run\_script

TripMapper

main\_ForTestingPurposesOnly

PlotInPolarCoordinates

Aircraft

FlightProfile

Means the begin-arrow module is imported into the end-arrow module

PlotInPolarCoordinates

Aircraft

TripMapper

LoadExistingUAMaerodromeInfrastructure

main\_ForTestingPurposesOnly

FlightProfile

LoadExistingUAMaerodromeInfrastructure

Aircraft

TripMapper

Aircraft

FlightProfile

**RunScript Function**

**Install Bokeh module\*\***

**Goal:** This is the main running script where we can ideally specify what kind of analysis we want to run

**Input(s):**

1. **Option:** specify what kind of analysis we want to run [type: integer]; 1: if you want to plot for a specific cruising altitude, 2: if you want to plot CLAP vs altitude
2. **SaveMap = False:** should the Bokeh map be saved? by default, it is False
3. **GenerateMapVideo = False:** should a video of the trip on the Bokeh map be generated? by default, it is False
4. **GeneratePolarPlotVideo = False:** should a video of the polar plot showing reachable ground footprint be generated? by default, it is False

**Output(s):**

1. If Option = 1: **PlotMultipleTrips** function in *main\_ForTestingPurposesOnly.py* file is called
2. If Option = 2: **PlotCLAP()** function in *main\_ForTestingPurposesOnly.py* is called

**NOTE:** make sure to change directory path to save the generated videos

**PlotMultipleTrips Function**

**Goal:** creates instances of trips and plots it on a bokeh map

**Input(s):**

1. **cruiseAltitudeInFeet:** cruising altitude in feet [type: float]
2. **Show = False:** show the bokeh map? By default, it is false

**What is inside this function?**

1. Create an instance of **TripMapper** Object (e.g., Chicago)
2. Create an instance of the **Aircraft** Object (e.g., Joby)
3. Get the reachable footprint by running the **method of Aircraft class called ReachableGroundFootprint**
4. Define the departure aerodrome ID (**dep**) and type (**depType**)
5. Define the arrival aerodrome ID (**arr**) and type (**depType**)
6. Define the waypoints list of the route in lat (**lonWP\_inDeg**) lon (**latWP\_inDeg**) and create a list called **WayPoints**
7. Draw the non-direct route trip on a Bokeh map, along with the reachable footprint as well, by **running the method of TripMapper class called DrawNonDirectRoutingTrip**
8. Output and show the map as an HTML file

**Output(s):**

1. **CLAP:** [type: float] output the CLAP value (%) for the trip, its waypoints, at the input cruising altitude in feet
2. **NOTE:** make sure to change directory path to save the plots

**Aircraft Class**

**Goal:** Here we can define the aircraft characteristics and compute its reachable ground footprint

1. **Input(s):**
   1. **ACname:** [type: string] Name of the aircraft
   2. **PAX:** [type: integer] Passenger capacity
   3. **Speed\_Cr:** [type: float] Max Cruise Speed in MPH
   4. **Range:** [type: float] Maximum range of the aircraft in miles
   5. **LD\_Ratio:** [type: float] Lift to Drag ratio of the aircraft
   6. **Max\_Flt\_Time:** [type: float] Maximum flight time of the aircraft in minutes
   7. **MTOW:** [type: float] Maximum takeoff weight in kg
   8. **KWh\_Batt:** [type: float] Battery energy capacity in kWh
   9. **S:** [type: float] wing surface area in meters cubed
2. **Attribute(s):**
   1. **Same as Inputs (Note:** ideally, in the near future, the input would be changed to a file path that loads the vehicle performance characteristics and assigns them to attributes)
3. **Method(s):**
   1. **Characteristics:** [type string] summarize the characteristics in a nice print format
   2. **ReachableGroundFootprint:**
      1. **Inputs:** 
         1. Altitude [type: float] altitude in feet
         2. mu\_deg: does not do anything in the code (can ignore this parameter)
         3. PowFailure: also does not do anything in the code yet (can ignore this parameter)
      2. **Outputs:** 
         1. X: [type: list] x-coordinate distance of footprint in meters
         2. Y: [type: list] y-coordinate distance of footprint in meters
         3. Radians: [type: list] polar plot angles in radians
         4. Radial: [type: list] polar plot radial distance in miles

The other methods listed below are responsible for helping the ReachableGroundFootprint method.

* 1. **FindTurnRadius**
  2. **FindTurnRadiusArcLength**
  3. **EstimateLiftandDrag**
  4. **EstimateSinkRate**
  5. **EstimateAltitudeLossDuringTurn**
  6. **Symmetry**
  7. **Reverse**
  8. **Combine**
  9. **PlotReachabilityFootprint**
  10. **PolarPlotReachabilityFootprint**
  11. **radians2degrees**
  12. **degrees2radians**

**FlightProfile Class**

**Goal:** Once the aircraft is defined, this class defines the flight profile of the vehicle for a trip from start to finish, with phases including: takeoff (A), climb (B), cruise (C), descend (D), land (E)

**Note**: Most of the below Attributes and Methods are explained in the comments section of the code in more detail

1. **Input(s):**
   1. **Aircraft:** [type: string] name of the aircraft
   2. **cruise\_alt:** [type: float] desired cruising altitude in feet
   3. **trip\_distance:** [type: float] trip distance (not displacement!) in miles
2. **Attribute(s):**
   1. **altitude\_cr:** [type: float]cruise\_alt is in feet, so converting here to meters
   2. **tripDistance**: [type: float]trip geodesic distance is converted from miles to meters
   3. **cruiseSpeed (ignore this attribute)**
   4. **velocity\_cr:** [type: float]cruising speed is converted from mph to m/s
   5. **dh:** [type: float]this is the climbing height in meters
   6. **dx:** [type: float]this is the horizontal distance traveled in climb phase, in meters
   7. **climbAngle:** [type: float]climb angle in radians
   8. **ROC:** [type: float]Rate of Climb is in m/s
   9. **ROD:** [type: float]Rate of Descent has the opposite sign of Rate of Climb, in m/s
   10. **segment\_A\_altitude:** [type: list]altitudes are in feet
   11. **segment\_A\_range:** [type: list]ranges are in miles
   12. **segment\_A\_horz\_vel:** [type: list]horizontal velocities are in meters per second
   13. **segment\_A\_vert\_vel:** [type: list]vertical velocities are in meters per second
   14. **segment\_B\_altitude:** [type: list]altitudes are in feet
   15. **segment\_B\_range:** [type: list]ranges are in miles
   16. **segment\_B\_horz\_vel:** [type: list]horizontal velocities are in meters per second
   17. **segment\_B\_vert\_vel:** [type: list]vertical velocities are in meters per second
   18. **segment\_C\_altitude:** [type: list]altitudes are in feet
   19. **segment\_C\_range:** [type: list]ranges are in miles
   20. **segment\_C\_horz\_vel:** [type: list]horizontal velocities are in meters per second
   21. **segment\_C\_vert\_vel:** [type: list]vertical velocities are in meters per second
   22. **segment\_D\_altitude:** [type: list]altitudes are in feet
   23. **segment\_D\_range:** [type: list]ranges are in miles
   24. **segment\_D\_horz\_vel:** [type: list]horizontal velocities are in meters per second
   25. **segment\_D\_vert\_vel:** [type: list]vertical velocities are in meters per second
   26. **segment\_E\_altitude:** [type: list]altitudes are in feet
   27. **segment\_E\_range:** [type: list]ranges are in miles
   28. **segment\_E\_horz\_vel:** [type: list]horizontal velocities are in meters per second
   29. **segment\_E\_vert\_vel:** [type: list]vertical velocities are in meters per second
3. **Method(s):**
   1. **PlotMissionProfile:** plots the segments A, B, C, D, E on an altitude vs range graph to represent the flight profile
   2. **FlightTime:** calculates flight time of the flight profile and returns it in minutes
   3. **EnergyConsumption:** calculates the energy consumption of the aircraft for the flight profile and returns in kWh
   4. **GivenRangeOutputAltitude:** as altitude changes when aircraft is climbing/descending, this function is used to get the altitude of the aircraft in meters given the position of the aircraft (miles) in the trip

**LoadExistingUAMaerodromeInfrastructure Class**

**Goal:** when this class is called, it loads all the existing UAM aerodrome infrastructures from a .xlsx file and outputs a list of longitude and latitudes for major, regional, heliports coordinate in the format (in Mercator coordinates) shown below:

Lat\_m\_regional = [lat1, lat2, …]

Lat\_m\_regional = [lon1, lon2, …]

Lat\_m\_major = [lat1, lat2, …]

Lat\_m\_major = [lon1, lon2, …]

Lat\_m\_heliports = [lat1, lat2, …]

Lon\_m\_heliports = [lon1, lon2, …]

***Inputs:*** file path location that contains the Lat, Lon coordinates in deg for the regional, major, and helipad

***Outputs:*** Lat, Lon coordinates of regional, major, heliports in both Mercator and Degree coordinates

**Note:** if you are calling a function inside another function, try using self.functionName.

1. **Initialization Inputs:** 
   1. **MetroArea:** e.g., “Chicago”, “Dallas”
2. **Attributes:**
   1. **FilePath:** e.g., “Datasets/” + MetroArea + “.xlsx” (Make sure this path is correctly defined in your machine)
   2. **Lat\_regional\_merc, Lon\_regional\_merc** = ReadExcelSheet(FilePath, “Regional”)
   3. **Lat\_major\_merc, Lon\_major\_merc** = ReadExcelSheet(FilePath, “Major”)
   4. **Lat\_heliports\_merc, Lon\_heliports\_merc** = ReadExcelSheet(FilePath, “Heliports”)
   5. **Lat\_regional\_deg, Lon\_regional\_deg** = ReadExcelSheet(FilePath, “Regional”)
   6. **Lat\_major\_deg, Lon\_major\_deg** = ReadExcelSheet(FilePath, “Major”)
   7. **Lat\_heliports\_deg, Lon\_heliports\_deg** = ReadExcelSheet(FilePath, “Heliports”)
3. **Methods:**
   1. **ReadExcelSheetInMercatorOnly:** read the excel sheet that is in degrees and convert to mercator
      1. Inputs: FilePath, SheetName
      2. Outputs: lat\_merc, lon\_merc
   2. **ReadExcelSheetInDegreesOnly:** read the excel sheet as it is, as default units are in degrees
      1. Inputs: FilePath, SheetName
      2. Outputs: lat\_deg, lon\_deg
   3. **InMercator Function:** call to convert (deg) to (Mercator) coordinates
      1. Inputs: lat\_deg, lon\_deg
      2. Outputs: lat\_merc, lon\_merc
   4. **Degree2Mercator Function:** this is the function that takes in a tuple (lat, lon) and converts to Mercator coordinates (Easting, Northing)
      1. Inputs: Coords
      2. Outputs: (x, y)
   5. **Mercator2Degrees:**
      1. Inputs: x, y
      2. Outputs: lat\_deg, lon\_deg

**TripMapper(LoadExistingUAMaerodromeInfrastructure) Class**

**Goal:** TripMapper oversees mapping the trip on a graph. It inherits LoadExistingUAMaerodromeInfrastructure class, so that the coordinates of existing aerodrome could be used for plotting and analysis

1. **Input(s):**
   * 1. **MetroName:** name of the metropolitan area (e.g., "Chicago", "Dallas")
     2. **MapType:** map style (e.g, "Satellite", "terrain", "OSM", etc.)
2. **Attribute(s):**
   * 1. **LoadExistingUAMaerodromeInfrastructure.\_\_init\_\_(self, MetroName)** inherits this class and the attributes related to the input metroname
     2. **MapType:** map style (e.g, "Satellite", "Map")
     3. **p:** data structure that contains the plotter information from running MapperInfrastructure Method
3. **Method(s):**
   * 1. **MapperInfrastructure**: maps out the metro area, on a specified map style, and the existing aerodromes using Bokeh library
     2. **TripDistance**: given two aerodromes, this function estimates the geodesic trip distance between these two points in miles, using GeodesicDistance Method
     3. **DrawGeodesicTrip**: (this function is a backup function if **DrawNonDirectRoutingTrip** is not working!) given two aerodromes, one departing and one arrival, this function plots the geodesic trip route on the map attribute "p"; also plots the reachable footprint of an aircraft under emergency conditions in most cases, a trip may not be on a geodesic route; so waypoints, in mercator coordinates, should be incorporated for defining non-direct routes; the number of waypoints is up to the user
     4. **DrawNonDirectRoutingTrip**: incorporating waypoints, this function does the same execution as Method 3, but for non-direct routing cases
        1. **Inputs**: DepType (e.g., “regional”), ArrType (e.g., “heliport”), idxDep (i.e., integer value airport id), idxArr (i.e., integer value airport id), Aircraft (e.g., “Joby”), CruiseAltitudeInFeet (i.e., integer value of desired cruise altitude in feet), WayPoints (array of lat, lon coordinates in degrees), Save=False
        2. **Outputs**: NoneType
     5. **PlotFlightProfiles**(Aircraft, CruiseAltitudeInFeet): plot the flight profile
        1. **Inputs:** Aircraft (e.g., “Joby”), CruiseAltitudeInFeet (i.e., desired cruising altitude)
        2. **Outputs:** FP1 (array of range vs. altitude data points, and more), see FlightProfile documentation for the detailed output list
        3. **NOTE:** make sure to change directory path to save the plots
     6. **FindClosestUAMaerodromeAlongTheRoute**()
        1. **Inputs:** None
        2. **Outputs:** Distance2ClosestContingencySite (miles), Direction (heading relative to North in degrees), Labeling (airport type: heliport, regional, major)
        3. **NOTE:** make sure to change directory path to save the plots
     7. **RotateReachableFootprintByTripHeading**(X, Y, teta)
        1. **Inputs:** X, Y, (x and y coordinates of the reachable footprint in SI unit meters), teta (rotation of the footprint according to the aircraft heading direction)
        2. **Outputs:** X\_new, Y\_new (rotated reachable footprint to line up with the aircraft heading direction corrected from SI unit meters to Mercator meters by factor of 5000/3730) (Note: I noticed this discrepancy when plotting the footprint on a map and came up with the scale factor to correct it accordingly)
     8. **PlotReachableRadiusAlongRoute**(Lat, Lon, X\_new, Y\_new, Save): plots the reachable footprint on the **p** map data structure defined above
        1. **Inputs:** Lat (route coordinates in degrees), Lon (route coordinates in degrees), X\_new (reachable footprint in x direction in mercator meters), Y\_new (reachable footprint in x direction in mercator meters), Save
        2. **Outputs:** NoneType
     9. **CalculateDetourRatio**()
     10. **ComputeCLAP**(self, X\_newArray, Y\_newArray, Distance2ClosestContingencySite, Direction)
     11. **WaypointsConnector**(self,LatLonFrom\_merc, LatLonTo\_merc, sampSize)
     12. **EstimateTripHeading**(self, delLat, delLon)
     13. **ShowMap**(self)
     14. **SaveMap**(self, Lat, Lon, imgNum, PatchLatj, PatchLonj)
         1. **NOTE:** make sure to change directory path to save the plots
     15. **GeodesicDistance**(self, lat1, long1, lat2, long2)
     16. **PlotContingencyLandingAssurance\_vs\_CruiseAltitude**(self, altitude, CLAP)
         1. **NOTE:** make sure to change directory path to save the plots