Viewing satellite tracks

This notebook explores the satellite track data computed from the raw [TLE data (./Viewing_TLEs.ipynb)](./Viewing_TLEs.ipynb) for the 12/2020 VAULT technical scenario, showing what data is available and how it can be accessed and visualized from Python. The notebook also acts as a runnable application that can be put on a server to allow users to explore the raw data interactively.

In [1]:
```
import pandas as pd
import numpy as np
import time, datetime, calendar
import colorcet as cc
import panel as pn
import datetime as dt
import holoviews as hv
from tables import open_file
from holoviews.operation.datashader import rasterize, dynspread
from holoviews.util.transform import easting_northing_to_lon_lat as en2ll
from holoviews.util.transform import lon_lat_to_easting_northing as ll2en
import skyfield
from skyfield.framelib import itrs
from skyfield.sgp4lib import EarthSatellite
from spatialpandas.geometry import PointArray
from spatialpandas import GeoDataFrame

hv.extension('bokeh')
```

Data preparation

First we'll load all the TLEs, then filter them by those for which tracks have been precomputed:

In [2]:
```
tle = pd.read_csv('data/satellite data/Cleaned TLE/tle2017.csv')
```

In [3]:
```
computed = open_file("data/satellite data/Indexed TLE/precomp2.h5", mode='r')
sat_group = computed.get_node("/sat")
num1, num2 = 205, 320
svals = [int(el[1:]) for el in dir(sat_group) if el.startswith('s')]
tle = tle[tle['norad_id'].isin(svals)]
```

Next, we'll convert the TLE location data at the epoch time to latitude and longitude, filtering out values that cannot be visualized in a Web Mercator projection. This string-processing task can take a while for a large file.

In [4]:
```
def modulo_lon(val):
    return (val+180) % 360 - 180

def compute_lat_lon(line1, line2):
        """Get the Lat/Lon at the TLE epoch"""
        sat = EarthSatellite(line1, line2)
        lat, long, _ = sat.at(sat.epoch).frame_latlon(itrs)
        return lat.degrees, long.degrees

def lat_lon_from_lines(lines, abs_max_lat=84):
    """Computes latitude/longitude and a mask to
    filter TLEs that don't work for Web Mercator
    (abs > 84 degrees by default)"""
    lons, lats, mask, inner_mask = [], [], [], []
    for line1, line2 in lines:
        lat, lon = compute_lat_lon(line1,line2)
        if None not in [lon, lat]:
            inner_mask.append(abs(lat) < abs_max_lat)
            mask.append(abs(lat) < abs_max_lat)
            lons.append(lon if lon < 180 else (lon - 360))
            lats.append(lat)
        else:
            mask.append(False)
    return np.array(lats)[inner_mask], np.array(lons)[inner_mask], mask
```

In [5]:
```
new_lines = [el.replace('None\n', '').split('\n')[:2] for el in tle['tle']] # Splitting the file
```

In [6]:
```
%%time
lats, lons, mask = lat_lon_from_lines(new_lines)
```

```
CPU times: user 33.4 s, sys: 8.54 ms, total: 33.4 s
Wall time: 33.5 s
```

In [7]:
```
%%time
x, y = ll2en(lons, lats)
```

```
CPU times: user 3.82 ms, sys: 0 ns, total: 3.82 ms
Wall time: 3.27 ms
```

Spatial index

To allow quick lookup of satellite by geographic location, we'll build a spatially indexed spatialpandas (https://github.com/holoviz/spatialpandas) GeoDataFrame (which can take a while for a large dataset, because it needs to sort all the datapoints).

In [8]:
```
%%time
sdf = GeoDataFrame({'geometry':PointArray((lons, lats)),'x':x,'y':y,
                    'norad_id':tle['norad_id'][mask],
                    'epoch_year': tle['epoch_year'][mask],
                    'epoch_day': tle['epoch_day'][mask],
                    })
```

```
CPU times: user 22.2 ms, sys: 22 µs, total: 22.2 ms
Wall time: 21.7 ms
```

Satellite trajectory data

Before running this notebook, satellite trajectories will need to have been precomputed externally. Here, we will just look up the data corresponding to a given TLE.

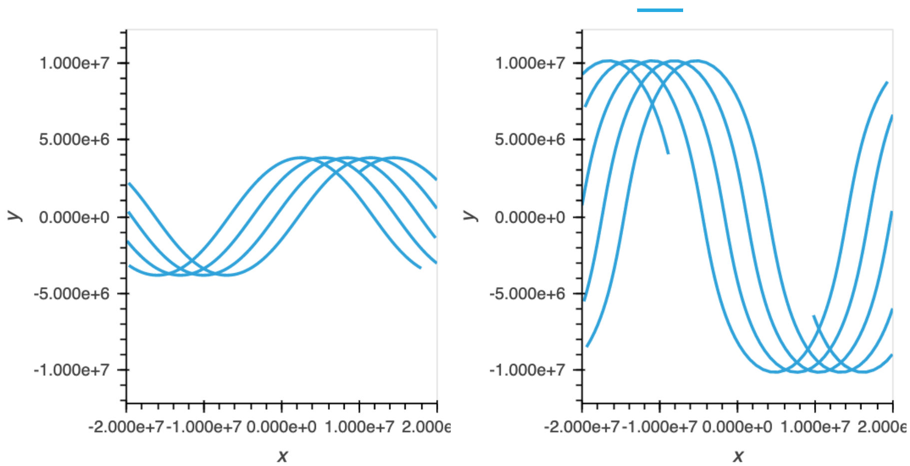
In [9]:
```
def get_track_around_TLE(sat_id, epoch_year, epoch_day, delta_seconds=4*60*60):
    tle_datetime = (dt.datetime(year=epoch_year, month=1, day=1)
                    + dt.timedelta(days=epoch_day-1))
    return get_precomputed_tracks(sat_id,
                                  tle_datetime-dt.timedelta(seconds=delta_seconds),
                                  tle_datetime+dt.timedelta(seconds=delta_seconds))

def get_precomputed_tracks(satellite, start, end):
    name = "s" + str(satellite)
    dataz = getattr(sat_group, name)[:]
    start_index = np.searchsorted(dataz[0, :], start.timestamp())
    end_index   = np.searchsorted(dataz[0, :], end.timestamp())
    return dataz[:, start_index: end_index]

def get_track(track, lat_clip=85.5):
    lat, lon = track[1,:], track[2,:]
    mask = np.abs(lat) > lat_clip
    lat[mask] = np.float('nan')
    lon[mask] = np.float('nan')
    lon = np.array([modulo_lon(el) for el in lon])

    eastings, northings = ll2en(lon,lat)
    # Heuristic to insert NaNs to break up Curve (prevent wrapping issues at date line)
    inds = np.where(np.abs(np.diff(eastings)) > 2e7)[0] # Big delta to split on
    inds += 1
    eastings  = np.insert(eastings,  inds, [float('nan') for i in range(len(inds))])
    northings = np.insert(northings, inds, [float('nan') for i in range(len(inds))])
    return hv.Curve((eastings, northings))
```

For example, here are the tracks for satellite IDs 205 and 320:

In [10]:
```
epoch_year, epoch_day = tle.iloc[0]['epoch_year'], tle.iloc[0]['epoch_day']
(get_track(get_track_around_TLE(205, epoch_year, epoch_day, delta_seconds=4*60*60)) +
 get_track(get_track_around_TLE(320, epoch_year, epoch_day, delta_seconds=4*60*60)))
```

Out[10]:

(https://bokeh.org/)



Satellite track visualizing app

Using the above lookup functions, we can now make an app that lets you click on a TLE record and plot the track of that satellite.

In [11]:
```
DELTA_SECONDS = 60*60 # Track length in time (seconds)

def mark_track(x,y):
    delta=0.1
    empty = hv.Curve([(0,0)]).opts(alpha=0)
    if None not in [x,y]:
        x, y = en2ll(x, y)
        row = sdf.cx[x-delta:x+delta, y-1:y+1]
        if len(row) == 0:
            return empty
        satid = int(row.iloc[0]['norad_id'])
        epoch_year = row.iloc[0]['epoch_year']
        epoch_day = row.iloc[0]['epoch_day']
        track = get_track_around_TLE(satid, epoch_year, epoch_day,
                                      delta_seconds=DELTA_SECONDS)
        return get_track(track).opts(color='red', alpha=1)
    else:
        return empty


tracks = hv.DynamicMap(mark_track, streams=[hv.streams.Tap()])
```

In [13]:
```
overlay = (hv.element.tiles.ESRI().opts(alpha=0.8, bgcolor='black')
 * dynspread(rasterize(hv.Points(zip(x,y)))).opts(width=900, height=600, cmap=cc.kbc[64:], cnorm='eq_hist')
 * tracks)
```

In [14]:
```
pn.Column("# Computed satellite tracks",
          "Click on one of the TLE records shown (blue dots) to select a satellite, "
          "and (after a short delay) a portion of that satellite's track will be shown.",
          overlay).servable()
```

Out[14]:

Computed satellite tracks

Click on one of the TLE records shown (blue dots) to select a satellite, and (after a short delay) a portion of that satellite's track will be shown.