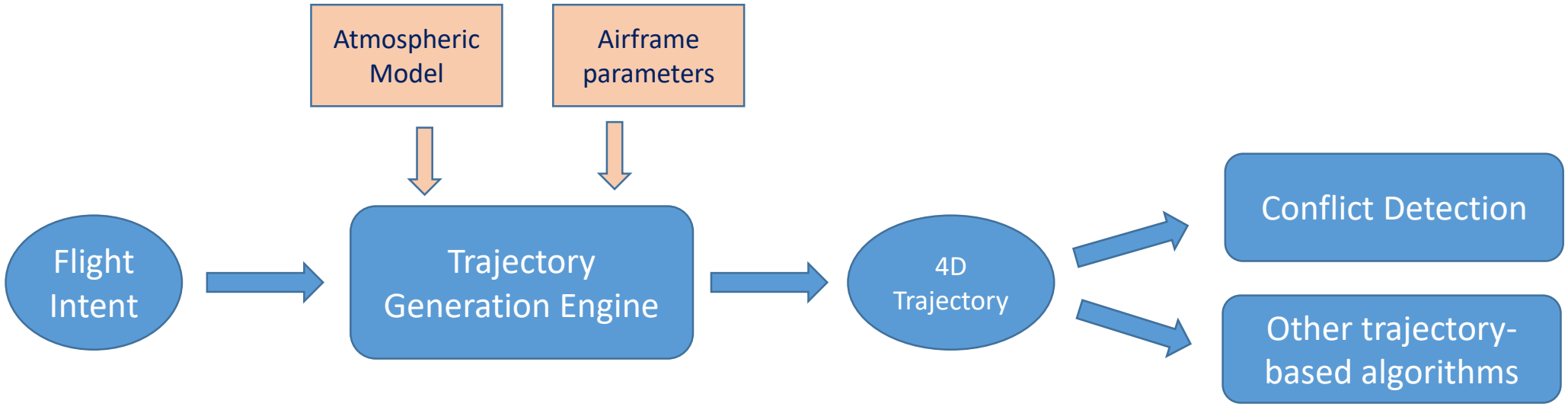
AMSC/CMSC 663/664
Mid Year Report

December 1, 2016

Jon Dehn

Project Goal

- Build a framework for testing compute-intensive algorithms in air traffic management



Progress

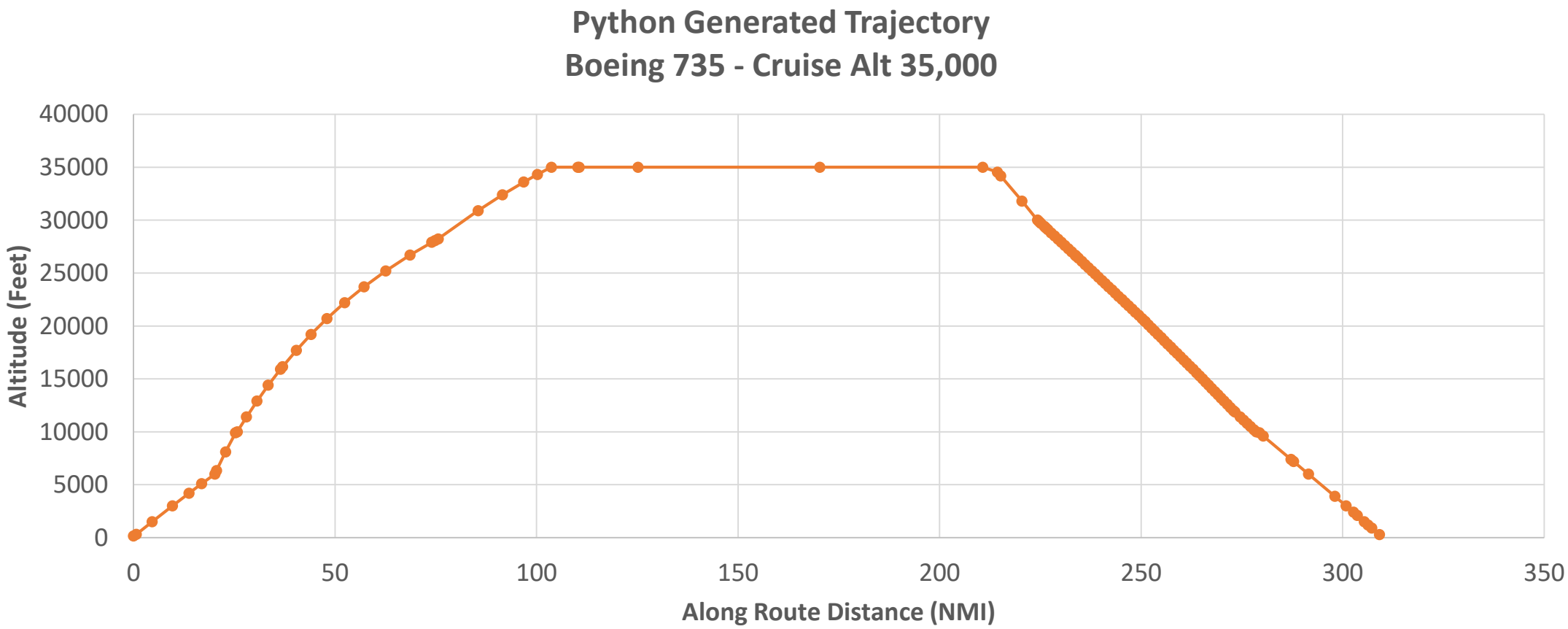
- **Flight Intent** (cruise altitude, speed, aircraft type, 2D path) is accessed by screen-scraping the Flight Aware website (flightaware.com)
- **Atmospheric Model** is obtained from NOAA (using 40 km grid; pressure levels):
  - ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/rap/prod
- **Airframe Parameters** are obtained from Eurocontrol's BADA website (license for this project obtained November 17)
- **Trajectory Generation Engine** has been coded in Python
  - Approximately 5000 source lines of code in 30 files in 6 packages

Review – essential trajectory generation equation

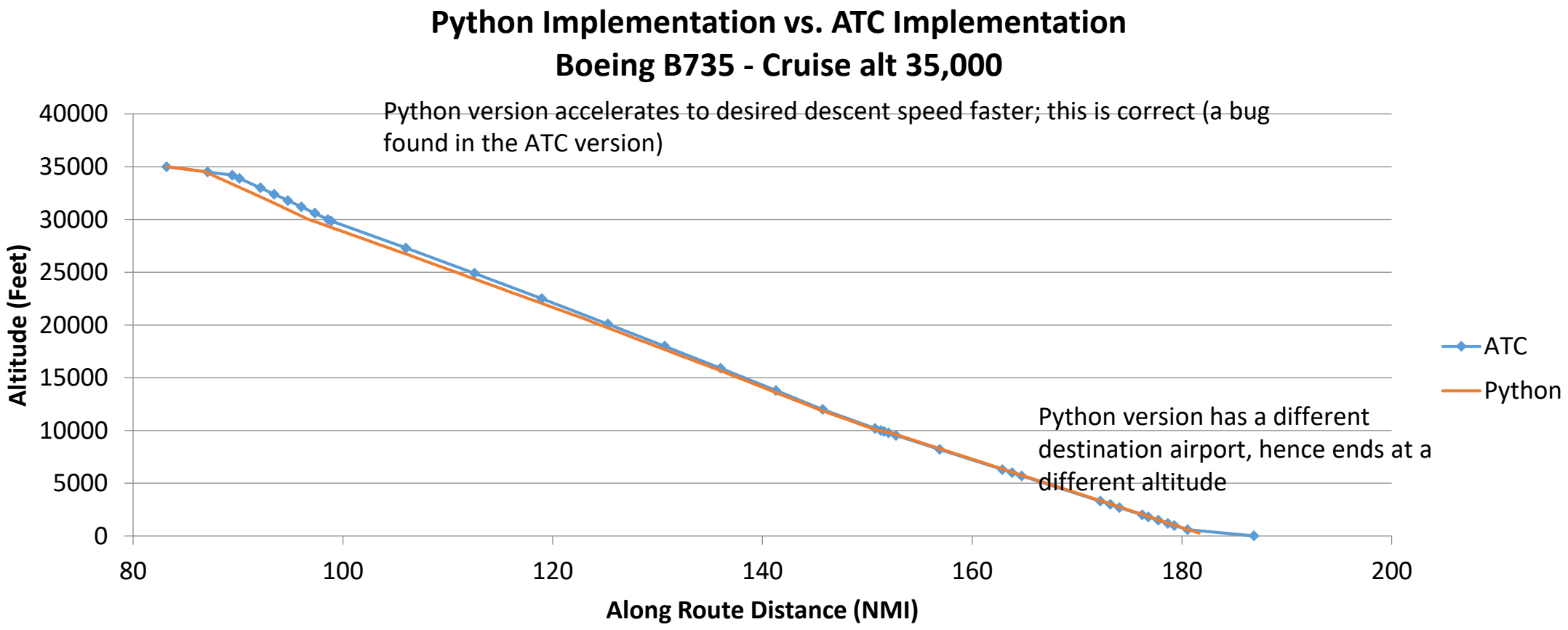
$$\frac{dH_p}{dt} = \frac{T - \Delta T}{T} \frac{(Thr - D)V_{TAS}}{mg} f(M)$$

- Python code implements this with first order Runge-Kutta (higher order Runge-Kutta still to be implemented)
- All low-level physics computations were compared against existing "Air Traffic Control (ATC)" code for correctness
- The generated trajectory was then be compared against a result obtained from prototype ATC code

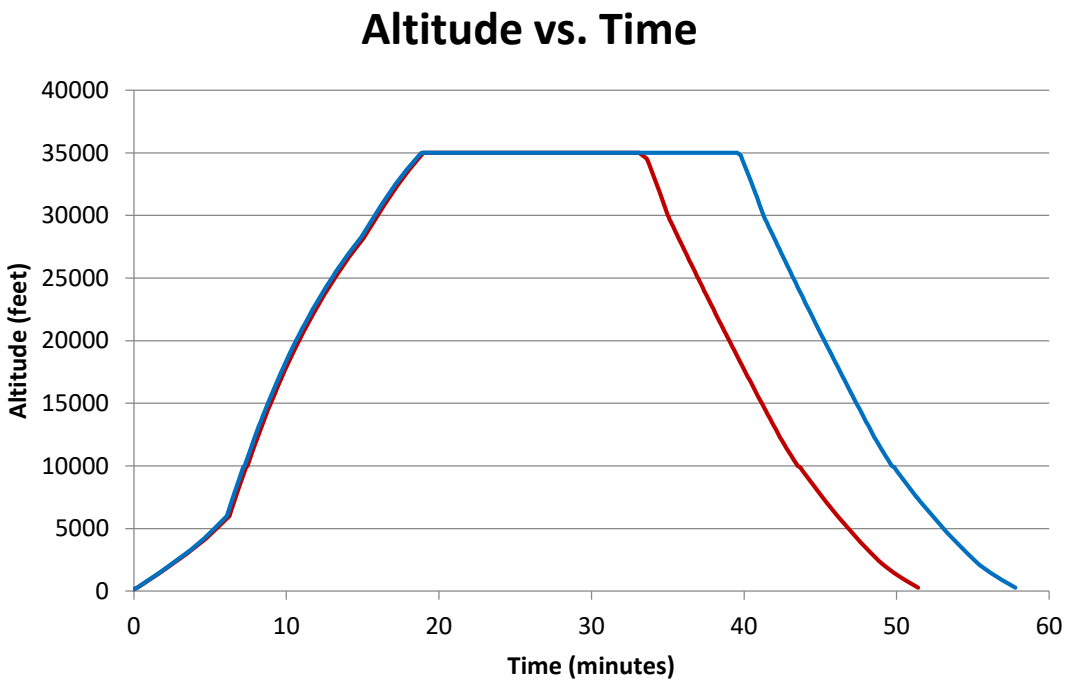
Results – General trajectory profile
(Altitude vs. Distance)

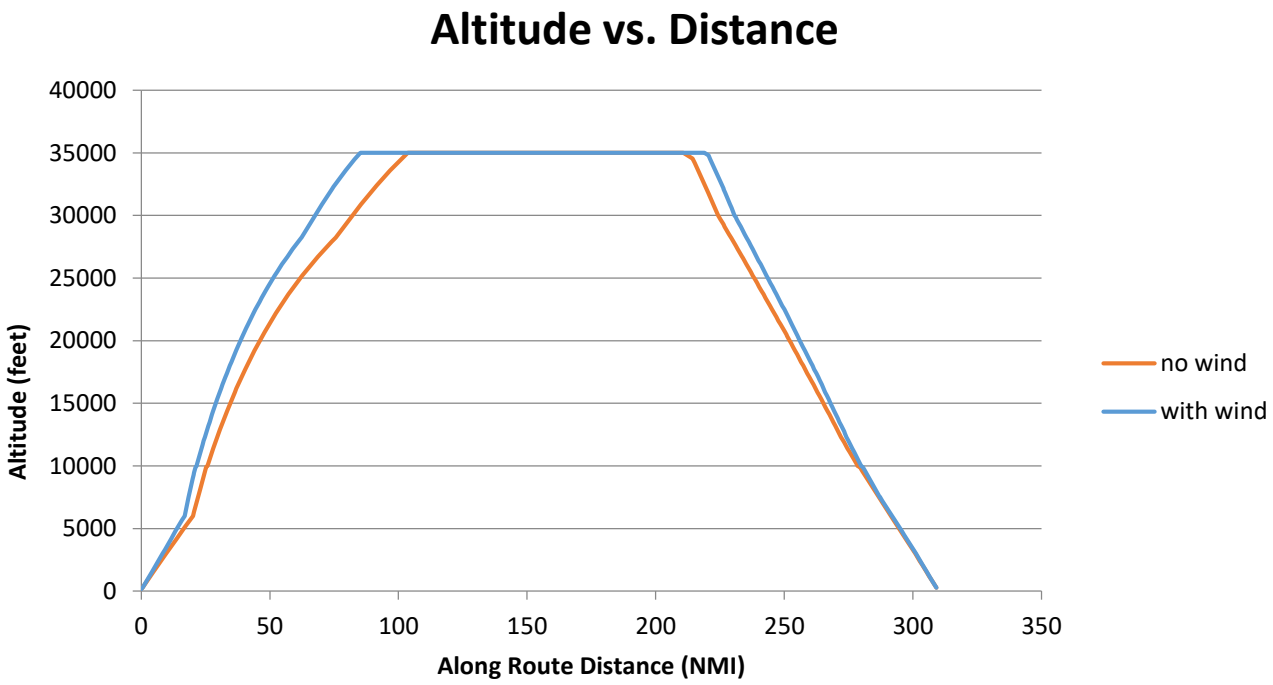


Results – compared to ATC version



Results – with wind, without wind





- Climb/Descent rate is identical; so top of climb is achieved at the same time, but sooner in distance with wind (Rate of Climb/Descent depends on true airspeed, not ground speed)
- Cruise phase takes longer in time, flying into a head wind
- Total flight time is more than 6 minutes longer

Challenges

- Learning Python (including the python *numpy* package for numeric processing); especially choosing time-efficient implementations
- Accessing NOAA “GRIB2” files from python on windows
  - Used a utility from NOAA (“degrib”) to convert to text CSV files to read in python
  - One hour weather data takes approximately 3 minutes to process; once done, it can be used in trajectory generation repeatedly
  - Processing includes reading CSV file, and constructing a spherical earth model version of that by interpolating the values in the file (which are in a Lambert Conformal Conic grid)

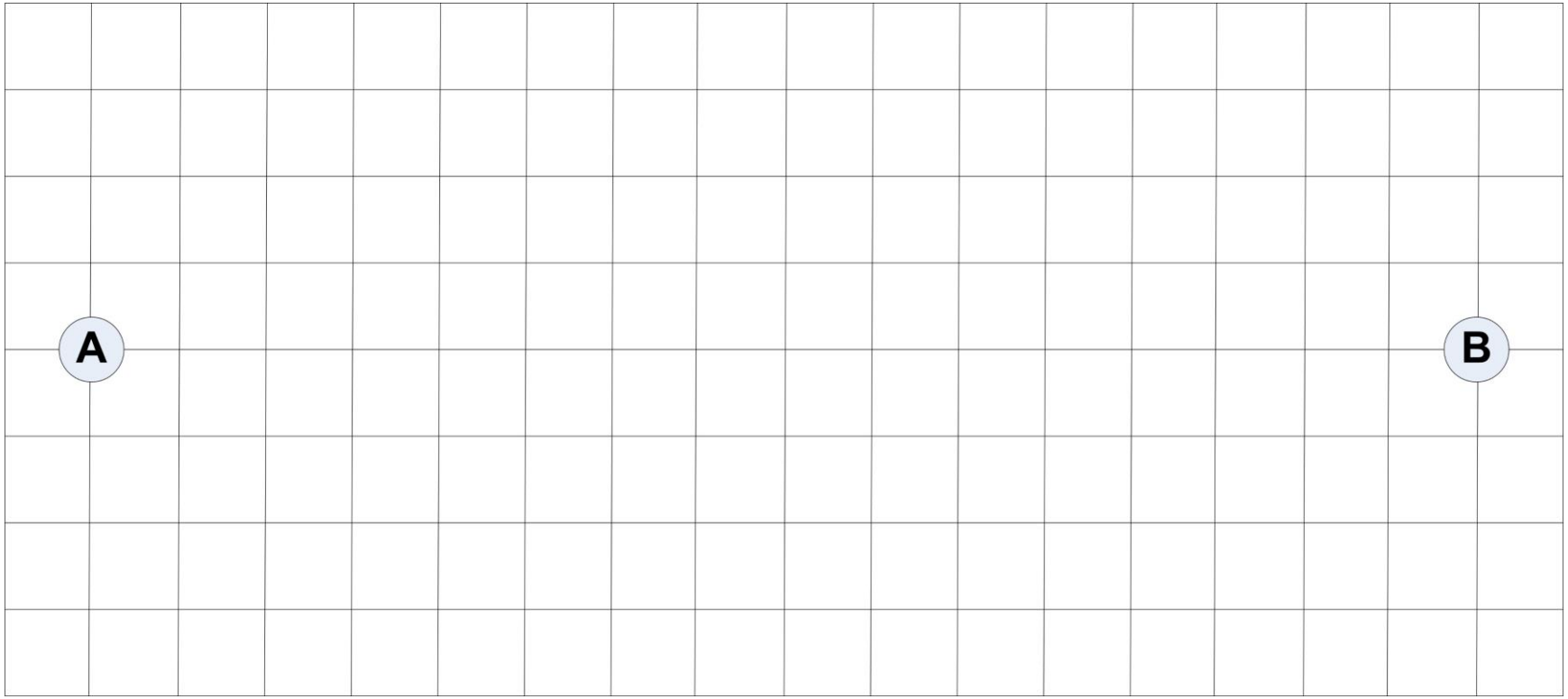
Particle Swarm Optimization (PSO) Algorithm

- Intent is to find the optimal (least fuel usage) trajectory between two fixed points, given the presence of winds
- PSO examines several possibilities through a field from point A to B, measuring fuel used for each path. Each possibility is a “particle”; examining all particles is one iteration.
- Points A and B will typically be after take off and before landing, as departure from and approach to airports are dictated by runway configurations
- PSO algorithms don’t guarantee that minimum solution will be found; rather they hope to find an acceptable solution in finite time

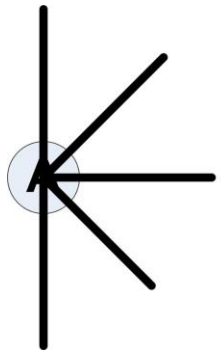
More PSO

- Path generation from A to B contain an element of randomness
- After PSO computes lat/long points, trajectories will be generated following those points, and fuel consumption will be calculated. The best choice for that iteration is then known.
- Subsequent iterations randomly vary starting points around that best choice, and algorithm concludes when the improvement is less than some epsilon.

Points A and B exist in a wind grid



Initialization

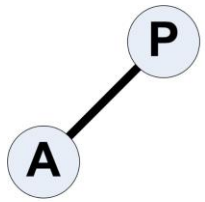


The algorithm will try several different starting directions, all beginning at point A

B

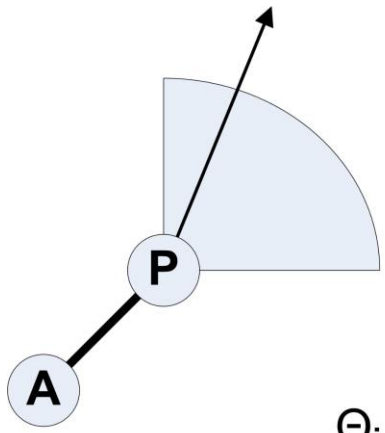
Each segment (starting at A) is the same length.

Initial step determines the first point



For each path modeled, the next point on the path will be determined by 3 factors

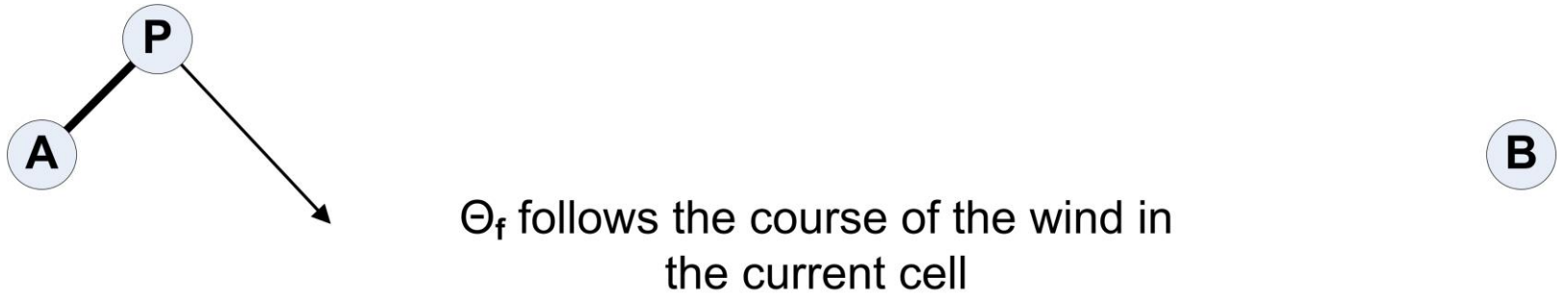
Three courses are used; θ_i (for inertia)



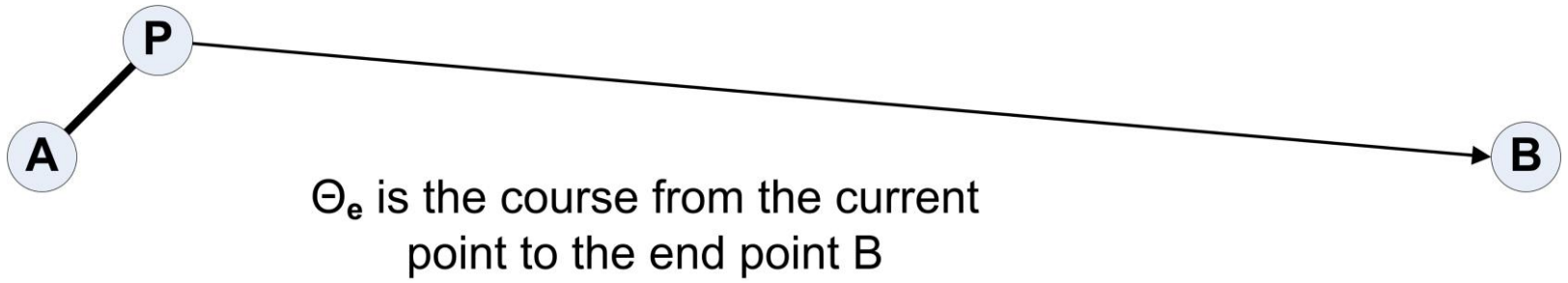
Θ_i is generally in the direction of the current course, with some randomness applied

B

θ_f (for fuel used)



θ_e (for end-point)



Next Point

- The exit course from our current point is a weighted sum of these three courses:

$$\theta = W_i\theta_i + W_e\theta_e + W_f\theta_f$$

The sum of all weights = 1.0

W_e is chosen to increase as the end point is approached:

$$W_e = (1 - \frac{dist(P,B)}{dist(A,B})$$

Weights

- Once W_e is determined, the other two weighting factors are chosen to make up the remainder of the weight, according to a pre-determined percentage; for example the fuel weight could get 70% of the remaining and the inertia weight could get 30%
- Increasing W_e as we get closer to point B ensures the path will converge on B.
- Resultant path will be some zig-zag line between A and B; this is then smoothed (as airplanes don't fly in zig-zags), a trajectory is generated following those points, and total fuel consumption calculated
- Finally, the trajectory with the lowest fuel consumption is chosen as the "best" path

Summary

- Work to date has produced a Trajectory Generation Engine that is good enough to perform other experiments
- Originally proposed four experiments:
  - Use of forecasted weather – on track for a **December** completion
  - Create optimal wind-aided trajectories using PSO – initial implementation to be done by end of **January**; final implementation by **end of semester**
  - Multi-threaded, perhaps on GPU cores, conflict detection – initial implementation to be done by end of **February**, final by **end of semester**
  - Compare BADA 3 vs. BADA 4 (which adds additional parameters to the thrust/drag equations) – this will be lower priority (it will require another Eurocontrol license request), and may not be achieved