ztf_data

February 29, 2020

[1]:
```
# Core
import pandas as pd
import numpy as np
from scipy.interpolate import CubicSpline

# Astronomy
from astropy.time import Time
from astropy.units import deg

# Utility
import os
from datetime import date
import time
from tqdm.auto import tqdm

# Plotting
import matplotlib.pyplot as plt
import matplotlib as mpl

# Libraries for getting Alerce data out of ZTF2 database
import json
import psycopg2
from alerce.api import AlerceAPI

# MSE imports
from utils import range_inc
from astro_utils import date_to_mjd, mjd_to_date
from ztf_data import load_ztf_det, load_ztf_det_year, ztf_det_add_dir,␣
↪load_ztf_det_all
from ztf_data import interp_ast_dir
from ra_dec import radec2dir, radec_diff, direction_diff
from horizons_files import load_obs_ast_jpl
from asteroid_data import make_data_one_file, get_earth_pos
```

[2]:
```
# Set plot style variables
mpl.rcParams['figure.figsize'] = [16.0, 10.0]
mpl.rcParams['font.size'] = 16
```

0.0.1 Load Detections from Alerce ZTF2 Database

[3]:
```
ztf = load_ztf_det_all()
```

```
Loaded ../data/ztf/ztf-detections.h5 from disk.
```

[4]:
```
# Review DataFram
ztf
```

[4]:

	ObjectID	CandidateID	mjd	ra	\
0	b'ZTF18acebhfp'	676397301515010013	58430.397303	41.357345	
1	b'ZTF18abodmwk'	596403415715010014	58350.403414	30.969721	
2	b'ZTF18abodmwk'	626428345715010011	58380.428345	30.969705	
3	b'ZTF18abodmwk'	630507595715015045	58384.507593	30.969940	
4	b'ZTF18abodmwk'	618384965715010022	58372.384965	30.969643	
…	…	…	…	…	
5459014	b'ZTF20aareruw'	1151532523515015015	58905.532523	253.007910	
5459015	b'ZTF20aarerwx'	1151533002615015009	58905.533009	232.886408	
5459016	b'ZTF20aarerww'	1151533002115010003	58905.533009	236.167899	
5459017	b'ZTF20aarervr'	1151526063515015015	58905.526065	286.235286	
5459018	b'ZTF18aajqsjs'	1151533002315015001	58905.533009	237.382168	

	dec	ux	uy	uz	magap	asteroid_prob
0	58.879488	0.387942	0.653853	0.649598	18.946699	0.865682
1	65.305308	0.358224	0.558644	0.748059	19.010401	0.855504
2	65.305294	0.358224	0.558644	0.748059	18.935900	0.855504
3	65.305305	0.358223	0.558645	0.748059	19.260401	0.855504
4	65.305179	0.358226	0.558644	0.748058	19.220200	0.855504
…	…	…	…	…	…	…
5459014	55.485537	-0.165587	-0.169403	0.971537	19.192400	0.608023
5459015	53.509617	-0.358833	-0.115301	0.926253	19.687099	0.559474
5459016	54.618457	-0.322375	-0.116973	0.939357	19.957001	0.392662
5459017	33.876902	0.232120	-0.509626	0.828494	19.049299	0.517241
5459018	53.766297	-0.318612	-0.135924	0.938089	18.847700	0.992615

```
[5697962 rows x 10 columns]
```

[5]:
```
# Review data types
ztf.dtypes
```

[5]:
```
ObjectID            |S12
CandidateID         int64
mjd               float64
ra                float64
dec               float64
ux                float64
uy                float64
uz                float64
```

```
magap            float64
asteroid_prob    float64
dtype: object
```

0.0.2 Summarize Observations by Month

[ ]:
```
# Extract the year-month tuple for each observation for summarizing
tt = Time(ztf.mjd, format='mjd')
isotimes = tt.iso
ym = np.array([isotime[0:7] for isotime in isotimes])
ym_ser = pd.Series(data=ym, index=ztf.index)

# Group data by month for monthly summary
obs_monthly = ztf.groupby(ym_ser)
obs_monthly_count = obs_monthly.size()
```

[ ]:
```
# Calculations for plot
month_strs = obs_monthly_count.index.values
x_values = np.arange(obs_monthly_count.size)
x_dates = [date(int(x[0:4]), int(x[5:7]), 1) for x in month_strs]
y_values = obs_monthly_count.values

# Plot the number of observations by month
fig, ax = plt.subplots()
ax.set_title('Alerce Asteroid Observations by Month')
ax.set_xlabel('Month')
ax.set_ylabel('Asteroid Observations')
# ax.bar(x=x_values, height=y_values, tick_label=month_strs, color='blue')
ax.bar(x=x_values, height=y_values, color='blue')
ax.set_xticks(x_values[::3])
ax.set_xticks(x_values, minor=True)
ax.set_xticklabels(month_strs[::3], minor=False)
# ax.legend()
ax.grid()
fig.savefig('../figs/alerce/alerce_ast_per_month.png', bbox_inches='tight')
plt.show()
```

0.0.3 Extract key Data from ZTF Frame

[6]:
```
# Extract mjd, ra, and dec of the ZTF observations as arrays of astropy angles
mjd_ztf = ztf.mjd.values
ra_ztf = ztf.ra.values
dec_ztf = ztf.dec.values
```

[7]:
```
# Extract directions of the ZTF observations as an Nx3 array
u_ztf = ztf[['ux', 'uy', 'uz']].values
```

0.0.4 Load JPL Asteroid Directions for Comparison to ZTF

[8]:
```
# Load DataFrame of JPL observations of asteroids from palomar
# dir_name_jpl = '../data/jpl/testing/hourly'
# jpl = load_obs_ast_jpl(ast_num0=1, ast_num1=16, observer_name='palomar',␣
 ↪dir_name=dir_name_jpl)
```

[9]:
```
# Load DataFrame of JPL observations of asteroids from geocenter
dir_name_jpl = '../data/jpl/testing/daily'
jpl = load_obs_ast_jpl(ast_num0=1, ast_num1=16, observer_name='geocenter',␣
 ↪dir_name=dir_name_jpl)
```

[10]:
```
jpl
```

[10]:

	asteroid_num	mjd	JulianDate	time_key	RA_jpl	DEC_jpl	\
0	1	55197.0	2455197.5	1324728	243.215442	-17.105913	
1	1	55198.0	2455198.5	1324752	243.625145	-17.196033	
2	1	55199.0	2455199.5	1324776	244.034084	-17.284935	
3	1	55200.0	2455200.5	1324800	244.442231	-17.372621	
4	1	55201.0	2455201.5	1324824	244.849560	-17.459094	
…	…	…	…	…	…	…	
3648	16	58845.0	2458845.5	1412280	332.583400	-12.175911	
3649	16	58846.0	2458846.5	1412304	332.963279	-12.044021	
3650	16	58847.0	2458847.5	1412328	333.344527	-11.910963	
3651	16	58848.0	2458848.5	1412352	333.727108	-11.776749	
3652	16	58849.0	2458849.5	1412376	334.110983	-11.641395	

	ux_jpl	uy_jpl	uz_jpl	RA_apparent	DEC_apparent	delta	\
0	-0.430702	-0.899812	0.069523	243.358581	-17.131844	3.437877	
1	-0.424384	-0.902835	0.069195	243.768548	-17.221645	3.430618	
2	-0.418063	-0.905804	0.068867	244.177730	-17.310227	3.423244	
3	-0.411741	-0.908720	0.068539	244.586099	-17.397589	3.415755	
4	-0.405417	-0.911583	0.068211	244.993632	-17.483728	3.408150	
…	…	…	…	…	…	…	
3648	0.867713	-0.496854	-0.014471	332.842722	-12.079671	3.017646	
3649	0.871109	-0.490873	-0.014611	333.222363	-11.947459	3.027660	
3650	0.874478	-0.484841	-0.014750	333.603364	-11.814086	3.037593	
3651	0.877821	-0.478758	-0.014889	333.985688	-11.679564	3.047446	
3652	0.881136	-0.472625	-0.015026	334.369299	-11.543908	3.057216	

	delta_dot	light_time
0	-12.468091	28.591952
1	-12.668422	28.531584
2	-12.868277	28.470254
3	-13.067723	28.407966
4	-13.266749	28.344720
…	…	…

```
3648  17.406609   25.096998
3649  17.268707   25.180280
3650  17.128990   25.262894
3651  16.987592   25.344833
3652  16.844636   25.426089

[58448 rows x 14 columns]
```

0.0.5 Load MSE Calculated Asteroid Directions for Comparison to ZTF

[ ]:
```
# Date range in ZTF data
mjd_min = np.min(mjd_ztf)
mjd_max = np.max(mjd_ztf)
print(f'ZTF data range: {mjd_min:9.3f} to {mjd_max:9.3f}')
```

[ ]:
```

```

[ ]:
```
# Load MSE asteroid data
inputs_mse, outputs_mse = make_data_one_file(0, 1000)

# Associated asteroid numbers
# ast_num_mse = np.arange(1, 1000, dtype=np.int32)
```

[ ]:
```
inputs_mse.keys()
```

[ ]:
```
outputs_mse.keys()
```

[ ]:
```
q_mse = outputs_mse['q']
q_mse.shape
```

[ ]:
```
# Range of asteroids to compare to ZTF
ast_num_min: int = 1
ast_num_max: int = 100

# Mask to overlap with selected asteroids for comparison
mask_mse_ast = (ast_num_min <= ast_num_mse) & (ast_num_mse <= ast_num_max)

# Extract times for MSE integration
mjd_mse = inputs_mse['ts'][0]

# Mask to overlap with the ZTF data
mask_mse_t = (mjd_min <= mjd_mse) & (mjd_mse <= mjd_max)

# Masked asteroid number and mjd
ast_num_src = ast_num_mse[mask_mse_ast]
mjd_src = mjd_mse[mask_mse_t]
```

[ ]:
```
ast_num_src
```

[ ]:
```
# mjd_src
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
# Alias inputs to interp_ast_dir
ast_num_src = jpl.asteroid_num.values
mjd_src = jpl.mjd.values
u_src = jpl[['ux_jpl', 'uy_jpl', 'uz_jpl']].values
ast_num_out = 1
mjd_out = mjd_ztf
```

[ ]:
```
# Splined direction of this asteroid according to JPL
u_jpl = interp_ast_dir(ast_num_src, mjd_src, u_src, ast_num_out, mjd_out)
```

[ ]:
```
u_jpl.shape
```

[ ]:
```
ztf
```

[ ]:
```
def compare_ztf_src(ztf: pd.DataFrame, ast_num_src: np.ndarray, mjd_src: np.
↪ndarray, u_src: np.ndarray):
    """
    Construct splined predicted asteroid directions from a source at desired␣
↪dates.
    INPUTS:
        ztf         : DataFrame of ZTF observations; columns must include mjd,␣
↪ux, uy, uz
        ast_num_src: asteroid numbers whose position is predicted by source;␣
↪shape (N,)
        mjd_src     : modified julian dates as of which direction is predicted␣
↪by source; shape (N,)
        u_src       : directions from observatory to asteroid predicted by␣
↪source; shape (N,3,)
    OUTPUTS:
        ast_nums    : array of distinct asteroids whose distance is compared;␣
↪shape (K,)
        angle_diff : difference in angle between each ZTF observation and␣
↪splined asteroid position; shape (N,K,)
    """
```

```
    pass
```

[ ]:

[ ]:
```
# Get distinct asteroid numbers in source
ast_nums = np.unique(ast_num_src)
K = ast_nums.size

# Number of rows in ZTF data
M = ztf.shape[0]

# Initialize empty array of distances
angle_diff = np.zeros(shape=(M,K))

# Array of times to be interpolated from ztf
mjd_out = ztf.mjd.values

# Extract directions of the ZTF observations as an Mx3 array
u_ztf = ztf[['ux', 'uy', 'uz']].values
```

[ ]:
```
ast_nums
```

[ ]:
```
K
```

[ ]:
```
M
```

[ ]:
```
# Iterate over asteroids one at a time
for k, ast_num in enumerate(ast_nums):
    # The interpolated direction of this asteroid at the ZTF observation times
    u_out = interp_ast_dir(ast_num_src=ast_num_src, mjd_src=mjd_src,␣
 ↪u_src=u_src,
                           ast_num_out=ast_num, mjd_out=mjd_out)
    # Distance from u_ztf to u_out
    u_dist = np.linalg.norm(u_out - u_ztf, axis=1)

    # Convert to arc seconds and save to column k
    # angle_diff[:, k] = 2.0*np.arcsin(u_dist*0.5)
    angle_diff[:, k] = u_dist
```

[ ]:
```
angle_diff.shape
```

[ ]:
```
np.min(angle_diff)
```

[ ]:
```
np.max(angle_diff)
```

[ ]:
```
u_dist.shape
```

[ ]:
```
np.max(u_out)
```

[ ]:
```
u_out.shape
```

[ ]:
```
u_out[0]
```

[ ]:
```
np.linalg.norm(u_out, axis=1)
```

[ ]:
```
u_out_sz = np.linalg.norm(u_out, axis=1)
```

[ ]:
```
np.argmax(u_out_sz)
```

[ ]:
```
i = np.argmax(u_out_sz)
```

[ ]:
```
u_out[i]
```

[ ]:
```
mjd[i]
```

[ ]:
```
mjd_out[i]
```

[ ]:
```
ast_num_out = ast_num
mask = (ast_num_src == ast_num_out)
```

[ ]:
```
x_spline = mjd_src[mask]
y_spline = u_src[mask]
u_spline = CubicSpline(x=x_spline, y=y_spline)
```

[ ]:
```
np.min(x_spline)
```

[ ]:
```
np.max(x_spline)
```

[ ]:
```

```