Untitled-1

```
# %%
import os
import glob
import requests
import zipfile

import numpy as np
import pandas as pd
from tqdm import tqdm
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.animation
import datetime
import math




# %%
def download_and_extract_zip(url):
    # Create a folder named .temp if it doesn't exist
    temp_folder = "temp"
    if not os.path.exists(temp_folder):
        os.makedirs(temp_folder)

    try:
        # Send a GET request to the URL to download the zip file
        response = requests.get(url)
        response.raise_for_status()

        # Extract the filename from the URL
        filename = os.path.join(temp_folder, os.path.basename(url))

        # Save the downloaded zip file to the temporary folder
        with open(filename, "wb") as file:
            file.write(response.content)

        # Extract the contents of the zip file to the temporary folder
        with zipfile.ZipFile(filename, "r") as zip_ref:
            zip_ref.extractall(temp_folder)

        os.remove(filename)

    except Exception as e:
        print(str(e))

# Prepare the download links and download all files
```

```python
url_list_cams = \
    [f"https://ceres.ta3.sk/iaumdcdb/dataDBs/video_offline/iaumdcCAMSv3_201{n}.csv.zip"
     for n in range(7)]

url_list_sonotaco = \
[f"https://ceres.ta3.sk/iaumdcdb/dataDBs/video_offline/iaumdcSNMv3_S{str(n).zfill(2)}.csv.zip"
     for n in np.arange(8, 23, 1)]

url_list = url_list_cams + url_list_sonotaco

for k in tqdm(url_list):
    download_and_extract_zip(k)

# %%
# Get all csv files
meteor_data_filepaths = glob.glob("temp/*.csv")

# %%
# Read in all files and combine them in one single dataframe
def combine_csv_files(filepaths):
    # Initialize an empty DataFrame to store the combined data
    combined_df = pd.DataFrame()

    for filepath in filepaths:
        try:
            # Read each CSV file and append it to the combined DataFrame
            df = pd.read_csv(filepath, delimiter=";")
            combined_df = pd.concat([combined_df, df], ignore_index=True)
        except Exception as e:
            print(f"Error reading file {filepath}: {str(e)}")

    return combined_df

df = combine_csv_files(meteor_data_filepaths)

# %%
# Danger zone: we want to check which columns contain NaN values. Now since Jupyter-Lab limits the
# output print, we are going to ... remove this limit. But note: if you want to display now a huge
# dataframe you are going to have a bad time
from IPython.display import display
print(f"Initial number of max columns: {pd.options.display.max_columns}")
print(f"Initial number of max rows: {pd.options.display.max_rows}")

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# %%
# Check if any values in each column are NaN
```

```
any_nans = df.isna().any()
all_nans = df.isna().all()

# Create a new DataFrame to store the results
nan_df = pd.DataFrame({'Any NaN values': any_nans, 'All NaN values': all_nans})

print(nan_df)

# Link ma gara check ganea if nan value wrong ayo vanea: https://ceres.ta3.sk/iaumdcdb//public/docs/parametersdescription.txt

# %%
df

# %%
# List of columns to drop
columns_to_drop = ["IID", "DB", "IC", "Ano", "delta_Dayy", "LS", "delta_LS",
                   "HB", "delta_HB", "HM", "delta_HM", "HE", "delta_HE",
                   "delta_RA", "delta_DECL", "Vi", "delta_Vi", "delta_Vg",
                   "delta_Vh", "delta_cZ", "delta_mv", "Qm", "Qa", "cZ",
                   "delta_q", "delta_e", "delta_a1", "delta_a", "delta_Qa",
                   "delta_i", "delta_arg", "delta_nod", "delta_pi", "sh",
                   "Mas", "delta_Mas", "lgM", "delta_lgM", "cor", "crh",
                   "mr", "delta_mr", "Hrf", "delta_Hrf", "LpA", "delta_LpA", "dur"]

# Drop only existing columns
df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)


# %%
# The longitude of perihelion (here: pi), contains NaN values ... around 50 % are not avaiable. But
# instead of dropping all rows are create 2 separate dataframes for the future. We still can use
# some parameters
df_orbit_compl = df[~df["pi"].isna()].copy()
df_orbit_error = df[df["pi"].isna()].copy()

# %%
# Print the resulting number of rows
print(len(df_orbit_compl))
print(len(df_orbit_error))

# %%
# Cross check if we still have NaN values in our "healthy" dataframe
print(any(df_orbit_compl.isna().all()))

# %%
# Create a folder for the data
folder = "meteor_data"
if not os.path.exists(folder):
```

```
    os.makedirs(folder)

# Store the dataframes
df_orbit_compl.to_csv("meteor_data/meteor_compl.csv")
df_orbit_error.to_csv("meteor_data/meteor_error.csv")

# %%
# Reminder how the data looks like
df_orbit_compl

# %%
 #We merge both dataframes and extract the columns of interest:
#     - RA: Right ascension of the radiant in degrees
#     - DECL: Declination of the radiant in degrees
#     - Vg: Geo-Centric velocity in km/s
#     - Yr, Mn, Dayy: Year, month and day+fraction of a day
df_radiants = pd.concat([df_orbit_compl[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]],
                         df_orbit_error[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]]])

# %%
# Checking the ranges of the RA and DECL values (we need to convert it for the matplotlib plotting
# function
print(f"Minimum RA value: {df_radiants.RA.min()}")
print(f"Maximum RA value: {df_radiants.RA.max()}")
print(f"Minimum DECL value: {df_radiants.DECL.min()}")
print(f"Maximum DECL value: {df_radiants.DECL.max()}")

# %%
# Convert to radians
df_radiants.loc[:, "RA_rad"] = np.radians(df_radiants["RA"])
df_radiants.loc[:, "DECL_rad"] = np.radians(df_radiants["DECL"])

# %%
# Add a column for the plot
df_radiants.loc[:, 'RA_rad4plot'] = \
    df_radiants['RA_rad'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)

# %%
# Add some styles
plt.style.use('dark_background')
plt.figure(figsize=(12, 8))
plt.subplot(projection="aitoff")

# Plot the radiants
plt.scatter(df_radiants['RA_rad4plot'], \
            df_radiants['DECL_rad'], color='white', linestyle='None', \
            alpha=.01, s=1)

# Convert the longitude values finally in right ascension hours
```

```
plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
                   '22 h', '20 h', '18 h', '16 h', '14 h'])

# Plot the labels
plt.xlabel('Right ascension in hours')
plt.ylabel('Declination in deg.')

# Add a grid
plt.grid(True)

# %%
# Read in the meteor data
df_orbit_compl = pd.read_csv("meteor_data/meteor_compl.csv",
                             index_col=0)
df_orbit_error = pd.read_csv("meteor_data/meteor_error.csv",
                             index_col=0)

# %%
# Add some styles
plt.style.use('dark_background')
fig = plt.figure(figsize=(12, 8))
plt.subplot(projection="aitoff")

# Add a color for the velocity values
cm = plt.colormaps.get_cmap('jet')

# Plot the radiants
cr = plt.scatter(df_radiants['RA_rad4plot'], \
            df_radiants['DECL_rad'], linestyle='None', \
            alpha=.01, s=1, c=df_radiants["Vg"].values, cmap=cm)

# Create a colormap
sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
                                                       max(df_radiants["Vg"].values)))


# Convert the longitude values finally in right ascension hours
plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
                   '22 h', '20 h', '18 h', '16 h', '14 h'])

# Plot the labels
plt.xlabel('Right ascension in hours')
plt.ylabel('Declination in deg.')

# Add a grid
plt.grid(True)

# Add the colorbar
```

```
ax = plt.gca()
color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
color_bar.set_alpha(1)
color_bar.set_label('Geocentric entry velocity in km/s')

# %%
# The Perseids's are active between July and September with a peak around the 12th of August. Let's
# filter the data +/- 7 days around the 12th to see, where the Perseids come from.
# Spoiler: the appear to come from ... Perseus ...

# First we add a datetime object column:
df_radiants.loc[:, "datetime"] = \
    df_radiants.apply(lambda x: datetime.datetime(year=int(x["Yr"]),
                                                  month=int(x["Mn"]),
                                                  day=math.floor(x["Dayy"])),
                                                  axis=1)

# %%
# And let's take a look:
df_radiants


# %%
# Now let's filter for the Perseids and let's check the plot again:
df_radiants_perseids_peak = \
df_radiants[(df_radiants["datetime"].dt.month == 8) &
            ((df_radiants["datetime"].dt.day >= 5) & (df_radiants["datetime"].dt.day <= 19))].copy()

# %%
df_radiants_perseids_peak

# %%
# Add some styles
plt.style.use('dark_background')
fig = plt.figure(figsize=(12, 8))
plt.subplot(projection="aitoff")

# Add a color for the velocity values
cm = plt.colormaps.get_cmap('jet')

# Plot the radiants
cr = plt.scatter(df_radiants_perseids_peak['RA_rad4plot'], \
            df_radiants_perseids_peak['DECL_rad'], linestyle='None', \
            alpha=.01, s=1, c=df_radiants_perseids_peak["Vg"].values, cmap=cm)

# Create a colormap
sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants_perseids_peak["Vg"].values),
```

```
                                                          max(df_radiants_perseids_peak["Vg"].values)))


# Convert the longitude values finally in right ascension hours
plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
                   '22 h', '20 h', '18 h', '16 h', '14 h'])

# Plot the labels
plt.xlabel('Right ascension in hours')
plt.ylabel('Declination in deg.')

# Add a grid
plt.grid(True)

# Add the colorbar
ax = plt.gca()
color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
color_bar.set_alpha(1)
color_bar.set_label('Geocentric entry velocity in km/s')

plt.title("Radiants around the Perseids' peak")

# %%
# Read in the meteor data
df_orbit_compl = pd.read_csv("meteor_data/meteor_compl.csv",
                             index_col=0)
df_orbit_error = pd.read_csv("meteor_data/meteor_error.csv",
                             index_col=0)

df_radiants = pd.concat([df_orbit_compl[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]],
                         df_orbit_error[["RA", "DECL", "Vg", "Yr", "Mn", "Dayy"]]])

# Now filter
df_radiants = df_radiants.loc[(df_radiants["Vg"]>=11) & (df_radiants["Vg"]<=73)].copy()

# Convert to radians
df_radiants.loc[:, "RA_rad"] = np.radians(df_radiants["RA"])
df_radiants.loc[:, "DECL_rad"] = np.radians(df_radiants["DECL"])

# Add a column for the plot
df_radiants.loc[:, 'RA_rad4plot'] = \
    df_radiants['RA_rad'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)

# First we add a datetime object column:
df_radiants.loc[:, "datetime"] = \
    df_radiants.apply(lambda x: datetime.datetime(year=int(x["Yr"]),
                                                  month=int(x["Mn"]),
                                                  day=math.floor(x["Dayy"])),
```

```
                                                   axis=1)

# Add now the day's fraction
df_radiants.loc[:, "datetime"] = \
    df_radiants.apply(lambda x: x["datetime"] + datetime.timedelta(days=x["Dayy"]%1), axis=1)

# %%
# Add a Day of Year (DOY) column, based on the datetime column
df_radiants.loc[:, "doy"] = df_radiants.datetime.dt.day_of_year.copy()

# %%
# Add some styles
plt.style.use('dark_background')
fig = plt.figure(figsize=(12, 8))

# Add aitoff projection
plt.subplot(projection="aitoff")

# Get axes
ax = plt.gca()

# Add a color for the velocity values
cm = plt.colormaps.get_cmap('jet')

filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==0].copy()

# Plot the radiants
cr = plt.scatter(filtered_df_radiants['RA_rad4plot'], \
                 filtered_df_radiants['DECL_rad'], linestyle='None', \
                 s=1, c=filtered_df_radiants["Vg"], cmap=cm)

# Create a colormap
sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
                                                        max(df_radiants["Vg"].values)))

# A function that creates a sky map per DOY
def update(frame):

    # Filter by the Day of Year
    filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==frame+1].copy()

    # Add the radiants in the plot of the filtered dataframe
    cr.set_offsets(filtered_df_radiants[['RA_rad4plot', "DECL_rad"]])
    cr.set_array(filtered_df_radiants["Vg"])

    # Add a title that indicates the DOY
    ax.set_title(f"DOY: {frame+1}")

    return cr

```

```python
# Convert the longitude values finally in right ascension hours
plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
                   '22 h', '20 h', '18 h', '16 h', '14 h'])

# Plot the labels
plt.xlabel('Right ascension in hours')
plt.ylabel('Declination in deg.')

# Add a grid
plt.grid(True)
# Add the colorbar
ax = plt.gca()
color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
color_bar.set_alpha(1)
color_bar.set_label('Geocentric entry velocity in km/s')

ani = matplotlib.animation.FuncAnimation(fig=fig, func=update, frames=365, interval=100)

# Save the animation as a GIF file
ani.save('scatter_animation.gif')

# %%
import os
import requests

# URLs of the SPICE kernel files
naif0012_url = "https://naif.jpl.nasa.gov/pub/naif/generic_kernels/lsk/naif0012.tls"
de432s_url = "https://naif.jpl.nasa.gov/pub/naif/generic_kernels/spk/planets/de432s.bsp"

# Paths to save the downloaded files
kernel_dir = "../kernels"
lsk_dir = os.path.join(kernel_dir, "lsk")
spk_dir = os.path.join(kernel_dir, "spk")

# Create directories if they don't exist
os.makedirs(lsk_dir, exist_ok=True)
os.makedirs(spk_dir, exist_ok=True)

# Download the naif0012.tls file
naif0012_path = os.path.join(lsk_dir, "naif0012.tls")
with requests.get(naif0012_url, stream=True) as r:
    r.raise_for_status()
    with open(naif0012_path, "wb") as f:
        for chunk in r.iter_content(chunk_size=8192):
            f.write(chunk)
print(f"Downloaded naif0012.tls to {naif0012_path}")

# Download the de432s.bsp file
de432s_path = os.path.join(spk_dir, "de432s.bsp")
```

```
with requests.get(de432s_url, stream=True) as r:
    r.raise_for_status()
    with open(de432s_path, "wb") as f:
        for chunk in r.iter_content(chunk_size=8192):
            f.write(chunk)
print(f"Downloaded de432s.bsp to {de432s_path}")

# Load SPICE kernels
import spiceypy
spiceypy.furnsh(naif0012_path)
spiceypy.furnsh(de432s_path)


# %%
# Load SPICE kernels
import spiceypy
spiceypy.furnsh("../kernels/lsk/naif0012.tls")
spiceypy.furnsh("../kernels/spk/de432s.bsp")

# %%
# Compute emphermis time
df_radiants.loc[:, "ET"] = \
    df_radiants.apply(lambda x: spiceypy.utc2et(x["datetime"].strftime("%Y-%m-%dT%H:%M:%S")),
                                                  axis=1)

# %%
# Compute the following vectors per ET:
# - Direction to Sun as seen from Earth
# - Direction to Apex as seen from Earth
# - Direction to Antihelion as seen from Earth
# - Direction to Anti-Apex as seen from Earth
df_radiants.loc[:, "Helion_vec"] = \
    df_radiants.apply(lambda x: np.array(spiceypy.spkgps(targ=10,
                                                         et=x["ET"],
                                                         ref="J2000",
                                                         obs=399)[0]),
                      axis=1)

# For the Apex we compute the velocity vector of Earth w.r.t. the Sun
df_radiants.loc[:, "Apex_vec"] = \
    df_radiants.apply(lambda x: np.array(spiceypy.spkgeo(targ=399,
                                                         et=x["ET"],
                                                         ref="J2000",
                                                         obs=10)[0][3:]),
                      axis=1)

# Antihelion: simply invert the Sun's vector
df_radiants.loc[:, "Antihelion_vec"] = \
    df_radiants.apply(lambda x: -1.0*x["Helion_vec"],
                      axis=1)
```

```python

# Antiapex: simply invert the Apex' vector
df_radiants.loc[:, "Antiapex_vec"] = \
    df_radiants.apply(lambda x: -1.0*x["Apex_vec"],
                      axis=1)

# %%
# Let's check the angles:
def angle_between_vectors(v1, v2):

    # Calculate the dot product of the vectors
    dot_product = np.dot(v1, v2)

    # Calculate the magnitude of the vectors
    v1_magnitude = np.linalg.norm(v1)
    v2_magnitude = np.linalg.norm(v2)

    # Calculate the cosine of the angle between the vectors
    cosine_of_angle = dot_product / (v1_magnitude * v2_magnitude)

    # Calculate the angle between the vectors
    angle = np.arccos(cosine_of_angle)

    # ... in degrees
    angle = np.degrees(angle)

    return angle

# We take the first entry only
antihelion_helion_angle = angle_between_vectors(df_radiants.iloc[0]['Helion_vec'],
                                                df_radiants.iloc[0]['Antihelion_vec'])

antiapex_apex_angle = angle_between_vectors(df_radiants.iloc[0]['Apex_vec'],
                                            df_radiants.iloc[0]['Antiapex_vec'])

helion_apex_angle = angle_between_vectors(df_radiants.iloc[0]['Helion_vec'],
                                          df_radiants.iloc[0]['Apex_vec'])

print(f"Angle between Helion and Antihelion direction in degrees: {antihelion_helion_angle}")
print(f"Angle between Apex and Antiapex direction in degrees: {antiapex_apex_angle}")
print(f"Angle between Apex and Helio direction in degrees: {helion_apex_angle}")

# %%
# Converting the vectors to sky coordiantes
df_radiants.loc[:, "Helion_RA"] = df_radiants["Helion_vec"].apply(lambda x: spiceypy.recrad(x)
[1])
df_radiants.loc[:, "Helion_Dec"] = df_radiants["Helion_vec"].apply(lambda x: spiceypy.recrad(x)
[2])
df_radiants.loc[:, 'Helion_RA_rad4plot'] = \
    df_radiants['Helion_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)
```

```
df_radiants.loc[:, "Antihelion_RA"] = df_radiants["Antihelion_vec"].apply(lambda x: 
spiceypy.recrad(x)[1])
df_radiants.loc[:, "Antihelion_Dec"] = df_radiants["Antihelion_vec"].apply(lambda x: 
spiceypy.recrad(x)[2])
df_radiants.loc[:, 'Antihelion_RA_rad4plot'] = \
    df_radiants['Antihelion_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else 
-1*x)

df_radiants.loc[:, "Apex_RA"] = df_radiants["Apex_vec"].apply(lambda x: spiceypy.recrad(x)[1])
df_radiants.loc[:, "Apex_Dec"] = df_radiants["Apex_vec"].apply(lambda x: spiceypy.recrad(x)[2])
df_radiants.loc[:, 'Apex_RA_rad4plot'] = \
    df_radiants['Apex_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)

df_radiants.loc[:, "Antiapex_RA"] = df_radiants["Antiapex_vec"].apply(lambda x: 
spiceypy.recrad(x)[1])
df_radiants.loc[:, "Antiapex_Dec"] = df_radiants["Antiapex_vec"].apply(lambda x: 
spiceypy.recrad(x)[2])
df_radiants.loc[:, 'Antiapex_RA_rad4plot'] = \
    df_radiants['Antiapex_RA'].apply(lambda x: -1*((x % np.pi) - np.pi) if x > np.pi else -1*x)

# %%
# Add some styles
plt.style.use('dark_background')
fig = plt.figure(figsize=(12, 8))

# Add aitoff projection
plt.subplot(projection="aitoff")

# Get axes
ax = plt.gca()

# Add a color for the velocity values
cm = plt.colormaps.get_cmap('jet')

filtered_df_radiants = df_radiants.loc[df_radiants["doy"]==180].copy()

# Plot the radiants
cr = plt.scatter(filtered_df_radiants['RA_rad4plot'], \
                 filtered_df_radiants['DECL_rad'], linestyle='None', \
                 s=1, c=filtered_df_radiants["Vg"], cmap=cm)

# Create a colormap
sm = plt.cm.ScalarMappable(cmap=cm, norm=plt.Normalize(min(df_radiants["Vg"].values),
                                                       max(df_radiants["Vg"].values)))

# Add the Sun, Antihelion, Apex and Antiapex direction
plt.plot(filtered_df_radiants.iloc[0]["Helion_RA_rad4plot"],
         filtered_df_radiants.iloc[0]["Helion_Dec"],
         marker="*", lw=0, markersize=10, color="yellow", label="Sun")

```

```
plt.plot(filtered_df_radiants.iloc[0]["Antihelion_RA_rad4plot"],
         filtered_df_radiants.iloc[0]["Antihelion_Dec"],
         marker="*", lw=0, markersize=10, color="orange", label="Antihelion")

plt.plot(filtered_df_radiants.iloc[0]["Apex_RA_rad4plot"],
         filtered_df_radiants.iloc[0]["Apex_Dec"],
         marker="^", lw=0, markersize=10, color="lightblue", label="Apex")

plt.plot(filtered_df_radiants.iloc[0]["Antiapex_RA_rad4plot"],
         filtered_df_radiants.iloc[0]["Antiapex_Dec"],
         marker="^", lw=0, markersize=10, color="red", label="Antiapex")

plt.legend()

# Convert the longitude values finally in right ascension hours
plt.xticks(ticks=np.radians(np.arange(-150, 180, 30)),
           labels=['10 h', '8 h', '6 h', '4 h', '2 h', '0 h', \
                   '22 h', '20 h', '18 h', '16 h', '14 h'])

# Plot the labels
plt.xlabel('Right ascension in hours')
plt.ylabel('Declination in deg.')

# Add a grid
plt.grid(True)

# Add the colorbar
ax = plt.gca()
color_bar = fig.colorbar(sm, orientation='horizontal', ax=ax)
color_bar.set_alpha(1)
color_bar.set_label('Geocentric entry velocity in km/s')

fig.savefig("dir_of_interests.png")


# %%


# %%




```