky map per DOY
376 def update(frame):
377
378     # Filter by the Day of Year
379     filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==frame+1].copy()
380
381     # Add the radiants in the plot of the filtered dataframe
382     cr.set_offsets(filtered_df_radiants[['RA_rad4plot', "DECL_rad"]])
383     cr.set_array(filtered_df_radiants["Vg"])
384
385     # Add a title that indicates the DOY
386     ax.set_title(f"DOY: {frame+1}")
387
388     return cr
389
```



```
390 # Convert the longitude values finally in right ascension hours
391 plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
392           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
393                 '22 h', '20 h', '18 h', '16 h', '14 h'])
394
395 # Plot the labels
396 plt.xlabel('Right ascension in hours')
397 plt.ylabel('Declination in deg.')
398
399 # Add a grid
400 plt.grid(True)
401 # Add the colorbar
402 ax = plt.gca()
403 color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
404 color_bar.set_alpha(1)
405 color_bar.set_label('Geocentric entry velocity in km/s')
406
407 ani = matplotlib.animation.FuncAnimation(fig=fig, func=update, frames=365, interval=100)
408
409 # Save the animation as a GIF file
410 ani.save('scatter_animation.gif')
411
412 # %%
413 import os
414 import requests
415
416 # URLs of the SPICE kernel files
417 naif0012_url = "https://naif.jpl.nasa.gov/pub/naif/generic_kernels/lsk/naif0012.tls"
418 de432s_url = "https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/de432s.bsp"
419
420 # Paths to save the downloaded files
421 kernel_dir = "../kernels"
422 lsk_dir = os.path.join(kernel_dir, "lsk")
423 spk_dir = os.path.join(kernel_dir, "spk")
424
425 # Create directories if they don't exist
426 os.makedirs(lsk_dir, exist_ok=True)
427 os.makedirs(spk_dir, exist_ok=True)
428
429 # Download the naif0012.tls file
430 naif0012_path = os.path.join(lsk_dir, "naif0012.tls")
431 with requests.get(naif0012_url, stream=True) as r:
432     r.raise_for_status()
433     with open(naif0012_path, "wb") as f:
434         for chunk in r.iter_content(chunk_size=8192):
435             f.write(chunk)
436 print(f"Downloaded naif0012.tls to {naif0012_path}")
437
438 # Download the de432s.bsp file
439 de432s_path = os.path.join(spk_dir, "de432s.bsp")
```

```

440 with requests.get(de432s_url, stream=True) as r:
441     r.raise_for_status()
442     with open(de432s_path, "wb") as f:
443         for chunk in r.iter_content(chunk_size=8192):
444             f.write(chunk)
445 print(f"Downloaded de432s.bsp to {de432s_path}")
446
447 # Load SPICE kernels
448 import spiceypy
449 spiceypy.furnsh(naif0012_path)
450 spiceypy.furnsh(de432s_path)
451
452
453 # %%
454 # Load SPICE kernels
455 import spiceypy
456 spiceypy.furnsh("../kernels/lsk/naif0012.tls")
457 spiceypy.furnsh("../kernels/spk/de432s.bsp")
458
459 # %%
460 # Compute empheremis time
461 df_radiants.loc[:, "ET"] = \
462     df_radiants.apply(lambda x: spiceypy.utc2et(x["datetime"].strftime("%Y-%m-%dT%H:%M:%S")),
463                       axis=1)
464
465 # %%
466 # Compute the following vectors per ET:
467 # - Direction to Sun as seen from Earth
468 # - Direction to Apex as seen from Earth
469 # - Direction to Antihelion as seen from Earth
470 # - Direction to Anti-Apex as seen from Earth
471 df_radiants.loc[:, "Helion_vec"] = \
472     df_radiants.apply(lambda x: np.array(spiceypy.spkgps(targ=10,
473                                                         et=x["ET"],
474                                                         ref="J2000",
475                                                         obs=399)[0]),
476                       axis=1)
477
478 # For the Apex we compute the velocity vector of Earth w.r.t. the Sun
479 df_radiants.loc[:, "Apex_vec"] = \
480     df_radiants.apply(lambda x: np.array(spiceypy.spkgeo(targ=399,
481                                                           et=x["ET"],
482                                                           ref="J2000",
483                                                           obs=10)[0][3:]),
484                       axis=1)
485
486 # Antihelion: simply invert the Sun's vector
487 df_radiants.loc[:, "Antihelion_vec"] = \
488     df_radiants.apply(lambda x: -1.0*x["Helion_vec"],
489                       axis=1)

```

```

490
491 # Antiapex: simply invert the Apex' vector
492 df_radiants.loc[:, "Antiapex_vec"] = \
493     df_radiants.apply(lambda x: -1.0*x["Apex_vec"],
494                       axis=1)
495
496 # %%
497 # Let's check the angles:
498 def angle_between_vectors(v1, v2):
499
500     # Calculate the dot product of the vectors
501     dot_product = np.dot(v1, v2)
502
503     # Calculate the magnitude of the vectors
504     v1_magnitude = np.linalg.norm(v1)
505     v2_magnitude = np.linalg.norm(v2)
506
507     # Calculate the cosine of the angle between the vectors
508     cosine_of_angle = dot_product / (v1_magnitude * v2_magnitude)
509
510     # Calculate the angle between the vectors
511     angle = np.arccos(cosine_of_angle)
512
513     # ... in degrees
514     angle = np.degrees(angle)
515
516     return angle
517
518 # We take the first entry only
519 antihelion_helion_angle = angle_between_vectors(df_radiants.iloc[0]['Helion_vec'],
520                                                df_radiants.iloc[0]['Antihelion_vec'])
521
522 antiapex_apex_angle = angle_between_vectors(df_radiants.iloc[0]['Apex_vec'],
523                                             df_radiants.iloc[0]['Antiapex_vec'])
524
525 helion_apex_angle = angle_between_vectors(df_radiants.iloc[0]['Helion_vec'],
526                                           df_radiants.iloc[0]['Apex_vec'])
527
528 print(f"Angle between Helion and Antihelion direction in degrees: {antihelion_helion_angle}")
529 print(f"Angle between Apex and Antiapex direction in degrees: {antiapex_apex_angle}")
530 print(f"Angle between Apex and Helio direction in degrees: {helion_apex_angle}")
531
532 # %%
533 # Converting the vectors to sky coordiantes
534 df_radiants.loc[:, "Helion_RA"] = df_radiants["Helion_vec"].apply(lambda x: spiceypy.recrad(x)
535                                                                    [1])
536 df_radiants.loc[:, "Helion_Dec"] = df_radiants["Helion_vec"].apply(lambda x: spiceypy.recrad(x)
537                                                                    [2])
538 df_radiants.loc[:, 'Helion_RA_rad4plot'] = \
539     df_radiants['Helion_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)

```

```

538
539 df_radiants.loc[:, "Antihelion_RA"] = df_radiants["Antihelion_vec"].apply(lambda x:
spiceypy.recrad(x)[1])
540 df_radiants.loc[:, "Antihelion_Dec"] = df_radiants["Antihelion_vec"].apply(lambda x:
spiceypy.recrad(x)[2])
541 df_radiants.loc[:, 'Antihelion_RA_rad4plot'] = \
542     df_radiants['Antihelion_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else
-1*x)
543
544 df_radiants.loc[:, "Apex_RA"] = df_radiants["Apex_vec"].apply(lambda x: spiceypy.recrad(x)[1])
545 df_radiants.loc[:, "Apex_Dec"] = df_radiants["Apex_vec"].apply(lambda x: spiceypy.recrad(x)[2])
546 df_radiants.loc[:, 'Apex_RA_rad4plot'] = \
547     df_radiants['Apex_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)
548
549 df_radiants.loc[:, "Antiapex_RA"] = df_radiants["Antiapex_vec"].apply(lambda x:
spiceypy.recrad(x)[1])
550 df_radiants.loc[:, "Antiapex_Dec"] = df_radiants["Antiapex_vec"].apply(lambda x:
spiceypy.recrad(x)[2])
551 df_radiants.loc[:, 'Antiapex_RA_rad4plot'] = \
552     df_radiants['Antiapex_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)
553
554 # %%
555 # Add some styles
556 plt.style.use('dark_background')
557 fig = plt.figure(figsize=(12, 8))
558
559 # Add aitoff projection
560 plt.subplot(projection="aitoff")
561
562 # Get axes
563 ax = plt.gca()
564
565 # Add a color for the velocity values
566 cm = plt.colormaps.get_cmap('jet')
567
568 filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==180].copy()
569
570 # Plot the radiants
571 cr = plt.scatter(filtered_df_radiants['RA_rad4plot'], \
572                 filtered_df_radiants['DECL_rad'], linestyle='None', \
573                 s=1, c=filtered_df_radiants["Vg"], cmap=cm)
574
575 # Create a colormap
576 sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
577                                                         max(df_radiants["Vg"].values)))
578
579 # Add the Sun, Antihelion, Apex and Antiapex direction
580 plt.plot(filtered_df_radiants.iloc[0]["Helion_RA_rad4plot"],
581          filtered_df_radiants.iloc[0]["Helion_Dec"],
582          marker="*", lw=0, markersize=10, color="yellow", label="Sun")
583

```

```
584 plt.plot(filtered_df_radiants.iloc[0]["Antihelion_RA_rad4plot"],
585           filtered_df_radiants.iloc[0]["Antihelion_Dec"],
586           marker="*", lw=0, markersize=10, color="orange", label="Antihelion")
587
588 plt.plot(filtered_df_radiants.iloc[0]["Apex_RA_rad4plot"],
589           filtered_df_radiants.iloc[0]["Apex_Dec"],
590           marker="^", lw=0, markersize=10, color="lightblue", label="Apex")
591
592 plt.plot(filtered_df_radiants.iloc[0]["Antiapex_RA_rad4plot"],
593           filtered_df_radiants.iloc[0]["Antiapex_Dec"],
594           marker="^", lw=0, markersize=10, color="red", label="Antiapex")
595
596 plt.legend()
597
598 # Convert the longitude values finally in right ascension hours
599 plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
600           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
601                  '22 h', '20 h', '18 h', '16 h', '14 h'])
602
603 # Plot the labels
604 plt.xlabel('Right ascension in hours')
605 plt.ylabel('Declination in deg.')
606
607 # Add a grid
608 plt.grid(True)
609
610 # Add the colorbar
611 ax = plt.gca()
612 color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
613 color_bar.set_alpha(1)
614 color_bar.set_label('Geocentric entry velocity in km/s')
615
616 fig.savefig("dir_of_interests.png")
617
618
619 # %%
620
621
622 # %%
623
624
625
626
```